International Journal on Advances in Internet Technology

2014 vol. 7 nr. 1&2

The *International Journal on Advances in Internet Technology* is published by IARIA.
ISSN: 1942-2652
journals site: http://www.iariajournals.org
contact: petre@iaria.org

Sponsored by IARIA
www.iaria.org

International Journal on Advances in Internet Technology
Volume 7, Number 1 & 2, 2014

Bill Grosky, University of Michigan-Dearborn, USA
Vic Grout, Glyndŵr University, UK
Jason Gu, Singapore University of Technology and Design, Singapore
Christophe Guéret, Vrije Universiteit Amsterdam, Nederlands
Frederic Guidec, IRISA-UBS, Université de Bretagne-Sud, France
Bin Guo, Northwestern Polytechnical University, China
Gerhard Hancke, Royal Holloway / University of London, UK
Arthur Herzog, Technische Universität Darmstadt, Germany
Rattikorn Hewett, Whitacre College of Engineering, Texas Tech University, USA
Nicolas Hidalgo, Yahoo! Research Latin America, France
Quang Hieu Vu, EBTIC, Khalifa University, Arab Emirates
Hiroaki Higaki, Tokyo Denki University, Japan
Eva Hladká, Masaryk University, Czech Republic
Dong Ho Cho, Korea Advanced Institute of Science and Technology (KAIST), Korea
Anna Hristoskova, Ghent University - IBBT, Belgium
Ching-Hsien (Robert) Hsu, Chung Hua University, Taiwan
Christian Hübsch, Institute of Telematics, Karlsruhe Institute of Technology (KIT), Germany
Chi Hung, Tsinghua University, China
Edward Hung, Hong Kong Polytechnic University, Hong Kong
Linda A. Jackson, Michigan State University, USA
Raj Jain, Washington University in St. Louis , USA
Edward Jaser, Princess Sumaya University for Technology - Amman, Jordan
Terje Jensen, Telenor Group Industrial Development / Norwegian University of Science and Technology, Norway
Yasushi Kambayashi, Nippon Institute of Technology, Japan
Georgios Kambourakis, University of the Aegean, Greece
Atsushi Kanai, Hosei University, Japan
Henrik Karstoft , Aarhus University, Denmark
Dimitrios Katsaros, University of Thessaly, Greece
Ayad ali Keshlaf, Newcastle University, UK
Reinhard Klemm, Avaya Labs Research, USA
Samad Kolahi, Unitec Institute Of Technology, New Zealand
Dmitry Korzun, Petrozavodsk State University, Russia / Aalto University, Finland
Evangelos Kranakis, Carleton University - Ottawa, Canada
Slawomir Kuklinski, Warsaw University of Technology, Poland
Andrew Kusiak, The University of Iowa, USA
Mikel Larrea, University of the Basque Country UPV/EHU, Spain
Frédéric Le Mouël, University of Lyon, INSA Lyon / INRIA, France
Nicolas Le Sommer, Université Européenne de Bretagne, France
Juong-Sik Lee, Nokia Research Center, USA
Wolfgang Leister, Norsk Regnesentral (Norwegian Computing Center), Norway
Clement Leung, Hong Kong Baptist University, Hong Kong
Man-Sze Li , IC Focus, UK
Longzhuang Li, Texas A&M University-Corpus Christi, USA
Yaohang Li, Old Dominion University, USA
Jong Chern Lim, University College Dublin, Ireland
Lu Liu, University of Derby, UK

Karim Mohammed Rezaul, Glyndwr University, UK
Leon Reznik, Rochester Institute of Technology, USA
Joel Rodrigues, Instituto de Telecomunicações / University of Beira Interior, Portugal
Simon Pietro Romano, University of Napoli Federico II, Italy
Michele Ruta, Politecnico di Bari, Italy
Jorge Sá Silva, University of Coimbra, Portugal
Farzad Salim, Queensland University of Technology, Australia
Sébastien Salva, University of Auvergne, France
Ahmad Tajuddin Samsudin, Telekom Malaysia Research & Development, Malaysia
Josemaria Malgosa Sanahuja, Polytechnic University of Cartagena, Spain
Luis Enrique Sánchez Crespo, Sicaman Nuevas Tecnologías / University of Castilla-La Mancha, Spain
Paul Sant, University of Bedfordshire, UK
Brahmananda Sapkota, University of Twente, The Netherlands
Alberto Schaeffer-Filho, Lancaster University, UK
Peter Schartner, Klagenfurt University, System Security Group, Austria
Rainer Schmidt, Aalen University, Germany
Thomas C. Schmidt, HAW Hamburg, Germany
Didier Sebastien, University of Reunion Island, France
Zary Segall, Chair Professor, Royal Institute of Technology, Sweden
Dimitrios Serpanos, University of Patras and ISI/RC ATHENA, Greece
Jawwad A. Shamsi, FAST-National University of Computer and Emerging Sciences, Karachi, Pakistan
Michael Sheng, The University of Adelaide, Australia
Kazuhiko Shibuya, The Institute of Statistical Mathematics, Japan
Roman Y. Shtykh, Rakuten, Inc., Japan
Patrick Siarry, Université Paris 12 (LiSSi), France
Jose-Luis Sierra-Rodriguez, Complutense University of Madrid, Spain
Simone Silvestri, Sapienza University of Rome, Italy
Åsa Smedberg, Stockholm University, Sweden
Vasco N. G. J. Soares, Instituto de Telecomunicações / University of Beira Interior / Polytechnic Institute of Castelo Branco, Portugal
Radosveta Sokullu, Ege University, Turkey
José Soler, Technical University of Denmark, Denmark
Victor J. Sosa-Sosa, CINVESTAV-Tamaulipas, Mexico
Dora Souliou, National Technical University of Athens, Greece
João Paulo Sousa, Instituto Politécnico de Bragança, Portugal
Kostas Stamos, Computer Technology Institute & Press "Diophantus" / Technological Educational Institute of Patras, Greece
Vladimir Stantchev, SRH University Berlin, Germany
Tim Strayer, Raytheon BBN Technologies, USA
Masashi Sugano, School of Knowledge and Information Systems, Osaka Prefecture University, Japan
Tae-Eung Sung, Korea Institute of Science and Technology Information (KISTI), Korea
Sayed Gholam Hassan Tabatabaei, Isfahan University of Technology, Iran
Yutaka Takahashi, Kyoto University, Japan
Yoshiaki Taniguchi, Osaka University, Japan
Nazif Cihan Tas, Siemens Corporation, Corporate Research and Technology, USA
Alessandro Testa, University of Naples "Federico II" / Institute of High Performance Computing and Networking

(ICAR) of National Research Council (CNR), Italy
Stephanie Teufel, University of Fribourg, Switzerland
Parimala Thulasiraman, University of Manitoba, Canada
Pierre Tiako, Langston University, USA
Ioan Toma, STI Innsbruck/University Innsbruck, Austria
Orazio Tomarchio, Universita' di Catania, Italy
Kurt Tutschku, University Blekinge Institute of Technology, Karlskrona, Sweden
Dominique Vaufreydaz, INRIA and Pierre Mendès-France University, France
Massimo Villari, University of Messina, Italy
Krzysztof Walkowiak, Wroclaw University of Technology, Poland
MingXue Wang, Ericsson Ireland Research Lab, Ireland
Wenjing Wang, Blue Coat Systems, Inc., USA
Zhi-Hui Wang, School of Softeware, Dalian University of Technology, China
Matthias Wieland, Universität Stuttgart, Institute of Architecture of Application Systems (IAAS),Germany
Bernd E. Wolfinger, University of Hamburg, Germany
Chai Kiat Yeo, Nanyang Technological University, Singapore
Mark Yampolskiy, Vanderbilt University, USA
Abdulrahman Yarali, Murray State University, USA
Mehmet Erkan Yüksel, Istanbul University, Turkey

International Journal on Advances in Internet Technology
Volume 7, Numbers 1 & 2, 2014

CONTENTS

A Comprehensive Evaluation of a Bitmapped XML Update Handler

Mohammed Al-Badawi, Abdallah Al-Hamdani, and Youcef Baghdadi
Department of Computer Science
Sultan Qaboos University
Muscat, Oman
{mbadawi, abd, ybaghdadi}@squ.edu.om

***Abstract*—XML (eXtensible Markup Language) update is problematic for many XML databases. The main issue tackled by the existing (and new) XML storages and indexing techniques is the cost reduction of updating the XML's hierarchal structure inside these storages. PACD (an acronym for Parent-Ancestor/Child-Descendent), as bitmapped XML processing technique introduced earlier, is an attempt in this direction. The technique brings the cost of updating the XML structure to the data representation level by introducing the 'next' and 'previous' axes as a mechanism to preserve the document order, and then using well-established matrix-based operations to manipulate the database transactions. This paper mainly provides a complexity analysis of the PACD update framework and presents a novel experimental evaluation method (in terms of comprehensiveness and completeness) for its update primitives. The outcomes of this evaluation have shown that the cost of eight update primitives (out of nine provided by PACD) locates under an acceptable range of a constant 'c', where 'c' is an extremely small number comparing to the number of nodes 'n' in the XML tree. Such good performance is lacked in the comparable techniques.**

Keywords-XML Databases; XML/RDBMS Mapping; XML Update; XML Indexing; Complexity Analysis; Experimental Design.

I. INTRODUCTION

Data stored in the extensible markup language (XML) containers (databases) is subject to update when circumstances change [1]. Unfortunately, handling XML updates is a common problem in the existing XML storages and optimization techniques. Relational approaches using node labeling techniques [2][3][4][5][6][7][8][9][10][11] [12][13] require a large number of renumbering operations in order to keep the node labels updated whenever a node is inserted, deleted or moved from one location to another in the XML tree. For the approaches that use path summaries to encode the XML hierarchical structure [14][15][16][17][18], an additional cost results from updating these summaries. In native XML approaches such as sequence based [19][20][21][22][23] and feature based techniques [24][25][26], the update problem is even worse. In the first case, the consequences of a single update operation (for example deleting a node) can affect thousands locations in the corresponding sequence depending on the node location in the XML tree. A similar problem occurs in the case of feature based techniques, which rely on encoding the relationship between the nodes and the different ePaths of the XML tree inside what is called feature-based matrices [24].

PACD is XML processing technique introduced in [28] [29] that brings the cost of updating the XML hierarchal structure to the data representation level by encoding these structures into a set of structure-based matrices each of which encodes a specific XPath [27] axis, plus two more axes specifically introduced by PACD to preserve the document order. Thus, PACD architecture combines some matrix-based operations along with the bit-wise operations to reduce the cost of querying and updating the structure of underlying XML file. This paper extends our previous work [1] by providing a detailed complexity analysis of the PACD Updates Query Handler (UQH). Unlike many existing studies, this paper presents a comprehensive evaluation process, which provides 1) a full algorithmic listing of all XML update primitives so that they can be re-used, 2) a detailed cost-analytical procedure of the XML update primitives, and 3) a supportive comprehensive experimental procedure that considers several testable aspects of the XML databases. Such evaluation method could be adopted by the XML research and development community to evaluate XML database processing techniques.

The paper starts by revisiting the PACD's framework in Section II. Then it introduces the UQH framework in Section III, while Section IV puts forward assumptions to facilitate the discussion of complexity analysis in the subsequent sections. Sections V to VII provide a detailed discussion of three types of update primitives: the insertion, deletion and change primitives, respectively. The overall complexity analysis and a supportive experimental evaluation are given in Sections VIII and IX, respectively. Section X concludes the paper.

II. BACKGROUND: PACD'S XML PROCESSING MODEL

PACD, introduced in [28][29], is a bitmap XML processing technique consisting of three main components: the Index Builder (IB; operations I.1-I.4), the Query Processor (QP) and the Update Query Handler (UQH). The IB (see Figure 1) shreds the XML hierarchal structure (derived by the XPath's thirteen axes and their extension; the Next and Previous axes [28]) into a set of binary relations each of which is physically stored as an n×n bitmap matrix. An entry in any matrix is '1' if there is a corresponding relationship between the coupled nodes or '0' otherwise [30][25]. The IB operations I.2-I.4 are responsible to reduce

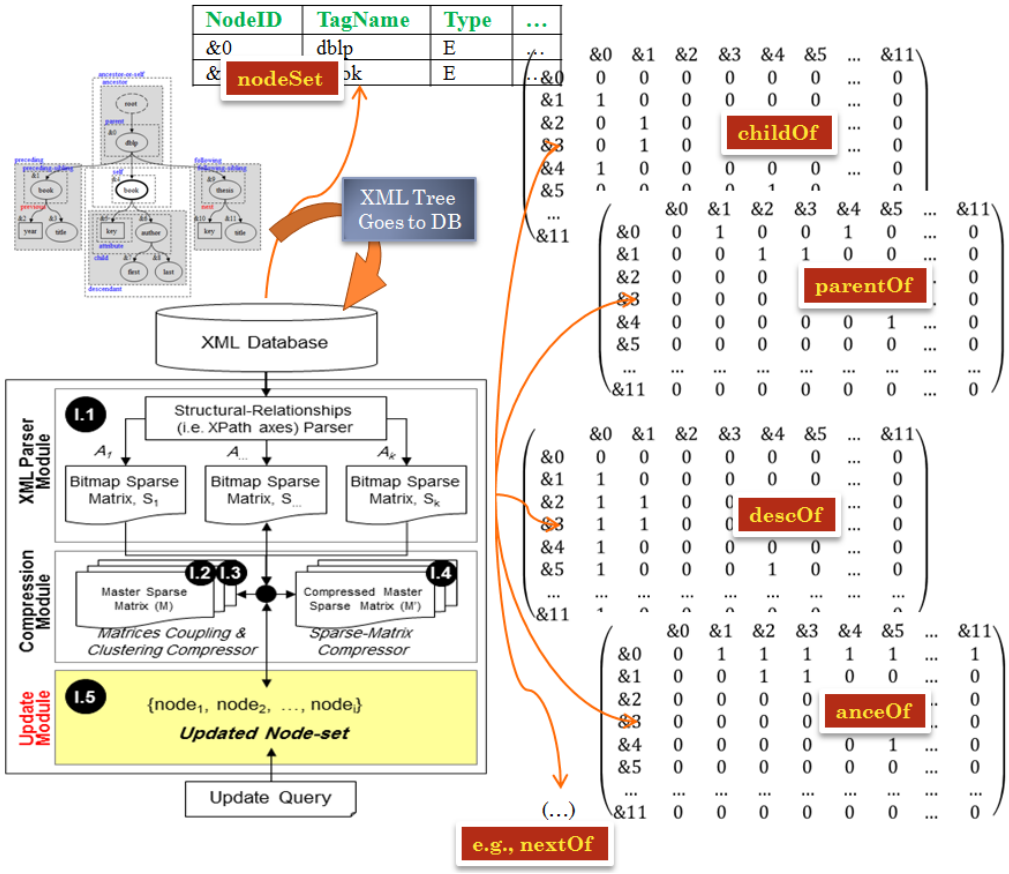


Figure 1. PACD Framework

the size of storing the XML structure by applying three levels of compression: the matrix-transformation level, the matrix-coupling level and the sparse-matrix compression level. More details about the data compression mechanism in particular and the IB in general can be found in [28][29].

On the other hand, the QP performs all operations related to the search-query execution. The full architecture of the QP was described in [29] but in brief, the process starts by analyzing the search-query statement to identify the affected nodes based on the twig structure. The process also identifies the query base matrices and draws an execution plan for the entire query, which eventually returns the results into a tabular-format (i.e., sub-matrices) and then converted to an XML data layout.

The next section describes the PACD's third component, that is the UQH, the core subject discussed in this paper.

III. The Update Handler

The PACD's UQH is responsible for all update operations, which includes the translation of the update query, the identification of update primitive(s), and the primitive execution.

Once the query is translated (e.g., from XQuery syntax to an SQL statement), the UQH starts identifying the node(s) that are affected by the update command/query. It navigates through the finite-state-machine (FSM) version of the update query in order to identify the affected node-set. Once the target node-set is known, the UQH determines and calls the appropriate update primitive (see Table I). PACD supports update primitives for single node insertion and deletion, twig insertion and deletion, and textual and structural contents changes.

The update primitive acts on all PACD's components including the NodeSet container and the structure based matrices (i.e., childOf, descOf and nextOf). Each update primitive executes certain instructions over each component such as adding new columns and rows and changing the bitmapped entries within the matrices. The cost of the update query execution will be the lump sum of the costs of executing all derived update primitives over each PACD's component. For example, an 'insert' primitive will involve adding one or more rows and columns to the bitmapped matrices, as well as adding one or more entries to the NodeSet container. Thus, the cost of the 'insert' operation becomes the cost of inserting the node information inside the NodeSet container plus the cost of inserting one row and column inside the childOf, descOf and nextOf matrices. More examples on using update primitives will be given later during the discussion of the update primitives.

The above steps are summarized in the algorithm provided in Figure 2, whereas Table I lists out the update primitives that are currently supported by PACD's UQH.

```
1   INPUT: update-query
2   OUTPUT: none
3   Construct the FSM execution plan of the corresponding twig
4   node-set = the returned node-set from the FSM execution
5   Using the update-query syntax, determine the update-primitive(s)
6   Call the update-primitive(s) with the obtained node-set:
7       Alter the NodeSet container;
8       Alter the childOf matrix;
9       Alter the descOf matrix;
10      Alter nextOf matrix;
11  End;
```

Figure 2. PACD Update Handler Algorithm

TABLE I: PACD UPDATE HANDLER PRIMITIVES

Insertion	insertLeaf	adds a leaf node
	insertNonLeaf	adds an internal node
	insertTwig	adds a single-rooted, connected sub-tree
Deletion	deleteLeaf	removes a leaf node
	deleteTwig	removes a single-rooted, connected sub-tree
Updating	changeName	renames an element or attribute name
	changeValue	edits the value (text) of an attribute (element)
	shiftNode	moves a node from one place to another
	shiftTwig	moves a single-rooted, connect sub-tree from one place to another

IV. ASSUMPTIONS AND AN ANALYTICAL PROCEDURE

This section lists some assumptions that are considered during the complexity and experimental results analysis. The analytical procedure of the experimental results is also described here.

A. Assumptions During the Analysis

During the analysis of the above XML update primitives, the cost of any update primitive counts the number of work-units done by the underlying system in order to update every PACD's component. So, each of the following operation is counted as a *single* work-unit:

- Operations on the NodeSet container:
  - ◊ Insert new record/row
  - ◊ Delete a record/row
  - ◊ Change one (or more) attributes/fields within the record/row
- Operations on a matrix-based component (e.g., childOf):
  - ◊ Insert a complete row or column
  - ◊ Delete an entire row or column
  - ◊ Change an entry of a matrix (i.e., change the status from '0' to '1' or vice versa)

As for illustration, inserting a leaf-node requires the insertion of a new record inside the NodeSet container (1 unit), the addition of one row and column to the childOf, descOf and nextOf matrices (6 units), and may change at most one entry in the nextOf matrix (1 unit). So the leaf-node insertion process costs 8 work-units (or hits).

In addition, the analyses provided in this paper were done based on the following assumptions:

- When a row or column is inserted into a matrix, its entries are set to zero by default with no extra cost.
- The cost of 'search' operations (locating the records) inside the PACD storage components; for example, fetching the node ID among the NodeSet container, is set to zero assuming that a very efficient lookup algorithm is used.
- The number of children at any arbitrary node in the XML tree is 'α', where α is a small number comparing to the number of nodes 'n' for very large XML databases
- The number of descendants at any arbitrary node in the XML tree can be estimated by multiplying the number of nodes 'n' by a fraction 'f', where $0 \le f \le 1$. The value of 'f' decreases exponentially as the context node goes from the root (where f=1) towards the leaf nodes (where f=0) [31].
- The given algorithms and their analyses are based on using the uncompressed PACD storage. Updating compressed PACD storage (which discussion is outside the scope of this paper) may involve additional steps and extra cost depending on the compression technique used.

Generally speaking, the above assumptions were made in order to simplify the analyses provided in the subsequent sections (Section V, VI and VII). The same assumptions also applied during the experimental result discussion in Section IX.

B. An Anlytical Procedure

During the discussion of each update primitive in the following sections, the usage of the primitive (including the function prototype), its pseudo-code, the complexity discussion, and one or two examples will be provided in separate subsections. Furthermore, all examples are based on the XML tree illustrated in Figure 3.

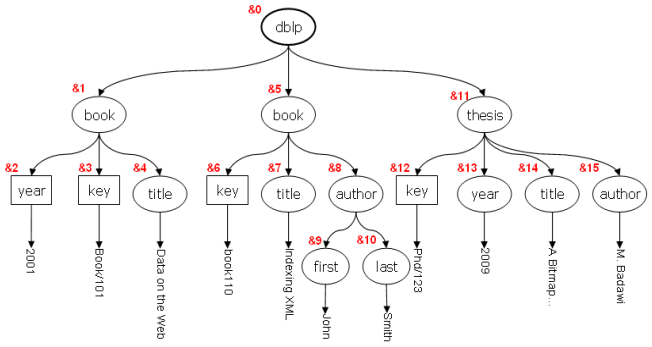


Figure 3. An XML Tree Example

V. INSERTION PRIMITIVES

This section discusses the three insertion primitives shown in Table I.

A. Leaf Node Insertion

1) Usage:

Syntax:	insertLeaf(node_info, parentID [,precID])
Description:	Inserts a node at the bottom-most level of the tree under

	the *parentID* node and next to *precID* node. Both the *parentID* and *precID* are identified by the UQH
Argument(s):	◊ node_info: all necessary information to fill the NodeSet record including the nodeID, tag/attribute name, node_type, and the value/textual content ◊ parentID: the ID of the parent node where the new node to be inserted ◊ precID: the ID of the preceding node. Must be specified in case of the order-preserving storage

2) Algorithm:

```
1  PROGRAM insertLeaf(node_info: nodeType, parentID:
   nodeIDType, precID: nodeIDType)
2    Get the next nodeID;
3    Insert the node information into NodeSet;
4    *-- update the childOf matrix:
5    Add a row and column to the 'childOf';
6    Set:
7        childOf[nodeID,parenID] = '1',
8    *--update the descOf matrix:
9    Add a row and column to the 'descOf';
10   Let: anceSet = {node(i), where descOf[parentID,i]
   = '1'} ∪ parentID;
11   For each i ∈ anceSet:
12       Set: descOf[nodeID,i] = '1';
13   *--update the nextOf matrix:
14   Add a row and column to the 'nextOf';
15   If precID ≠ null:
16       Let: temp = node(i), where nextOf[i,precID] =
   '1';
17       Set: nextOf[nodeID,precID] = '1';
18       If temp ≠ null:
19           Set: nextOf[temp,precID] = '1';
20 PROGRAM_END.
```

3) Complexity Analysis:

Based on the assumption given above, inserting the node's information into the NodeSet container requires one hit (line 3), whereas updating the childOf matrix requires three hits: two to add a row and column (line 5) and one to set the child/parent relationship between the new node and the parentID (line 7). Similarly, updating the descOf matrix requires 2+h hits: two to add a row and column (line 9) and a maximum of 'h' hits (*where 'h' is the maximum height of the XML tree*) to set the descendant/ancestor relationship between the new node and its ancestor list, which is calculated in Line 10 (see Lines 11-12). In terms of the nextOf matrix, besides the two hits that are required to insert a row and column to the matrix (line 14), the program makes two additional hits to update the previous/next relationship (lines 17 and 19). So the total work-units required to insert a leaf node in an XML tree of height 'h' is 10+h. This is a very small number 'c' comparing to the number of nodes 'n'; thus the complexity is of order O(c).

4) Example: Using the database in Figure 3, insert the 'year' information (e.g., 2003) to the book identified by the key 'book/110', where the 'year' information must precede the 'author' information (result given in Figure 4).

The cost breakdown is:

NodeSet	childOf	descOf	nextOf	**Total**
1	3	4	4	**12 hits**

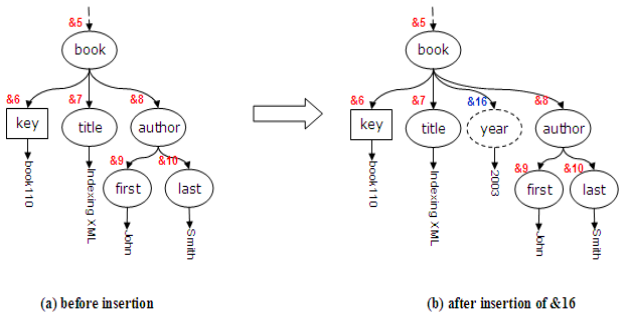


Figure 4. A Leaf Node Insertion Example

B. Non-Leaf Node Insertion

1) Usage:

Syntax:	insertNonLeaf(node_info, parentID [,precID])
Description:	Inserts a node at any level of the tree except the lowest level. The *parentID* and the *precID* are identified by the UQH prior calling the primitive. At this stage, this primitive is only used to add additional level between a parent and the complete set of its children. Subdividing the parentID's children between the existing parent and the new node is left to further investigation.
Argument(s):	◊ node_info: all necessary information to fill the NodeSet record including the nodeID, tag/attribute name, node_type, and the value/textual content ◊ parentID: the ID of the parent node where the new node to be inserted ◊ precID: the ID of the preceding node. Must be specified in case of the order-preserving storage

2) Algorithm:

```
1  PROGRAM insertNonLeaf(node_info:nodeType,
   parentID:nodeIDType,precID: nodeIDType)
2    Get the next nodeID;
3    Insert the node information into NodeSet;
4    *-- update the childOf matrix:
5    Add a row and column to the 'childOf';
6    Let: childSet = {node(i), where chilOf[i,parentID]
   = '1'}
7    For each i ∈ childSet:
8       Set: chilOf[i,nodeID] = '1';
9    Set: childOf[nodeID,parentID] = '1';
10   *--update the descOf matrix:
11   Add a row and column to the 'descOf';
12   Let: anceSet = {node(i), where descOf[parentID,i]
   = '1'} ∪ parentID;
13   Let: descSet = {node(j), where descOf[j,parentID]
   = '1'};
14   For each i ∈ anceSet:
15      Set: descOf[nodeID,i] = '1';
16   For each j ∈ descSet:
17      Set: descOf[j,nodeID] = '1';
18   *--update the nextOf matrix:
19   Add a row and column to the 'nextOf';
20   If precID ≠ null:
21      Let: temp = {node(i), where nextOf[i,precID] =
   '1'};
22      Set: nextOf[nodeID,precID] = '1';
23      If temp ≠ null:
24         Set: nextOf[temp,precID] = '1';
25 PROGRAM_END.
```

3) Complexity Analysis:

This primitive also requires one hit to insert inside the NodeSet container (line 3). However, more work is required to update the childOf matrix because the children of the parental node 'parentID' have to be assigned to the new node. So the number of hits required to update the childOf matrix is '1+α', where 'α' is the number of children of the context node at an arbitrary level in the XML tree.

To update the descOf matrix, the primitive has to assign the ancestors of the 'parentID' to the new node 'nodeID' (lines 14-15) and the descendants of the 'parentID' as descendant from the new node (lines 16-17). The first process requires no more than 'h' hits, while the cost of the second process may extend to 'n' hits; but in reality it only requires a factor of 'n' hits depending on the insertion level (see Section IV). Finally, the cost of updating the nextOf matrix is the same for updating the nextOf matrix in the previous primitive (lines 22 and 24).

4) Example: Using the database in Figure 3, assign the current author of the book titled 'Indexing XML' to be the first author of the book so that other authors can be added later. This requires adding a parent node called 'au_det' for the 'first' and 'last' nodes under the original 'author' node (result given in Figure 5).

The cost breakdown is:

NodeSet	childOf	descOf	nextOf	**Total**
1	5	6	0	**12 hits**

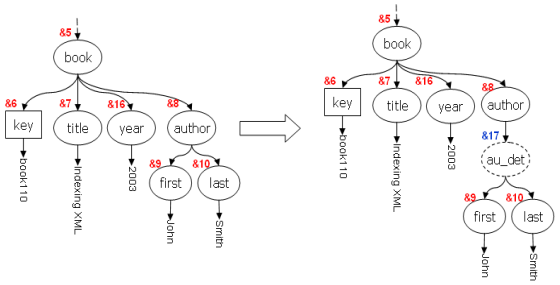


Figure 5. An Non-leaf Node Insertion Example

C. Twig Insertion

1) Usage:

Syntax:	insertTwig(twig_info, parentID [,precID])
Description:	Inserts a sub-tree of 'm' nodes under the *parentID* and after the *precID*. Both the parentID and the precID are determined by the UQH, and the twig is only inserted at bottom-most nodes
Argument(s):	◊ twig_info: all necessary information to fill the NodeSet record including the nodeID, tag/attribute names, node types, and the value/textual contents ◊ parentID: the ID of the parent node where the new twig to be inserted ◊ precID: the ID of the preceding node. Must be specified in case of the order-preserving storage

2) Algorithm:

The twig insertion can be modeled as inserting multiple-connected nodes. In other words, inserting a twig of 'm' nodes requires 'm' times the cost of inserting a single leaf-node and can be performed by the same algorithm in Section V(C) starting at the twig root node.

3) Complexity Analysis:

The cost of this primitive is 'm' times the cost of inserting a single leaf-node, where 'm' is the number of nodes inside the inserted twig.

4) Example: Using the database in Figure 3, add second author information (i.e., including the 'first' and 'last' name) to the book titled 'Indexing XML' (result given in Figure 6)

The cost breakdown is:

NodeSet	childOf	descOf	nextOf	**Total**
3	9	14	8	**34 hits**

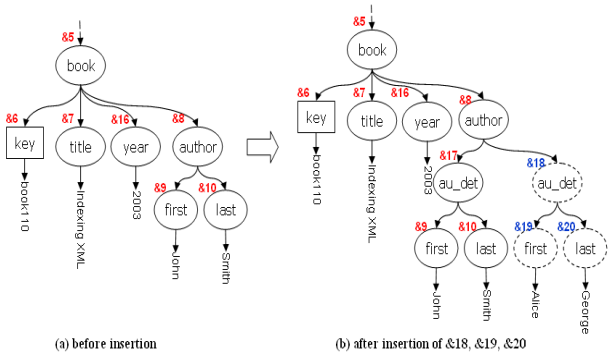


Figure 6. A Twig Insertion Example

D. Insertion Primitives Summary

Table II summarizes the number of work-units required to conduct the insertion primitives.

VI. Delete Primitives

PACD currently supports the 'deleteLeaf' and 'deleteTwig' primitives. These are discussed below.

A. Leaf Node Deletion

1) Usage:

Syntax:	deleteLeaf(nodeID)
Description:	Deletes a node from the lowest level of the tree labeled with *nodeID* that is returned by the UQH
Argument(s):	◊ nodeID: the unique node ID of the deleted node

2) Algorithm:

```
1   PROGRAM deleteLeaf(nodeID: nodeIDType)
2     *-- update the childOf matrix:
3     Locates the corresponding row and column of the
    nodeID inside the 'childOf';
4     Remove the row and column from the 'childOf';
5     *--update the descOf matrix:
6     Locates the corresponding row and column of the
    nodeID inside the 'descOf';
7     Remove the row and column from the 'descOf';
8     *--update the nextOf matrix:
9     Let:
10       next = {node(i), where nextOf[i,nodeID] = '1'};
11       prev = {node(j), where nextOf[nodeID,j] = '1'};
12    Locates the corresponding row and column of the
    nodeID inside the 'nextOf';
13    Remove the row and column from the 'nextOf';
14    If next ≠ null AND prev ≠ null:
```

```
15      Set: nextOf[next,prev] = '1';
16    *--update the NodeSet container:
17    Locate the corresponding record of the nodeID
   inside the 'NodeSet';
18    Delete the nodeID;
19  PROGRAM_END.
```

3) Complexity Analysis:

Deleting a leaf node is simple and straightforward. In the childOf and descOf matrices, after locating the row and column IDs of the target node, the update process simply removes that row and column. Thus, the process involves two work units for each matrix. Regarding the deletion from the nextOf matrix, a special consideration is required when the target node has previous (line 11) and next (line 10) siblings. In this case, an extra hit is required to assign the *next* node of the target node to be the *next* node of the *previous* node of the target node. Finally, to remove the node from the NodeSet container, the system performs one work unit after locating the record of the target node (line 15). So the 'deleteLeaf' primitive does not do more than *eight* work units to remove a node from the PACD's storage.

4) Example: Using the database in Figure 3, remove the author's last-name from the book identified by the key 'book/110' (result given in Figure 7).

The cost breakdown is: (Note: the node ID &10 will be recycled)

childOf	descOf	nextOf	NodeSet	**Total**
2	2	2	1	**7 hits**

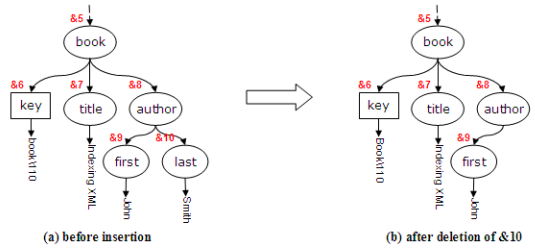


Figure 7. A Leaf Node Deletion Example

B. Twig Deletion

1) Usage:

Syntax:	deleteTwig(twigRootNodeID)
Description:	Deletes a connected sub-tree rooted at 'twigNRootNodeID' from the XML tree. The twigRootNodeID is returned by the UQH process
Argument(s):	◊ twigRootNodeID: the node ID of twig's root node

2) Algorithm:

```
1   PROGRAM deleteTwig(twigRootNodeID: nodeIDType)
2     *-- reconnect the next_of list of the nextOf
   matrix:
3     Let:
4        next = {node(i), where nextOf[i,twigRootNodeID]
   = '1'};
5        prev = {node(j), where nextOf[twigRootNodeID,j]
   = '1'};
6     If next ≠ null AND prev ≠ null:
7        Set: nextOf[next,prev] = '1';
8     *--identify all the node inside the deleted twig:
9     Let: descSet = {node(i), where descOf[i,
   twigRootNodeID] = '1'} ∪ twigRootNodeID;
10    *--remove row and columns from all matrices, and
   the node_info from the NodeSet :
11    For each i ∈ descSet:
12       Locates the corresponding row and column of the
   nodeID inside the 'childOf';
13       Remove the row and column from the 'childOf';
14       Locates the corresponding row and column of the
   nodeID inside the 'descOf';
15       Remove the row and column from the 'descOf';
16       Locates the corresponding row and column of the
   nodeID inside the 'nextOf';
17       Remove the row and column from the 'nextOf';
18       Locate the corresponding record of the nodeID
   inside the 'NodeSet';
19       Delete the nodeID;
20  PROGRAM_END.
```

3) Complexity Analysis:

Deleting a twig of 'm' nodes is very similar to deleting a leaf-node except that the cost is multiplied by 'm'. Furthermore, deleting a twig will involve only one reconnection process over the previous/next relationship. This process is performed to rearrange the previous/next relationship of the *previous* and the *next* nodes of the *root* node of the target twig (lines 3-7). So the maximum cost of the 'deleteTwig' primitive is '1+[m×(2+2+2+1)]' work units, where 'm' is the number of nodes inside the deleted twig.

4) Example: Using the database in Figure 3, remove the complete author's information from the book identified by the key 'book/110' (result given in Figure 8). Note: this will remove the nodes '&8' and '&9'.

The cost breakdown is:

childOf	descOf	nextOf	NodeSet	**Total**
4	4	4	2	**14 hits**

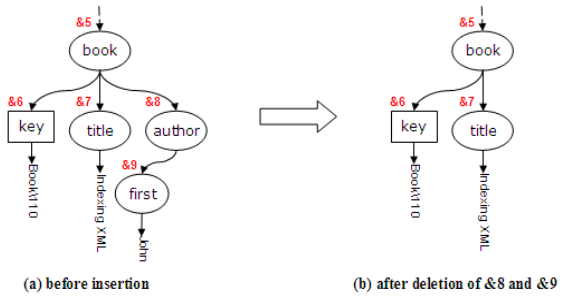


Figure 8. A Twig Deletion Example

C. Deletion Primitives Summary

Table III summarizes the number of work-units required to conduct the deletion primitives.

VII. CHANGE PRIMITIVES

Table I introduced four change primitives that can be used to rename node description (i.e., element and attribute names), change the value of a node, and move a node or a twig from one place to another inside the XML tree.

A. Node Description Change

1) Usage:

Syntax:	changeName(nodeID|oldName,newName)
Description:	Renames a node (identified by the *nodeID*) or a set of nodes (that have the same name identified by *oldName*) to the new name *newName*
Argument(s):	◊ nodeID: the node ID of a particular node ◊ oldName: the element or attribute name of a set of nodes ◊ newName: the new name to be assigned to the changed nodes

2) Algorithm:

```
1  *-----Case1: changing particular node's name:
2  PROGRAM changeName(nodeID: nodeIDType, newName:
   string)
3    *-- update the NodeSet container:
4    Locates the corresponding record of the nodeID in
   the 'NodeSet';
5    Replace the 'name' attribute by the 'newName';
6  PROGRAM_END;
7
8  *-----Case2: changing a set of  nodes' name:
9  PROGRAM changeName(oldName: string, newName: string)
10   *-- update the NodeSet container:
11   Let: updateSet = {node(i), where NodeSet.Name =
   oldName};
12   For each node ∈ updateSet:
13       Replace the 'name' attribute by the 'newName';
14 PROGRAM_END;
```

3) Complexity Analysis:

PACD separates the XML textual content representation from the structural content representation and manage them in a different storage component. The former (which includes the node ID, tag/attribute name, type and value) are arranged in the NodeSet container that stores the node information in a separate record. This arrangement makes it easier for the *textual-based* change operations such as the 'changeName' to alter node's record regardless the complexity of the XML's hierarchal structure. So, to change the name of a particular node, it will be sufficient to allocate that node in the NodeSet container and change the 'Name' attribute (lines 4-5). In the case of changing multiple node names such as changing an attribute or element name, the whole nodes labelled with that name class have to be changed. So, the 'changeName' primitive initially identifies all the nodes that share the same name (line 11) and then alters the 'Name' attribute of all identified nodes (lines 12-13). The complexity of this process depends on the distribution of the tag/attribute name in the XML tree, which might be estimated or obtained from the XML schema.

4) Example1: Using the database in Figure 3, change the name of the node 'thesis' to be 'phdthesis'.

This query changes the tag name of the node &11 from 'thesis' to 'phdthesis' with the cost of one work-unit.

B. Node Value Change

1) Usage:

Syntax:	changeValue(nodeID|oldName,newValue)
Description:	change the textual contents of a node (identified by the *nodeID*) or a set of nodes (that have the same name identified by *oldName*) to the new value *newValue*
Argument(s):	◊ nodeID: the node ID of a particular node ◊ oldName: the element or attribute name of a set of nodes ◊ newValue: the new textual content to be assigned to the nodes

2) Algorithm:

The algorithm of this primitive is identical to the one in Section VII-A(2).

3) Complexity Analysis:

The cost of this primitive is similar to the 'changeName' primitive, see Section VII-A(3).

4) Example: Using the database shown in Figure 3, change the publication year for the book labelled with 'Book/101' to be '2000' instead of '2001'.

This query changes the value of the node &2 from '2001' to '2000' with the cost of one work-unit.

5) Example: Using the database in Figure 3, change the 'title' of all publications to the uppercase.

In this query, the 'oldName' parameter is 'title' and the 'newValue' parameter is a function that converts its argument to the uppercase. The query will perform three work units in total.

C. Single Node Shifting

In the context of XML tree, single node shifting is only meaningful when the node is a leaf node. This can be used to transfer information from one block to another, for example to swap the first and second books' ID as in Figure 3. The NodeSet information is not affected by this primitive.

1) Usage:

Syntax:	shiftNode(nodeID,newParentID[,leftID])
Description:	Moves the node labeled with *nodeID* to under the node *newParentID*. If the exact location is required, the preceding node at the new location (i.e., *'leftID'*) must be specified
Argument(s):	◊ nodeID: the node to be moved ◊ newParentID: the parent node at the new location ◊ leftID: the preceding node at the new location

2) Algorithm:

```
1  PROGRAM shiftNode(nodeID: nodeIDType, newParentID:
   nodeIDType, leftID: nodeIDType)
2    *-- update the childOf matrix:
3    Let: oldParentID = {node(i), where
   childOf[nodeID,i] = '1'};
4    Set:
5        childOf[nodeID,newParentID] = '1';
6        childOf[nodeID,oldParentID] = '0';
7    *--update the descOf matrix:
8    Let:
9        oldAnceSet = {node(i), where descOf[nodeID,i]
   = '1'};
10       newAnceSet = {node(j), where
   descOf[newParentID,j] = '1'} ∪ newParentID;
11   For each node i ∈ newAnceSet:
12       Set: descOf[nodeID,i] = '1';
13   For each node i ∈ oldAnceSet:
14       Set: descOf[nodeID,i] = '0';
15   *--update the nextOf matrix:
16   Let:
17       next_of_nodeID = {node(i), where
```

```
   nextOf[i,nodeID] = '1'};
18       prev_of_nodeID = {node(j), where
   nextOf[nodeID,j] = '1'};
19       next_of_leftID = {node(i), where
   nextOf[i,leftID] = '1'};
20       prev_of_leftID = {node(j), where
   nextOf[leftID,j] = '1'};
21    Set (if any combination is not null):
22       nextOf[next_of_nodeID,prev_of_nodeID] = '1';
23       nextOf[nodeID,prev_of_nodeID] = '0';
24       nextOf[nodeID,leftID] = '1';
25       nextOf[next_of_leftID,nodeID] = '1';
26       nextOf[leftID,prev_of_leftID] = '0';
27       nextOf[next_of_leftID,leftID] = '0';
28 PROGRAM_END.
```

3) Complexity Analysis:

Moving a leaf node from one place to another releases the child/parent relationship between the node and its original parent and creates a new child/parent relationship between the node and the new parent. This requires two hits (lines 5 and 6). Similarly, in the descOf matrix, the shifting process releases the descendant/ancestor relationship between the node and its original ancestor list, and creates a new set of descendant/ancestor relationships between the node and the ancestors of the new parent. This requires no more than '2×h' hits, where 'h' is the maximum height of the XML tree (lines 11-14).

Updating the previous/next relationship for the 'shiftNode' is a bit complicated but it requires no more than six hits to release the old previous/next relationships and to set up the new ones (lines 22-27).

4) Example: Using the database in Figure 3, move the publication year of book 'book/101' to be the publication year for the book 'Book/110' (see Figure 9).

The cost breakdown is:

childOf	descOf	nextOf	**Total**
2	4	4	**10 hits**

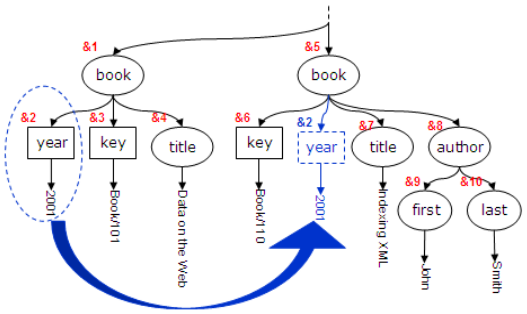


Figure 9. A Leaf Node Shifting Example

D. Twig Shifting

Twig shifting operations are useful when a sub-tree is moved from one parent to another without deleting the sub-tree and creating it again under the new parent.

1) Usage:

Syntax:	shiftTwig(twigRootID,newParentID[,leftID])
Description:	Moves a sub-tree (twig) rooted at the *twigRootID* to be a sub-tree under the node *newParentID*. If the exact location is required, the preceding node at the new location (i.e., *leftID'*) must be specified
Argument(s):	◊ twigRootID: the root of the twig to be moved ◊ newParentID: the parent node at the new location ◊ leftID: the preceding node of twig's root node at the new location

2) Algorithm:

```
1  PROGRAM shiftTwig(twigRootID: nodeIDType,
   newParentID: nodeIDType, leftID: nodeIDType)
2    *-- update the childOf matrix:
3    Let: oldParentID = {node(i), where
   childOf[twigRootID,i] = '1'};
4    Set:
5        childOf[twigRootID,newParentID] = '1';
6        childOf[twigRootID,oldParentID] = '0';
7    *--update the descOf matrix:
8    Let:
9        twigNodeSet = {node(1..m), where node(i) ∈
   twig};
10       oldAnceSet = {node(i), where
   descOf[twigRootID,i] = '1'};
11       newAnceSet = {node(j), where
   descOf[newParentID,j] = '1'} ∪ newParentID;
12   For each node i ∈ newAnceSet:
13       For each node j ∈ twigNodeSet:
14          Set: descOf[j,i] = '1';
15   For each node i ∈ oldAnceSet:
16       For each node j ∈ twigNodeSet:
17          Set: descOf[j,i] = '0';
18   *--update the nextOf matrix:
19   Let:
20       next_of_ twigRootID = {node(i), where
   nextOf[i, twigRootID] = '1'};
21       prev_of_ twigRootID = {node(j), where
   nextOf[twigRootID,j] = '1'};
22       next_of_leftID = {node(i), where
   nextOf[i,leftID] = '1'};
23       prev_of_leftID = {node(j), where
   nextOf[leftID,j] = '1'};
24   Set (if any combination is not null):
25       nextOf[next_of_twigRootID,prev_of_twigRootID]
   = '1';
26       nextOf[twigRootID,prev_of_twigRootID] = '0';
27       nextOf[twigRootID,leftID] = '1';
28       nextOf[next_of_leftID, twigRootID] = '1';
29       nextOf[leftID,prev_of_leftID] = '0';
30       nextOf[next_of_leftID,leftID] = '0';
31 PROGRAM_END.
```

3) Complexity Analysis:

Similar to the 'nodeShift' primitive, the 'twigShift' primitive makes two amendments to the structure of the childOf matrix: one to release the child/parent relationship between the twig's old parent and its root, and another to set up the child/parent relationship between the twig's new parent and its root (lines 5-6). When updating the descOf matrix, the cost is multiplied by 'm' during the 'twigShift' operation because the primitive has to deal with 'm' nodes rather than a single node as in 'nodeShift' primitive (lines 12-17). The cost of updating the nextOf matrix is same for both the 'nodeShift' and 'twigShift' primitives (lines 24-30).

TABLE II: COST SUMMARY OF THE INSERTION PRIMITIVES

Operation	Growth in		# of Work-Units (Hits)				
	NodeSet	*Matrix*	*NodeSet*	*childOf*	*descOf*	*nextOf*	*Max. Complexity*
insertLeaf	1 rec. more	1 row more 1 col more	1	2+1	2+h	2+2	O(c)
insertNonLeaf	1 rec. more	1 row more 1 col more	1	2+α	2+f×n	2+2	O(f×n)
insertTwig (m nodes)	m rec. more	m row more m col more	m	m.(2+1)	m.(2+h)	m.(2+2)	O(m.c)

n= total number of nodes in the XML tree
h= the maximum height of the XML tree (# of levels)
α= the maximum breadth-degree (i.e., number of children) of any XML node
f= a number between 0 and 1, where 'f×n' is the number of descendants at an arbitrary node
c= is very small number comparing to 'n' such that, for large XML databases, $\lim_{n\to\infty}\frac{c}{n}=0$

TABLE III: COST SUMMARY OF THE DELETION PRIMITIVES

Operation	Growth in		# of Work-Units (Hits)				
	NodeSet	*Matrix*	*NodeSet*	*childOf*	*descOf*	*nextOf*	*Max. Complexity*
deleteLeaf	1 rec. less	1 row less 1 col less	1	2	2	2+1	O(c)
deleteTwig (m nodes)	m rec. less	m rows less m cols less	m	m×2	m×2	m×(2+1)	O(m.c)

n= total number of nodes in the XML tree
c= is very small number comparing to 'n' such that , for large XML databases, $\lim_{n\to\infty}\frac{c}{n}=0$

TABLE IV: COST SUMMERY OF THE CHANGE PRIMITIVES

Operation	Growth in		# of Work-Units (Hits)				
	NodeSet	*Matrix*	*NodeSet*	*childOf*	*descOf*	*nextOf*	*Max. Complexity*
chnageName	none	none	1 or k	0	0	0	O(k)
changeValue	none	none	1 or k	0	0	0	O(k)
nodeShift	none	none	0	2	2×h	6	O(c+2.h)
twigShift (m nodes)	none	none	0	m×2	m×2×h	6	O(c+m×2×[h+1])

n= total number of nodes in the XML tree
k= the number of nodes per tag/attribute name (usually much smaller than 'n')
h= the height of the XML tree (# of levels)
c= is very small number comparing to 'n' such that , for large XML databases, $\lim_{n\to\infty}\frac{c}{n}=0$

4) Example: Using the database in Figure 3, move the author information of book 'book/110' to be the author for the book 'Book/101' (see Figure 10).

The cost breakdown is:

childOf	descOf	nextOf	**Total**
2	12	2	**16 hits**

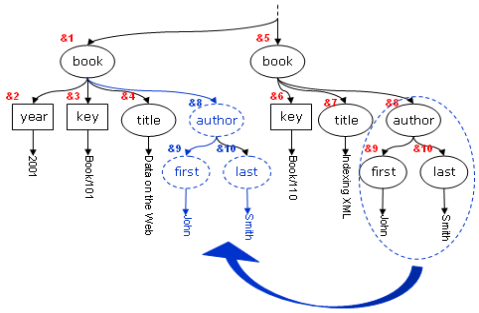


Figure 10. A Twig Shift Example

E. Change Primitives Summary

Table IV summarizes the number of work-units required to conduct each change primitives.

VIII. OVERALL COMPLEXITY DISCUSSION

The analysis provided in Sections V, VI and VII shows that the cost of all update-primitives over the PACD's uncompressed data representation locates in acceptable limits in general. Of the update primitives discussed, the highest update complexity is only a fraction of the number of nodes (i.e., 'n') and this only happens during the *rarely-used* operation 'insertNonLeaf'. The cost of other update operations ranges between a very small constant 'c' and 'm×c' in the case of manipulating a *twig* of size 'm' nodes.

From the technical point of view, the bitmapped XML structure (see Section II) and the introduction of the *previous/next* axes has played a major role in such cost reduction. Unlike node-labeling based techniques [32][33][34][8][12][13], the use of the nextOf matrix (to encode the document order) has narrowed the spread of label

changes to consider only the adjacent nodes of the targeted node. Also encoding the basic XML structures (i.e., the child/parent and descendant/ancestor relationships) using the bitmapped *node-pairs* (i.e., the childOf and descOf matrices) has reduced the high cost and complexity that result from using: (1) *path-summaries* [35][36][37][16][17][18], and (2) *sequences* [20][19][22] to encode such structures. The analysis has shown that the number of changes in the childOf structure is bounded by a small constant 'c' (where 'c' is a very small number comparing to 'n', the total number of nodes in the XML tree) in most cases except the 'insertNonLeaf' primitive, which requires 'α' number of hits depending on the node's *breadth degree*. On the other hand, the same primitive may perform up to 'n' hits over the descOf matrix, however in real situations that number is fractioned by small number 'f', which ranges between '0' and '1' (usually $0 \leq f \leq ½$ for real, well designed XML databases).

Another source of cost reduction in the PACD's update transactions is the separation between the textual content representation and the XML hierarchal structure representation. The *content-based* primitives only affect the NodeSet container while the *structure-based* update primitives affect the bitmapped matrices. This is not valid in the case of path-summary and sequence-based techniques, where the underlying path-summary or sequence has to be changed. In general, the number of hits over the NodeSet container is limited by the number of targeted nodes except when amending a tag/attribute name or a node value for a set of nodes that share the same tag/attribute name. In this case, the cost is limited by the number of nodes that share the same tag/attribute name, which is also considered small comparing to the entire XML tree.

IX. A COMPARATIVE STUDY

To evaluate and support the analysis provided above, this section presents an experimental study conducted to compare the PACD's performance (in terms of the update handling) against two representative mapping techniques. The section initially provides the experiment setup, including the list of the used update queries, the structure of the compared techniques, and the underlying test databases. Then it presents the experimental results and their discussion for each query in a separate section counting the number of work-units done by the query over the test databases. Finally, the section lists out the main finding from the experiment.

A. The Experiment Setup

A comparative experiment between the performance PACD technique and two representative XML techniques from the literature is conducted to support the above complexity analyses. The experiment executes 6 update queries –as a representation of the above update primitives- translated over 3 XML databases for the 3 selected XML techniques. The 6 update queries are listed in Table V while the characteristics of the 3 XML databases are given in Table VI. Table VII shows the XML/RDBMS mapping schema of the three compared techniques, PACD, XParent [36] and Edge [38], while other specifications of these techniques can be found in [29], [36] and [38], respectively.

The experiment (see the result summary in Table VIII) counts the number of changes (hits) done over the technique data storage (2nd column of Table VIII), and lists them per query ID in separate columns over each XML database. The number of hits, over all components, is summed up in the last 3 rows of Table VIII.

Finally, the experiment was conducted using a stand-alone Intel Pentium-IV machine with 3.6GHz dual processor and 1GB of RAM. The machine was operated by MS Windows XP SP3, and the translation of all XML queries to the corresponding RDBMS queries was executed using MS FoxPro database engine. Furthermore, data indices were used whenever applicable over the three techniques to leverage their performance with the power of RDBMS. Such HW/SW setup was counted to have no influence on the generated results.

TABLE V. THE EXPERIMENT'S UPDATE QUERIES

Query ID	Query Description
U1	Insert an Atomic Value, i.e., leave node
U2	Insert a Non-atomic Value, i.e., non-leave or internal node
U3	Delete an Atomic Value , i.e., leave node
U4	Delete a Non-atomic Value , i.e., non-leave or internal node
U5	Change an Atomic Value, i.e., the textual content of a node
U6	Change a Non-atomic Value, i.e., tag-name

TABLE VI. FEATURES OF THE USED XML DATABASES

	DBLP [39]	XMark [40]	Treebank [41]
Size (#of nodes)	2,439,294	2,437,669	2,437,667
Depth(#of levels)	6	10	36
Min Breadth†	2	2	2
Max Breadth	222,381	34,041	56,385
Avg Breadth†	11	6	3
#of Elements	2,176,587	1,927,185	2,437,666
#of Attributes	262,707	510,484	1

TABLE VII. THE EXPERIMENTAL COMPARABLE XML TECHNIQUES

Technique	Components (XML/RDBMS Mapping Schema)
PACD	XMLNodes(nodeID, type, tagID) XMLSym(tagID, desc) XMLValues(nodeID, value) nextOf(nextID, prevID) childOf(childID, parented) descOf(descID, anceID) ** childOF+descOf= OIMatrix(Source, Target, relType)*
Edge	Edge(source, target, ordinal, label, flag, value)
XParent	labelPath(pathID, length ,PathDesc) element(pathID, ordinal, nodeID) data(pathID, ordinal, nodeID, value) dataPath(nodeID, parented) ancestors(nodeID, anceID, level)

Query: U1_DBLP

Insert a new author called “New Author” of the *inproceedings* publication identified by the key “conf/ecai/BeeringerAHMW94”. The new author must be the first in the author list of that publication.

Query: U1_XMark

Insert a new location called “New Location” for the item identified by “item38683” from South America.

Query: U1_TreeBank

Insert a new ‘CC’ element (textual-value is “New CC”) under the ‘NP’ element which has a child called ‘CC’ that stores “IQXQwyLNRrdOEoHUpfWNbB==” and the existing ‘CC’ element can be reached by the path ‘/FILE/EMPTY/S/S/VP/NP/VP/PP/NP/CC’. The new ‘CC’ element must precede the existing ‘CC’ element.

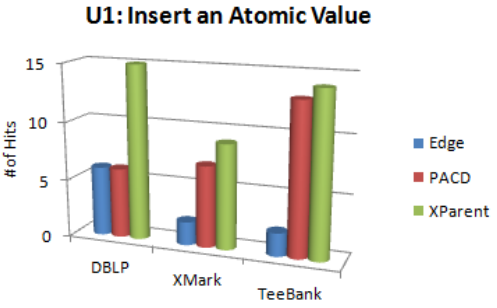


Figure 11. Performance of the "Insert an Atomic Value" Query

Query: U2_DBLP

Insert a new root element called “nRoot” to the DBLP100 database.

Query: U2_XMark

Insert a new root element called “nRoot” to the XMark100 database.

Query: U2_TreeBank

Insert a new root element called “nRoot” to the TREE100 database.

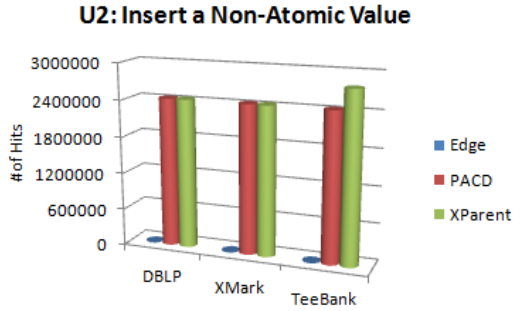


Figure 12. Performance of "Insert a Non-Atomic Value" Query

B. *Result Discussion*

This section discusses the experimental results. Each subsection discusses the results of a particular XML update query including the syntax of the executed query, a graphical representation of the results, and a brief analysis of each technique performance. The final remarks about these analyses are given in Section IX.C.

1) *Inserting an Atomic Value*

The three queries in Figure 11 insert a new leaf-node at levels 3, 5 and 10 of the DBLP, XMark and TreeBank databases, respectively. The queries were designed to act at a distance of 30% from the root-node.

The graph shows that PACD required six, seven and thirteen amendments to the underlying relational schema in order to execute this query. The number of amendments is mainly controlled by the level number where the insertion was applied. For example, the 6 operations required by PACD over DBLP are distributed as follows: 1 insertion to the ‘XMLNodes’ table and another insertion to the ‘XMLValues’ table because the node contains a textual value. Three operations were also required to update the ‘OIMatrix’ table while the last operation was required to link the new node to its next node at the ‘nextOf’ table. As discussed earlier, PACD requires at most two operations to update the ‘nextOf’ table for any leaf-node insertion, and at most ‘h’ to update the ‘OIMatrix’ table for the same process, where ‘h’ is the maximum number of levels in the XML tree.

Query: U3_DBLP

Delete the author named “Antje Beeringer” from the inproceedings record that is identified by the key “conf/ecai/BeeringerAHMW94”.

Query: U3_XMark

Delete the location for the item identified by “item38683” from South America. The deleted element is storing the content “United States”.

Query: U3_TreeBank

Delete the ‘CC’ element (textual-value is “lQXQwyLNRrdOEoHUpfWNbB==”) which can be reached by the path ‘/FILE/EMPTY/S/S/VP/NP/VP/PP/NP/CC’.

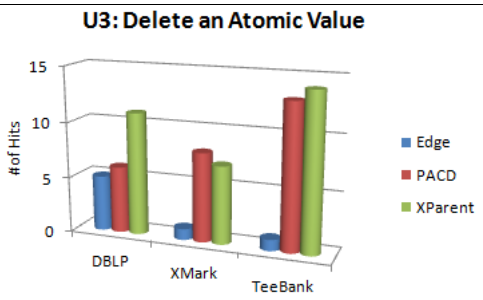


Figure 13. Performance of "Delete an Atomic Value” Query

Query: U4_DBLP

Delete the entire record of an article labeled with the KEY “tr/gte/TM-0332-11-90-165”.

Query: U4_XMark

Delete the entire record of an item labelled with the ID “item7” from Africa.

Query: U4_TreeBank

Delete the entire record of the element ‘PP’ that is reachable by the path ‘/FILE/EMPTY/S/VP/S/VP/NP/VP/NP/PP’ and its child element ‘TO’ has the value “6fc25UxSwWg9Pz+yyR6wi8==”.

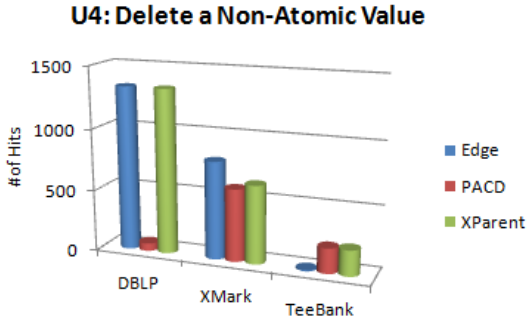


Figure 14. Performance of “Delete a Non-Atomic Value" Query

Compared to other techniques, PACD is always better than XParent because the later required one more insertion to the ‘data’ table and more *change* operations to keep the document order updated at the ‘ordinal’ attributes of the ‘elem’ and ‘data’ tables. On the other hand, Edge performance is either same as PACD or better. This superiority is determined by the fact that Edge encodes are far less of XML structure, which in turn affects its query performance. In summary, when linked with the queries-range coverage, PACD appears to have the best update performance for this type of query among the three techniques.

2) Inserting Non-Atomic Value

These queries (Figure 12) insert virtual root-nodes to the three XML databases, respectively. The virtual new root insertion is used here for three reasons. Firstly, the operation was chosen to represent the process of inserting non-leaf nodes, which may occur at any level in the XML tree. Secondly, inserting at the top most level can give a logical comparison between the number of operations required by the operation rather than inserting at lower locations in different XML databases. Finally, it will be easier to observe the XML updater behavior and judge its performance with relation to the database size and other XML features.

For this particular query, PACD required ‘2+n’ amendments to the underlying relational schema, where ‘n’ is the number of nodes in the XML tree. The ‘n’ operations were required to insert the parent/child and descendant/ancestor relationships of the new node, whereas the other two operations were used to insert the

Query: U5_DBLP

Change the book title identified by the key "phd/Mumick91" from "Query Optimization in Deductive and Relational Databases" to be "Query Optimization in Deductive and Relational Databases: Modified".

Query: U5_XMark

Change the street name of a person's address by the key "person0" from "85 Geniet St" to be "85 Geniet St: Modified".

Query: U5_TreeBank

Change the value stored in 'IN' element which is reachable by the path "/FILE/S/VP/SBAR/S/NP/SBAR/S/VP/S/VP/VP/NP/PP/NP/PP/NP/VP/PP/IN" from "CrtsNRQX7cNqOsWbpvPMgO==" to "CrtsNRQX7cNqOsWbpvPMgO==:Modified".

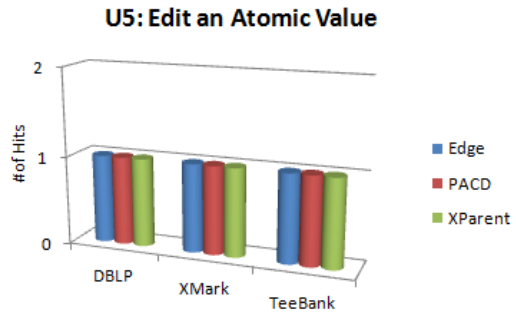


Figure 15. Performance of "Edit an Atomic Value" Query

Query: U6_DBLP

Change the description of the element/attribute name 'key' records to 'ID_NUM'.

Query: U6_XMark

Change the description of the element/attribute name 'id' records to 'ID_NUM'.

Query: U6_TreeBank

Change the description of the element/attribute name 'PP' records to 'PPPP'.

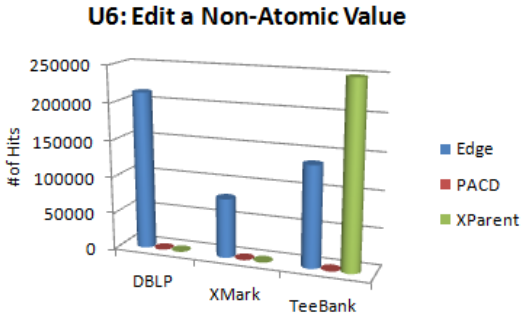


Figure 16. Performance of "Edit a Non-Atomic Value" Query

corresponding record in the 'XMLNodes' and 'XMLSym' tables because the new node was assumed to have distinct tag name from the existing tag/attribute list.

Due to its restricted XML mapping algorithm, Edge had only one amendment to execute this update operation. This amendment was required to insert the new node's record into the underlying mapping schema without affecting the 'ordinal' attribute because the root node logically has no siblings. XParent workload on the other hand was slightly higher than PACD. XParent required extra "s" operations to update the 'labelPath' table where "s" is the number of records in that table, which stores the corresponding XML schema summary. In general, XParent's number of operations exceeds PACD ones by the number of the records affected inside the underlying XPath summary, and also the relative position of the inserted node amongst its siblings.

3) Delete an Atomic Value

These queries (Figure 13) delete a single leaf node from each XML database. The deleted nodes were located at levels 3, 5 and 10 of the DBLP, XMark and TreeBank databases, respectively; and they were 30% away from the root node each database. In addition, the deleted nodes were chosen to have at least one sibling node of the same tag-name so that the impact of the deletion process on the document order can be calculated.

PACD performed '2+p+2' amendments to the underlying mapping schema of all XML database types where 'p' is the level number of the node deleted. The first 2 operations were required to remove the corresponding records from the

TABLE VIII. THE EXPERIMENTAL RESULTS

Tech. Name	Query / Tables	DBLP						XMark						Treebank					
		U1	U2	U3	U4	U5	U6	U1	U2	U3	U4	U5	U6	U1	U2	U3	U4	U5	U6
Edge	edge	6	1	5	78815	1	213634	2	1	1	792	1	80316	2	1	1	9	1	136545
PACD	XMLSym	0	1	0	0	0	1	0	1	0	0	0	1	0	1	0	0	0	1
	XMLNodes	1	1	1	13	0	0	1	1	1	48	0	0	1	1	1	8	0	0
	XMLValues	1	0	1	12	1	0	1	0	1	24	1	0	1	0	1	4	1	0
	OIMatrix	2	n	2	26	0	0	4	n	4	514	0	0	9	n	9	186	0	0
	nextOf	2	0	2	11	0	0	1	0	2	2	0	0	2	0	2	5	0	0
XParent	elem	6	1	4	78815	0	0	2	1	1	48	0	0	2	1	2	9	0	0
	data	6	0	4	12	1	0	2	0	1	24	1	0	2	0	2	4	1	0
	labelpath	0	146	0	0	0	8	0	252	0	0	0	9	0	338750	0	0	0	248480
	dataPath	1	1	1	13	0	0	1	1	1	48	0	0	1	1	1	9	0	0
	ancestor	2	n	2	26	0	0	4	n	4	514	0	0	9	n	9	186	0	0
Edge	Total	6	1	5	78815	1	213634	2	1	1	792	1	80316	2	1	1	9	1	136545
PACD	Total	6	n+2	6	62	1	1	7	n+2	8	588	1	1	13	n+2	13	203	1	1
XParent	Total	15	n+148	11	78866	1	8	9	n+254	7	634	1	9	14	n+338752	14	208	1	248480

n=2437669, is the number of nodes inside each XML database and it should be unified for all databases

'XMLNodes' and 'XMLValues', respectively, and last 2 operations were required to update the 'nextOf' table while the 'p' operations were conducted on the 'OIMatrix' table to remove the node's corresponding records. In general, the number of operations for this type of update is determined by the level number of the target node in the XML tree with a maximum cost of '2+h+2' where 'h' is the maximum number of levels in the XML tree.

The performance of Edge and XParent in update was close to PACD's. In XParent, the update handler requires '2×rs' more operations (where 'rs' is the number of the right-hand side siblings of the node) to update the document-order inside the 'elem' and 'data' tables, while Edge's processor required '1+2×rs' operations to remove the node from the list and amend the siblings' ordinal attribute.

In summary, PACD appears more efficient for this type of queries due the document order preserving mechanism.

4) Delete a Non-Atomic Value

The action of these queries (Figure 14) was conducted at levels two, five and ten of the DBLP, XMark and TreeBank databases, respectively. These queries were included to test the performance of deleting a sub-tree from the master XML tree. The number of nodes in the target sub-trees was selected to be very small compared to the master XML tree so that identifying the number of records affected became easy. The DBLP's sub-tree consisted of 13 nodes distributed over 2 levels, and the XMark's sub-tree had 48 nodes distributed over 7 levels while the number of nodes and levels in the TreeBank's sub-tree were 8 and 4, respectively. All sub-tree nodes were combinations of atomic and non-atomic nodes.

PACD required 13 and 12 operations to remove the DBLP's sub-tree from the nodes and values lists, respectively, 26 operations to update the parent/child and descendant/ancestor relationships, and 11 operations to update the 'nextOf' container. These figures were determined by three factors. Firstly, the sub-tree size determined the number of 'delete' operations from both the 'XMLNodes' and 'XMLValues' tables. Secondly, the breadth and the depth as well as the level of the sub-tree's root node all controlled the number of update operations of the 'OIMatrix' table. Finally, the number of update operations at the 'nextOf' table was mainly controlled by the breadth of the sub-tree including, at most, 2 operations to re-link the left and right hand side nodes for the previous/next relationship. In general, PACD generates a manageable number of changes for this type of queries especially when the update happens at the low levels of the XML tree.

On the other hand, Edge and XParent performed 1270 times more operations compared to PACD for the DBLP's query, and the three techniques were close to each other for the XMark's query while PACD and XParent were 22 times higher than Edge for the TreeBank's query. These figures support the above conclusion that the number of operations is determined by the size of the deleted sub-tree and its location in the master XML tree. In general, PACD's document-order encoding mechanism had a clear impact in reducing the number of changes that are required to conduct sub-tree deletion operations.

5) Edit an Atomic Value

This is the cheapest update query (Figure 15) that can be ever conducted by any technique tested. All techniques over all database types have made exactly one amendment to their relational schema storage. In this case, PACD needs to update the 'XMLValues' table, Edge also updates the corresponding record in its orphan table while XParent needs to change the record inside the 'data' table.

6) Edit a Non-Atomic Value

These queries (Figure 16) can be used to alter the tags and attributes names without affecting the document's hierarchal structure. The experiment has chosen to alter the name of some elements/attributes, which were widely repeated in each XML database to show the importance of minimizing the cost of such update queries. The DBLP's

query was designed to change all 'KEY' attributes, and the XMark's query was designed to change all 'ID' attributes, while the TreeBank's query was deigned to change the name of the recursive element 'PP'. The 'KEY', 'ID' and 'PP' tokens were repeated 213,634, 80,316 and 136,545 times, respectively inside the corresponding XML databases.

In general, the statistics show that the number of amendments conducted by PACD was always 1 because PACD stores all database tokens only once. On the other hand, the number of amendments in Edge's table was determined by the number of elements/attributes that hold the same name, while the number of amendments in XParent environment was determined by the number of XPath expressions that contain the element/attribute name. So, for Edge, the number of changes was 213,634, 80,316 and 136,545 over the DBLP, XMark and TreeBank databases respectively, while XParent performed 8, 9 and 248,480 changes over the same set of XML databases. The high number of changes produced by XParent over the TreeBank database was due the recursive properties of the element 'PP' inside the XML schema.

C. Main Findings

The experiment discussed here has evaluated the PACD's update primitives by executing six XML update queries over three different XML databases. The evaluation process examined the performance of PACD over each XML database and compared it with Edge's and XParent's performance over the same database set.

Comparing to other techniques, and taking into account the queries-range coverage, PACD appeared having the best performance for most of the queries in all situations. The experiment has also shown that the performance of XParent and Edge was delayed by the cost of the document order persevering mechanism. PACD eliminates this cost by encoding the previous/next relationship that requires at most 2 changes for any type of query/operation that concerns about document-order.

X. Conclusion

This paper has discussed the PACD's updating framework, which is managed by a set of low cost update primitives. Once an update query is issued, the Update Query Handler (UQH) process identifies the target node-set and the necessary update primitive(s). The translation of an update query may generate one or more update primitives each of which may alter one or more XML nodes. The UQH currently can generate nine update primitives divided into three categories; the insert, delete, and change primitives.

This paper has provided a comprehensive complexity analysis of the PACD's update primitives supported by illustrative examples for each update primitive. The paper also presented an experimental evaluation process to support the analysis and generalize conclusions based on the generated results.

Both analysis and experimental results provided in this paper have shown that the computation cost of the XML updates can be improved using the PACD's update primitives, which specifically act on its data-storage. The summary of the complexity discussion is given in Tables II, III and IV, while the full experimental result summary is depicted in Table VIII.

Besides, the paper has supplied a full algorithmic listing of the XML update primitives under the PACD environment, along with a comprehensive evaluation method (and the results), which can be recycled by the XML research community to test and evaluate the XML database developments. Such level of details is rarely found in the existing literature.

References

[1] M. Al-Badawi and A. Al-Hamadani, "A Complexity analysis of an XML update framework," in Proceedings of ICIW 2013, Rome, Italy, 2013, pp. 106-113, ISSN: 2308-3972, ISBN: 978-1-61208-280-6.

[2] T. Härder, M. Haustein, C. Mathis, and M. Wagner, "Node labelling schemes for dynamic XML documents reconsidered," International Journal of Data Knowledge Engineering, vol. 60, issue 1, 2007, pp. 126-149.

[3] P. O'Neil, E. O'Neil, S. Pal, I. Cseri, G. Schaller, and N. Westbury, "ORD-PATHs: insert-friendly XML node labels," In proceeding of ACM/SIGMOD international conference on Management of Data, 2004, pp. 903-908.

[4] W. Shui, F. Lam, D. Fisher, and R. Wong, "Querying and marinating ordered XML data using relational databases," in Proceedings of the 16th Australasian database conference - vol. 39, Newcastle, Australia, 2005, pp. 85-94.

[5] I. Tatarrinov, S. Viglas, K. Beyer, J. Shanmugasundaram, E. Shekita, and C. Zhang, "Storing and querying ordered XML using a relational database system," ACM/SIGMOD Record, Madison, Wisconsin, 2002, pp. 204-215.

[6] H. Wang, H. He, J. Yang, P. Yu, and J. Yu, "Dual labeling: Answering graph reachability queries in constant time," in Proceedings of the International conference of Data Engineering, 2006, pp. 75-86.

[7] C. Zhang, J. Nsughton, D. DeWitt, Q. Luo, and G. Lohman, "On supporting containment queries in relational database management systems," in Proceedings of the 2001 ACM SIGMOD international conference on Management of Data, California, USA, 2001, pp. 425-436.

[8] J. K. Min, J. Lee, and C. W. Chung, "An efficient XML encoding and labeling method for query processing and updating on dynamic XML data," Advance in Databases: Concepts, Systems and Applications, LNCS, vol. 4443, 2009, pp. 715-726.

[9] S. Sakr, "A prime number labeling scheme for dynamic ordered XML trees," in Proceedings of the Intelligent Data Engineering nd Automated Learning, LNCS, vol. 5326, 2008, pp. 378-386.

[10] J. Lu, X. Meng, and T. W. Ling, "Indexing and querying XML using extended Dewey labeling scheme," Journal of Data & Knowledge Engineering, vol. 70, issue 1, 2011, pp. 35-59.

[11] L. Xu, T. Wang Ling, and H. Wu, "Labeling dynamic XML documents: an order-centric approach," IEEE Transactions on Knowledge and Data Engineering, vol. 24, issue 1, 2012, pp. 100-113.

[12] J. Liu, Z. M. Ma, and L. Yan, "Efficient labeling scheme for dynamic XM trees," Information Sciences, vol. 221, 2013, pp. 338-354.

[13] R. Lin, Y. Chang, and K. Chao, "A compact and efficient labeling scheme for XML documents," Database Systems for Advanced Applications, LNCS, vol. 7825, 2013, pp. 269-283.

[14] Q. Chen, A. Lim, and K. Ong, "D(K)-Index: An adaptive structural summary for graph-structured data," in Proceedings of the 2003 ACM SIGMOD international conference on Management of data, CA, USA, 2003, pp. 134-144.

[15] C. Chung, J. Min, and K. Shim, "APEX: An adaptive path index for XML data," in Proceedings of the 2002 ACM SIGMOD international conference on Management of data, Madison, Wisconsin, 2002, pp. 121-132.

[16] S. Haw and C. Lee, "Extending path summary and region encoding for efficient structural query processing in native XML databases," Journal of Systems and Software, vol. 82, issue 6, 2009, pp. 1025-1035.

[17] A. Arion, A. Bonifati, I. Manolescu, and A. Pugliese, "Path summaries and path partioning in modern XML databases," World Wide Web, vol. 11, issue 1, 2008, pp. 117-151.

[18] M. Sadoghi, I. Burcea, and H. A. Jacobsen "A gneric boolean predicated XPath expression matcher," in Proceedings of the 14th Int. Conf. on Extending Database Technology, 2011, pp. 45-56.

[19] J. Kwon, P. Rao, B. Moon, and S. Lee, "Fast XML document filtering by sequencing twig patterns," ACM Transactions on Internet Technology (TOIT), vol. 9, issue 4, Article 13, 2009, pp. 13.1-13.51.

[20] H. Wang and X. Meng, "On sequencing of tree structures for XML indexing," in Proceedings of the 21st international conference on Data Engineering, 2005, pp. 372-383.

[21] H. Wang, X. Wang, and W. Zeng, "A research on automaticity optimization of KeyX index in native XML database," in Proceedings of the 2008 international conference on Computer Science and Software Engineering, 2008, pp. 700-703.

[22] W. Li, J. Jang, G. Sun, and S. Yue "A new Sequence-Based approach for XML data query," in Proceedings of the 2013 Chinese Intelligent Automation Conf., LNEE, vol. 256, 2013, pp. 661-670.

[23] H. Al-Jmimi,A. Barradah, and S. Mohammed "Sibiling labeling scheme for updating XML dynamically," in Proceedings of the 4th Int. Conf. on Computer Engineering and Technology, vol. 40, 2012, pp. 21-25.

[24] J. Yoon, S. Kim, G. Kim, and V. Chakilam, "Bitmap-based indexing for multi-dimensional multimedia XML document," in Proceedings of the 5th International Conference on Asian Digital Libraries-ICADL2002, Singapore, 2002, pp. 165-176.

[25] N. Zhang, M. Özsu, I. Ilyas, and A. Aboulnaga, "FIX: feature-based indexing technique for XML documents," in Proceedings of the 22nd international conference on VLDB, vol. 32, Seoul, Korea, 2006, pp. 259-270.

[26] R. Senthilkumar and A. Kannan, "Query and update support for indexing and compressed XML (QUICX)," Recent Trends in wireless and Mobile Networks Communication in computer and Information Science, vol. 162, 2011, pp. 414-428.

[27] J. Clark and S. DeRose, XML Path Language (XPath)-Version 1.0, [Online] Available online: http://www.w3.org/TR/xpath/, [Accessed on: 25/05/2014].

[28] M. Al-Badawi, H. Ramadhan, S. North, and B. Eaglestone, "A performance evaluation of a new bitmap-based XML processing approach over RDBMS," Int. J. of Web Engineering and Technology, vol. 7, no. 2 , 2012, pp. 143 – 172.

[29] M. Al-Badawi, B. Eaglestone, and S. North, "PACD: A bitmap-based approach for processing XML data," WebIST'09, Lisbon, Portugal, 2009, pp. 66-71.

[30] H. He, H. Wang, J. Yang, and P. Yu, "Compact reachability labeling for graph-structured data," in Proceedings of the 14th ACM international conference on Information and knowledge management, Bremen, Germany, 2005, pp. 594-601.

[31] T. Bray, J. Paoli, C. Sperbeg-McQueen, E. Maler, and F. Yergeau, Extensible Markup Language (XML) 1.0 (Fourth Edition), [Online] Available online: http://www.w3.org/TR/REC-xml/, [Last accessed on: 25/05/2014].

[32] J. Yun and C. Chung, "Dynamic interval-based labelling scheme for efficient XML query and update processing," Journal of Systems and Software, vol. 81, issue 1, 2008, pp. 56-70.

[33] J. Lu, T. Ling, C. Chan, and T. Chen, "From region encoding to extended dewey: on efficient processing of XML twig pattern matching," in Proceedings of the 31st International Conference on VLDB, Trondheim, Norway, 2005, pp. 193-204.

[34] X. Wu, M. Lee, and W. Hsu, "A prime number labeling scheme for dynamic ordered XML trees," in Proceedings of the 20th international conference on Data Engineering, 2004, pp. 66-78.

[35] R. Goldman and J. Widom, "DataGuides: enabling query formulation and optimaization in semistructured database," in Proceedings of the 23rd international conference on VLDB, 1997, pp. 436-445.

[36] H. Jiang, H. Lu, W. Wang, and J. Yu, "XParent: an efficient RDBMS-based XML database system," International conference on Data Engineering, CA, USA, 2002, p. 2.

[37] M. Yoshikawa, T. Amagasa, T. Shimura, and S. Uemura, "XRel: a path-based approach to storage and retrieval of XML documents using relational databases," ACM/IT., vol. 1, issue 1, NY, USA, 2001, pp. 110-141.

[38] D. Florescu and D. Kossmann "A Performance Evaluation of alternative Mapping Schemas for Storing XML Data in a Relational Database," TR:3680, May 1999, INRIA, Rocquencourt, France, pp. 1-24.

[39] DBLP, The DBLP Computer Science Biblography, [Online] Available at http://dblp.uni-trier.de/, [Last accessed on 24/05/2014].

[40] A. Schmidt, F. Waas, M. Kersten, D. Carey, I. Manolescu, and R. Busse. "XMark: a benchmark for XML data management," International conference on Very Large Data Bases, Hong Kong, China, 2002, pp. 974-985.

[41] PennProj, The Penn Treebank Project, [Online] Available online at http://www.cis.upenn.edu/~treebank/, [Last accessed on: 25/05/2014].

Internet of Threads: Processes as Internet Nodes

Renzo Davoli
Computer Science and Engineering Department
University of Bologna
Bologna, Italy
Email: renzo@cs.unibo.it

***Abstract*—In the beginning, Internet and TCP/IP protocols were based on the idea of connecting computers, so the addressable entities were networking adapters. Due to the evolution of networking and Internet services, physical computers no longer have a central role. Addressing networking adapters as if they were the true ends of communication has become obsolete. Within Internet of Threads, processes can be autonomous nodes of the Internet, i.e., can have their own IP addresses, routing and QoS policies, etc. In other words, the Internet of Threads definition enables networked software appliances to be implemented. These appliances are processes able to autonomously interoperate on the network, i.e., the software counterpart of the Internet of Things' objects. This paper will examine some usage cases of Internet of Threads, discussing the specific improvements provided by the new networking support. The implementation of the Internet of Threads used in the experiments is based on Virtual Distributed Ethernet (VDE), Contiki and View-OS. All the software presented in this paper has been released under free software licenses and is available for independent testing and evaluation.**

***Keywords*-Internet; IP networks; Virtual Machine Monitors**

I. INTRODUCTION

The design of the Internet is dated back to the beginning of the 1960s. The main goal and role of the Internet was to interconnect computers. It was natural, though, to consider the network controller of computers as the *addressable entities*. The endpoints of any communication, those having an Internet Protocol (IP) address were the hardware controllers [2]. This *ancient* definition is still used in the modern Internet. Processes and threads are identified by their *ports*, a sub-structure of the addressing scheme, as they are considered dependent on the hardware (or virtual) computer they are running on.

In the common scenario when a client application wants to connect to a remote internet service, it first gets the IP address of the server providing the service, and then the client application opens a connection to a specific well-known (or pre-defined) port at that address.

Thus, the DNS maps a logical name (e.g., www.whitehouse.gov or ftp.ai.mit.edu) to the IP address of a network controller of a computer that provides the service. This emphasis on the hardware infrastructure providing the service is obsolete: the main focus of the Internet nowadays is on services and applications. The whole addressing scheme of the Internet should change to address this change of perspective.

By Internet of Threads (IoTh) we mean the ability of processes to be addressable as nodes of the Internet, i.e., in IoTh processes play the same role as computers, being IP endpoints. They can have their own IP addresses, routing and QoS policies, etc.

On IPv4, IoTh usage can be limited by the small number of available IP addresses overall, but IoTh can reveal all its potential in IPv6, whose 128-bit long addresses are enough to give each process running on a computer its own address.

This change of perspective reflects the current common perception of the Internet itself. Originally, Internet was designed to connect remote computers using services like remote shells or file transfers. Today users are mainly interested in specific networking services, no matter which computer is providing them. So, in the early days of the Internet, assigning IP addresses to the networking controllers of computers was the norm, while today the addressable entity of the Internet should be the process which provides the requested service.

For a better explanation, let us compare the Internet to a telephone system. The original design of the Internet in this metaphor corresponds to a fixed line service. When portable phones were not available, the only way to reach a friend was to guess where he/she could be and try to call the nearest line. Telephone numbers were assigned to places, not to people. Today, using portable phones, it is simpler to contact somebody, as the phone number has been assigned to a portable device, which generally corresponds to a specific person.

In the architecture of modern Internet services, there are already exceptions to the rule of assigning IP addresses to physical network controllers.

- Virtual Machines (VM) have virtual network controllers, and each virtual controller has its own IP address (or addresses). This extension is, from some perspectives, similar to IoTh. In fact, it is possible to run a specific VM for each networking service. This approach, although possible, would clearly be ineffective. It is highly resource demanding in terms of: main memory to emulate the RAM of the VM; disk space for the virtual secondary memory of the VM; time since the VM has to run an entire Operating System Kernel providing processor scheduling, memory managing, etc.
- Each interface can be assigned several IP service ori-

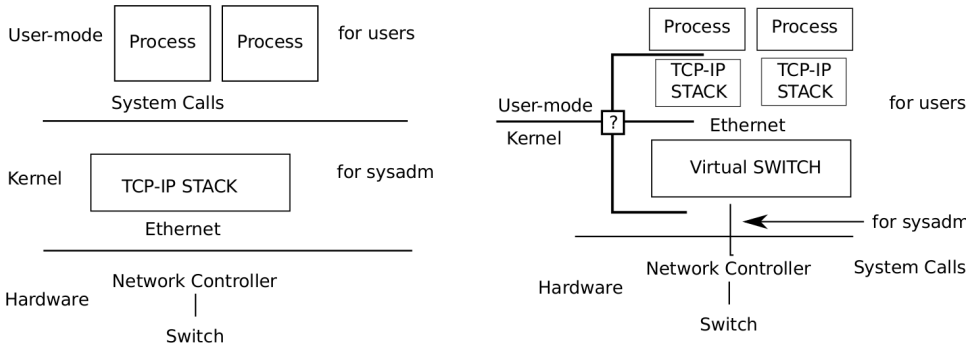


Fig. 1. Different perspectives on the networking support: the standard OS support is on the left side, IoTh is on the right side

ented addresses. Thus, it is possible to define different addresses, each one corresponding to a specific service. For example, if a DNS maps www.mynet.org to 1.2.3.4, and ftp.mynet.org to 1.2.3.5, it is possible to assign both addresses to the same controller. Addresses can be assigned to a specific process using the *bind* system call. This approach is commonly used in High Availability (HA) servers, where there is the need to migrate services from one host to another in case of faulty hardware or software [3]. This approach:

  - requires a complex daemon configuration (each socket must be bound to the right address) and network filtering (e.g., using *iptables* [4]) to prevent services from being reached using the wrong logical address (e.g., http://ftp.mynet.org);
  - provides the way to migrate service daemons in a transparent manner for clients, but it does require a non trivial procedure to delete addresses and network filtering rules on one server and then define the same addresses and filtering rules on the other, prior to starting the new daemon process.

- Linux Containers (LXC), as well as Solaris Zones [5], [6], allow system administrators to create different operating environments for processes running on the same operating system kernel. Among the other configurable entities for containers, it is possible to define a specific network support, and to create virtual interfaces of each container (flag CLONE_NEWNET of clone(2)). The definition and configuration of network containers, or zones, are privileged operations for system administrators only. This feature appears very close to the intents of IoTh. However, in LXC the creation of a networking virtual interface, thus the setting of a new IP address, is a system administration operation. In fact, it requires the process to own the CAP_NET_ADMIN capability, the same required to modify the configuration of the physical controller of the hosting computer. In IoTh, the creation of a networking environment for a process is as simple as a library function call. In this way, a process defines its IP address(es) as an ordinary user operation. IoTh provides Network Access Control to prevent abuses of networking services at the virtual Data-Link layer (in general a Virtual Ethernet). A process can define its interfaces and its IP addresses as the owner of a Personal Computer (PC) can assign any IP address to the interfaces of their PC connected to a Local Area Network (LAN). If the IP address is wrong, or inconsistent with the addresses running on the LAN, that PC cannot communicate. And even if the address is correct, it is possible to set up firewalls to define what that PC can do and what cannot be done. In IoTh each process can play the role of the PC in the example. In the same way, virtual firewalls can be set up to define which are the permitted networking services.

The paper will develop as follows: Section II introduces the design and implementation of Ioth, followed by a discussion in Section III. Related work is described in Section IV. Section V is about usage cases. Section VI introduces an innovative way to assign IP addresses and Section VII is about practical examples. Section VIII discusses the security issues related to IoTh and Section IX provides some performance figures of a proof-of-concept implementation. The paper ends with some final considerations about future work.

II. Design and Implementation of IoTh

The role and the operating system support of the Data-Link networking layer must be redesigned for IoTh. Processes cannot be plugged to physical networking hubs or switches as they do not have hardware controllers (in the following the term *switch* will be used to reference either a switch or a hub, as the difference is not relevant to the discussion). On the other hand, it is possible to provide processes with virtual networking controllers and to connect these controllers to virtual switches. Figure 1 depicts the different perspectives on the networking support. The focus of Fig. 1 is to show how IoTh changes the Operating System (OS) support for networking: what is provided by the hardware vs. what is implemented in software, what is shared throughout the system vs what is process specific and what is implemented as kernel code vs. what runs in user-mode.

The typical networking support is represented on the left side of Fig. 1. Each process uses a networking Application Program Interface (API), usually the Berkeley sockets API [7],

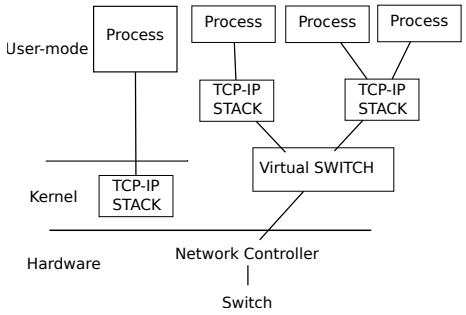


Fig. 2. A more complex scenario of IoTh usage

to access the services provided by a single shared stack, or by one of the available stacks for zones or LXC (see Section I). The TCP-IP stack is implemented in the kernel and directly communicates with the data-link layer to exchange packets using the physical LAN controllers.

In IoTh, represented on the right side of the figure, unprivileged processes can send data-link packets using virtual switches, able to dispatch data-link packets from process to process and between processes and virtual interfaces (e.g., tuntap interfaces) of the hosting OS. Virtual switches can also be interfaced to physical networking controllers, but this latter operation is privileged and requires specific capabilities (CAP_NET_ADMIN).

So, the hardware-software boundary has been moved downwards in the IoTh design. In fact, the data-link networking (commonly the Ethernet) includes software components in IoTh, i.e., virtual switches, for unprivileged user processes. In IoTh the virtual switches are shared components between user processes, while the TCP-IP stacks (or, in general, the upper part of the networking stacks, from the networking layer up) are process specific. It is also possible for a group of processes to share one TCP-IP stack, but in the IoTh design this is just an implementation choice and no longer an OS design issue or system administration choice.

The kernel/user-mode code boundary is flexible in IoTh: both the virtual ethernet switches and the TCP-IP stacks can be implemented in the kernel or not. A virtual switch can be a standard user-mode process or a kernel service, while a TCP-IP stack is a library that can be implemented in the kernel to increase its performance.

Figure 2 shows a more complex scenario that is more consistent with a real usage case of IoTh. In fact, the traditional support for networking and the IoTh approach co-exist in a system. Following the telephony systems example of the introduction (see Section I), fixed line services and portable phones interoperate.

The OS running on a computer still needs a computer specific TCP-IP stack and IP addresses to be used when a system administrator needs to configure some systemwide service. In the telephone service metaphor we use fixed lines when we want to be sure to call a specific place. Internet services (like ftp, web, MTA, etc.) can be reached using their own IP addresses. These services are the portable phones of the metaphor.

IoTh can be used for networking clients, too. Several virtual switches running on the same hosting system can be connected to different networks and can provide different services, e.g., can have different bandwidths or routing. Additionally, several virtual switches can be active on the laptop of a professor attending a conference, one connected by an encrypted Virtual Private Network (VPN) to his/her University's remote protected intranet and another directly connected to the physical, maybe wi-fi, LAN of the conference center.

Users of IoTh enabled OS, like the professor in the example, will be able to choose between the available networking services in their applications (browser, voip clients, etc.) just as they currently choose the printer.

III. Discussion

All the concepts currently used in Local Area Neworking can be applied to IoTh networking.

Virtual switches define virtual Ethernets. Virtual Ethernets can be bridged with Physical Ethernets, so that workstations, personal computers or processes running as IoTh nodes are indistinguishable as endnodes of Internet communication. Virtual Ethernets can be interconnected by Virtual Routers. It is possible to use DHCP [8] to assign IP addresses to processes, to use IPv6 stateless autoconfiguration, [9], to route packets using NAT [10], to implement packet filtering gateways, etc. This is a non-exaustive list of protocols and services running on a real Ethernet that work on a virtual Ethernet, too. It is also possible to create complex virtual networking infrastructures composed by several virtual Ethernets such as virtual application level firewalls with De Militarized Zone (DMZ) networks and virtual bastion hosts.

IoTh can support the idea of network structure consolidation in the same way that the Virtual Machines provided the idea of Server consolidation. Complex networking topologies can be virtualized, thus reducing the costs and failure rates of a hardware infrastructure. Today it is common for large companies to substitute their servers with virtual machines, creating, in this way, their internal cloud, or moving their servers to an external system-as-a-service (SaaS) cloud. This choice permits a more flexible and economically effective management of the servers and, at the same time, all the investments in terms of software can be preserved, as it is possible to move each server to a virtual counterpart, while mantaining the same software architecture: operating system type and version, libraries, etc. IoTh adds one more dimension to this consolidation process: it is possible by IoTh to virtualize not only each server as such, but also the pre-existing networking infrastructure.

Network consolidation is just an example of IoTh as a tool for compatibility with the past. In this example each process joining the virtual networks is just a virtual machine or a virtual router or firewall. The granularity of an Internet node is flexible in IoTh. A virtual machine can be an Internet node, but each browser, bit-torrent tracker, web server or mail transport agent (MTA) can be an Internet node, too.

Through IoTh, there is no difference between local and remote Inter Process Communication (IPC) services. A process can have its own IP address(es) and can interoperate with other processes using standard protocols and standard ports. Several processes running on the same host can use the same port, since each one uses different IP addresses. The same IPC protocols can be used regardless of the host on which the process is running: nothing changes, whether the communicating processes are running on the same host or on different, perhaps remote, computers. This allows a simpler migration of services from one machine to another.

Each process in IoTh can have its user interface implemented as an Internet service. This means, for example, that it is possible to create programs which register their IP addresses in a dynamic DNS, and where their web user interface is accessible through a standard browser.

IoTh is, from this perspective, the software counterpart of Internet of Things (IoT [11]). In IoT hardware gadgets are directly connected to the network. IoT objects interact between themselves and with users through standard Internet protocols. IoTh applies the same concept to processes, i.e., to software objects as if they were virtual IoT gadgets. These IoTh-enabled processes using internet protocols to interoperate can be called networked virtual appliances. If they were implemented on specific dedicated hardware objects, they would become things, according to the definition of IoT.

IV. Related Work

IoTh uses and integrates several concepts and tools already available in the literature and in free software repositories.

A. Virtual Ethernet Services

IoTh is based on the availability of virtual data-link layer networking services, usually virtual Ethernet services, as Ethernet is the most common data-link standard used. Virtual Ethernet networking was first introduced as a networking support for virtual machines. Originally, each virtual machine monitor program was provided with its specific virtual networking service. Some of them were merely an interface to a virtual networking interface (tuntap [12]). The kernel of the hosting operating system had to manage the bridging/switching or routing between virtual machines and real networks. User-mode Linux (UM-L [13]) introduced the idea of network switch as a user process interconnecting several UM-L VMs.

Virtual Distributed Ethernet (VDE [14]) extended the virtual switch idea in many ways:

- VDE created a virtual plug library to interconnect different types of VMs to the same virtual switch. Currently, User-Mode Linux, qemu [15], kvm [16], virtualbox [17] natively support VDE in their mainstream code. Virtuallyi, any VM that supports tuntap can also be connected to VDE using the virtual tuntap library.
- VDE switches running on different hosts can be connected by virtual cables to form an extended virtual Ethernet LAN. All the VM connected to one of the interconnected switches regard the others as if they all were on the same LAN.
- VDE provides support for VLANs, Fast spanning tree for link fault tolerance, remote management of switches, etc.
- VDE is a service for users: the activation of a VDE switch, the connection of a VM to a switch, or the interconnection of remote switches, are all unprivileged operations.

VDE switches were first implemented using user-level processes, but there is an experimental version of faster VDE switches running as kernel code (kvde_switch). Although this implementation runs as kernel code, kernel switches can be started and managed by unprivileged users and processes. kvde_switch is based on Inter Process Networking (IPN [18]) sockets, a general purpose support for broadcast/multicast IPC between processes.

The idea of a general purpose virtual Ethernet switch for virtual machines has been implemented by some other projects, too:

- OpenVswitch [19] is a virtual Ethernet switch for VMs implemented at kernel level. OpenVswitch has VLAN and QoS support. It has been designed to be a fast, flexible support for virtual machines running on the same host. It does not support distributed virtual networks, and requires root access for its configuration.
- Vale [20] is a very fast support for virtual networking, based on the netmap [21] API. It uses shared memory techniques to speed-up the communication between the VMs. Vale, like OpenVswitch, does not directly support distributed networks and must be managed by system administrators.

B. TCP-IP stacks

As described in the introduction, the TCP-IP networking stack is generally unique in a system and it is considered as a shared systemwide service provided by the kernel. The implementation of the TCP-IP stack can be found in the kernel source code of all the free software kernels. In Linux, the IPv4 stack is in the directory /net/ipv4 and the IPv6 stack is in /net/ipv6. Alpine [22] is an early approach to network virtualization for protocol development. In fact, Alpine used a customized BSD kernel running as a user process to serve as a partial virtual machine monitor to virtualize the system calls for networking. TCP-IP stack implementations as libraries are common as software tools for embedded system design. Many manufacturers of embedded system platforms provide their own TCP-IP libraries as part of their development kits. Some of these libraries have been released as free software. Unfortunately, many of these libraries provide minimal or partial implementation of the networking stacks to be used for specific purposes only and are tailored or optimized for a specific embedded system hardware architecture. Adam Dunkels wrote two general purpose and free licensed TCP-IP stacks for embedded systems: uIP [23] and LWIP (Light Weight IP) [24]. uIP is a very compact stack for microcontrollers having limited resources, while LWIP is a more complete implementation for

powerful embedded machines. LWIP was initially designed for IPv4, but a basic support for IPv6 has recently been added. In 2005, when LWIP did not support IPv6 yet, VirtualSquare labs created a fork of LWIP, named LWIPv6 [25]. LWIPv6 then evolved independently and is now a library supporting both IPv4 and IPv6 as a single hybrid stack, i.e., differently from the dual-stack approach, LWIPv6 manages IPv4 packets as a subcase of IPv6 packets. When LWIPv6 dispatches an IPv4 packet it creates a temporary IPv6 header, used by the stack, which is deleted when the packet is later delivered. LWIPv6 is also able to support several concurrent TCP-IP stacks. It has features like packet filtering, NAT (both NATv4 and NATv6), slirp (for IPv4 and IPv6) [26].

C. Process/OS interface

In this work, we use two different approaches to interface user processes with virtual stacks and virtual networks. A way to create networked software appliances is to run entire operating systems for embedded computers as processes on a server. Contiki [27], or similar OSs, can be used to implement new software appliances from scratch. This approach cannot be used to interface existing programs (e.g., an existing web server like Apache) to a virtual network, unless the software interface for networking is completely rewritten to support virtual networking.

ViewOS [28] is a partial virtualization project. View-OS virtualizes the system calls generated by the programs, so unmodified binary programs can run in the virtualized environment. ViewOS supports the re-definition of the networking services at user level. Processes running in a View-OS partial virtual machine, where the networking has been virtualized, will use a user-mode stack instead of the kernel provided one. Server, client and peer-to-peer programs can run transparently on a View-OS machine as if they were running just on the OS, but using a virtualized stack instead of the kernel stack.

Another project provides network virtualization in the NetBSD environment: Rump Anykernel [29]. The idea of Rump is to provide user-mode environments where kernel drivers and services can run. Rump provides a very useful structure for kernel code implementation and debugging, as entire sections of the kernel can run unmodified at user level. In this way, it is possible to test unstable code without the risk of kernel panic.

At the same time, Rump provides a way to run kernel services, like the TCP-IP stack, at user level. It is possible to reuse the kernel code of the stack as a networking library, or as a networking deaemon at user level. Antti Kantee, the author of Rump, named this idea Anykernel. In Rump, it is possible to run each device driver or system service in three different ways: as kernel code, as user-mode code embedded in the application process, or as a user-mode server. These three operational modes respectively correspond to three kernel architectures, when applied to all the drivers and services: monolithic kernels, exokernels and microkernels. Then an Anykernel is a kernel where the kernel architecture is flexible and can be independently decided on each driver or server.

D. Multiple Stack support

Some IoTh applications require the ability for one process to be connected to several TCP-IP stacks at the same time. The Berkeley sockets API has been designed to support only a single implementation for each protocol family.

ViewOS and LWIPv6 use an extension of the Berkeley sockets API, *msocket* [30], providing the support for multiple protocol stacks for the same protocol family. A new system call *msocket* is an extended version of the *socket* system call: *msocket* has one more (leading) argument, the pathname of a special file which defines the stack to use. *msocket* is back compatible with the standard Berkeley sockets API: it is possible to define and modify the current default stack of a process. The *socket* system call will use the current default stack. The default stack is part of the process status, and it is inherited in the case of *fork* or *execve*. Existing programs using only *socket* can work on any available stack, one at a time.

V. Usage Cases

This section describes some general usage cases of IoTh. A complete description of the experiments, including all the details to test the results, can be found in Section VII.

A. Client Side usage cases

- Co-existence of multiple networking environments. This feature can be used in many ways. For example, it is possible to have a secure VPN connected to the internal protected network of an institution or a company (an intranet) on which it is safe to send sensitive data and personal information, and a second networking environment to browse the Internet.
  As a second example, technicians who need to track networking problems may find it useful to have some processes connected to the faulty service, while a second networking environment can be used to look for information on the Internet, or to test the faulty network by trying to reach the malfunctioning link from the other end.
- Creation of networking environments for IPC. Many programs have web user interfaces for their configuration (e.g., CUPS or xmbc). Web interfaces are highly portable and do not require specific graphics libraries to run. Using IoTh it is possible to create several Local Host Networks (LHN), i.e., virtual networks for IPC only, to access the web interfaces of the running processes. LHN can have access protection, e.g., an LHN to access the configuration interface of critical system daemons can be accessible only by *root* owned processes. All daemons can have their own IP address, logical name and run their web based configuration interface using port 80.
  Let us take the CUPS example. The web interface of CUPS is available at the port 631 on localhost, and allows users to read the status of the printers. System administrators can add or reconfigure printers, CUPS asks for a password authentication for these operations. Using IoTh it is possible to define the FQDN *cups.localhost*

to be a static IP address in the file /etc/hosts. An IoTh version of CUPS could join two LHN using *msockets* and use the same IP address on both. One virtual network is for system administrators, the other for users. User browsers can be connected to the LHN for system administrators if they are enabled, otherwise they can use the other LHN. All the configuration options will be disabled by CUPS for all accesses from the user LHN. The same approach as CUPS could be used for many system daemons providing a web interface. In this way, the LHN for system administration works as a single-sign-on channel for all the daemons. In fact, system administrators will not be required to be authenticated by a password, as only processes owned by a privileged account can join that LHN.

- Per user IP addresses. It is possible to use IoTh to assign an IP address to each user working on the same server. This is useful, for example, in critical environments, where tracking the responsibility of all the activities on the network is required. In multiuser and multitasking operating systems using a shared TCP-IP stack it is not generally possible, unless the configuration forces the system to log the mapping between each TCP connection or UDP datagram, and the owner of the process which requested the networking operation. It would be clearly a very expensive procedure in terms of processing power and storage requirements. This problem has been described in detail in User Level Networking (ULN) [31]. Through IoTh, each user can be given their own VDE, where their TCP-IP stacks or their processes can be connected. The router of user VDEs can assign IP addresses, or a range of IP addresses, to each user. In this way, there will be a direct mapping between each IP address and the user responsible for it. Likewise, it is possible to track the personal computers connected to a public network. Because of the distributed nature of VDE, the IP addresses of a user can be assigned to processes running on different hosting computers.
  IP addresses assigned per user allows for the implementation of differentiated services. Users or classes of users can be assigned different QoS and access constraints.

B. Server side usage cases

- Virtual hosting is a well-known feature of several networking servers: the same server provides the same kind of service for multiple domains. Apache web server is one of the most common examples of daemons supporting virtual hosting. The same Apache process can work as a web server for many domains. The target IP address used by the client, or the address specified in the GET request of the html protocol, can be used by Apache to assign each request to a domain.
  IoTh generalizes this idea. It is possible to run several instances of the same networking daemon, giving each one its IP address. It is possible to run several pop, imap, DNS, web, MTA, etc., daemons, each one using its own stack. All the daemons will use their standard port numbers.
  It may be objected that it is already possible to obtain a similar result by assigning all the IP addresses to the networking controller of the server (or to a controller) and configuring each daemon to bind its specific IP address. In this way, one shared TCP-IP stack manages all the addresses, and each daemon selects the one to use. Unfortunately, this makes this approach complex and prone to error for system administrators.
  For example, the difference between this approach and the IoTh method is clear when the IP address of a daemon must be modified. Several configuration files must be edited in a consistent manner, using one shared stack to complete the required operation: /etc/network/interfaces, the daemon's configuration files and, in well configured systems, the *iptables* directives of the firewalling rules.
  Using IoTh it is sufficient to change the daemon's address.
  It is possible to envisage daemons designed for IoTh that run several concurrent networking stacks and provide specific services, depending on the stack they receive the requests from.
- Service migration in IoTh is as simple as stopping the daemon process on one host and starting it on another one. In fact, a daemon process can have its embedded networking stack, so its IP address and its routing rules are just configuration parameters of the daemon process itself. A VDE can provide a virtual Ethernet for all the processes running on several hosts. Stopping the daemon process on one server and activating it later on a second server providing the same VDE is, in the virtual world, like unplugging the Ethernet cable of a computer from a switch and plugging it into a port of another switch of the same LAN (the ports of both switches should belong to the same VLAN).
- With IoTh it is possible to design network daemons which change their IP addresses in a dynamic way. One Time IP address (OTIP) [32] applies to IP addresses the same technique used for passwords in One Time Password (OTP) services [33]. In OTP, the password to access a service changes over time, and the client must compute the current password to be used to access the service. This is common for protecting on-line operations on bank accounts. When a customer wants to use his/her account, the current password must be copied from the display of a small key-holder device. OTIP uses the same concept to protect private services accessible on the Internet. A daemon process changes its IP address dynamically over time and all its legitimate users can compute its current IP address using a specific tool, and connect. Port scan traces and network dumps cannot provide useful information for malicious attacks, because all the addresses change rapidly.
  This method has mainly been designed for IPv6 networks. In fact, the current server address can be picked up as one

of the valid host addresses available on the local network, most of the time among 2^{64} possible addresses. Clearly, a 2^{64} address space is too large for attackers to try a brute force enumeration attack on all the available addresses, and even if they eventually succeeded, the retrieved addresses would have to be exploited before their validity expires and the servers move to new addresses.

C. Other usage cases

IoTh allows us to use several networking stacks. These stacks can be several instances of the same stack, or different stacks. In fact, it is possible to have different implementations of TCP-IP stacks or stacks configured in different ways, available at the same time. Processes can choose which one is best suited to their activities.

This feature can be used in different ways:

- Using an experimental stack as the single, shared stack of a remote computer can partition the remote machine in cases of a malfunctioning of the stack itself. IoTh enables the coexistence of the stack under examination with a reliable production stack, which can be used as a safe communication channel.
- Processes can have different networking requirements. For example, communicating peers on a high latency link need larger buffers for the TCP sliding window protocol. It is possible to configure each stack and fine tune its parameters for the requirements of each process, as each process can have its own stack.

VI. Hash-Based Addresses

The deployment of IoTh based services requires the definition of several IP addresses. In fact, the number of IP addresses used by IoTh can be orders of magnitude higher than the number of IP addresses currently assigned only to hardware or virtual controllers.

Although each (numerical) IP address could be defined by some form of autoconfiguration (stateful or stateless [8], [9]), the real problem is to provide the mapping between each Fully Qualified Domain Name (FQDN) of a service and the corresponding IPv6 address of the process providing the requested service. In other words, the management of a high number of IoTh addresses by the standard Domain Name Service (DNS) procedures can be complex, error prone and time consuming for system administrators.

In [35], we propose a novel IP address self-configuration scheme based on a 64-bit hash encoding of the FQDN to be used as the host part of the IPv6 address. The complete IPv6 address will then be composed by the network prefix of the (virtual) LAN followed by the hash encoding of the FQDN.

The above referenced paper shows how the name resolution of hash-based addresses can be managed automatically, requiring system administrators the provide solely a unique FQDN for each service. The hash-based IP addressing self assigning method is completely consistent with the name resolution protocols in use on the Internet [34]. So, existing networking clients and DNS implementations can seamlessly interoperate with DNS providing addresses using the hash-based name resolution.

It is worth noting that hash encoding might generate collisions. Thus, multiple FQDNs can correspond to the same hash-based IP address. The same problem can arise in OTIP generated addresses (see Section V) where two processes could temporarily acquire the same address. In both [35] and [32] there is a discussion about the statistical relevance of the problem. Being an instance of the well-known birthday paradox problem, the probability of hash collision can be estimated as follows:

$$Pr[(n, m)] \approx 1 - e^{-\frac{m^2}{2n}} \quad (1)$$

where m is the number of elements choosing the same random key amongst n possible keys.

Figure 3 shows the probability function (1) plotted for up to 1Mi (i.e., 2^{20}) computers. The probability of two addresses colliding is less than one in 30 million for a LAN connecting more than 1 million hosts. In more realistic cases, network connecting less than one thousand nodes, the probability has the order of magnitude of one in $3 \cdot 10^{14}$.

Although very unfrequent, collisions of hash-defined addresses may happen, but such collisions can be detected by DNS servers and reported to system administrators who can change some of the FQDNs of the services to solve the problem. On the contrary, for OTIP it is not possible to completely avoid the collision problem, which, however, could only cause extremely unlikely temporary unreachability state of the services involved in the collision.

VII. Practical examples

This section presents some practical experiments on IoTh. These experiments are based on several tools which have been designed or extended to provide a working proof-of-concept of IoTh. For an independent testing of the results published in this paper, the source code of all the software is available under free software licenses on public repositories.

The tools used for the experiments include

- Virtual Distributed Ethernet: vde_switch, vde_plug;
- LWIPv6: the stack and sliprv6;
- Contiki: including the VDE interface;
- View-OS: umview or kmview, umnet (network virtualization), umnetlwipv6 (interface to lwipv6), umfuse (virtual file system support), umfusefile (single file virtualization).

A. Example 1: client side IoTh

.

This example shows how to use IoTh to run a browser on its own network.

First of all, start a VDE switch, a ViewOS VM and connect the VDE to a remote network. It is sufficient to have an unprivileged user account on far.computer.org, slirpvde6 is able to route the entire subnet.

```
$ vde_switch -daemon -sock /tmp/vde1
$ umview xterm &
$ dpipe vde_plug /tmp/vde1 = \
```

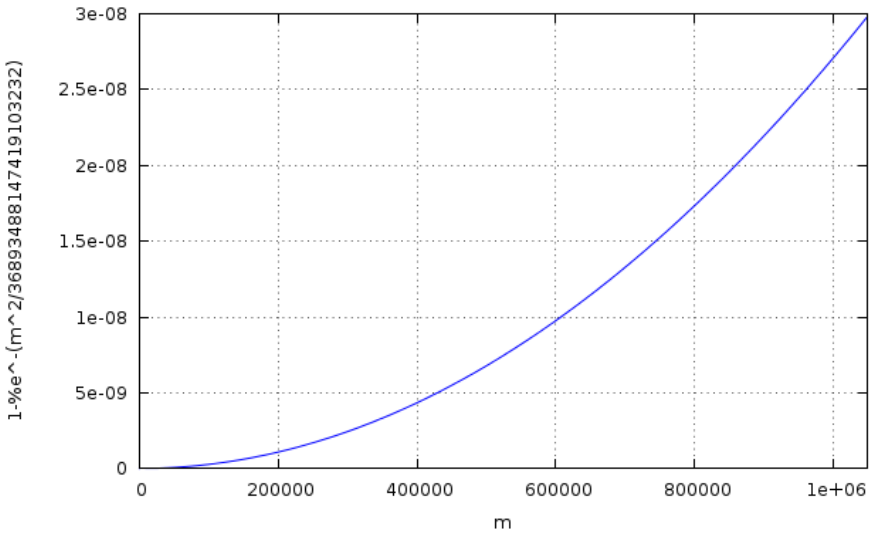


Fig. 3. Probability of address collision in a 64 bits hash

```
> ssh far.computer.org slirpvde6 -D -N -
```

The new xterm is running in the View-OS VM. In that xterm type:

```
$ um_add_service umnet umfuse
$ mount -t umnetlwipv6 none /dev/net/default
$ mount -t umfuseramfile -o ghost \
> /etc/resolv.conf /etc/resolv.conf
$ ip link set vd0 up
$ /sbin/udhcpc -i vd0 -q \
> -s ~/etc/udhcpc/default.script
$ firefox new
```

The first command loads the View-OS modules for network and file system virtualization. View-OS uses the mount operation to load a new network stack. It becomes the default stack for the VM because the target of the mount operation is `/dev/net/default`. `umfuseramfile` is used to virtualize `/etc/resolv.conf`: in this way, the virtual file is writable in this View-OS VM, and it is possible to define the DNS server to be used by the VM.

Then the following commands activate the virtual controller vd0, start a DHCP client to request a dynamic address for vd0, and start a browser.

All the connections of the browser will be seen from the Internet as if they were generated by `far.computer.org`

Figure 4 shows an example in which one browser is connected to the local networking service (the one in Bologna), while a second browser uses IoTh and is connected to an American network (the one in Kansas City).

B. Example 2: server side virtual network appliance

This example shows how to start virtual networked appliances. More specifically, the programs used in this example are a simple web server, based on Contiki, and a simple virtual network-attached storage (NAS), based on LWIPv6.

The source code for the Contiki example is included in the subversion *(svn)* repository of VDE on sourceforge [36] as a patch to the Contiki source tree. From the patched version of Contiki it is possible to generate the test program named `webserver-example.minimal-net-vde`.

As a first step, launch a VDE switch connected to a tap interface and assign an IP address to it.

```
$ sudo vde_switch -d -s /tmp/vde2 -tap tap0
$ sudo ip link set tap0 up
$ sudo ip addr add 192.168.100.1/24 dev tap0
```

Then start the Contiki web server:

```
export CONTIKIIP=192.168.100.2
export CONTIKIMASK=255.255.255.0
export CONTIKIVDE=/dev/vde2
webserver-example.minimal-net-vde
```

From a browser it is now possible to reach the Contiki web server at its own IP address, as shown in Figure 5.

The next step shows how to migrate the Contiki web server to another hosting machine. Then compile the Contiki web server on the other host, or copy the executable file, if the architecture of the remote machine is compatible with the local one. Open a new terminal window and create a VDE cable to the remote machine (far.machine).

```
$ dpipe -d vde_plug /tmp/vde2 = \
> ssh far.machine vde_plug /tmp/vde2[]
```

Open an ssh connection on the remote machine and start the Contiki web server using the same sequence of commands described here above. Now reload the web page on your browser. Nothing seems to change. The page is overwritten by an identical one, but that page is now provided by the Contiki server running on the remote machine.

The NAS example uses LWIPv6. The source code for this example is available from the wiki site of VirtualSquare [37].

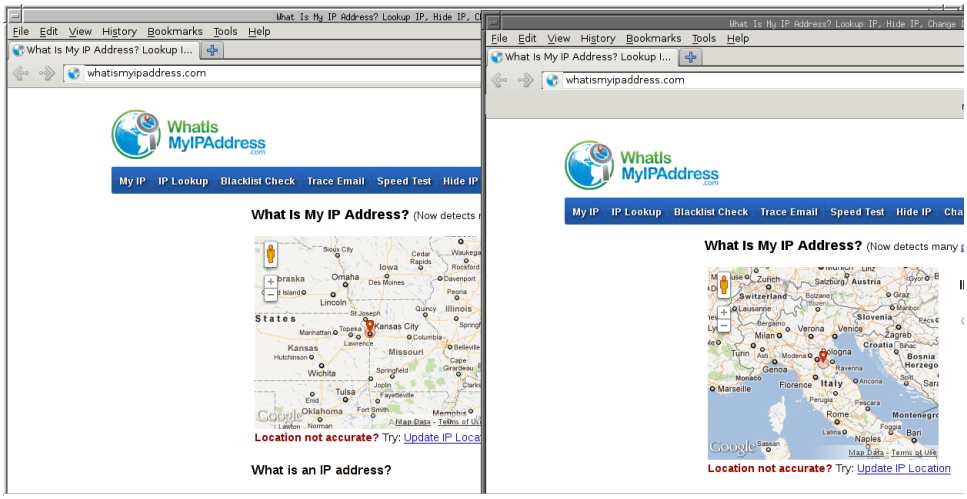


Fig. 4. Client side example of IoTh: the browsers are using two different stacks. The one in Kansas City uses a user-mode TCP-IP stack and a virtual networking switch.

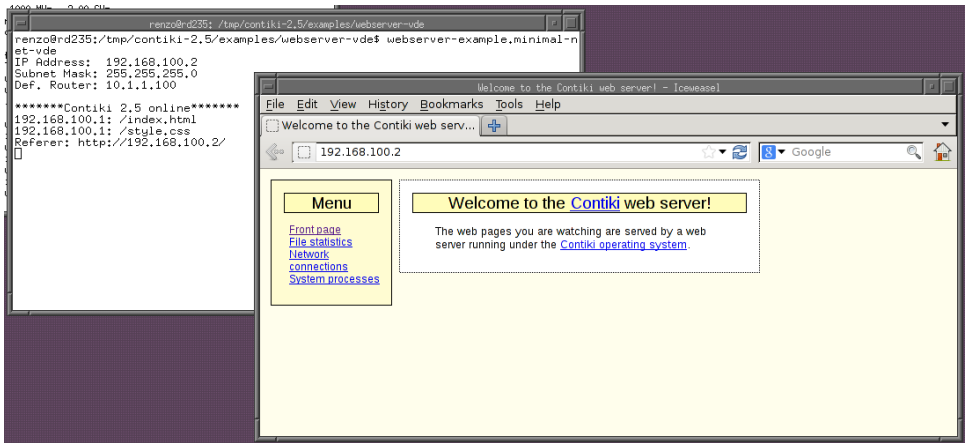


Fig. 5. A server side example of IoTh usage: the web page of the browser in foreground is provided by the contiki based virtual networked appliance started in the terminal window in background.

On a computer where the LWIPv6 library has already been installed (it can be generated from the sources available on the svn repository or simply installed as a packet for Debian Sid users), compile the webnas4 example.

Change your current directory to a subtree of the file system which does not contain private data, as its contents will be exported, and start webnas4 as follows:

```
$ /path/to/webnas4 \
> 192.168.100.3/24+02:01:02:03:04:05 /dev/vde2
```

All the contents of the current directory are now exported to the VDE network and accessible via web by loading the page `http://192.168.100.3` on the browser.

It is possible to use static global IP addresses for the Contiki address and for the NAS example, and to define a default router for both as follows:

```
$export CONTIKIDR=192.168.100.1
```

for Contiki and the option `route:192.168.100.1` for webnas4, using a convenient global IP prefix instead of 192.168.100. In this way, both virtual network appliances can be reached by any Internet user. These examples use IPv4 to shorten the address in the commands and to make them more readable, although they can be modified to use IPv6 instead.

VIII. Security Considerations

Several aspects of security must be taken into consideration in IoTh.

It is possible to limit the network access possibilities of an IoTh process and restrict the network services it can use. In fact, each IoTh process must be connected to a virtual local network to communicate, and virtual local networks have access control features. In VDE, for example, the permission to access a network is defined using the standard access control mechanisms of the file system. Each VDE network can be restricted to specific users, groups using the file permissions or Access Control Lists (ACL). The interaction between processes connected to a VDE and the other networks (or the entire Internet) can be regulated by specific configurations of

TABLE I
COMPARISON IN BANDWIDTH (MB/S) BETWEEN A KERNEL STACK AND IOTH

	10MB kernel	10MB IoTh	20MB kernel	20MB IoTh	40MB kernel	40MB IoTh
localhost	116	29.9	118	35.9	136	37.4
network 1Gb/s	104	41.9	112	49.0	112	51.7
network 100Mb/s	11.2	11.0	11.1	11.0	11.1	11.0

the virtual routers used to interconnect that VDE.

Limiting the IoTh process access to networking is just one aspect of IoTh security. It protects the environment from the effects of faulty, buggy or malicious processes.

It is also possible to consider the positive effects of IoTh with respect to protection from external attacks. Port scanning [38] is a method used by intruders to get information about a remote server, planned to be a target for an attack. A port scan can reveal which daemons are currently active on that server, then which security related bugs can be exploited.

This attack method is based on the assumption that all the daemons are sharing the same IP stack and the same IP addresses. This assumption is exactly the one negated by IoTh. Port scanning is almost useless in IoTh, since an IP address is daemon specific, so it would reveal nothing more than the standard ports used for that service. When IoTh is applied to IPv6, the process IP address on a VDE network can have a 64-bit prefix and 64 bits for the node address. A 64-bit address space is too large for a brute force address scan to be effective.

There are also other aspects of security to be considered regarding the effects of IoTh on the reliability of the hosting system. Daemon processes run as unprivileged user processes in IoTh. They do not even require specific capabilities to provide services on privileged ports (CAP_NET_BIND_SERVICE to bind a port number less than 1024). The less privileged a daemon process is, the smaller the damage it may cause in cases when the daemon is compromised (e.g., by a buffer overflow attack).

In some cases, IoTh can limit the effects of Denial of Service (DoS) attacks. In fact, DoS attacks may succeed by overloading not only the communication channels, but also the TCP-IP stacks of the target machine. In this latter case, in IoTh, a maximum load boundary for the daemon process can confine the effects of the attack to the target service, which is overloaded by the high rate of requests, while the other daemons running on the same host would be much less affected.

IX. PERFORMANCE OF IOTH

IoTh provides a new viewpoint on networking. As this paper has shown in the previous sections, IoTh allows a wide range of new applications. IoTh flexibility obviously costs in terms of performance. A fair analysis of IoTh performance has to consider the balance between the costs of using this new feature and the benefits it gives. In the same way, processes run faster on an Operating System not supporting Virtual Memory, but, for many applications, the cost of Virtual Memory is worthwhile because you can run a greater number of processes. The IoTh approach can co-exist with the standard management

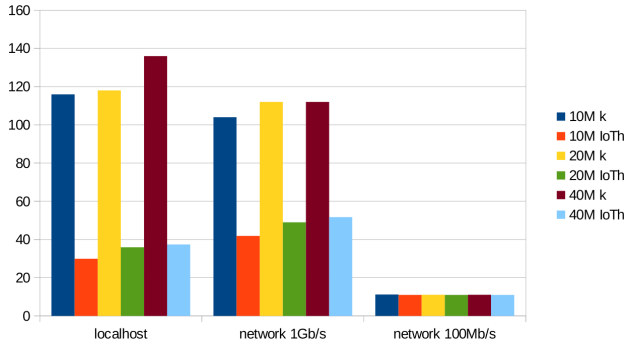


Fig. 6. A graphical view of Table I data

of IP addresses and services. System administrators can decide which approach is more suitable for each service.

Table I shows the comparison of the bandwidth of a TCP connection between the Linux Kernel TCP-IP stack implementation and a IoTh implementation based on VDE and LWIPv6. The program used for the test is the NAS example of the previous section. The test set includes the measure of the bandwidth for file transfers of 10MB, 20MB and 40MB between processes running on the same host, on hosts connected by a 100Mb/s LAN and by a 1Gb/s LAN. The test environment consists of two GNU-Linux boxes (Debian SID distribution), Linux 3.2 kernel, NetXtreme BCM5752 controller, dual core Core2Duo processor running at 2Ghz, HP ProCurve Switches 1700 and 1810G. The files have been transferred using wget.

From the table and from the graph (Fig. 6) it is possible to see that IoTh can reach a sustained load of about 50MB/s, so the overhead added by the new approach is appreciable only on very fast communication lines. On a 100Mb/s LAN the difference is minimal. The improved performance for larger file transfers is caused by the constant startup cost (socket opening, http protocol, etc.), which is distributed on a longer operation. On localhost or on fast networks, the bandwidth of IoTh is about a quarter to a half of the bandwidth reached by the kernel.

It is worth considering that, in this test, both VDE and LWIPv6 run at user level. These are the performance values of the less efficient implementation structure of IoTh. Kernel level implementations of the TCP-IP stack library, and of the virtual networking switch engine, will increase the performance of IoTh.

X. CONCLUSION AND FUTURE WORK

IoTh opens up a range of new perspectives and applications in the field of Internet Networking.

IoTh unifies the role of networking and IPC, so it can play an important role in the design of future applications: distributed applications and interoperating processes can use the same protocols to communicate.

The research on IoTh is currently working in two different directions: the design of new IoTh applications and the evolution of the infrastructures to support IoTh.

The challenge of supporting new IoTh based services creates a need to analyze the TCP-IP protocols, in order to evaluate if and how these protocols, designed for physical networks, need to be modified or updated to be effective in IoTh. An example of a question that needs to be evaluated is whether the DNS protocol can have specific queries or features for IoTh.

On the other hand, IoTh requires an efficient infrastructure, able to provide a virtual networking (Ethernet) service to processes. This support must be optimized by integrating the positive results of currently available projects and then extending them to provide new services. For example, the speed of Vale [20] should be interfaced with the flexibility of VDE. There are several features in use on real networks that could be ported on virtual networks, e.g., port trunking. The research should also consider new efficient ways of interconnecting the local virtual networks to provide a better usage of virtual links, both for efficiency and for fault tolerance.

All the software presented in this paper has been released under free software licenses and has been included in the Virtual Square tutorial disk image [39]. This disk image can be used to boot a Debian SID GNU-Linux virtual machine. All the software tools and libraries used in this paper have already been installed and the source code of everything not included in the standard Debian distribution is also available in the disk image itself.

ACKNOWLEDGMENTS

I would like to express my gratitude to all the software designers and developers of the VirtualSquare Lab who have shared my daydreaming about the virtualization of everything, patiently following all my brainstorming. This paper is an extended version of [1], published by IARIA in the proceedings of The Eighth International Conference on Internet and Web Applications and Services (ICIW 2013). I wish to thank all the anonymous reviewers of IARIA for their detailed comments and suggestions.

REFERENCES

[1] R. Davoli, "Internet of threads," in *ICIW 2013, The Eighth International Conference on Internet and Web Applications and Services*. IARIA, 2013, pp. 100–105.

[2] J. Postel, "DoD standard Internet Protocol," RFC 760, Internet Engineering Task Force, Jan. 1980, obsoleted by RFC 791, updated by RFC 777. [Online]. Available: http://www.ietf.org/rfc/rfc760.txt 02.25.2014

[3] F. Piedad and M. Hawkins, *High Availability: Design, Techniques and Processes*, ser. Harris Kern's Enterprise Computing Institute Series. Prentice Hall, 2000.

[4] P. McHardy *et al.*, "Netfilter," http://www.netfilter.org.

[5] D. Price and A. Tucker, "Solaris zones: Operating system support for consolidating commercial workloads," in *Proceedings of the 18th USENIX conference on System administration*, ser. LISA '04. Berkeley, CA, USA: USENIX Association, 2004, pp. 241–254.

[6] LXC team, "lxc linux containers," http://lxc.sourceforge.net/ 02.25.2014.

[7] IEEE and The Open Group, "Posix.1 2008," http://pubs.opengroup.org/onlinepubs/9699919799/ 02.25.2014.

[8] S. Alexander and R. Droms, "DHCP Options and BOOTP Vendor Extensions," RFC 2132 (Draft Standard), Internet Engineering Task Force, Mar. 1997, updated by RFCs 3442, 3942, 4361, 4833, 5494. [Online]. Available: http://www.ietf.org/rfc/rfc2132.txt

[9] S. Thomson and T. Narten, "IPv6 Stateless Address Autoconfiguration," RFC 2462 (Draft Standard), Internet Engineering Task Force, Dec. 1998, obsoleted by RFC 4862. [Online]. Available: http://www.ietf.org/rfc/rfc2462.txt

[10] P. Srisuresh and K. Egevang, "Traditional IP Network Address Translator (Traditional NAT)," RFC 3022 (Informational), Internet Engineering Task Force, Jan. 2001. [Online]. Available: http://www.ietf.org/rfc/rfc3022.txt

[11] K. Ashton, "That 'Internet of Things' thing," *RFID Journal*, vol. 22, pp. 97–114, 2009.

[12] M. Krasnyansky, "Universal tun/tap device driver," 1999, linux Kernel Documentation: Documentation/networking/tuntap.txt.

[13] J. D. Dike, "User-mode linux," in *Proc. of 2001 Ottawa Linux Symp. (OLS)*, Ottawa, 2001.

[14] R. Davoli, "Vde: Virtual distributed ethernet," in *Proceedings of the First International Conference on Testbeds and Research Infrastructures for the DEvelopment of NeTworks and COMmunities*, ser. TRIDENTCOM '05. Washington, DC, USA: IEEE Computer Society, 2005, pp. 213–220.

[15] F. Bellard, "Qemu project," http://fabrice.bellard.free.fr/qemu/.

[16] A. Kivity, "kvm: the Linux virtual machine monitor," in *OLS '07: The 2007 Ottawa Linux Symposium*, Jul. 2007, pp. 225–230.

[17] Oracle inc., "Oracle vm virtualbox," https://www.virtualbox.org/.

[18] R. Davoli, "Inter process networking (ipn)," http://wiki.virtualsquare.org/wiki/index.php/IPN, 2007.

[19] Open vSwitch team, "Open vswitch," http://openvswitch.org/ 02.15.2014.

[20] L. Rizzo and G. Lettieri, "Vale, a switched ethernet for virtual machines," University of Pisa, Italy, Tech. Rep., 2012. [Online]. Available: http://info.iet.unipi.it/∼luigi/papers/20120608-vale.pdf 02.25.2014

[21] L. Rizzo, "Revisiting network i/o apis: the netmap framework," *Commun. ACM*, vol. 55, no. 3, pp. 45–51, 2012.

[22] D. Ely, S. Savage, and D. Wetherall, "Alpine: A user-level infrastructure for network protocol development," in *Proc. of 3rd USENIX Symp. on Internet Technologies and Systems (USITS01)*, March 2001, san Francisco.

[23] A. Dunkels, "Full tcp/ip for 8-bit architectures," in *Proceedings of the 1st international conference on Mobile systems, applications and services*, ser. MobiSys '03. New York, NY, USA: ACM, 2003, pp. 85–98.

[24] A. Dunkels, L. Woestenberg, K. Mansley, and J. Monoses, "Lwip," http://savannah.nongnu.org/projects/lwip 02.25.2014.

[25] R. Davoli, "Lwipv6," http://wiki.virtualsquare.org/wiki/index.php/LWIPV6 02.25.2014, 2007.

[26] K. Price, "Slirp, the ppp/slip-on-terminal emulator," http://slirp.sourceforge.net 02.25.2014.

[27] A. Dunkels, B. Gronvall, and T. Voigt, "Contiki - A Lightweight and Flexible Operating System for Tiny Networked Sensors," in *Proceedings of the 29th Annual IEEE International Conference on Local Computer Networks*, ser. LCN '04. Washington, DC, USA: IEEE Computer Society, 2004, pp. 455–462.

[28] L. Gardenghi, M. Goldweber, and R. Davoli, "View-os: A new unifying approach against the global view assumption," in *Proceedings of the 8th international conference on Computational Science, Part I*, ser. ICCS '08. Berlin, Heidelberg: Springer-Verlag, 2008, pp. 287–296.

[29] A. Kantee, "Flexible operating system internals: The design and implementation of the anykernel and rump kernels," 2012, doctoral Dissertation, Aalto Univerisity, Finland.

[30] R. Davoli and M. Goldweber, "msocket: multiple stack support for the berkeley socket api," in *SAC '12: Proceedings of the 27th Annual ACM Symposium on Applied Computing*. New York, NY, USA: ACM, 2012, pp. 588–593.

[31] A. Pira, E. Tassi, and R. Davoli, "Managing User Level Networking-Personal IP networks," in *Proc. of ICN 04 - International Conference on Networking*, April 2004.

[32] R. Davoli, "Otip: One time ip address," in *ICSNC 2013, The Eighth International Conference on Systems and Networks Communications*. IARIA, 2013, pp. 154–158.

[33] D. M'Raihi, S. Machani, M. Pei, and J. Rydell, "TOTP: Time-Based One-Time Password Algorithm," RFC 6238 (Informational), Internet Engineering Task Force, May 2011. [Online]. Available: http://www.ietf.org/rfc/rfc6238.txt

[34] P. Mockapetris, "Domain names - implementation and specification," RFC 1035 (Standard), Internet Engineering Task Force, Nov. 1987, updated by RFCs 1101, 1183, 1348, 1876, 1982, 1995, 1996, 2065, 2136, 2181, 2137, 2308, 2535, 2845, 3425, 3658, 4033, 4034, 4035, 4343, 5936, 5966, 6604. [Online]. Available: http://www.ietf.org/rfc/rfc1035.txt

[35] R. Davoli, "Ipv6 hash-based addresses for simple network deployment," in *AFIN 2013, The Fifth International Conference on Advances in Future Internet*. IARIA, 2013, pp. 15–20.

[36] ——, "Vde sourceforge home page," http://vde.sourceforge.net.

[37] R. Davoli *et al.*, "Virtual square wiki," http://wiki.virtualsquare.org/.

[38] Fyodor Vaskovich (Gordon Lyon), "The art of port scanning," *Phrack*, vol. 7, no. 51, 1997.

[39] R. Davoli, "Virtual square tutorial disk image," http://wiki.virtualsquare.org/wiki/index.php/Virtual_Square_Tutorial_Disk_Image 02.25.2014.

A Robust Approach to Large Size Files Compression using the MapReduce Web Computing Framework

Sergio De Agostino
Computer Science Department
Sapienza University
Rome, Italy
Email: deagostino@di.uniroma1.it

***Abstract*—Lempel-Ziv (LZ) techniques are the most widely used for lossless file compression. LZ compression basicly comprises two methods, called LZ1 and LZ2. The LZ1 method is the one employed by the family of Zip compressors, while the LZW compressor implements the LZ2 method, which is slightly less effective but twice faster. When the file size is large, both methods can be implemented on a distributed system guaranteeing linear speed-up, scalability and robustness. With Web computing, the MapReduce model of distributed processing is emerging as the most widely used. In this framework, we present and make a comparative analysis of different implementations of LZ compression. An alternative to standard versions of the Lempel-Ziv method is proposed as the most efficient one for large size files compression on the basis of a theoretical worst case analysis, which evidentiates its robustness.**

Keywords-web computing; mapreduce framework; lossless compression; string factorization; worst case analysis

I. Introduction

Lempel-Ziv (LZ) techniques are the most widely used for lossless file compression and preliminary results on the distributed implementation, shown in this paper, were presented in [1], [2], [3]. LZ compression [4], [5], [6] is based on string factorization. Two different factorization processes exist with no memory constraints. With the first one (LZ1) [5], each factor is independent from the others since it extends by one character the longest match with a substring to its left in the input string. With the second one (LZ2) [6], each factor is instead the extension by one character of the longest match with one of the previous factors. This computational difference implies that while sliding window compression has efficient parallel algorithms [7], [8], [9], [10], LZ2 compression is hard to parallelize [11] and less effective in terms of compression. On the other hand, LZ2 is more efficient computationally than sliding window compression from a sequential point of view. This difference is maintained when the most effective bounded memory versions of Lempel-Ziv compression are considered [9], [12]. While the bounded memory version of LZ1 compression is quite straightforward, there are several heuristics for limiting the work-space of the LZ2 compression procedure in the literature. The "least recently used" strategy (LRU) is the most effective one. Hardness results inside Steve Cook's class (SC) have been proved for this approach [12], implying the likeliness of the non-inclusion of the LZ2-LRU compression method in Nick Pippenger's class (NC). Completeness results in SC have also been obtained for a relaxed version of the LRU strategy (RLRU) [12]. RLRU was shown to be as effective as LRU in [13] and, consequently, it is the most efficient one among the Lempel-Ziv techniques.

Bounding memory is very relevant with distributed processing and it is an important requirement of the MapReduce model of computation for Web computing. A formalization of this model was provided in [14], where further constraints are formulated for the number of processors, the number of iterations and the running time. However, such constraints are a necessary but not sufficient condition to guarantee a robust linear speed-up. In fact, interprocessor communication is allowed during the computational phase and experiments are needed to verify an actual speed-up. Distributed algorithms for the LZ1 and LZ2 methods approximating in practice their compression effectiveness have been realized in [9], [15], [16], where the stronger requirement of no interprocessor communication during the computational phase is satisfied. In fact, the approach to a distributed implementation in this context consists of applying the sequential procedure to blocks of data independently.

In Sections II and III, we describe the Lempel-Ziv compression techniques and their bounded memory versions respectively. Section IV sketches past work on the study of the parallel complexity of Lempel-Ziv methods leading to the idea of relaxing the least recently used strategy. In Section V, we present the MapReduce model of computation and introduce further constraints for a robust approach to a distributed implementation of LZ compression on the Web. Section VI makes a comparative analysis of different implementations of LZ compression in this framework. A worst case analysis of standard LZ2 compression is given in Section VII and an alternative to the standard versions is proposed as the most efficient one for large size files compression. Conclusions and future work are given in Section VIII.

II. Lempel-Ziv Data Compression

Lempel-Ziv compression is a dictionary-based technique. In fact, the factors of the string are substituted by *pointers* to copies stored in a dictionary which are called *targets*. LZ1 (LZ2) compression is also called the sliding (dynamic) dictionary method.

A. LZ1 Compression

Given an alphabet A and a string S in A^*, the LZ1 factorization of S is $S = f_1 f_2 \cdots f_i \cdots f_k$, where f_i is the shortest substring which does not occur previously in the prefix $f_1 f_2 \cdots f_i$, for $1 \leq i \leq k$. With such factorization, the encoding of each factor leaves one character uncompressed. To avoid this, a different factorization was introduced (LZSS factorization) where f_i is the longest match with a substring occurring in the prefix $f_1 f_2 \cdots f_i$ if $f_i \neq \lambda$, otherwise f_i is the alphabet character next to $f_1 f_2 \cdots f_{i-1}$ [17]. f_i is encoded by the pointer $q_i = (d_i, \ell_i)$, where d_i is the displacement back to the copy of the factor and ℓ_i is the length of the factor (LZSS compression). If $d_i = 0$, l_i is the alphabet character. In other words the dictionary is defined by a window sliding its right end over the input string, that is, it comprises all the substrings of the prefix read so far in the computation. It follows that the dictionary is both *prefix* and *suffix* since all the prefixes and suffixes of a dictionary element are dictionary elements. The position of the longest match in the prefix with the current position can be computed in real time by means of a suffix tree data structure [18], [19].

B. LZ2 Compression

The LZ2 factorization of a string S is $S = f_1 f_2 \cdots f_i \cdots f_k$, where f_i is the shortest substring which is different from every previous factor. As for LZ1 the encoding of each factor leaves one character uncompressed. To avoid this a different factorization was introduced (LZW factorization) where each factor f_i is the longest match with the concatenation of a previous factor and the next character [20]. f_i is encoded by a pointer q_i to such concatenation (LZW compression). LZ2 and LZW compression can be implemented in real time by storing the dictionary with a trie data structure. Differently from LZ1 and LZSS, the dictionary is only prefix.

C. Greedy versus Optimal Factorization

The pointer encoding the factor f_i has a size increasing with the index i. This means that the lower is the number of factors for a string of a given length the better is the compression. The factorizations described in the previous subsections are produced by greedy algorithms. The question is whether the greedy approach is always optimal, that is, if we relax the assumption that each factor is the longest match can we do better than greedy? The answer is negative with suffix dictionaries as for LZ1 or LZSS compression. On the other hand, the greedy approach is not optimal for LZ2 or LZW compression. However, the optimal approach is NP-complete [21] and the greedy algorithm approximates with an $O(n^{\frac{1}{4}})$ multiplicative factor the optimal solution [22].

III. Bounded Size Dictionary Compression

The factorization processes described in the previous section are such that the number of different factors (that is, the dictionary size) grows with the string length. In practical implementations instead the dictionary size is bounded by a constant and the pointers have equal size. While for LZSS (or LZ1) compression this can be simply obtained by sliding a fixed length window and by bounding the match length, for LZW (or LZ2) compression dictionary elements are removed by using a deletion heuristic. The deletion heuristics we describe in this section are FREEZE, RESTART, SWAP, LRU [23] and RLRU [12]. Then, we give more details on sliding window compression.

A. The Deletion Heuristics

Let $d + \alpha$ be the cardinality of the fixed size dictionary, where α is the cardinality of the alphabet. With the FREEZE deletion heuristic, there is a first phase of the factorization process where the dictionary is filled up and "frozen". Afterwards, the factorization continues in a "static" way using the factors of the frozen dictionary. In other words, the LZW factorization of a string S using the FREEZE deletion heuristic is $S = f_1 f_2 \cdots f_i \cdots f_k$ where f_i is the longest match with the concatenation of a previous factor f_j, with $j \leq d$, and the next character. The shortcoming of this heuristic is that after processing the string for a while the dictionary often becomes obsolete. A more sophisticated deletion heuristic is RESTART, which monitors the compression ratio achieved on the portion of the imput string read so far and, when it starst deteriorating, restarts the factorization process. Let $f_1 f_2 \cdots f_j \cdots f_i \cdots f_k$ be such factorization with j the highest index less than i where the restart operation happens. Then, f_j is an alphabet character and f_i is the longest match with the concatenation of a previous factor f_h, with $h \geq j$, and the next character (the restart operation removes all the elements from the dictionary but the alphabet characters). This heuristic is used by the Unix command Compress since it has a good compression effectiveness and it is easy to implement. Usually, the dictionary performs well in a static way on a block long enough to learn another dictionary of the same size. This is what is done by the SWAP heuristic. When the other dictionary is filled, they swap their roles on the successive block.

The best deletion heuristic is LRU (last recently used strategy). The LRU deletion heuristic removes elements from the dictionary in a continuous way by deleting at each step of the factorization the least recently used factor that is not a proper prefix of another one. In [12], a relaxed version

(RLRU) was introduced. RLRU partitions the dictionary in p equivalence classes, so that all the elements in each class are considered to have the same "age" for the LRU strategy. RLRU turns out to be as good as LRU even when p is equal to 2 [13]. Since RLRU removes an arbitrary element from the equivalence class with the "older" elements, the two classes (when p is equal to 2) can be implemented with a couple of stacks, which makes RLRU slightly easier to implement than LRU in addition to be more space efficient. SWAP is the best heuristic among the "discrete" ones.

B. Compression with Finite Windows

As mentioned at the beginning of this section, LZSS (or LZ1) bounded size dictionary compression is obtained by sliding a fixed length window and by bounding the match length. A real time implementation of compression with finite window is possible using a suffix tree data structure [24]. Much simpler real time implementations are realized by means of hashing techniques providing a specific position in the window where a good appriximation of the longest match is found on realistic data. In [25], the three current characters are hashed to yield a pointer into the already compressed text. In [26], hashing of strings of all lengths is used to find a match. In both methods, collisions are resolved by overwriting. In [27], the two current characters are hashed and collisions are chained via an offset array. Also the Unix gzip compressor chains collisions but hashes three characters [28].

C. Greedy versus Optimal Factorization

Greedy factorization is optimal for compression with finite windows since the dictionary is suffix. With LZW compression, after we fill up the dictionary using the FREEZE or RESTART heuristic, the greedy factorization we compute with such dictionary is not optimal since the dictionary is not suffix. However, there is an optimal semi-greedy factorization which is computed by the procedure of Figure 1 [29], [30]. At each step, we select a factor such that the longest match in the next position with a dictionary element ends to the rightest. Since the dictionary is prefix, the factorization is optimal. The algorithm can even be implemented in real time with an augmented trie data structure [29].

j:=0; i:=0
repeat forever
  for $k = j + 1$ **to** $i + 1$ **compute**
    $h(k)$: $x_k ... x_{h(k)}$ is the longest match in the k^{th} position
  let k' be such that $h(k')$ is maximum
  $x_j ... x_{k'-1}$ is a factor of the parsing; $j := k'$; $i := h(k')$

Figure 1. The semi-greedy factorization procedure.

IV. LZ Compression on a Parallel System

LZSS (or LZ1) compression can be efficiently parallelized on a PRAM EREW [7], [8], that is, a parallel machine where processors access a shared memory without reading and writing conflicts. On the other hand, LZW (or LZ2) compression is P-complete [11] and, therefore, hard to parallelize. Decompression, instead, is parallelizable for both methods [31]. The asymmetry of the pair encoder/decoder between LZ1 and LZ2 follows from the fact that the hardness results of the LZ2/LZW encoder depend on the factorization process rather than on the coding itself.

As far as bounded size dictionary compression is concerned, the "parallel computation thesis" claims that sequential work space and parallel running time have the same order of magnitude giving theoretical underpinning to the realization of parallel algorithms for LZW compression using a deletion heuristic. However, the thesis concerns unbounded parallelism and a practical requirement for the design of a parallel algorithm is a limited number of processors. A stronger statement is that sequential logarithmic work space corresponds to parallel logarithmic running time with a polynomial number of processors. Therefore, a fixed size dictionary implies a parallel algorithm for LZW compression satisfying these constraints. Realistically, the satisfaction of these requirements is a necessary but not a sufficient condition for a practical parallel algorithm since the number of processors should be linear. The SC^k-hardness and SC^k-completeness of LZ2 compression using, respectively, the LRU and RLRU deletion heuristics and a dictionary of polylogarithmic size show that it is unlikely to have a parallel complexity involving reasonable multiplicative constants [12]. In conclusion, the only practical LZW compression algorithm for a shared memory parallel system is the one using the FREEZE, RESTART or SWAP deletion heuristics. Unfortunately, the SWAP heuristic does not seem to have a parallel decoder. Since the FREEZE heuristic is not very effective in terms of compression, RESTART is a good candidate for an efficient parallel implementation of the pair encoder/decoder on a shared memory parallel system and even on a system with distributed memory. However, in the context of distributed processing of massive data with no interprocessor communication the LZW-RLRU technique turns out to be the most efficient one. We will see these arguments more in detail in the next two sections.

V. The MapReduce Model of Computation

The MapReduce programming paradigm is a sequence $P = \mu_1\rho_1 \cdots \mu_R\rho_R$, where μ_i is a mapper and ρ_i is a reducer for $1 \leq i \leq R$. First, we describe such paradigm and then discuss how to implement it on a distributed system. Distributed systems have two types of complexity, the interprocessor communication and the input-output mechanism. The input/output issue is inherent to any parallel algorithm and has standard solutions. In fact, in [14] the sequence P

does not include the I/O phases and the input to μ_1 is a multiset U_0 where each element is a $(key, value)$ pair. The input to each mapper μ_i is a multiset U_{i-1} output by the reducer ρ_{i-1}, for $1 < i \leq R$. Mapper μ_i is run on each pair (k, v) in U_{i-1}, mapping (k, v) to a set of new $(key, value)$ pairs. The input to reducer ρ_i is U'_i, the union of the sets output by μ_i. For each key k, ρ_i reduces the subset of pairs of U'_i with the key component equal to k to a new set of pairs with key component still equal to k. U_i is the union of these new sets.

In a distributed system implementation, a key is associated with a processor (a node in the Web). All the pairs with a given key are processed by the same node but more keys can be associated to it in order to lower the scale of the system involved. Mappers are in charge of the data distribution since they can generate new key values. On the other hand, reducers just process the data stored in the distributed memory since they output for a set of pairs with a given key another set of pairs with the same given key.

To add the I/O phases to P, we extend the sequence to $\mu_0\mu_1\rho_1 \cdots \mu_R\rho_R\mu_{R+1}\rho_{R+1}$, where (λ, x) is the unique $(key, value)$ pair input to μ_0 with λ the default initial (and final) key and x the input data. μ_0 distributes such data generating the multiset U_0 (μ_1 is the identity function or can be seen as a second step of the input phase). Finally, μ_{R+1} maps U_R to a multiset where all the pair elements have the same key λ and ρ_{R+1} reduces such multiset to the pair (λ, y) with y output data.

The following complexity requirements are stated as necessary for a practical interest in [14]:

- R is polylogarithmic in the input size n;
- the number of processors (or nodes in the Web) involved is O($n^{1-\epsilon}$) with $0 < \epsilon < 1$;
- the amount of memory for each node is O($n^{1-\epsilon}$);
- mappers and reducers take polynomial time in n.

As mentioned in the introduction, such requirements are necessary but not sufficient to guarantee a speed-up of the computation. Obviously, the total running time of mappers and reducers cannot be higher than the sequential one and this is trivially implicit in what is stated in [14]. The non-trivial bottleneck is the communication cost of the computational phase after the distribution of the original input data among the processors and before the output of the final result. This is obviously algorithm-dependent and needs to be checked experimentally since R can be polylogarithmic in the input size. The only way to guarantee with absolute robustness a speed-up with the increasing of the number of nodes is to design distributed algorithms implementable in MapReduce with $R = 1$. Moreover, if we want the speed-up to be linear then the total running time of mappers and reducers must be O($t(n)/n^{1-\epsilon}$) where $t(n)$ is the sequential time. These stronger requirements are satisfied by the distributed implementations of the several versions of LZ compression discussed in the next section, except for one of them, which requires $R = 2$.

VI. LZ Compression on the Web in MapReduce

We can factorize blocks of length ℓ of an input string in O(ℓ) time with O(n/ℓ) processors, using any of the bounded memory compression techniques. Such distributed algorithms are suitable for a small scale system but due to their adaptiveness, they work on a large scale parallel system only when the file size is large.

A. Sliding Window Compression in MapReduce

With the sliding window method, ℓ is equal to kw where k is a positive integer and w is the window length [9], [15], [16]. The window length is usually several thousands kilobytes. The compression tools of the Zip family, as the Unix command "gzip" for example, use a window size of at least 32K bytes. From a practical point of view, we can apply something like the gzip procedure to a small number of input data blocks, achieving a satisfying degree of compression effectiveness and obtaining the expected speed-up on a real parallel machine. Making the order of magnitude of the block length greater than the one of the window length guarantees robustness on realistic data. It follows that the block length in our parallel implementation should be about 300 kB and the file size should be about one third of the number of processors in megabytes.

In the MapReduce framework, we implement the distributed procedure above with a sequence $\mu_0\mu_1\rho_1\mu_2\rho_2$ where μ_0 and $\mu_2\rho_2$ are the input and output phases, respectively. Let $X = X_1 \cdots X_m$ be the input string where X_i is a substring that has the same length $\ell \geq 300$ kB for $1 \leq i \leq m$. The complexity requirements of the MapReduce model are satisfied by the fact that ℓ is allowed to be strictly greater than 300 kB. The input to μ_0 is the pair $(0, X)$ mapping this element to the set U_0 of pairs $(1, X_1) \cdots (m, X_m)$. U_0 is mapped to itself by μ_1 and ρ_1 reduces (i, X_i) to (i, Y_i) where Y_i is the LZSS coding of X_i for $1 \leq i \leq m$. Finally, μ_2 maps each element (i, Y_i) of its input $U_1 = \{(1, Y_1) \cdots (m, Y_m)\}$ to $(0, Y_i)$ and ρ_2 outputs $(0, Y)$, where $Y = Y_1 \cdots Y_m$. Obviously, the stronger requirements for a linear speed-up, stated in the previous section, are satisfied by this program.

Decompression in MapReduce is simmetrical. To decode the compressed files on a distributed system, it is enough to use a special mark occurring in the sequence of pointers where the coding of a block ends. The input phase distributes among the processors the subsequences of pointers coding each block.

B. LZW Compression Distributed Algorithms

As far as LZW compression is concerned, it was originally presented with a dictionary of size 2^{12}, clearing out the dictionary as soon as it is filled up [20]. The Unix command "compress" employs a dictionary of size 2^{16} and works with the RESTART deletion heuristic. The block length needed to fill up a dictionary of this size is approximately 300 kB. As previously mentioned, the SWAP heuristic is the best deletion heuristic among the discrete ones. After a dictionary is filled up on a block of 300 kB, the SWAP heuristic shows that we can use it efficiently on a successive block of about the same dimension, where a second dictionary is learned. A distributed compression algorithm employing the SWAP heuristic learns a different dictionary on every block of 300 kB of a partitioned string (the first block is compressed while the dictionary is learned). For the other blocks, block i is compressed statically in a second phase using the dictionary learned during the first phase on block $i-1$. But, unfortunately, the decoder is not parallelizable since the dictionary to decompress block i is not available until the previous blocks have been decompressed. On the other hand, with RESTART we can work on a block of 600 kB where the second half of it is compressed statically. We wish to speed up this second phase though, since LZW compression must be kept more efficient than sliding window compression. In fact, it is well-known that sliding window compression is more effective but slower. If both methods are applied to a block of 300 kB and LZW has a second static phase to execute on a block of about the same length, it would no longer have the advantage of being faster. We showed how to speed up in a scalable way this second phase on an extended star network (a tree of height 2) in time $O(km)$ with $O(n/km)$ processors, where k is a positive integer and m is the maximum factor length [2], [15].

In [15], during the input phase the central node of the extended star (that is, the root of the tree) broadcasts a block of length 600 kB to each adjacent processor. Then, for each block the corresponding processor broadcasts to the adjacent leaves a sub-block of length $m(k+2)$ in the suffix of length 300 kB, except for the first one and the last one which are $m(k+1)$ long. Each sub-block overlaps on m characters with the adjacent sub-block to the left and to the right, respectively (obviously, the first one overlaps only to the right and the last one only to the left). Every processor stores a dictionary initially set to comprise only the alphabet characters. The first phase of the computation is executed by processors adjacent to the central node. The prefix of length 300 kB of each block is compressed while learning the dictionary. At each step of the LZW factorization process, each of these processors sends the current factor to the adjacent leaves. They all adds such factor to their own dictionary. After compressing the prefix of length 300 kB of each block, all the leaves have a dictionary stored which has been learned by their parents during such compression phase.

Let us call a *boundary match* a factor covering positions of two adjacent sub-blocks stored by leaf processors. Then, the leaf processors execute the following algorithm to compress the suffix of length 300 kB of each block:

- for each block, every corresponding leaf processor but the one associated with the last sub-block computes the boundary match between its sub-block and the next one ending furthest to the right, if any;
- each leaf processor computes the optimal factorization from the beginning of its sub-block to the beginning of the boundary match on the right boundary of its sub-block (or the end of its sub-block if there is no boundary match).

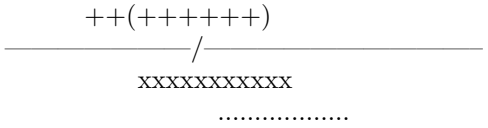


Figure 2. The making of a surplus factor.

Stopping the factorization of each sub-block at the beginning of the right boundary match might cause the making of a surplus factor, which determines the approximation factor $(k+1)/k$ with respect to any factorization. Indeed, as it is shown in Figure 2, the factor in front of the right boundary match (sequence of x's) might be extended to be a boundary match itself (sequence of plus signs) and to cover the first position of the factor after the boundary (dotted line).

In [32], it is shown experimentally that for $k = 10$ the compression ratio achieved by such factorizarion is about the same as the sequential one. Results were presented for static prefix dictionary compression but they are valid for dynamic compression using the LZW technique with the RESTART deletion heuristic. Indeed, experiments were realised compressing similar files in a collection using a dictionary learned from one of them. This is true even if the second step is greedy, since greedy is very close to optimal in practice. Moreover, with the greedy approach it is enough to use a simple trie data structure for the dictionary rather than the augmented suffix trie data structure of [29] needed to implement the semi-greedy factorization in real time. Therefore, in [2] after computing the boundary matches the second part of the parallel approximation scheme was substituted by the following procedure:

- each leaf processor computes the static greedy factorization from the end of the boundary match on the

left boundary of its sub-block to the beginning of the boundary match on the right boundary.

Considering that typically the average match length is 10, one processor can compress down to 100 bytes independently. Then, compressing 300 kB involves a number of processors up to 3000 for each block. It follows that with a file size of several megabytes or more, the system scale has a greater order of magnitude than the standard large scale parameter, making the implementation suitable for an extreme distributed system. We wish to point out that the computation of the boundary matches is very relevant for the compression effectiveness when an extreme distributed system is employed since the sub-block length becomes much less than 1 kB.

With standard large scale systems the sub-block length is several kilobytes with just a few megabytes to compress and the approach using boundary matches is too conservative for the static phase. In fact, a partition of the second half of the block does not effect on the compression effectiveness unless the sub-blocks are very small since the process is static. In conclusion, we proposed in [2] a further simplification of the algorithm for standard small, medium and large scale distributed systems.

Let $p_0 \cdots p_n$ be the processors of a distributed system with an extended star topology. p_0 is the central node of the extended star network and $p_1 \cdots p_m$ are its neighbors. For $1 \leq i \leq m$ and $t = (n-m)/m$ let the processors $p_{m+(i-1)t+1} \cdots p_{m+it}$ be the neighbors of processor i.

$B_1 \cdots B_m$ is the sequence of blocks of length 600 kB partitioning the input file. Denote with B_i^1 and B_i^2 the two halves of B_i for $1 \leq i \leq m$. Divide B_i^2 into t sub-blocks of equal length. The input phase of this simpler algorithm distributes for each block the first half and the sub-blocks of the second half in the following way:

- broadcast B_i^1 to processor p_i for $1 \leq i \leq m$
- broadcast the j-th sub-block of B_i^2 to processor $p_{m+(i-1)t+j}$ for $1 \leq i \leq m$ and $1 \leq j \leq t$

Then, the computational phase is:

in parallel for $1 \leq i \leq m$

- processor p_i applies LZW compression to its block, sending the current factor to its neigbors at each step of the factorization
- the neighbors of processor p_i compress their blocks statically using the dictionary received from p_i with a greedy factorization

As for the sliding window method, decoding the compressed file on a distributed system requires the presence of a special mark occurring in the sequence of pointers each time the coding of a block ends. The input phase distributes the subsequences of pointers coding each block among the processors. If the file is encoded by an LZW compressor implemented with one of the two procedures described in this subsection, a second special mark indicates for each block the end of the coding of a sub-block. The coding of the first half of each block is stored in one of the neighbors of the central node while the coding of the sub-blocks are stored into the corresponding leaves. The first half of each block is decoded by one processor to learn the corresponding dictionary. Each decoded factor is sent to the corresponding leaves during the process, so that the leaves can rebuild the dictionary themselves. Then, the dictionary is used by the leaves to decode the sub-blocks of the second half.

C. LZW Compression in MapReduce

In the MapReduce framework, the program sequence could be $\mu_0\mu_1\rho_1\mu_2\rho_2\mu_3\rho_3$ where $\mu_0\mu_1$ and $\mu_3\rho_3$ are the input and output phases, respectively. Let $X = X_1Y_1 \cdots X_mY_m$ be the input string where X_i and Y_i are substrings having the same length $\ell \geq 300$ kB for $1 \leq i \leq m$ and $Y_i = B_{i,1} \cdots B_{i,r}$ such that $B_{i,j}$ is a substring that has the same length $\ell' \geq 1000$ for $1 \leq j \leq r$. The complexity requirements of the MapReduce model will be satisfied by the fact that ℓ is allowed to be strictly greater than 300 kB and ℓ' strictly greater than 1000 bytes. Keys are pairs of positive integers. The input to μ_0 is the pair $((0,0), X)$, which is mapped to the set U_0 of pairs $((0,1), X_1Y_1)$, $\cdots$, $((0,m), X_mY_m))$, as input to μ_1. Then, μ_1 maps U_0 to the set U_0' of pairs $((0,1), X_1)$, $((1,1), B_{1,1})$, $\cdots$, $((1,r), B_{1,r})$, $\cdots$, $((0,m), X_m))$, $((m,1), B_{m,1})$, $\cdots$, $((m,r), B_{m,r})$. ρ_1 reduces $((0,i), X_i)$ to a set of two $(key, value)$ pairs, that is, $\{((0,i), Z_i), ((0,i), D_i)\}$, where Z_i and D_i are respectively the LZW coding of X_i and the dictionary learned during the coding process. On the other hand, $((i,j), B_{i,j})$ are reduced to themselves by ρ_1 for $1 \leq i \leq m$ and $1 \leq j \leq r$. The second iteration step $\mu_2\rho_2$ works as the identity function when applied to $((0,i), Z_i)$. μ_2 works as the identity function when applied to $((i,j), B_{i,j})$ as well. Instead, $((0,i), D_i)$ is mapped by μ_2 to $((i,j), D_i)$ for $1 \leq j \leq r$. Then, ρ_2 reduces the set $\{((i,j), B_{i,j}), ((i,j), D_i)\}$ to $((i,j), Z_{i,j})$ where $Z_{i,j})$ is the coding produced by the second phase of LZW compression with the static dictionary D_i. Finally, μ_3 maps (i, Z_i) to $((0,0), Z_i)$ and $((i,j), Z_{i,j})$ to $((0,0), Z_{i,j})$. Then, ρ_3 outputs $((0,0), Z)$ where $Z = Z_1Z_{1,1} \cdots Z_{1,r} \cdots Z_mZ_{m,1} \cdots Z_{m,r}$.

The program described does not compute boundary matches since we assumed the length of the sub-blocks to be at least 1000 bytes. When the length is between a hundred and a thousand bytes, the mapper μ_1 distributes overlapping sublocks and the reducer ρ_2 computes the boundary matches before completing the factorization process.

The communication cost during the computational phase of the MapReduce program above is determined by μ_2. The

dictionary D_i is sent from the node associated with the key $(0, i)$ to the node associated with the key (i, j), in parallel for $1 \leq i \leq m$ and $1 \leq j \leq r$. Each factor f in D_i can be represented as pc where p is the pointer to the longest proper prefix of f (an element of D_i) and c is the last character of f. Since the standard sizes for the dictionary and the alphabet are respectively 2^{16} and 256, three bytes can represent a dictionary element. Conservatively, at least ten nanoseconds are spent to send a byte between nodes. Therefore, the communication cost to send a dictionary is at least 30 (2^{16}) nanoseconds, which is about two milliseconds. Considering the fact that 300 kB are compressed usually in about 30 milliseconds by a Zip compressor and in about 15 milliseconds by an LZW compressor, the communication cost is acceptable. This is also true for decompression, since the decoder is symmetrical as explained in the previous subsection.

D. A Comparative Analysis

We have described four different implementations of Lempel-Ziv data compression with the MapReduce framework. One implementation uses the sliding window technique while the other three are variants of the LZW compressor. The distributed implementations have irrelevant communication cost during the computational phase and keep the same characteristics of the sequential one on a single block of the distributed data. Therefore, LZW compression is less effective but faster than sliding window compression. In order to improve the effectiveness of LZW compression, the length of a single block of the distributed data is twice the one of the sliding window implementation. This can be done since the higher speed of the LZW compressor is kept in virtue of the fact that the compression of the second half of the block is not adaptive. Therefore, the distributed system can be arbitrarily scaled up when the second half is processed and there is no relevant slow-down. The first of the three distributed implementations proposed for the LZW compressor has a preprocessing phase and a nearly-optimal approach to the compression of the second half of the block. However, we observe with the second implementation that we can relax on the quasi-optimality of the approach since a left to right greedy algorithm performs well in practice. Finally, we notice that the preprocessing phase is needed only if the size of the distributed system is beyond standard large scale and a third implementation for standard large scale systems is presented, which is almost as simple as the one for the sliding window technique.

VII. LZW Compression and Worst Case Analysis

The approaches to LZW compression described above are not robust when the data are highly disseminated [3]. However, when compressing large size files even on a large scale system the size of the blocks distributed among the nodes is larger than 600 kB. In order to increase robustness when the data are highly disseminated, the most appropriate approach is to apply a procedure where no static phase is involved. Therefore, new dictionary elements should be learned at every step while bounding the dictionary size. We show worst case analyses proving this fact, concluding that LZW-RLRU compression is the most suitable in this context since it is the most efficient one.

A. Worst Case for the Standard Distributed Implementation

In [2], the notions of bounded memory on-line decodable optimal LZW compression for the FREEZE and RESTART heuristics were introduced.

A *feasible* d-frozen LZW factorization $S = f_1 \cdots f_k$ is a feasible LZW factorization, where the number of different concatenations of a factor with the next character is $\leq d$. We define *optimal* d-frozen LZW factorization to be the feasible d-frozen LZW factorization with the snallest number of factors. Computing the optimal solution in polynomial time is quite straightforward if the degree of the polynomial time function is the dictionary size but it is obviously unpractical and a better algorithm is not known.

A *feasible* d-restarted LZW factorization $S = f_1 \cdots f_j \cdots f_i \cdots f_k$ is a feasible LZW factorization such that if j and i are consecutive indices where the restart operation happens, then the number of different concatenations of a factor with the next character is $\leq d$ between f_j and f_i. We define *optimal* d-restarted LZW factorization to be the feasible d-restarted LZW factorization with the smallest number of factors. A practical algorithm to compute the optimal solution is obviously not known as for the optimal d-frozen LZW factorization.

The compression models just introduced employ dictionaries with size bounded by the FREEZE and RESTART heuristics, respectively. The on-line greedy factorizations are obviously feasible. Moreover, feasible factorizations are the ones produced by the distributed algorithms described in the previous section. In this section, we give upper bounds to the approximation multiplicative factor. A trivial upper bound to the approximation multiplicative factor of every feasible factorization with respect to the optimal one is the maximum factor length of the optimal string factorization, that is, the height of the trie storing the dictionary. Such upper bound is $\Theta(d)$, where d is the dictionary size ($O(d)$ follows from the feasibility of the factorization and $\Omega(d)$ from the factorization of the unary string). There are strings for which the on-line greedy d-frozen LZW factorization is a $\Theta(d)$ approximation of the optimal one. Indeed, if we bound the dictionary size to $d + 2$ and consider the input binary string $(\prod_{i=0}^{d/2-1} ab^i ba^i)(\prod_{i=1}^{d} a^{d/2})$ then the on-line greedy *d-frozen* LZW factorization is a, b, ab, ba, abb, baa, $\cdots$ ab^i, ba^i, $\cdots$ $ab^{d/2-1}$, $ba^{d/2-1}$, a, a, $\cdots$, a while the optimal *d-frozen* LZW factorization is a, b, ab, b, a, abb, b, aa, $\cdots$ ab^i, b, a^i, $\cdots$ $ab^{d/2-1}$, b, $a^{d/2-1}$, $a^{d/2}$, $a^{d/2}$, $\cdots$ $a^{d/2}$. It

follows that the cost of the greedy factorization is $d + d^2/2$ while the cost of the optimal one is $5d/2 - 1$.

The feasible d-restarted LZW factorizations output by the distributed algorithms of the previous section can be as bad as the greedy solution using the frozen dictionary in the worst case. Indeed, if we apply any of such distributed algorithms to the input block of length d^2

$$b^{d^2/4-d/2}(\prod_{i=0}^{d/2-1} ab^iba^i)(\prod_{i=1}^{d} a^{d/2})$$

the dictionary is filled up by the greedy factorization process applied to the first half of the block, that is, $b^{d^2/4-d/2}(\prod_{i=0}^{d/2-1} ab^iba^i)$. Such factorization is: b, bb, $\cdots$, b^ℓ, $b^{\ell'}$, a, b, ab, ba, abb, baa, $\cdots$ ab^i, ba^i, $\cdots$ $ab^{d/2-1}$, $ba^{d/2-1}$ where $\ell' \leq \ell+1$ and the dictionary size is $d+\ell+3$. The static factorization of the second half is $a, a, \cdots a, a$ and the total cost of the factorization of the block is $\ell+1+d+d^2/2$ which is $\Theta(d^2)$. On the other hand, the cost of the optimal solution on the block is $\ell + 5d/2$, which is $\Theta(d)$. Observe that the $O(d)$ approximation multiplicative factor depends on the static phase and this happens when the dictionary learned on the first half of the block performs badly on the second half, that is in practice, when the data are highly disseminated. We will show in the next subsection that the on-line greedy d-restarted LZW factorization performs much better in the worst case, suggesting a more robust approach to distributed computing.

B. Worst Case Analysis of the Sequential Implementation

During the learning process before freezing and eventually restarting the dictionary, the on-line greedy factorization is the only feasible factorization producing factors which are all different from each other, that is, the number of factors equals the number of dictionary elements. This is the property we use to prove our result.

Theorem. The on-line greedy d-restarted LZW factorization is an $O(\sqrt{d})$ approximation of the optimal one, where d is the dictionary size.

Proof. Without loss of generality, we can assume the restart operation happens as soon as the dictionary is filled up during the greedy factorization process, since the static phase monitors the performance of the procedure. Let S be a string of length n and T be the trie storing the dictionary of factors of the optimal d-restarted LZW factorization Φ of S between two consecutive positions, where the restart operation happens. Each dictionary element (but the alphabet characters) corresponds to the concatenation of a factor f of the optimal factorization with the first character of the next factor, that we call an *occurrence* of the dictionary element (node of the trie) in Φ. We call an element of the dictionary, built by the greedy process, *internal* if its occurrence is contained in the occurrence of a node of T and denote with M_T the number of internal occurences. The number of non-internal occurences is less than the number of factors of Φ. Therefore, we can consider only the internal ones. An occurrence f' of the greedy factorization internal to a factor f of Φ is represented by a subpath of the path representing f in T. Let u be the endpoint at the lower level in T of this subpath (which, obviously, represents a prefix of f). Let $d(u)$ be the number of subpaths representing internal phrases with endpoint u and let $c(u)$ be the total sum of their lengths. All the occurences of the greedy factorization are different from each other between two consecutive positions, where the restart operation of the greedy procedure happens. Since two subpaths with the same endpoint and equal length represent the same factor, we have $c(u) \geq d(u)(d(u)+1)/2$. Therefore

$$1/2 \sum_{u \in T} d(u)(d(u)+1) \leq \sum_{u \in T} c(u) \leq 2|\Phi|H_T$$

where H_T is the height of T, $|\Phi|$ is the number of phrases of Φ and the multiplicative factor 2 is due to the fact that occurrences of dictionary elements may overlap. We denote with $|T|$ the number of nodes in T; since $M_T = \sum_{u \in T} d(u)$, we have

$$M_T^2 \leq |T| \sum_{u \in T} d(u)^2 \leq |T| \sum_{u \in T} d(u)(d(u)+1) \leq 4|T||\Phi|H_T$$

where the first inequality follows from the fact that the arithmetic mean is less than the quadratic mean. Then

$$M_T \leq \sqrt{4|T||\Phi|H_T} = |\Phi|\sqrt{\frac{4|T|H_T}{|\Phi|}} \leq 2|\Phi|\sqrt{H_T}$$

The statement of the theorem follows from the fact that the height of the trie is $\Theta(d)$ in the worst case. q. e. d.

The theorem suggests an approach restarting the dicionary as soon as it is filled up, which is more robust but in some cases (when the data are quite homegeneous) a little less effective in terms of compression effectiveness. Therefore, on a distributed system each processor stores a block of data and applies the on-line greedy LZW factorization adding a new element to the dictionary at each step. Obviously, blocks are short enough to observe the dictionary size bound d. From the the statement of the theorem in the previous section, such approach outputs an $O(\sqrt{d})$ approximation of the optimal solution since it computes the on-line greedy d-restarted factorization. If the file size is very large and the bound to the dictionary size is reached by one processor before the end of its block, a "least recently used" strategy can be applied to remove dictionary elements to preserve

robustness. The relaxed version of LZW-LRU compression using only two eqivalence classes is the one we propose as the most suitable and efficient for large size files lossless compression.

C. LZW-RLRU2 Compression: A Robust Approach

The relaxed version of the LRU heuristic using p equivalence classes is:

> RLRUp: When the dictionary is not full, label the i^{th} element added to the dictionary with the integer $\lceil i \cdot p/k \rceil$, where k is the dictionary size minus the alphabet size and $p < k$ is the number of labels. When the dictionary is full, label the $i-th$ element with p if $\lceil i \cdot p/k \rceil = \lceil (i-1)p/k \rceil$. If $\lceil i \cdot p/k \rceil > \lceil (i-1)p/k \rceil$, decrease by 1 all the labels greater or equal to 2. Then, label the $i-th$ element with p. Finally, remove one of the elements represented by a leaf with the smallest label.

In other words, RLRU works with a partition of the dictionary in p classes, sorted somehow in a fashion according to the order of insertion of the elements in the dictionary, and an arbitrary element from the oldest class with removable elements is deleted when a new element is added. Each class is implemented with a stack. Therefore, the newest element in the class of least recently used elements is removed. Observe that if RLRU worked with only one class, after the dictionary is filled up the next element added would be immediately deleted. Therefore, RLRU would work like FREEZE. But for $p = 2$, RLRU is already more sophisticated than SWAP since it removes elements in a continuous way and its compression effectiveness compares to the original LRU. Therefore, LZW-RLRU2 is the most efficient approach to compress on the Web or any other distributed system when the size of the input file is very large. In the MapReduce framework, a program sequence $\mu_0\mu_1\rho_1\mu_2\rho_2$ implements it as the one for the LZSS compressor explained in Section VI. A sequence of the same length works symmetrically for decompression.

VIII. Conclusion

We showed how to implement Lempel-Ziv data compression in the MapReduce framework for Web computing. An alternative to standard versions of the Lempel-Ziv method is proposed as the most efficient one for large size files compression. The robustness of the approach is evidentiated by a theoretical worst case analysis of the standard techniques. Moreover, scalability is preserved since no interprocessor communication is required. It follows that a linear speed-up is guaranteed during the computational phase. With arbitrary size files, scaling up the system is necessary to preserve the efficiency of LZW compression but with very low communication cost if the data are not highly disseminated. The MapReduce framework allows in theory a higher degree of communication than the one employed in the procedures presented in this paper. In [14], it has been shown how the PRAM model of computation can be simulated in MapReduce under specific constraints with the theoretical framework. These constraints are satisfied by several PRAM Lempel-Ziv compression and decompression algorithms designed in the past [8], which are suitable for arbitrary size highly disseminated files. As future work, it is worth investigating experimentally if any of these PRAM algorithms (which are completely different from the ones presented in this paper) can be realized with MapReduce in practice on specific files.

References

[1] S. De Agostino, "Compressing Large Size Files on the Web in MapReduce," Proceedings International Conference on Internet and Web Applications and Services (ICIW), 2013, pp. 135-140.

[2] S. De Agostino, "LZW Data Compression on Large Scale and Extreme Distributed Systems," Proceedings Prague Stringology Conference, 2012, pp. 18-27.

[3] S. De Agostino, "Bounded Memory LZW Compression and Distributed Computing: A Worst Case Analysis," Proceedings Festschrift for Borivoj Melichar, 2012, pp. 1-9.

[4] A. Lempel and J. Ziv, "On the Complexity of Finite Sequences," IEEE Transactions on Information Theory, vol. 22, 1976, pp. 75-81.

[5] A. Lempel and J. Ziv, "A Universal Algorithm for Sequential Data Compression," IEEE Transactions on Information Theory, vol. 23, 1977, pp. 337-343.

[6] J. Ziv and A. Lempel, "Compression of Individual Sequences via Variable-Rate Coding," IEEE Transactions on Information Theory, vol. 24, 1978, pp. 530-536.

[7] M. Crochemore and W. Rytter, "Efficient Parallel Algorithms to Test Square-freeness and Factorize Strings," Information Processing Letters, vol. 38, 1991, pp. 57-60.

[8] S. De Agostino, "Parallelism and Dictionary-Based Data Compression," Information Sciences, vol. 135, 2001, pp. 43-56.

[9] L. Cinque, S. De Agostino, and L. Lombardi, "Scalability and Communication in Parallel Low-Complexity Lossless Compression," Mathematics in Computer Science, vol. 3, 2010, pp. 391-406.

[10] S. De Agostino, "Lempel-Ziv Data Compression on Parallel and Distributed Systems," Algorithms, vol. 4, 2011, pp. 183-199.

[11] S. De Agostino, "P-complete Problems in Data Compression," Theoretical Computer Science, vol. 127, 1994, pp. 181-186.

[12] S. De Agostino and R. Silvestri, "Bounded Size Dictionary Compression: SC^k-Completeness and NC Algorithms," Information and Computation, vol. 180, 2003, pp. 101-112.

[13] S. De Agostino, "Bounded Size Dictionary Compression: Relaxing the LRU Deletion Heuristic," International Journal of Foundations of Computer Science, vol. 17, 2006, pp. 1273-1280.

[14] H. J. Karloff, S. Suri, and S. Vassilvitskii, "A Model of Computation for MapReduce," Proc. SIAM-ACM Symposium on Discrete Algorithms (SODA 10), SIAM Press, 2010, pp. 938-948.

[15] S. De Agostino, "LZW versus Sliding Window Compression on a Distributed System: Robustness and Communication," Proc. INFOCOMP, IARIA, 2011, pp. 125-130.

[16] S. De Agostino, "Low-Complexity Lossless Compression on High Speed Networks," Proc. ICSNC, IARIA, 2012, pp. 130-135.

[17] J. A. Storer and T. G. Szimansky, "Data Compression via Textual Substitution," Journal of ACM, vol. 24, 1982, pp. 928-951.

[18] M. Rodeh, V. R. Pratt, and S. Even, "Linear Algorithms for Compression via String Matching," Journal of ACM, vol. 28, 1980, pp.16-24.

[19] E. M. Mc Creight, *A Space-Economical Suffix Tree Construction Algorithm,* Journal of ACM, vol. 23, 1976, pp. 262-272.

[20] T. A. Welch, "A Technique for High-Performance Data Compression," IEEE Computer, vol. 17, 1984, pp. 8-19.

[21] S. De Agostino and J. A. Storer, "On-Line versus Off-line Computation for Dynamic Text Compression," Information Processing Letters, vol. 59, 1996, pp. 169-174.

[22] S. De Agostino and R. Silvestri, "A Worst Case Analisys of the LZ2 Compression Algorithm," Information and Computation, vol. 139, 1997, pp. 258-268.

[23] J. A. Storer, Data Compression: Methods and Theory, Computer Science Press, 1988.

[24] E. R. Fiala and D. H. Green, "Data Compression with Finite Windows," Communications of ACM, vol. 32, 1988, pp. 490-505.

[25] J. R. Waterworth, "Data Compression System," US Patent 4 701 745, 1987.

[26] R. P. Brent, "A Linear Algorithm for Data Compression," Australian Computer Journal, vol. 19, 1987, pp. 64-68.

[27] D. A. Whiting, G. A. George, and G. E. Ivey, "Data Compression Apparatus and Method," US Patent 5016009, 1991.

[28] J. Gailly and M. Adler, http://www.gzip.org, 1991.

[29] A. Hartman and M. Rodeh, "Optimal Parsing of Strings," In: Apostolico, A., Galil, Z. (eds.) Combinatorial Algorithms on Words, Springer, 1985, pp. 155-167.

[30] M. Crochemore and W. Rytter, Jewels of Stringology, World Scientific, 2003.

[31] S. De Agostino, "Almost Work-Optimal PRAM EREW Decoders of LZ-Compressed Text," Parallel Processing Letters, vol. 14, 2004, pp. 351-359.

[32] D. Belinskaya, S. De Agostino, and J. A. Storer, "Near Optimal Compression with respect to a Static Dictionary on a Practical Massively Parallel Architecture," Proceedings IEEE Data Compression Conference, 1995, pp. 172-181.

Correlation and Consolidation of Distributed Logging Data in Enterprise Clouds

Sven Reissmann, Dustin Frisch, Christian Pape, and Sebastian Rieger
University of Applied Sciences Fulda
Department of Applied Computer Science
Fulda, Germany
{sven.reissmann, dustin.frisch, christian.pape, sebastian.rieger}@cs.hs-fulda.de

***Abstract*—Due to the availability of virtualization technologies and related cloud infrastructures, the amount and also the complexity of logging data of systems and services grow steadily. Automated correlation and aggregation techniques are required to support a contemporary processing and interpretation of relevant logging data. In the past, this was achieved using highly centralized logging systems. Based on this fact, the paper introduces a prototype for an automated semantical correlation, aggregation and condensation of logging information. The prototype relies on a NoSQL storage back-end that is used to persist consolidated messages of distributed logging sources in a highly performant manner. This step of consolidation includes strategies for minimizing long-term storage, and by using correlation techniques also offers possibilities to detect anomalies in the stream of processed messages. In this context, we will present the special requirements of handling scalable logging systems in highly dynamic infrastructures like enterprise cloud environments, which provide dynamic systems, services and applications.**

***Keywords*—Syslog Correlation; Log Analysis; Anomaly Detection; Monitoring; Enterprise Cloud.**

I. Introduction

The vast rise of virtualization technologies and the related wide availability of virtual machines (VM) has increased the amount of logging data over the past years [1]. In addition to virtual machines themselves, cloud infrastructures, in which they are deployed, also deliver new services and applications in a fast and highly dynamic manner, producing logging data that is needed to monitor their state and service quality. This leads to a growth of logging sources and the demand for logging systems to dynamically handle new sources and collect the corresponding data. Each new source provides detailed logging information and increases the overall amount of logging data. Typically, logging data will be compressed and also anonymized at short intervals if personally identifiable information is included. Also, outdated log entries can be removed, but the number of logging sources (e.g., the number of virtual machines) themselves often cannot be reduced. For instance, in a virtualized cloud infrastructure where servers, storage and also the network are virtualized, each system, service and application should at least provide a minimal set of logging data to allow an effective analysis of its status and relevant events during service operation.

To support this analysis and evaluation across logging information originating from a large number of different distributed source systems, correlation techniques offer a way to group similar systems and applications. Furthermore, correlation can be used for the aggregation of logging data, hence providing a condensation based on its relevance. In [1], we introduced a solution to persist logging data that originated from syslog sources into a NoSQL-based (Not only SQL) database by enhancing existing solutions. For correlation and consolidation purposes, this data was also enriched with meta information before providing the data for distributed analysis and evaluation. This article elaborates on the implementation and concepts outlined in [1] and presents extensions to use different, e.g., structured logging sources and increase the efficiency of our correlation engine. Furthermore, an evaluation test-bed to enhance the scalability of our implementation in OpenStack-based enterprise cloud environments is given. Additionally, we present possibilities to use information from external network and system management solutions to enrich the events being correlated.

The article is laid out as follows. In the next section, the state-of-the-art of distributed logging in cloud environments is described. Section III gives examples of related work and research projects that also focus on improving the management and analysis of logging data in distributed cloud environments. Requirements for the correlation and consolidation of logging data in enterprise clouds are defined in Section IV. In Section V, the implementation of a prototype for log correlation and consolidation in cloud environments is presented. It provides aggregation and condensation of logging data in cloud environments by correlating individual events from distributed sources. At the end of the section, an example of the usage of our prototype for the log analysis in cloud environments is given. Section VI describes the deployment of our prototype in an OpenStack test-bed. The performance of our prototypical implementation is evaluated in Section VII using multiple test cases. In the last section of this article, a conclusion is drawn and aspects for future research are outlined.

II. State-of-the-Art

The following sections give an overview on the evolution of logging methods in distributed environments using aggregation and consolidation techniques for standard logging mechanisms like syslog. Also, the advantages of evolving NoSQL databases in this area are outlined.

A. Distributed Logging in Cloud Environments

Current cloud service providers offer a variety of monitoring mechanisms. For example, Amazon Web Services (AWS)

and RackSpace both provide monitoring and alarms for their virtual machines. In the basic version, these services monitor several performance metrics, like central processing unit (CPU), input/output (I/O), and network utilization. Advanced versions (e.g., Amazon CloudWatch [2]) allow the customers to check the current status of services running in the virtual machines and to define custom metrics and alarms that can be monitored using individual application programming interfaces (API) of the cloud service provider. While these APIs could be used to send particular events and alarms, there is no specific service to handle the aggregation, correlation and management of logging data generated and provided by the operating systems and services running in the virtual machines. Furthermore, the individual APIs currently vary from provider to provider. Hence, it is not possible to use a unified monitoring implementation across different cloud service providers. This also hinders the establishment of enterprise clouds that should allow the integration of private or hybrid cloud services operated by public cloud service customers, as these solutions again use individual monitoring techniques. An appropriate standard to address the issue of cloud service provider-independent logging, is currently in the works at the Internet Engineering Task Force (IETF) [3].

Until such open standards are available, distributed logging in cloud environments could be carried out by developing specific logging mechanisms for the infrastructures, platforms or applications (IaaS, PaaS, SaaS) used in the cloud. The drawback of this approach would be the required software development and maintenance effort. Moreover, the individual APIs developed by the customers are likely to need a migration to upcoming cloud logging standards in the near future. The more appropriate approach, therefore, could be to extend existing and established logging services to support distributed cloud scenarios. The de-facto standard logging service offered in every predefined Linux-based virtual machine image by existing cloud providers is syslog, which is described in the next section. In fact, syslog is also the basis for the upcoming Internet-Draft [3] focusing on cloud-based logging services. Logging data can be stored and structured in NoSQL-based databases as described in Section II-C.

B. Log Aggregation and Consolidation with syslog

Syslog defines a distributed logging solution for generating, processing and persisting host- and network-related events. As a key feature, it allows one to separate the software that generates events from the system that is responsible for processing, storing and analyzing those events. Since its introduction, the syslog protocol has evolved into the de-facto standard for processing and forwarding logging events on UNIX-based systems and network devices (i.e., routers, firewalls). However, there has not been any standardization of the protocol characteristics for quite a long time, which has led to incompatibilities across vendor-specific implementations.

The state of the protocol (*Berkeley Software Distribution (BSD) Syslog Protocol*), including the most commonly used message structure and data types, has been documented by the IETF in RFC 3164 [4]. Each syslog packet starts with a so-called PRI (priority) part representing the severity of the message as well as the facility that generated the message. Severity and facility are together numerically coded using decimal values; the priority value, which is placed between angle brackets at the very beginning of the message, is calculated by first multiplying the facility value by 8 and then adding the severity value. RFC 3164 specifies eight possible values for the severity of a message, as well as 24 facility values, which are assigned to some of the operating systems' daemons (i.e., the mail subsystem). The second part of a syslog packet, following immediately after the trailing bracket of the PRI part, is called HEADER and includes a TIMESTAMP and HOSTNAME field, each followed by one single space character. TIMESTAMP represents the local time when a message was generated using the format "Mmm dd hh:mm:ss", while HOSTNAME may only contain the hostname [5], IPv4 address [6], or IPv6 address [7] of the producer of the message. Finally, the MSG part of a syslog packet consists of two fields, known as TAG and CONTENT, where CONTENT contains the actual message, while TAG is the name of the program that generated it. The TAG value can be distinguished from the message or other optional information by a colon or left square bracket depending on the presence of an optional (but commonly used) combination of the program name with its process ID. Listing 1 shows an example of an RFC 3164 compliant syslog message.

```
<34>Oct 11 22:14:15 mymachine su: 'su root'
    failed for lonvick on /dev/pts/8
```

Listing 1. Example of an RFC 3164 compliant syslog message.

While syslog messages are typically stored in files on the generating host's local filesystem, we outlined above that the impact of virtualization technologies and the corresponding growth of logging sources indicate that a centralized collection and analysis of syslog events is of essential importance. Otherwise, an overall rating of nearly identical messages originating from different sources (i.e., from a cloud service that spreads over multiple hosts) would be a difficult task. As a matter of fact, centralized logging infrastructures and the utilization of relays to cascade syslog servers in large environments were considerations in the early development of syslog. Historically, the BSD Syslog Protocol [4] uses the User Datagram Protocol (UDP) to transport messages over the network, which may lead to the imperceptible loss of important events. To address this issue, the use of a reliable delivery mechanism for syslog has been proposed shortly after the documentation of the protocol characteristics [8]. Additionally, security features like Transport Layer Security (TLS) and cryptographic signatures have been proposed to assure the integrity and authenticity of the data during transport of the messages from the sending hosts [9][10][11]. Figure 1 shows an example of a centralized logging environment, where a number of physical or virtual servers send their messages to a central syslog server, which then stores these messages appropriately.

A major drawback of the previously described syslog protocol regarding correlation and analysis capabilities is the unstructured nature of the messages' format. In 2009, a standard for a syslog protocol was proposed in RFC 5424 [12], which obsoletes the previously described but still widely used BSD Syslog Protocol. The new protocol specifies the PRI part of a syslog packet in the same way as its predecessor, but includes it into the HEADER part of the packet together

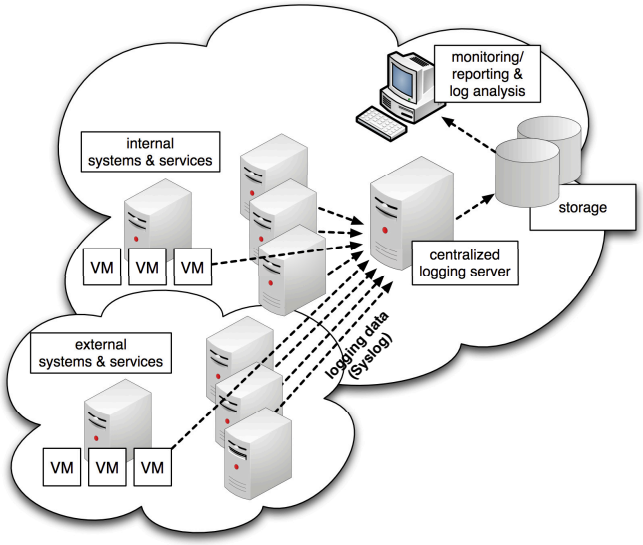


Fig. 1. Centralized logging of distributed systems and services.

with additional information such as the name and process ID of the generating application and a message ID. Another feature of the new HEADER format is the use of a formalized TIMESTAMP value as standardized in RFC 3339 [13]. Most important, however, is the possibility to add structured data into a syslog packet, allowing to express information in a well defined and easily machine-parsable format. An example of a syslog message containing both structured data and a regular free text message is expressed in Listing 2.

```
<165>1 2003-10-11T22:14:15.003Z
       mymachine.example.com
    evntslog - ID47 [exampleSDID@32473 iut="3"
    eventSource="Application" eventID="1011"]
    BOMAn application event log entry...
```

Listing 2. Example of an RFC 5424 compliant syslog message.

Among the wide range of available syslog servers capable of processing the previously delineated protocols, syslog-ng [14] and rsyslog [15] are some of the most widely used solutions on UNIX-based operating systems. Both implement RFC 3164 as well as RFC 5424 message transport, various security features such as cryptographic signatures, message encryption, and the ability to convert messages from one format to another. The rsyslog server, which provides an open-source implementation of the syslog protocol, was selected for our research as it provides interesting features like a large number of input and output modules to support a wide range of logging sources and novel storage back-ends like MongoDB, Hadoop Distributed File System (HDFS), and Elasticsearch. Furthermore, message normalization and modification techniques make it possible to parse incoming messages and add structured information to them, which enables post-processing applications to apply correlation techniques and filtering rules. In our test-bed, which we describe in detail in later sections of this article, we make use of rsyslog's input filters, message normalization provided by liblognorm, and message forwarding capabilities using various output modules.

C. Log Management utilizing NoSQL Databases

Regarding the persistence of logging data, important aspects are performance and scalability of the storage system, especially as in many cases there is a tremendous amount of data to be stored. Further, the flexibility to add new logging sources that may introduce new data structures (i.e., when providing individual structured data as described in the previous section) is a crucial requirement. While various storage back-ends are available for centralized logging environments, we would argue that only a certain type of these systems qualifies by offering reasonable write performance and, even more important, allowing complex analysis on the stored data. That class of storage systems is called NoSQL, which refers to a type of data storage that became an interesting alternative to Structured Query Language (SQL) databases over the past couple of years, especially for storing huge amounts of information such as logging data.

Relational databases demand the structure of the data to be specified at the time the database is created. This means that creating a relational database for data that does not easily map into a fixed table-layout (e.g., different log formats from distributed sources) is not a simple task. NoSQL datastores, in contrast, provide little or no pre-defined schema, allowing the structure of the data to be modified or extended at any time. This property qualifies NoSQL for persisting structured syslog information, where on the one hand, the data is well formatted, but at the same time, the structured key-value pairs of the individual syslog sources may vary depending on the type of service or system that generated the message.

While the name NoSQL makes it appear that the lack of SQL is the most important difference, there are a number of other characteristics that distinguish this type of storage system from the well-known SQL database management systems (DBMS). In almost all cases, a simple query interface is used for storing, retrieving or modifying data, rather than an SQL processor. While the query language of an SQL DBMS itself does not necessarily have negative effects on performance, it can be quite difficult to write efficient queries when trying to do complex operations on the data, e.g., joining a large number of tables. Another important difference of NoSQL compared to SQL DBMS is the scalability over hundreds of hosts, which is achieved in exchange for giving up 100% ACID (Atomicity, Consistency, Isolation, Durability) semantics. Instead, NoSQL guarantees consistency only within certain boundaries or within a specific record. At the same time, scalability leads to high availability by doing transparent failover and recovery using mirror copies. Of course, not all copies are guaranteed to be up to date, because of the previously mentioned lack of the ACID compliance.

Although there are different implementations of NoSQL datastores that fit completely different needs, all are designed to fit new requirements, like storing unstructured data or performing full-text search queries. In [16], Cattell presents an overview of the available types of NoSQL technologies and names some of the actual implementations of the various technologies. Looking at the distinct approaches of NoSQL, three main types can be identified. Key-value stores allow one to store unstructured data in a single distributed key-value index for all the data. The data is typically stored as binary large object (BLOB) without being interpreted and

can be accessed by a client interface, which provides basic operations like insert, delete and index lookups. Key-value stores achieve high performance when querying for keys, but are not very well suited to perform searches on the stored objects themselves. Going one step further, extensible record stores, also referred to as column-oriented datastores, allow the storage of closely related data in an extendable row-and-column layout. Scalability is achieved by splitting both rows, through shards on the primary key, and columns, by using pre-defined column groups. Besides using a row and column model, extensible record stores are not bound to the restrictions of the highly structured table layout SQL uses, as new attributes can easily be added at any time. Still, they allow one to store data in a more detailed structure compared to the previously mentioned key-value stores. A third type of NoSQL is referred to as document-based datastores or document stores. Here, document is not to be confused with an actual document in the traditional sense. Instead, these types of databases are able to store collections of objects, each of which may have a completely independent number of attributes of various types. The structure of new documents can be extended at any time, meaning that documents may consist of any number of key-value pairs of any length. Like the previous, document stores can partition the data over multiple machines for scalability and replication of the data. Document stores provide a high degree of flexibility and interoperability, but like other NoSQL systems, they do not provide ACID transactional conformance.

Looking at the requirements for storing logging data and being able to do complex analysis of the data later on, not all of the previously named NoSQL technologies are suitable. Key-value stores as well as column-oriented databases allow highly efficient queries for the data using their keys, while not being applicable for doing full-text search queries and correlation on the stored data. Document stores, in contrast, allow highly efficient search queries on the stored data objects as well as the documents' keys. Further, the flexibility to add any newly formatted document at any time is an important feature when storing individually structured syslog messages. For the implementation of our prototype, we use Elasticsearch [17], a document store that offers a high-performance, full-featured text search engine based on Apache Lucene [18].

III. Related Work

The challenge of persisting and evaluating decentralized logging data has been in the focus of many research publications. For instance, the evaluation of decentralized logging information in IaaS, PaaS and SaaS cloud environments was described in [19] and [20]. Also, an Internet-Draft is in development [3] covering processing of syslog messages from distributed cloud applications. Besides the requirements by these new highly distributed applications, there is also a challenge for analysis and structuring of logging information. Existing solutions for automated log analyzers comply with only some of these requirements [21]. Therefore, Jayathilake [21] recommends structuring logging data and extracting the contained information. In this context, NoSQL databases are best suited for handling these variable fields. These databases provide an adaptive approach of persisting data and allow the use of different table schemata or, e.g., a document-based approach to storing key-value pairs. As outlined in [22] and [23], the evaluation and rating itself can be automated by event correlation and event detection techniques. Both publications also describe the usage of the correlation solution Drools, which we use for our research. Correlation techniques help to reduce and consolidate the logging data so that only a condensed representation including relevant information, required for analysis and evaluation, will be persisted. As described in [23], a reduction of syslog data by up to 99% is possible. A solution based on the NoSQL database MongoDB using MapReduce to correlate and aggregate logging data in distributed cloud analysis farms is described in [24]. This solution, however, lacks event correlation and detection techniques.

IV. Requirements for Correlation of Logging Data in Enterprise Clouds

In the introduction of this article, we described that centralized logging environments tend to produce a tremendous number of logging events at the central logging server. To manage the storage of all these events and provide a way to perform a fast analysis on the stored data, the use of the previously mentioned NoSQL datastores seems obvious. However, looking at the amount of data that has to be manually analyzed and evaluated, the question arises whether it is possible to automate the process of evaluating the relevance of certain syslog events or even reduce the amount of data that will be stored. The latter is only reasonable if it can be guaranteed that no valuable information will be lost after the reduction of messages. In the next sections, we are going to describe in detail our approach for automatic evaluation and reduction of syslog events. Also, a method for identifying interesting types of events for which a correlation or consolidation is possible is proposed at the end of this section.

A. Correlating Distributed Logs in Enterprise Clouds

The core objective of our previous work was the processing of data provided via syslog and the identification of important events in the network or on individual hosts. For instance, the sequence of messages of an ongoing Secure Shell (SSH) brute-force attack illustrates the demand for an automated rating of messages. During a brute-force attack, the SSH daemon generates a log entry for each invalid login attempt. These messages are delivered to a centralized syslog server, indicating individual failed login attempts. However, the relatively small number of such specific events might become lost in the large total number of syslog messages received from all distributed sources. At the same time, a single message indicating that an SSH login attempt was successful will definitely be hard to identify among the huge number of syslog messages denoting failed login attempts while the brute-force attack is running. Figure 2 shows a visualization of authentication messages collected by a central syslog server used to monitor servers, network devices and storage systems for students' web servers at the University of Applied Sciences Fulda.

The resulting graph gives an overview of the general behavior and basic noise of these logging messages. The Holt-Winters [25] algorithm was used to compute and adapt a confidence band over time, which represents the normal behavior of the time-series data. If specific values are violating this confidence band for a number of periods in a given time window, these values are marked as failures or aberrant

Fig. 2. Holt-Winters based vertical bands representing high occurrences of SSH login attempts resulting in ”Failed password” messages.

behavior. This is also visualized in the graphical output by yellow vertical areas (vertical bands in the graph). In this example, the peaks were results of distributed brute-force attacks with common user account names, such as root, admin or test.

An IT operator analyzing the logging data, however, is not interested in displaying each individual login attempt, but rather wants to know whether the brute-force attack led to a successful login into one of his systems. To answer this kind of question, the syslog messages must be filtered for data of corresponding SSH daemons and searched for failed login attempts that are followed by a successful login. Thus, a system registering a large number of failed login attempts, and finally a successful login, might experience a successful brute-force attack, while the possibility of an attack rises along with the number of failed login attempts. The time-consuming and costly search for attack patterns like this can be simplified by an automated rating of syslog messages. To identify individual messages describing similar events from different operating systems and platforms, it is required to normalize syslog data before correlating and persisting them. For the lookup of SSH login attempts, it is sufficient to examine single individual syslog messages. In order to identify a completed attack, a sequence of these matching messages must be investigated. If the conditions for a successful attack are met, it is possible to generate a new prioritized syslog message to support the immediate detection of these security threats in the network. By using the flexibility of structured data in syslog messages we described earlier, it is possible to add certain tags to the parsed or newly generated messages. These tags may be used later on, in order to support analysis by allowing the application of filters, e.g., to separate newly injected messages from the normal syslog messages after they are persisted all together into the Elasticsearch cluster.

B. Consolidating Logging Data from Distributed Services

A second goal of our work was to reduce the number of messages that actually get persisted into the Elasticsearch cluster. This may seem subordinate against the backdrop of increasing computing performance and concepts like BigData. Reducing the actual amount of data that gets persisted, however, still results in faster and easier analysis, even when using the novel NoSQL techniques. In practice, we actually observe the reduction of stored messages in long-term storage as a requirement for many companies. Such reduction techniques basically delete messages of a certain age or do not persist messages below a certain severity. However, this results in a loss of valuable contextual information, which is why we would argue that such simple consolidation mechanisms are not practicable for logging data. Our novel approach first provides a grouping mechanism, where identical or recurring events are summarized. Based on those groups we are able to generate new syslog messages containing a dense representation of all the valuable information of the initial messages, thus allowing us to actually drop those messages without losing any contextual information. Using our solution, it is possible to reduce the number of messages that need to be stored at the central database server, and therefore we are able to improve the performance of this system without losing information. Furthermore, it is possible to manipulate the severity of the newly generated messages, in order to increase their value for later analysis.

An example application of such modification of syslog messages might be the detection of the previously mentioned brute-force attack, which results in a flood of low-priority security event messages. By generating a single message of high priority — telling an administrator what the actual attack looked like, judging from the number of login attempts, the duration of the attack and the actual result — we produce information that supports estimating the situation and the next steps to be taken. In addition, regardless of waiving all the failed login attempts at the central database server, further investigation is still possible by performing an exact analysis of the attack using the logfiles stored at the individual server that was under attack. A second example of consolidating messages would be the correlation of application access logs. For instance, in a cloud environment new machines will be spawned on demand, so several systems provide a single service in a cooperative way. An example could be a number of dynamically started Hypertext Transfer Protocol (HTTP) servers receiving requests via a load balancer. The requests on the individual servers will be logged to the centralized syslog server, but these individual events must be aggregated, e.g., to support the decision process of starting or stopping virtual machines running the HTTP servers. The access log messages can be correlated to an access count per timeslot and it is also possible to count active HTTP servers by differentiating distinct logging sources. As already illustrated in the previous example, it is again not necessary to persist the original access messages. The correlated logging information is useful to evaluate the load on all servers and can also be used to determine whether running machines have to be stopped or new ones need to be started.

Taking the logging information into account, a confident decision can be made that goes beyond the possibilities of network-based load balancing and failover techniques. A more generic approach would be to use correlation engines like Drools to count messages matching a set of rules for specific timeslots and to generate diagrams for these kinds of messages. This approach allows one to compare different timeslots and to answer questions like “Were the same number of cron jobs executed on Monday and Tuesday?”. Also, a visual representation of these results, e.g., as presented in [26], could be possible with the benefit of easily identifying anomalies at first sight.

C. Identifying logging events of interest

When trying to automatically process syslog messages with the objective of evaluating their importance by using correlation techniques, one has to start by identifying sequences of messages that point to certain events. We did a manual analysis of such events by using large amounts of syslog information provided by the General Students' Committee of the University of Applied Sciences Fulda and a number of virtual servers we previously set up to collect and simulate specific attacks. To analyze the dataset, we implemented a small Java-based application that counts similar messages in five-minute intervals based on regular expressions. The results of these calculations are stored in a round-robin archive using RRDtool [27], which is able to generate a graphical representation of the data. This simplified approach to represent logging data in a diagram allows us to use algorithms applicable for time series data, e.g., forecasting or anomaly-detection algorithms. The RRDtool utility itself implements an exponential smoothing algorithm for forecasting and anomaly detection also known as Holt-Winters Aberrant Behavior Detection [28]. We used this algorithm to identify aberrant behavior in the syslog data of specific applications we monitored. An example of this method based on counting warning messages from the postfix/smtpd process on mail servers is shown in Figure 3. Investigating the logging data to find the cause of the peaks indicated in the graph led back not only to temporary name resolution problems, but also to authentication issues on this host.

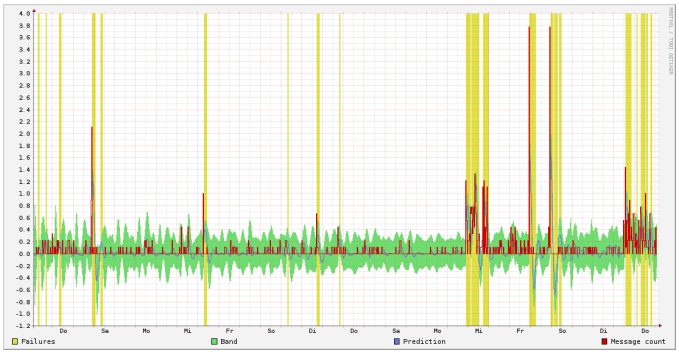


Fig. 3. Holt-Winters based vertical bands representing high occurrences of postfix/smtpd warning messages.

On one hand, the visualization of logging data, as shown in Figure 3, can be used to support network operators trying to find and investigate problems in the monitored network. On the other hand, it is a convenient approach to identify interesting logging data in order to construct suitable rules for aggregation, correlation, and of course anomaly detection. It is also imaginable that this use of time-series algorithms provides a way to realize an automated generation of rulesets for detecting aberrant behavior in logging data. One prerequisite, however, is the feasibility of such algorithms to work on highly-structured data, which allows them to unambiguously identify certain events, as a similar method shows [29].

V. Implementation of Log Correlation and Consolidation in Cloud Environments

To facilitate the analysis of security event messages in distributed cloud environments and to reduce the amount of logging data that needs to be permanently stored, we presented a log correlation and consolidation prototype in [1]. In this section we will give an overview of the functional principle of the prototype and an example application of our solution by showing the correlation of logging data being generated during an SSH brute-force attack as described in Section IV-A.

A. Mode of Operation

As we already mentioned, our prototypical implementation uses rsyslog [15] as the central syslog server, which receives and normalizes syslog messages originating from distributed sources. Since the majority of the compute cloud providers offer syslog-based logging in their VMs, our entire approach using rsyslog can be used to correlate the logging data across multiple and heterogeneous cloud environments (e.g., cloud federations or hybrid clouds). The Complex Event Processing (CEP) Engine Drools Fusion [30] is used as a basis for a correlation and consolidation prototype, which was implemented in Java. Finally, we use the Elasticsearch [17] document store for permanent persistence of all the received and correlated syslog messages.

As pictured in Figure 4, syslog messages are sent from the distributed clients — either virtual machines or physical hosts — to the rsyslog server using a TLS connection over TCP. While this will slightly increase the overhead for processing and transmitting the messages, it guarantees reliable delivery as well as authenticity and privacy of the received messages.

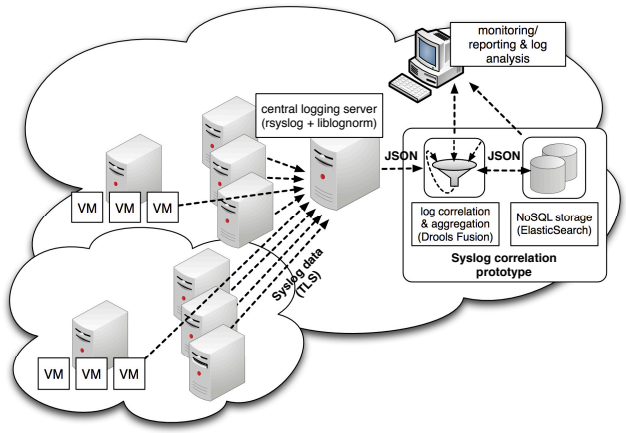


Fig. 4. Correlation and aggregation of logging data at a central syslog server.

Normalization of the received messages is performed at the central syslog server using liblognorm [31], a high-performance log normalization library for rsyslog. The use of this library enables us to parse incoming messages from different sources by specifying normalization rules in a syntax similar to regular expressions before relaying the messages to the designated output module. From the parsed messages, we then extract and process valuable information and add it back to the original syslog message using an RFC 5424 [12] compliant structured log format. It is obvious that this normalization step is needed to assure that messages originating from different clients, which may use a slightly different message format for the same type of event, can be handled in a unified way. More important, the identification of diverse

messages having the same meaning in terms of correlation can be identified and denoted by adding unique tags to the structured information of the original messages, which can easily be interpreted by our correlation prototype.

Listing 3 shows an example of rules that can be used with liblognorm to normalize syslog messages indicating successful or failed SSH password login attempts. During normalization, the values for username, source IP address and TCP port number as well as the SSH protocol version used in the authentication attempt are identified and provided as structured data in the processed syslog messages. Most important, a list of tags (labels) can be added to the message, indicating the type of event that was denoted by the parsed message. In our example, the tags SSHSUCCESS or SSHFAILURE are added to messages depending on the type of event, whereas these tags may be assigned to multiple various messages pointing to the same security event. We illustrate this by the two latter rules, which both get assigned the tag SSHFAILURE. For performance reasons, the complete rule base gets merged into an optimized parser tree by liblognorm, allowing effective analysis of the received messages.

```
rule=SSHSUCCESS: Accepted password for %user:
    word% from %ip:ipv4% port %port:number% %
    protocol:word%

rule=SSHFAILURE: Failed password for %user:
    word% from %ip:ipv4% port %port:number% %
    protocol:word%

rule=SSHFAILURE: Failed password for invalid
    user %user:word% from %ip:ipv4% port %port:
    number% %protocol:word%
```

Listing 3. Example normalization rules for matching SSH password authentication attempts.

The normalization step still has a heavy impact on performance due to the string matching operations that must be performed on all incoming syslog messages. For this reason, we make use of the ruleset feature in rsyslog, which allows us to have multiple input queues for message submission. In each queue we apply only a subset of specific normalization rules based on the type of the incoming messages. On the one hand, this minimizes the number of rules that need to be matched against the incoming messages; on the other hand, it allows us to distribute our whole prototype over multiple cloud instances as described in more detail in Section VI. The decision to use rulesets would also allow us to forward messages directly to the storage back-end in case they are not affected by correlation. However, an evaluation study we present in Section VII shows that in case of an Elasticsearch back-end this has a negative impact on performance.

After normalizing, the logging information is serialized using JavaScript Object Notation (JSON) and forwarded to an appropriate rsyslog output module, which connects to our correlation prototype. Listing 4 depicts an example of a structured syslog message, providing the most useful information from the originating syslog message through a data substructure and the type of event through its list of tags. The prototype embodies a correlation engine, which analyzes the messages and also instantly transfers them into an Elasticsearch cluster for permanent persistence. Therefore, our prototype utilizes the Elasticsearch Java API to become part of the cluster as a transparent node not storing data itself, but forwarding it to an appropriate data nodes.

```
{
  "data": {
        "protocol": "ssh2",
        "port": "54548",
        "ip": "10.0.23.4",
        "user": "root"
  },
  "time": "2014-01-29T16:06:00.000",
  "host": "test.example.com",
  "facility": "auth",
  "severity": "info",
  "program": "sshd",
  "message": "Failed password for root from
      10.0.23.4 port 54548 ssh2",
  "tags": [ "SSHFAILURE" ]
}
```

Listing 4. Structured syslog message of a failed SSH password authentication attempt.

Our implementation of the correlation prototype is based on the Complex Event Processing (CEP) Engine Drools Fusion [30], which supports temporal reasoning on a stream of data, allowing us to extend events with a property containing the time of occurrence provided by the syslog message's timestamp value. This enables us to define rules that may consider a number of similar messages in a specific time interval when evaluating the importance of incidents at the monitored systems. The rules can easily be extended to provide arbitrarily complex correlation and consolidation techniques. A more detailed example of possible rules used in the correlation engine is shown in Section V-B.

Drools automatically keeps track of all events that any of the rules may apply to, using an in-memory cache. Messages that do not match any of the rules, in contrast, will be removed from the in-memory cache. However, messages matching at least one rule are correlated, and the result is stored with a flag representing the successful correlation and a reference to original messages (that have been correlated) in the Elasticsearch cluster. By periodically searching for successful correlation flags in the Elasticsearch cluster and pruning the original messages they refer to, we can achieve a consolidation.

B. Example Application

As an example application of our prototype we are showing the correlation of logging data being generated during an SSH brute-force attack as described in Section IV-A. The aim is to generate a new syslog message of high priority in the case of a successful SSH login, which follows immediately after a certain number of failed logins, hence pointing to an SSH brute-force attack, which possibly succeeded. The detection of this scenario should be carried out completely autonomously by our prototype. In order to detect such a scenario, first we need to match the corresponding syslog messages of failed and successful SSH logins. These messages need to be isolated and filtered from the stream of logging data originating from the syslog server to trigger certain operations on them. Since we

already normalized incoming syslog messages at the central syslog server using liblognorm, we can easily recognize all interesting events by their list of tags, which we previously added to the structured data values of the messages. An example of an unsuccessful SSH authentication attempt message including its additional tag was already shown in Listing 4.

To accomplish the detection of a successful SSH brute-force attack we utilize the temporal reasoning features of Drools Fusion [30], which allow us to construct rules that describe particular events by a sequence of specific syslog messages within a certain time. First, we need to detect an ongoing SSH brute-force attack, before we can then search for subsequent successful logins. Therefore, we specify a rule entitled ”SSH brute-force attempt” as shown in Listing 5, which will match syslog messages containing the value SSHFAILURE in the list of their tags. Furthermore, by evaluating the values of the producing host and the username used to authenticate, we build up multiple in-memory queues of various authentication attempts. If we can match a series of ten failed messages within a one minute time window concerning a specific user account at one specific system, we assume a certain chance of an ongoing brute-force attack. Our current implementation shown in Listing 5 then retracts all messages from the time frame and generates a new syslog message with the facility ”security” and severity ”warning”, containing the message ”SSH brute-force attack” together with further additional information. Beyond that, the new value BRUTEFORCE is added to the list of message tags, allowing us to easily identify the message in a second rule described below.

```
rule "SSH brute-force attempt"
no-loop
when
  Message ($host: host,
           $user: data["user"])
  $atts: CopyOnWriteArrayList(size >= 10)
    from collect(
      Message(tags contains "SSHFAILURE",
              host == $host,
              data["user"] == $user)
      over window:time(1m))
then
  Message last = (Message) $atts.get($atts.
      size() - 1);

  for (Object f : $atts) {
    retract(f);
  }

  insert(messageFactory(last)
    .setTime(last.getTime())
    .setSeverity(Message.Severity.WARNING)
    .setFacility(Message.Facility.SECURITY)
    .setMessage("SSH brute-force attack " +
        "for @{data.user} from @{data.ip}")
    .addTag("BRUTEFORCE")
    .message());
end
```

Listing 5. Drools fusion rule to detect running SSH brute-force attacks.

It would also be possible to postpone the persistence of the logging data until the correlation is finished, to store all messages related to the attack with a higher priority (e.g., emergency) in the Elasticsearch cluster. Another possibility would be to persist the generated messages in addition to the existing ones instead of retracting the original messages. As described earlier, messages still needed for correlation are automatically kept in the in-memory cache by Drools Fusion according to the rules we defined, whereas messages that no longer match any of the rules get removed from the cache, self-controlled by Drools.

A second rule ”Successful SSH brute-force attack”, which we illustrate in Listing 6 matches successful SSH logins after a brute-force attempt was recognized. This is done by detecting a successful authentication message with the tag SSHSUCCESS within a ten-second window after a message containing the tags SSHFAILURE and BRUTEFORCE was recognized, containing the same username respectively. In that case, the message indicating the successful login is altered by increasing the severity to ”emergency” and adding another value ”INCIDENT” to the list of tags, to ease traceability of the security event message. Additionally, a short description is appended to the textual message to make it more meaningful.

```
rule "Successful SSH brute-force attack"
no-loop
when
  $att: Message(tags contains "SSHFAILURE",
                tags contains "BRUTEFORCE",
                $host: host,
                $user: data["user"])
  $suc: Message(host == $host,
                data["user"] == $user,
                tags contains "SSHSUCCESS",
                this finishes[10s] $att)
then
  $att.addTag("INCIDENT");
  $att.setSeverity(Severity.EMERGENCY);
  $att.setMessage($att.getMessage()
    + " [bruteforce]");

  update($att);
end
```

Listing 6. Drools fusion rule to detect successful SSH brute-force attacks.

Figure 5 illustrates how the detection of a successful SSH brute-force attack is displayed in the user interface of our prototype. In the example shown, the correlated message has been logged in addition to the individual login attempt messages. Obviously, the increased severity of the security event is more likely to be recognized by an operator inspecting the syslog messages than events with the regular severity info.

Severity	Facility	Message
emerge	security	Accepted password for root from 10.0.23.4 port 54548 ssh2 [bruteforce]
info	auth	Failed password for root from 10.0.23.4 port 54548 ssh2
warn	security	SSH brute-force attack for root from 10.0.23.4
info	auth	Connection closed by 10.0.23.4 [preauth]
info	auth	Failed password for root from 10.0.23.4 port 54548 ssh2
info	auth	Failed password for root from 10.0.23.4 port 54548 ssh2
info	auth	Failed password for root from 10.0.23.4 port 54548 ssh2

Fig. 5. Extract of our prototype’s user interface showing the detection of a successful SSH brute-force attack.

Instead of altering the matching syslog message, it would also be reasonable to generate a completely new one as shown in Listing 5. An even more interesting approach that we briefly discuss in Section VIII is to trigger an event in an existing network management system like OpenNMS [32].

VI. Using the prototype to enable log correlation in an OpenStack Cloud Environment

The in-memory cache of Drools Fusion limits the number of events our engine is able to correlate. Also, a single engine instance limits the scalability, which is a major drawback in cloud environments that should rather scale elastically both up and down with respect to performance. To overcome these limitations and also to test our implementation in a cloud environment, we integrated our correlation engine in a Rackspace Private Cloud [33] test-bed based on OpenStack Havana [34]. Our syslog correlation prototype shown in Figure 4 was set up as an OpenStack Nova VM instance using an Ubuntu 12.04.3 LTS Cloud Image. Subsequently, Java 1.7 and the code for our prototype (named jCorrelat) were installed.

A. Scalability of our prototype in an OpenStack environment

While the CPU performance and memory capacity of the VM instance could be resized using OpenStack Nova, or respectively the hypervisor beneath it, a comprehensive scale-out solution in the cloud demands the ability to scale over multiple instances. Hence, multiple instances running our prototype should be launched. To balance the load across these VM instances, an additional implementation that distributes the log messages across the instances was evaluated. Furthermore, we considered as an option using the load-balancing facility of OpenStack's networking component Neutron. While both solutions are feasible, the practicability to enhance the scalability of our prototype using these approaches is rather limited. This is due to the fact that, if the correlation were carried out across multiple VM instances, these instances would need some sort of shared or distributed memory. While there are solutions for Java-based, distributed, in-memory data stores, e.g., Terracotta BigMemory [35] or Hazelcast [36], these solutions are rather expensive and also increase the complexity of the management of our solution.

Therefore, we chose a simpler approach that forwards logging information to the corresponding VM instance based on the application it was received from. To accomplish this distribution, multiple output modules were defined in the centralized rsyslog logging server. Figure 6 depicts the usage of multiple VM instances of our correlation engine in an OpenStack test-bed. Each instance receives logging data for a specific application. If the correlation of logging data originating from different applications is required, logging information from multiple applications could be sent to a single VM instance as shown in Figure 6. If the correlation needs more memory, e.g., due to complex rulesets that need to store the logging data for a long period in the in-memory database, distributed memory techniques as described above, e.g., using [35] or [36] could be added. This way, the in-memory cache for the correlation engine stores the logging data across multiple VM instances. Using queries to external data stores for the correlation, e.g., searching in the persisted messages in Elasticsearch, did not prove successful in our tests, as the latency added to each incoming log message is too high to handle bursts of logging information coming from the applications.

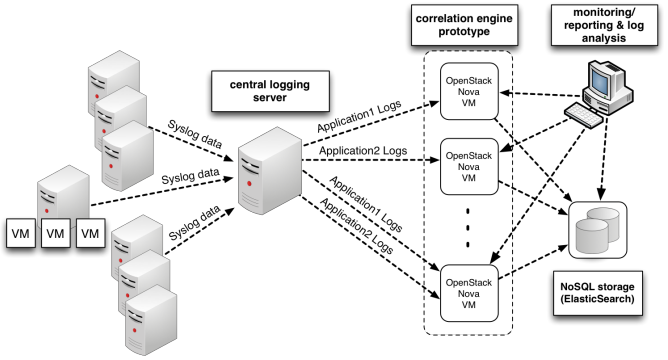


Fig. 6. Scale-out solution for our correlation engine using OpenStack.

As described in Section V, our correlation engine is based on Drools Fusion. Though discussions about a distributed setup can be found in the Drools forums (e.g., the discontinued Drools Grid), Drools is currently using an extended Rete algorithm [37] for its rule engine that relies on shared memory and is therefore limited to a single machine. Hence, our solution to use separate Drools VM instances in OpenStack Nova is currently sufficient for our experiments. Another interesting possibility could be the integration with the Storm distributed and fault-tolerant realtime computation framework [38].

Since we presented our first research results in [1], the OpenStack Havana release has brought two new components that can be used for our distributed syslog correlation engine shown in Figure 6. The first is Heat, which offers an OpenStack orchestration API. Using this API, the setup we described in Figure 6 can be implemented as a Heat template. This template describes the entire environment and VM instance configuration for our correlation engine prototype, hence allowing an automatic deployment within an OpenStack environment. This orchestration could also be combined with configuration management tools like Chef [39] that handle automatic preparation of the operating system and our correlation engine within the VM instance. Furthermore, Heat manages the lifecycle needs of the instance, e.g., scaling instances up and down, but especially stopping and starting new ones. This way, Heat supports an auto-scaling mechanism that enables us to automatically start more VM instances of our correlation engine, e.g., in case of an increasing logging volume.

The auto-scaling mechanism can also include the distribution of logging data originating from a specific application. For example, it is possible to include the name of the application in the name of the VM instance, which can then be used by the central logging server to distribute the logging data to specific correlation instances. As the auto-scaling could lead to high resource usage, the maximum number of instances should be limited. Also, an accounting mechanism for the VM instance usage is necessary. Such an accounting mechanism is offered by another component included in OpenStack Havana, named Ceilometer, which is closely integrated with Heat. Ceilometer offers a standardized interface to collect measurements and accounting information within OpenStack. This opens up an interesting approach to ensure the accounting of our instances.

A promising concept is also proposed by the integration of an Elasticsearch storage driver for Ceilometer in one of the forthcoming OpenStack releases [40]. This way, a standardized way to collect logging information in an OpenStack environment could be combined with the advanced log analysis and long-term storage capabilities offered by Elasticsearch as described in Section II-C. In [40], syslog and logging information coming from other applications within an OpenStack environment has already been considered. If this project would find its way into a new version of OpenStack, we could integrate our correlation engine in between, thereby getting a standardized interface to log data in an OpenStack environment, while also enabling the correlation and consolidation of logging events.

B. Integration of non-syslog logging data

Regarding the OpenStack enterprise cloud test-bed we described in the previous section, besides consuming syslog based logging data, OpenStack also produces its own individual type of log files. OpenStack log files use a structured log format that contains a timestamp, the process ID, the severity and the log message in each line [41]. The log message often spreads across multiple lines as the log is filled with Python tracebacks. One way of using this log data in our correlation engine would be to send it directly to our prototype. As the structured log format used by OpenStack is not fundamentally different from the syslog format described in Section II-B, the adaptation of our prototype to consume OpenStack's Python-based logs is easy to implement. However, the most appropriate way to integrate OpenStack logging in our correlation engine is the configuration of OpenStack to use centralized logging to our rsyslog server, as described in [41]. This can also be implemented for a variety of other application-specific non-syslog logging mechanisms, e.g., log4j, log4net [42]. Due to syslog being the de-facto standard especially for logging in Linux based environments, a large number of application-specific log files and formats can be sent to or consumed by current syslog server implementations. Another benefit of converting application-specific logs to syslog messages, beyond the centralization of the logging data, is normalization as described in Section V and an overall decrease in complexity due to the unified logging.

C. Correlation with information from external management systems

While logging data that is used for the correlation in our prototype should be converted to a syslog-based format (as described in the previous section), information originating from management systems like network management and monitoring, system management, service management or facility management could be valuable to correlate events in the log with events from these external management systems. An example could be the physical location of the node that runs the service in a data center or context information like scheduled downtimes for a service during which certain events should be ignored. This sort of filtering also increases the performance of the correlation as rules could be tailored to take such contextual factors (e.g., downtimes) into account.

Since we use JSON to send the logging data to our correlation prototype, as described in Section V, events coming from external management systems could be injected using a simple TCP or web-service interface. Such a service could be used either to push events from the management system to our correlation engine prototype or to periodically pull information from the management systems. In our OpenStack cloud environment, for example, we use OpenNMS [32] as a network management system that collects monitoring information from the systems and network equipment. By using the events from external network, system and service management systems, the output of the correlation engine could also in turn be used as a feedback for these systems. One example could be the automatic creation of a trouble ticket if a ruleset in our prototype is positively evaluated. Normally, such an automatic creation could lead to a large number of open tickets, but due to the correlation and especially consolidation of events offered by our prototype, the quality of the information and integration with trouble ticket or service management systems could instead be improved.

VII. Performance Evaluation

In this section, we present the results of our performance evaluation study in which we monitored the number of processed syslog packets considering various storage back-ends for permanent data persistence.

A. Test-bed setup

For the performance evaluation we set up a test-bed using the Rackspace Private Cloud as delineated in Section VI. Our prototype runs on an Ubuntu 12.04.3 LTS Cloud Image, which has been assigned four Intel i7 CPU cores at 2.80 GHz, a total of 8 GB memory and a solid-state drive (SSD) storage device. On the same system we have set up rsyslog as the central syslog server together with liblognorm, which we utilize to apply the normalization rules described in Section V. Permanent persistence of syslog messages is done using various storage back-ends that are also set up on the same virtual machine and available to our prototype and the rsyslog server through a TCP socket.

The transport of syslog messages from distributed sources is simulated using loggen [43], a syslog message generation tool provided by the syslog-ng project for testing purposes, which has been installed on a remote machine. The tool may be configured to generate random syslog messages as well as injecting real messages by doing loop-reads on a prepared file. For message submission we provide a dedicated 1 Gbit/s Ethernet connection to the generating machine, connecting it over TCP. The use of any security features like TLS or signatures was renounced in our test in favor of throughput performance.

In order to compare the performance impact of our correlation and consolidation prototype, we set up various storage back-ends and ran the test explained below several times, each time configuring an appropriate rsyslog output module. The individually used storage back-ends and the corresponding output modules are presented in Table I.

B. Evaluation of the test-beds' peak performance

According to Gerhards [44], the rsyslog server is capable of processing up to 250k messages per second over the network. To prove the correctness of our test-bed setup, in a first

TABLE I. UTILIZATION OF STORAGE BACK-ENDS AND RSYSLOG OUTPUT MODULES

Backend type	Output module	Description
File	omfile	Persist messages to a single file on disk
MySQL	ommysql	Persist messages into MySQL using the corresponding TCP socket
Elasticsearch	omelasticsearch	Persist messages into Elasticsearch using the corresponding REST API
jCorrelat (1)	omtcp	Forward messages to our prototype, which persists them into Elasticsearch without applying any correlation rules
jCorrelat (2)	omtcp	Forward messages to our prototype, which persists them into Elasticsearch after applying the appropriate correlation rules

experiment we examined the actual peak performance without any bottlenecks like normalization and correlation or disk I/O latency. Therefore, we configured loggen to generate random syslog messages of different sizes (256, 512 and 1024 byte) and submit these to the central syslog server as fast as possible using six concurrent TCP stream-sockets. The syslog server immediately forwards the received messages to the /dev/null device using the file output module and does not apply any further processing like message normalization or correlation techniques.

Figure 7 depicts our result, showing that the number of messages processed by the central syslog server is dependent on the message size. At an average message size of 512 byte, we get rather close to the number that was stated in [44]. In addition, the graph on the right side of Figure 7 illustrates that we are able to saturate the 1 Gbit/s Ethernet link to about 85.5% of capacity in all of the test cases.

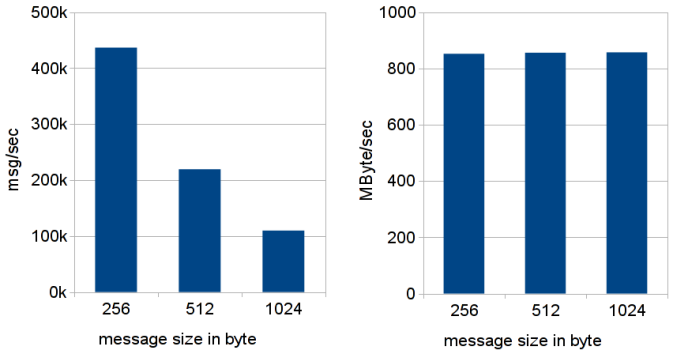


Fig. 7. Maximum syslog packet throughput (left) and bandwidth utilization (right) without any normalization and correlation.

C. Evaluation of the correlation prototype performance

This time, our test setup uses liblognorm at the central syslog server to apply normalization rules as described in Section V. For message generation we again use loggen, which we advised to do loop-reads on a prepared file containing real syslog messages that were previously collected in our test environment. The file contains a total of 20,000 messages including a vast variety of different syslog facilities, one-third of which are authentication related messages including various SSH brute-force attempts.

Figure 8 summarizes the results of our tests. It clearly shows that the best performance can be achieved when writing to a single syslog file on disk, which is not surprising as it does not involve the overhead of sending messages over a TCP socket, like in the other test cases. However, the use of the file storage back-end is included only for comparison, as it appears obvious that its application is not practicable in a centralized syslog environment due to the huge amount of messages from different sources and the lack of any reasonable analysis capabilities. Also, when persisting messages into MySQL, we experienced good performance, but analyzing the logging data afterwards is not easy as the read performance decreases with the number of stored messages. A countermeasure would be indexing, but this in turn creates a heavy burden on write performance. Message persistence into Elasticsearch was done using rsyslog's corresponding output module, which uses the Elasticsearch representational state transfer (REST) interface to insert logging data. As shown in Figure 8, the write performance of this method is poorer than writing to MySQL, most likely because the REST interface involves the overhead of using HTTP when submitting messages.

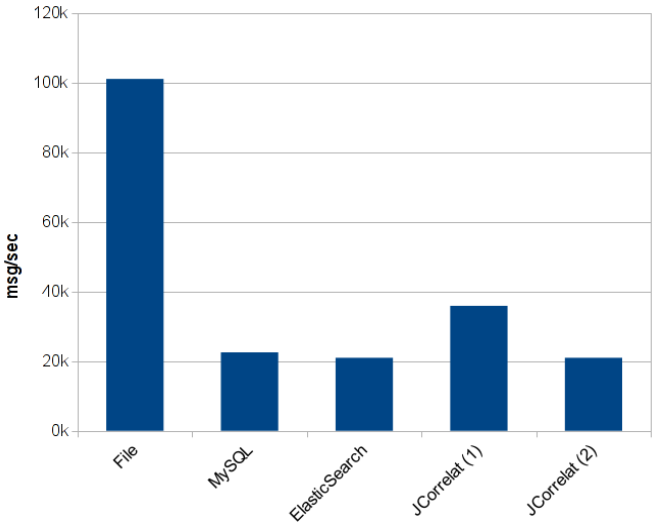


Fig. 8. Comparison of message throughput with different storage back-ends. Our prototype appears twice showing its performance without any correlation rules installed (jCorrelat (1)) and with the rules for SSH brute-force detection applied (jCorrelat (2)).

In the case of sending messages to our prototype, we first measured the message throughput without performing message correlation by simply removing any rules, hence operating in transparent mode. In contrast to using the Elasticsearch REST interface directly via an appropriate rsyslog output module, we get much better performance when using our prototype without applying any correlation. The reason is the usage of the Elasticsearch Java API in our prototype, which allows it to join the Elasticsearch cluster, acting as a transparent node by simply forwarding all messages to the actual data nodes. This way, we are able to decrease the overhead of sending via REST and consequently increase performance.

Finally, when activating message correlation in our prototype as described in Section V-B, we achieve roughly the same throughput we would get by sending messages using the Elasticsearch REST interface or the MySQL back-end, but with the benefit of increased significance of the persisted security event messages. For this reason, we argue that the

application of our prototype is perfectly suitable for logging environments that are already using one of the discussed SQL or NoSQL back-end types. In addition, a feature we described briefly in Section V-A but did not use in our performance evaluation is rsyslog's rulesets facility. It allows normalization rules to be applied only to messages matching a certain ruleset (i.e., matching the syslog facility auth), which would reduce the application of normalization rules to only one-third of the syslog messages we sent to our prototype. As normalization uses string parsing operations, which are known to be CPU intensive, using this method would allow us to reduce normalization overhead and increase performance even further.

VIII. Conclusion and Future Work

In the previous sections, we presented a solution to automatically correlate and consolidate syslog messages containing logging data from distributed sources in cloud environments. Besides evaluating the requirements for such implementations and defining an appropriate concept, a prototype was developed. The prototype addresses the requirements for correlation and consolidation of distributed logging sources in today's enterprise cloud environments. It supports the proper condensation of log messages by grouping individual messages. The achieved reduction improves the performance of processing and analyzing logging data, especially in distributed environments with many systems (typically virtual machines) sending similar logging information.

Existing monitoring solutions could be enhanced to use the presented prototype as a filter, improving the quality and relevance of the logging data (e.g., by using escalation techniques, traps, or sending messages regarding detected events) as shown in the example of an SSH brute-force attack in Sections IV-A and V-B. The integration of the prototype with existing network monitoring tools (e.g., OpenNMS, Splunk) is one of the next steps for our research. An interesting starting point could be their interfaces to correlate events, i.e., to perform a root-cause analysis, that could be extended to consume relevant events that were filtered from the distributed logging data by our prototype.

A current limitation regarding the amount of logging data that can be correlated is the available memory. Theoretically, the prototype could also use data that is already stored in the NoSQL storage for the correlation to overcome this limitation. While this approach has a negative impact on performance, it could on the other hand dramatically increase the accuracy of complex correlation over long-term data. The enhancement could be easily implemented using Elasticsearch's API not only for the analysis but also while filtering and before persisting the logged data in the NoSQL database. In the next version of our prototype, we will implement this extension and evaluate the performance impact (regarding latency to store a log entry and overall throughput of the correlation engine). For example, we could integrate this approach into the OpenStack-based cloud environment presented in Section VI.

Our predefined ruleset outlined in this paper can easily be generalized to fit the requirements of other use cases. In our ongoing evaluation we will therefore contrast the results of our prototype to comparative work being presented in [22], [23] and [24]. Another possible topic for future research is the integration of existing knowledge-based systems and automated reasoning as developed, e.g., for network anomaly and intrusion detection systems (IDS) [45]. Even more interesting could be the integration of existing work that has been published regarding the detection of anomalies in syslog messages. Makanju et. al. [46] describe a promising solution to detect anomalies in logging data of high performance clusters (HPC). Administrators can confirm the detected anomalies to correlate them with error conditions and trigger a consolidation. These techniques could also facilitate the definition of correlation rules as patterns are detected without prior configuration.

Syslog-based event forecasting, as described in [29], could be another promising option for our prototype. The prototype could be used to enhance the information being evaluated to generate the forecast, but can also consume the forecasting data. This way, existing rulesets could be augmented. Furthermore, the definition of rules could be simplified by automatically deriving rules from the forecasts, which have been submitted to our prototype. A starting point for further research could be the use of confidence bands generated by the Holt-Winters algorithm shown in Section IV-C. To detect failures and error conditions in cloud environments this has already been proposed in [47]. We will evaluate the extension of this approach to allow for the correlation and aggregation of logging data in enterprise cloud environments.

Since we published our first research results in [1], Amazon Web Services released the Kinesis cloud service, which offers real-time processing of streaming data at large scale [48]. While this solution seems to offer a promising alternative for the correlation and consolidation of logging data within the Amazon Cloud, moving large amounts of logging data from enterprise clouds towards Amazon presents an obstacle. Additionally, Kinesis is based on streaming, rather than temporal reasoning or complex event processing as described in Section V-B. The scalability of Kinesis, however, is paramount, and we are evaluating integration of techniques like Twitter Storm [38] in our solution. In this respect, Esper [49] also presents an interesting alternative to Drools Fusion and offers high scalability when used in combination with Storm. The visualization and analysis of logging data in an Elasticsearch cluster could also perhaps be improved by enabling time-based comparisons and corresponding plots using Kibana [50], which integrates seamlessly with Elasticsearch.

References

[1] C. Pape, S. Reissmann, and S. Rieger, "RESTful Correlation and Consolidation of Distributed Logging Data in Cloud Environments," In Proceedings of the Eighth International Conference on Internet and Web Applications and Services (ICIW), 2013, pp. 194–199.

[2] Amazon Web Services Inc., "Amazon CloudWatch," https://aws.amazon.com/cloudwatch/ 2014.05.30.

[3] G. Golovinsky, D. Birk, and S. Johnston, "Syslog extension for cloud using syslog structured data - draft-golovinsky-cloud-services-log-format-03," Internet-Draft, IETF, 2012.

[4] C. Lonvick, "RFC 3164: The BSD syslog protocol," Request for Comments, IETF, 2001.

[5] P.V. Mockapetris, "RFC 1034: Domain names - concepts and facilities," Request for Comments, IETF, 1987.

[6] P.V. Mockapetris, "RFC 1035: Domain names - implementation and specification," Request for Comments, IETF, 1987.

[7] R. Hinden and S. Deering, "RFC 2373: IP Version 6 Addressing Architecture," Request for Comments, IETF, 1998.

[8] C. Lonvick, "RFC 3195: Reliable Delivery for syslog," Request for Comments, IETF, 2001.

[9] F. Miao, Y. Ma, and J. Salowey, "RFC 5425: Transport Layer Security (TLS) Transport Mapping for Syslog," Request for Comments, IETF, 2009.

[10] K. E. Nawyn, "A security analysis of system event logging with syslog," As part of the Information Security Reading Room, SANS Institute, 2003.

[11] F. von Eye, D. Schmitz, and W. Hommel, "SLOPPI - A Framework for Secure Logging with Privacy Protection and Integrity," In Proceedings of the Eighth International Conference on Internet Monitoring and Protection (ICIMP), 2013, pp. 14–19.

[12] R. Gerhards, "RFC 5424: The syslog protocol," Request for Comments, IETF, 2009.

[13] G. Klyne and C. Newman, "RFC 3339: Date and Time on the Internet: Timestamps," Request for Comments, IETF, 2002.

[14] BalaBit IT-Security, "Multiplatform Syslog server and logging daemon," http://www.balabit.com/network-security/syslog-ng 2014.05.30.

[15] R. Gerhards, "The enhanced syslogd for linux and unix rsyslog," http://www.rsyslog.com, 2014.05.30.

[16] R. Cattell, "Scalable SQL and NoSQL data stores," ACM SIGMOD Record 39, no. 4, 2011, pp. 12–27.

[17] Elasticsearch Global BV, "Open Source Distributed Real Time Search & Analytics," http://www.elasticsearch.org, 2014.05.30.

[18] Apache Software Foundation, "Apache Lucene," http://lucene.apache.org, 2014.05.30.

[19] R. Marty, "Cloud application logging for forensics," In Proceedings of the 26th Symposium on Applied Computing, ACM, 2011, pp. 178–184.

[20] A. Rabkin and R. Katz, "Chukwa: A system for reliable large-scale log collection," In Proceedings of the 24th International Conference on Large Installation System Administration, USENIX, 2010, pp. 1–15.

[21] D. Jayathilake, "Towards structured log analysis," In Proceedings of the International Joint Conference on Computer Science and Software Engineering (JCSSE), 2012, pp. 259–264.

[22] A. Müller, C. Göldi, B. Tellenbach, B. Plattner, and S. Lampart, "Event correlation engine," Department of Information Technology and Electrical Engineering - Master's Thesis, Eidgenössische Technische Hochschule Zürich, 2009.

[23] M. Grimaila, J. Myers, R. Mills, and G. Peterson, "Design and analysis of a dynamically configured log-based distributed security event detection methodology," In the Journal of Defense Modeling and Simulation: Applications, Methodology, Technology, vol. 9, no. 3, 2012, pp. 1–23.

[24] J. Wei, Y. Zhao, K. Jiang, R. Xie, and Y. Jin, "Analysis farm: A cloud-based scalable aggregation and query platform for network log analysis," In Proceedings of the International Conference on Cloud and Service Computing (CSC), 2011, pp. 354–359.

[25] J. D. Brutlag, "Aberrant Behavior Detection in Time Series for Network Monitoring," In Proceedings of the 14th Systems Administration Conference (LISA), 2000, pp. 139–146.

[26] K. Fukuda, "On the use of weighted syslog time series for anomaly detection," In Proceedings of the International Symposium on Integrated Network Management (IM), IFIP/IEEE, 2011, pp. 393–398.

[27] T. Oetiker, "RRDtool," http://oss.oetiker.ch/rrdtool/, 2014.05.30.

[28] C. C. Holt, "Forecasting seasonals and trends by exponentially weighted moving averages," In International Journal of Forecasting, vol. 20, no. 1, 2004, pp. 5–10.

[29] A. Clemm and M. Hartwig, "NETradamus: A forecasting system for system event messages," In Proceedings of Network Operations and Management Symposium (NOMS), IEEE, 2010, pp. 623–630.

[30] JBoss Community, "Drools - The Business Logic integration Platform," http://www.jboss.org/drools/, 2014.05.30.

[31] R. Gerhards, "A syslog normalization library," http://www.liblognorm.com, 2014.05.30.

[32] OpenNMS Group, "The OpenNMS project," http://www.opennms.org 2014.05.30.

[33] Rackspace Inc., "Private Cloud Computing, Storage & Hosting by Rackspace & Openstack," http://www.rackspace.com/cloud/private/, 2014.05.30.

[34] OpenStack Foundation, "OpenStack Open Source Cloud Computing Software," http://www.openstack.org, 2014.05.30.

[35] Terracotta, Inc., "BigMemory Terracotta," http://terracotta.org/products/bigmemory/, 2014.05.30.

[36] Hazelcast, "Hazelcast: In-Memory Data Grid," http://www.hazelcast.com, 2014.05.30.

[37] R. B. Doorenbos, "Production matching for large learning systems," PhD Thesis, University of Southern California, 1995.

[38] Apache Software Foundation, "Storm, Distributed and fault-tolerant realtime computation," http://storm-project.net, 2014.05.30.

[39] Chef, "Chef, IT automation for speed and awesomeness," http://www.getchef.com/chef/, 2014.05.30.

[40] OpenStack LaunchPad, "OpenStack Telemetry (Ceilometer), Elasticsearch Storage Driver Support," https://blueprints.launchpad.net/ceilometer/+spec/elasticsearch-driver/, 2014.05.30.

[41] OpenStack Foundation, "OpenStack Operations Guide - Chapter 13. Logging and Monitoring," http://docs.openstack.org/trunk/openstack-ops/content/logging_monitoring.html, 2014.05.30.

[42] Apache Software Foundation, "Apache logging services," http://logging.apache.org, 2014.05.30.

[43] BalaBit IT Security, "The syslog-ng Open Source Edition Guide," http://www.balabit.com/sites/default/files/documents/syslog-ng-ose-3.3-guides/en/syslog-ng-ose-v3.3-guide-admin-en/html/loggen.1.html, 2014.05.30.

[44] R. Gerhards, "rsyslog: going up from 40K messages per second to 250K," http://www.gerhards.net/download/LinuxKongress2010rsyslog.pdf, 2014.05.30.

[45] M. Salem, S. Reissmann, and U. Buehler, "Persistent Dataset Generation using Real-Time Operative Framework," IEEE International Conference on Computing, Networking and Communications (ICNC), IEEE, 2014, pp. 1023–1027.

[46] A. Makanju, A. Nur Zincir-Heywood, and E. E. Milios, "Interactive Learning of Alert Signatures in High Performance Cluster System Logs," In Proceedings of Network Operations and Management Symposium (NOMS), IEEE, 2012, pp. 52–60.

[47] Y. Watanabe, H. Otsuka, M. Sonoda, S. Kikuchi, and Y. Matsumoto, "Online failure prediction in cloud datacenters by real-time message pattern learning," In Proceedings of the 4th International Conference on Cloud Computing Technology and Science (CloudCom), IEEE, 2012, pp. 504–511.

[48] Amazon Web Services Inc., "Amazon Kinesis," http://aws.amazon.com/kinesis/, 2014.05.30.

[49] EsperTech Inc., "Esper - Complex Event Processing," http://esper.codehaus.org, 2014.05.30.

[50] Elasticsearch Global BV, "Kibana," http://www.elasticsearch.org/overview/kibana/, 2014.05.30.

Chord-Cube: Music Visualization and Navigation System with an Emotion-Aware Metric Space for Temporal Chord Progression

Shuichi Kurabayashi
Faculty of Environment and Information Studies
Keio University
5322 Endo, Fujisawa, Kanagawa 252-0882, Japan
kurabaya@sfc.keio.ac.jp

Tatsuki Imai
Faculty of Environment and Information Studies
Keio University
5322 Endo, Fujisawa, Kanagawa 252-0882, Japan
t10109ti@sfc.keio.ac.jp

Abstract—**In this paper, we propose an interactive music search-and-navigation system, called Chord-Cube, which visualizes musical similarities on the basis of temporal chord progression. Our proposed system offers an interactive navigation mechanism that allows users to find their desired music intuitively, by visualizing music items in a three-dimensional (3D) space. Each axis in this 3D space corresponds to three types of chord progression phases: Introductive-melody, Continued-melody, and Bridge, which are typical structures in pop and rock music. Users can utilize this 3D space to find their desired song by placing their favorite song at the point of origin and obtaining the semantic distance between the input song and other songs. We have conducted two experimental studies, in which we compared our proposed navigation system with the conventional manual trial-and-error manner, to evaluate the extent to which our visual navigation method improves music retrieval. The results of experiments show that our visual navigation method successfully increases the retrieval performance for pop and rock music.**

Keywords—Music, Visualization, Navigation, 3D, Database.

I. INTRODUCTION

In this paper, we propose an interactive music search-and-navigation system called the Chord-Cube system and its implementation using modern Web technologies. The Chord-Cube system was originally proposed in [1], and this paper is an extended version of the paper [1]. Our system utilizes traditional music theory including tonality and chord progression, which determines the impression of music, to interpret user's feelings about the music. It employs chord progression in songs as a fundamental feature for calculating similarities among music items because we consider chord progression as one of the most important factors in determining the overall mood of a song. By leveraging this music theory knowledge, we develop an intuitive music navigation method to find the desired music by visualizing music-to-music relationships from the viewpoint of temporal similarities in chord progression. Our proposed system provides an interactive 3D cube that visualizes the relative distance among music items by calculating their similarities.

Music has traditionally been regarded as one of mankind's most important forms of cultural heritage. Concomitant with the rapid advances in computing technologies, many songs are being digitized and stored in online libraries and on personal devices. The proliferation of portable and personal devices such as tablet computers and smartphones has resulted in them being frequently used to listen to music. This proliferation and diversity of digital media has increased the demand for effective music retrieval systems [2]. By enhancing the retrieval capability of music, we believe a wider scope can be provided for the sharing of human cultures.

The change in emotion in a song over time is one of the most important factors in selecting music to be played on modern mobile music players and smartphones. Young people, in particular, select music in accordance with their location and mood. To support such intuitive and emotion-oriented music selection, a player that can utilize smart content analysis to extract the movements of musical elements that have profound effects on human perception is needed.

However, current music database systems implemented in online music stores such as the iTunes Music Store and Sony's Music Unlimited do not support such perception-oriented retrieval methods. Consequently, because users often store thousands of music files in the cloud, it is difficult for them to locate their desired songs intuitively, even if they know the details of the desired music. Owing to the temporal nature of music, developing an effective music search environment, in which users can retrieve specific music samples using intuitive queries, is difficult because in order to search a temporal structure, the system has to recognize the changing features of the contents in a context-dependent manner.

Interactive and visual-oriented search mechanisms that do not use text-based search methods are promising because users often memorize music contents with a spatial metaphor. However, a music information retrieval (MIR) method that can reflect the emotions being felt by users as they listen to the music is needed. Such a retrieval environment must have an interactive and navigational user interface that can visualize context-dependent relationships between songs dynamically and in accordance with the user's viewpoint. For this purpose, we have developed Chord-Cube, which visualizes musical similarities calculated by considering emotive movements in temporal chord progression. Whereas traditional MIR systems [3] focus on finding the most relevant song or similar songs by computing similarities or relevance according to extracted features, our proposed system focuses on providing an integrated toolkit for comparing songs in order to create a visualization of implicit interrelationships on the basis of emotional characteristics.

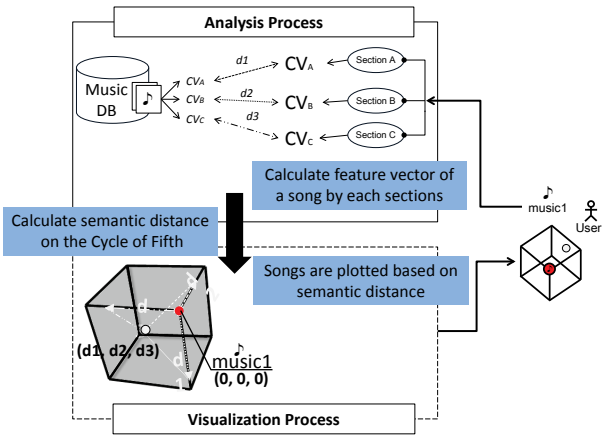


Figure 1. Overview of the Chord-Cube system for visualizing tonality-based distances of songs and navigating users to retrieve their desired song using a query song.

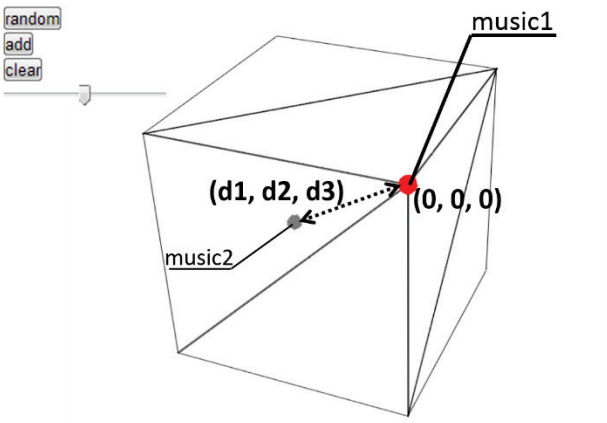


Figure 2. Overview of Chord-Cube visualization.

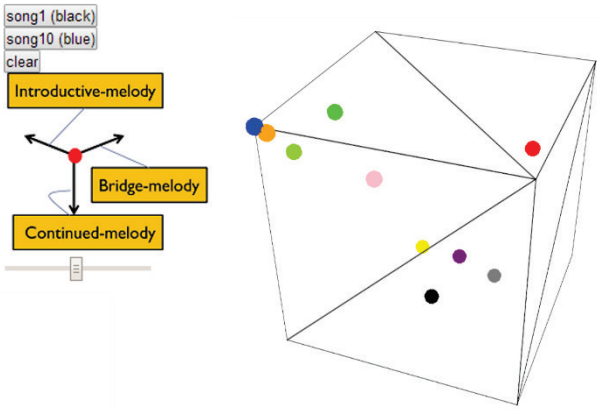


Figure 3. Example of Chord-Cube visualization. Each colored sphere represents a song. A user can operate this cube from any desired perspective.

A unique feature of our proposed system is its "chord-vector space," in which the distance between musical chords can be calculated by analyzing the impressive behaviors of chord progression. Our system uses distances, which are calculated in the chord-vector space, to represent the degree of similarity among songs. As shown in Figure 1, each dimension of this graphical space corresponds to a degree of similarity of chords within three respective sets of song section types: "introductive-melody," "continued-melody," and "bridge-melody." Introductive-melody is a beginning of the musical piece. Continued-melody is an interlude between the introduction and bridge section. A bridge is a representative section that expresses the salient feature, and it is one of the most impressive sections in a musical composition.

This cube is a three-dimensional object inside which songs are displayed as points. Our cube accepts an initial song as a point of origin in the cube. Users can choose any song as their point of origin. The system then plots other songs inside the cube by reflecting the distance between each song and the song at the point of origin.

We implemented a prototype of our system utilizing modern HTML5 technologies. The implemented prototype system assumes that it will be applied to the online music store as a front-end user interface. It utilizes WebGL, which is a standardized API for rendering interactive 3D graphics within web browsers without the use of any plug-ins. In the system, a dynamic distance calculation method that applies chord progression data is implemented in JavaScript. Thus, this system provides a fundamental framework for implementing the user interface (UI) of an online music database system.

The remainder of this paper is organized as follows. Section II discusses related research. Section III presents several motivating examples that demonstrate how our system can be utilized to retrieve unknown songs. Section IV gives an architectural overview of the system, Section V demonstrates its fundamental data structures, Section VI defines its core functions, and Section VII outlines its prototype implementation. Section VIII discusses our feasibility studies conducted. Finally, Section IX concludes this paper.

II. Related Work

In this paper, we present a system architecture that aims to improve the effectiveness of music retrieval approaches by visualizing multi-aspect similarities among songs. Conventional music database systems that are available on the Internet utilize metadata, such as genre and artist name, as indexing keys. However, such fundamental metadata are not sufficient to retrieve music without detailed knowledge of the target data. Consequently, content-based retrieval and advanced query interpretation methods have been developed to find music. In content-based music retrieval methods, a user inputs a raw music file as a query that the system analyzes and extracts several significant features from in order to identify equivalent or highly similar music samples in a database. As an example of the content-based music retrieval method, there are several input materials such as humming [4][5][6] and chords [7][8]. The content-based method has advantages in terms of ease of input and the ability to generate a large amount of information reflecting musical content. Because content-based technologies are very effective in retrieving musical equivalents to input queries, they are widely used for copyright protection in online music sharing services.

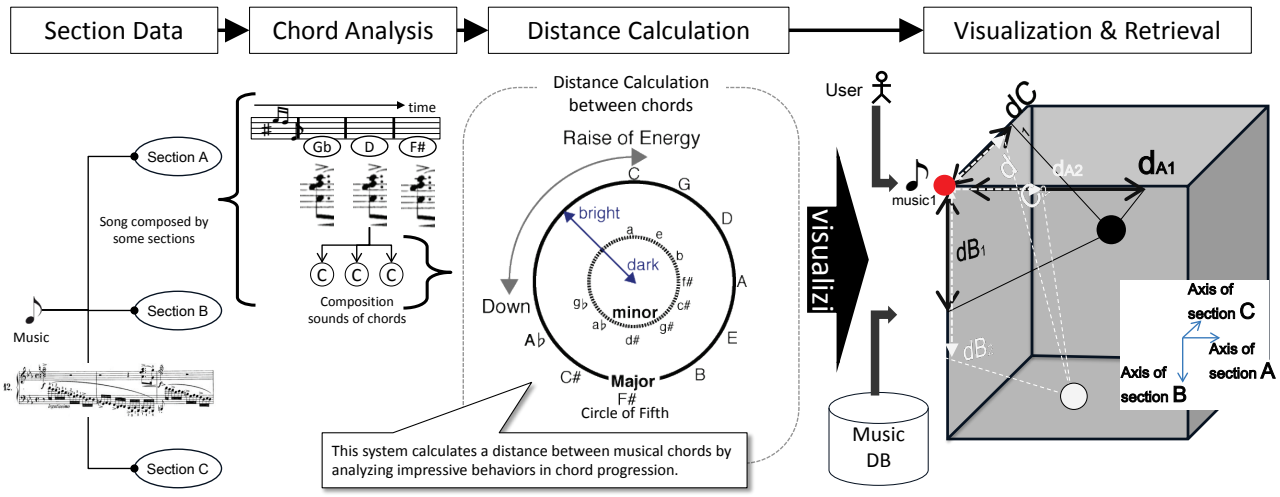


Figure 4. System architecture of Chord-Cube for visualizing tonality-based relevance among songs.

However, ordinary users also want to be able to find new and unknown music more easily, and a method for retrieving music that is similar but not exactly equal to a query would be most helpful in attaining this goal. A wide variety of visualization techniques have been proposed in the context of content-based MIR [9]. For example, Pampalk et al. [10], proposed an interface for discovering artists by using a ring-like structured visual UI, Knees et al. [11] developed a method for visually summarizing the contents of music repositories, and Stober et al. [12] proposed an interface that can conduct music searches on the basis of unclearly defined demands.

Dittmar et al. [13] introduced "GlobalMusic2One," a portal site for visualizing songs by using two-dimensional similarity maps for explorative browsing and target-oriented searches. To enhance the retrieval effectiveness of songs, a music recommendation filter [14] utilizing a user's personal preferences and a method for managing songs in mobile environments [15] has also been proposed. From the aspect of musical structure analysis, several music visualization systems have been developed [16][17][18]. These methods use the color sense of tonality to view the harmonic structure and relationships between key regions in a musical composition. Imai et al. [19] also proposed tonality-based visualization as a means of enhancing the find-ability of music.

The most significant difference between conventional approaches and our approach is that our system focuses on the development of a method for emotion-based music visualization. Conventional visualization MIR methods automatically extract content information from audio signals by applying signal processing and machine learning techniques, whereas our system analyzes emotional transitions by capturing the progression of chords as a trajectory of "how the music sounds." Our system can calculate the evolving distance between two chord vectors as a continuous comparison along a timeline. Another significant innovation delivered by our method is the use of an interactive 3D visualization space. This visualization method configures a 3D cube around an example query serving as an origin vertex point, and displays each musical item according to its relevance score relative to the example query.

III. MOTIVATING EXAMPLE

Our Chord-Cube system is envisioned for use in scenarios such as the following. Imagine that a user has 1,000 songs in his/her smartphone and s/he desires to select songs that are similar in emotion to a specific song in the smartphone. In this case, the user could retrieve a desired song by inputting a sample song and browsing the visualized 3D cube where relevant songs are located close to the input song. As shown in Figure 2, when the user inputs a song, the system positions the song at vertex (0, 0, 0) of the cube. Further, the other songs are plotted within the 3D cube, as shown in Figure 3, indicating the distance between the query song at the vertex and the various points in the cube. Users can then compare songs from various perspectives as follows: similarity in "introductive-melody," similarity in "continued-melody," and similarity in "bridge-melody." Users can rotate this cube to find the most desirable song.

Another scenario that exemplifies the objective of our system relates to the user experience aspect. First, the user selects his/her favorite song in a smart device, such as an iPad or an Android tablet. Then, the system generates the cube showing the relevant songs around the selected song. The user then draws an oval from within which songs are selected and added to the playlist. Such a spatial approach to defining a playlist is effective because of the distance metrics in our cube visualization. Because our visualization mechanism shows dynamically measured semantic distances between music items rather than relevance rankings, the visualized music space provides an intuitive interface for users to choose new music samples of interest.

IV. SYSTEM ARCHITECTURE

A. Architectural Overview

Figure 4 gives an architectural overview of our Chord-Cube system. Music navigation within the Chord-Cube system is achieved through integration of music content analysis and relevance visualization. The overall system comprises a

distance calculation module and a visualization module. In order to extract the chord features of a music sample, the distance calculation module inputs the music sample as a query for analysis. The module then computes the distances between the chord features extracted from the query and each music item within the database on the basis of a key distance calculation technology that can measure the distance between two chords according to their respective temporal contexts (i.e., chord progressions). To define the relationship between chord combinations and progressions, we have developed a matrix-based data structure.

B. Emotive Distance Calculation Using Circle of Fifth

The system calculates the similarity of songs using the impressive motion defined in Circle of Fifth [20]. The center of Figure 4 illustrates the distance metrics used by the Circle of Fifth [20]. This circle represents the relations of closeness or similarity and distance between tonal elements. In this circle, two adjacent tonalities have similar impressions, but opposite face tonalities have opposite impressions. By tracing a trajectory of chords within the Circle of Fifth, the system can calculate and represent the manner in which the music affects a listener's emotional perceptions. Figure 11 shows a visualization of tonality changing in a music item. The horizontal axis corresponds to timeline, whereas the vertical axis corresponds to the tonality relevance score. This chart shows 12 types of major tonalities and 12 types of minor tonalities. As shown in the chart, one musical composition contains continuous changes in tonality. To detect the emotional changes in music, it is important to trace this tonality behavior. To analyze the change in tonality, well-studied key-finding algorithms, such as the Krumhansl-Schmuckler algorithm [20] and the Temperley algorithm [21][22], can be used. We implemented the Krumhansl-Schmuckler algorithm in the Chord-Cube system.

In order to make selection of the desired music easy, the system displays the calculated distances between samples in a 3D graphical user interface. The visualization module constructs a virtual cubic space consisting of axes corresponding to three music structures typically found in J-pop music: introductive-melody, continued-melody, and bridge. The input query is placed at the origin, while target music items are located within the space according to their respective relevance scores; thus, the most relevant music item is located the closest to the origin, while irrelevant music items are scattered further away.

C. Visualization Process

The system performs chord progression oriented music visualization using the following steps:

Step-1. A user inputs a song as a criterion for finding new songs (Figure 5).

Step-2. The system divides the song's chord progression into component sounds (Figure 6).

Step-3. Using a method based on the cycle of fifths, the semantic distances between components are calculated and placed within a feature vector, called the chord vector (Figure 7).

Step-4. The inner products between the chord vectors of each section are calculated to determine the similarities between each of the sections (Figure 8).

Step-5. The relevance of each song is then plotted within a 3D cube in order to present an intuitive visualization of the distance between the song at the vertex and the various points in the cube (Figure 9).

Step-6. Further retrieval can be done by translating another song within the cube to the vertex in order to create a new relevance comparison based on the selected song as the origin (Figure 10).

These visualization mechanisms allow users to retrieve a desired song from an intuitive visual space based on its similarity in chord progression to the reference query song at the vertex.

V. Data Structures

Our system contains three fundamental components: A) musical instrument digital interface (MIDI) song data, B) chord progression, and C) component sounds distance matrix.

A. MIDI Song Data

This system uses standardized MIDI data format as the primary data format for storing music data in a file system. MIDI stores note-on signals and corresponding note-off signals sequentially because MIDI was developed in order to automate keyboard-type instruments. The system represents a MIDI file as $F := \{n_1(t, p, d), n_2, \ldots, n_k\}$, where n_i represents the i-th note, whose attributes are t: the start time of the note, p: the pitch of the note, and d: the time duration of the note. F is a sequential set of k-tuple data.

Our system provides a matrix structure that represents the continuous changing and distribution of pitch in the target music data. We call the data structure a music pitch matrix. The pitch matrix is a 128 by n matrix that is given as the data matrix. MIDI specification defines the domain of the pitch value as zero to 127. A musical composition is expressed as a set of m timelines. Each timeline is characterized by a note on information for zero to 127 pitch levels. When the 12-th note is on in an m-th section, $c_{[12, m]}$ is one. The pitch matrix P is defined as follows:

$$P := \begin{pmatrix} c_{[0,0]} & \cdots & c_{[0,n]} \\ \vdots & \ddots & \vdots \\ c_{[m,0]} & \cdots & c_{[m,n]} \end{pmatrix} \quad (1)$$

where $c_{[i,j]}$ denotes the status of the j-th pitch at the i-th time duration. We implemented the MIDI analysis modules for converting MIDI into a musical score-like data structure by using our MediaMatrix system [23], a stream-oriented database management system.

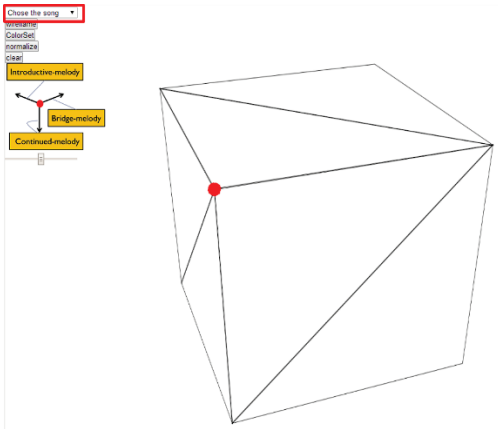


Figure 5. Querying Step-1: A user chooses a song as an origin point.

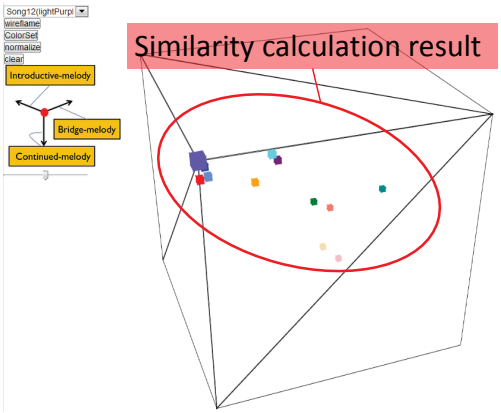


Figure 6. Querying Step-2: System visualizes similarity calculation results

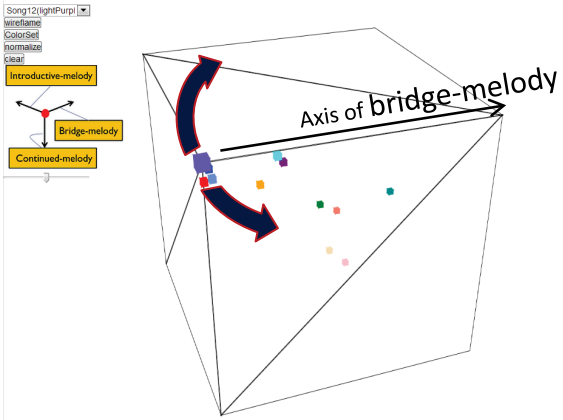


Figure 7. Querying Step-3: User rotates the cube about the axis of the bridge-melody

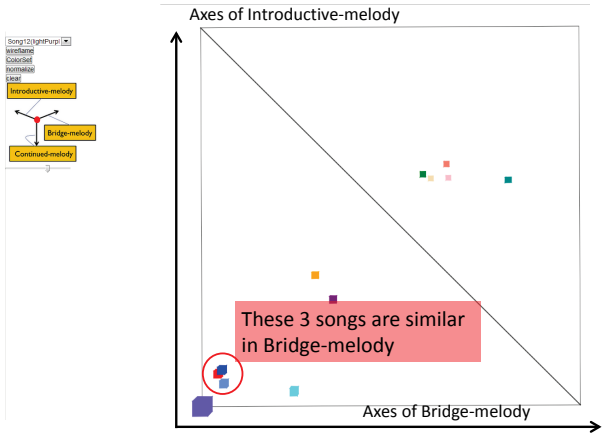


Figure 8. Querying Step-4: User can get information about songs with similar bridge-melody.

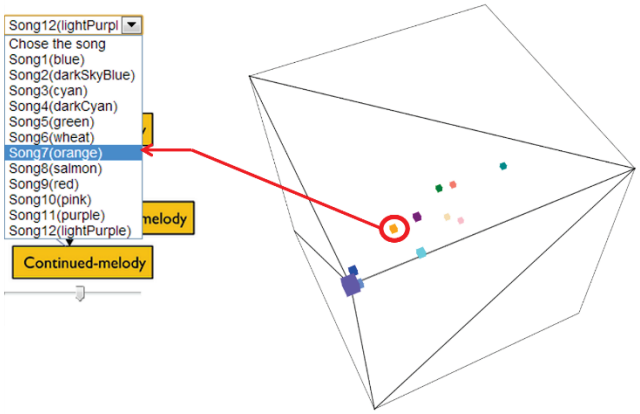


Figure 9. Querying Step-5: User selects another song for new criteria of visualization

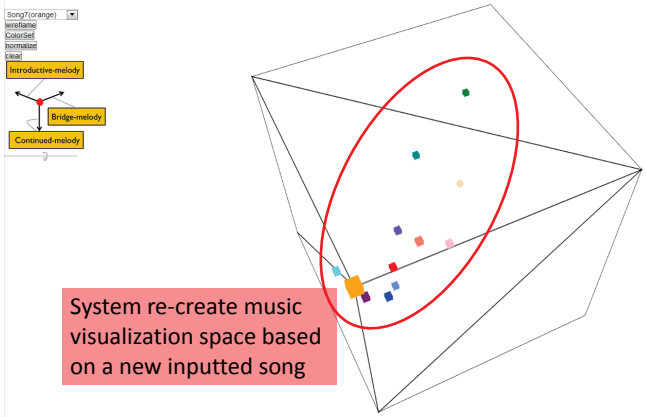


Figure 10. Querying Step-6: System re-creating similarity visualization space on the basis of the new criteria

B. *Chord Progression*

Chord progression refers to continuous chord changes over time. We adopt the concept of tonality, which is a musical system that is constructed by sound elements, such as harmonies and melodies [20]. Different tonalities have different impressions. In one musical composition, tonality changes from section to section. It is important to support this change in tonality in a music database because changes in tonality causes changes in impression.

A music item is modeled as a sequence that consists of chords. More specifically, we define an item Music (M) as a data structure consisting of a sequence of chords (c). Music M_i is defined by the following equation:

$$M_i := \langle c_0, c_0, c_1, \cdots, c_n \rangle \quad (2)$$

where n is the number of chords. A chord is a 12-tuples relevance score, where each tuple corresponds to a specific type of tonality such as C and $C^{\#}$. Therefore, we define a chord (c) as a data structure based on correlation of k-th tonality (v_k). Chord c_j is defined by the following equation:

$$c_j := \langle v_1, v_2, \ldots, v_{12} \rangle \quad (3)$$

where v_k corresponds to the k-th tonality; hence, there are 12 values in this vector.

C. Component Sounds Distance Matrix

Chord progressions are composed of three or more overlapping sounds. We call these overlapping sounds "component sounds." We have developed a correlation matrix that defines the movement distance for each combination of tonalities. Figure 12 shows a component sounds distance matrix designed using the Circle of Fifths. The component sounds distance matrix is a 12 × 12 matrix that is given as the data matrix. The size of the matrix corresponds to the number of tonality types defined in the Circle of Fifth. In this matrix, a larger value signifies a stronger correlation. Thus, C and $C^{\#}$ (0.83) are more correlative than C and $D^{\#}$ (0.50). The component sounds distance matrix T is defined as follows:

$$\mathrm{T} := \begin{pmatrix} d_{[1,1]} & \cdots & d_{[1,12]} \\ \vdots & \ddots & \vdots \\ d_{[12,1]} & \cdots & d_{[12,12]} \end{pmatrix} \quad (4)$$

where $d_{i,j}$ denotes a correlation value for the j-th and i-th tonalities.

Our Chord-Cube system uses this matrix to calculate the similarity between songs, based on their component sounds, by multiplying the number of occurrences of each particular sound by its respective distance. As a result, we can obtain a vector representing the strength of the sounds in the song. We call this vector the "chord vector." The system then constructs a chord-vector space consisting of the calculated 12-dimensional values. The system calculates the relevance of two songs by measuring the distance of two chord vectors, which represent how the chords change in each song. In addition, the system can compare songs according to their sectional contents, such as introductive-melody, continued-melody, and bridge-melody, by calculating a chord vector based on each section of a song.

VI. CORE FUNCTIONS

Our system contains four fundamental components: A) a chord detector for converting a pitch matrix into chord progression array, B) a chord-vector generation module for generating a vector data by analyzing the chord progression array, C) a distance calculation module applied to determine the semantic distance of songs, and D) visualization module.

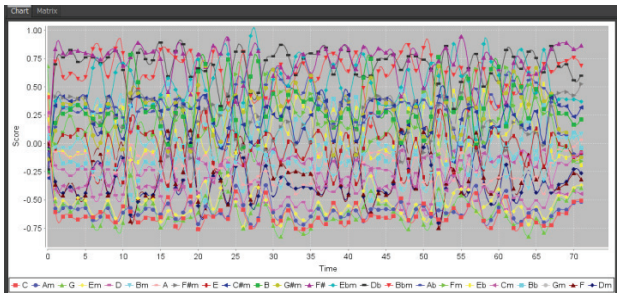


Figure 11. A visualization of tonality changing in one music item. The tonality changes with time.

	C	C#	D	D#	E	F	F#	G	G#	A	A#	B
C	0	0.83	0.33	0.50	0.67	0.17	1	0.17	0.67	0.50	0.33	0.83
C#	0.83	0	0.83	0.33	0.50	0.67	0.17	1	0.17	0.67	0.50	0.33
D	0.33	0.83	0	0.83	0.33	0.50	0.67	0.17	1	0.17	0.67	0.50
D#	0.50	0.33	0.83	0	0.83	0.33	0.50	0.67	0.17	1	0.17	0.67
E	0.67	0.50	0.33	0.83	0	0.83	0.33	0.50	0.67	0.17	1	0.17
F	0.17	0.67	0.50	0.33	0.83	0	0.83	0.33	0.50	0.67	0.17	1
F#	1	0.17	0.67	0.50	0.33	0.83	0	0.83	0.33	0.50	0.67	0.17
G	0.17	1	0.17	0.67	0.50	0.33	0.83	0	0.83	0.33	0.50	0.67
G#	0.67	0.17	1	0.17	0.67	0.50	0.33	0.83	0	0.83	0.33	0.50
A	0.50	0.67	0.17	1	0.17	0.67	0.50	0.33	0.83	0	0.83	0.33
A#	0.33	0.50	0.67	0.17	1	0.17	0.67	0.50	0.33	0.83	0	0.83
B	0.83	0.33	0.50	0.67	0.17	1	0.17	0.67	0.50	0.33	0.83	0

Figure 12. Component sound distance matrix representing distance between each sound based on tonality.

A. Chord Detector

The system provides a fundamental function to convert a pitch matrix into chord progression arrays. The function f_{map} extracts chords by detecting three or more overlapping sounds in the pitch matrix. We define f_{map} (P_i) that inputs a pitch matrix P_i as follows:

$$f_{map}(P_i) \rightarrow M_i \quad (5)$$

where M_i denotes a sequence of chords. The detailed definition of M_i is given in Section V-B, equations (2) and (3).

B. Chord Vector Generation

The system generates a chord vector by summing the matrix consisting of the products of the semantic distance of each sound on the cycle of fifths with the number of occurrences of that sound, as defined by

$$f_{CV}(d, e) := \left(\sum_{i=1}^{12} d_{[i,1]} \cdot e_{[i]}, \quad \cdots, \quad \sum_{i=1}^{12} d_{[i,12]} \cdot e_{[i]} \right) \quad (6)$$

where d represents the distance between the component sounds, while e represents the number of occurrences of each component sound. The chord vector thus generates and stores a correlation between all component sounds in each section.

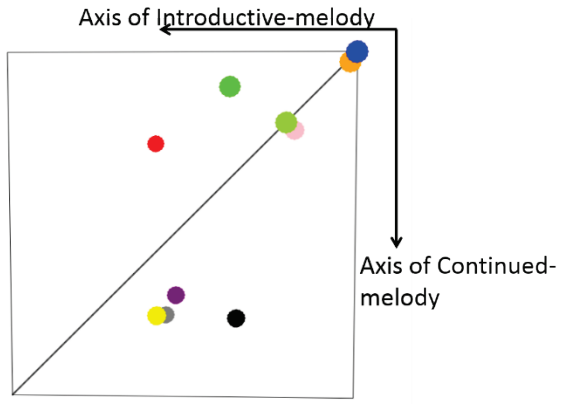


Figure 13. The system facilitates comparison of music items from multiple perspectives. In this case, a user compares from introductive-melody and continued-melody.

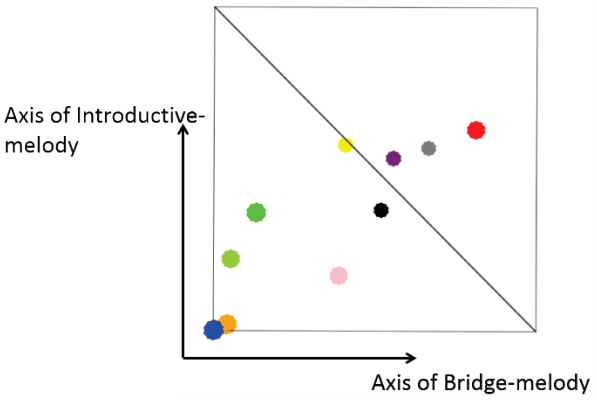


Figure 14. The system facilitates comparison of music items from multiple perspectives. In this case, a user compares from introductive-melody and bridge-melody.

C. Distance Calculation

As stated in the previous section, the chord-vector matrix is derived by multiplying the component sounds distance matrix with the number of occurrences of each sound; this result consists of a 12-dimensional vector representing the strength of each sound within a section. The system compares songs in terms of their representative features encoded in the 12-dimensional distance metric space ("chord-vector space") by their respective chord vectors. Distances between sections are calculated from the inner products of vectors, using

$$f_{distance}(CV_1 \cdot CV_2) := \sum_{i=1}^{12} CV_{1[i]} \cdot CV_{2[i]} \qquad (7)$$

where CV_1 and CV_2 are the chord vectors of two different songs.

D. Visualization Module

The system utilizes the chord vector to compare user-selected songs to all songs in the music database. Defining each section of music1 (i.e., a user-imported song) as S1a, S1b, and S1c, and of music2 (another song in the database) as S2a, S2b, and S2c, the similarity calculation function distance between S1a and S2a is calculated as d1, the distance between S1b and S2b is d2, and the distance between S1c and S2c is d3. If, on the 3D space consisting of the respective song section type, music1 is located at the origin (0, 0, 0), then the coordinates for music2 can be represented as (d1, d2, d3). Thus, the system can visualize the distances between songs as Cartesian distances in a solid body called the "Chord-Cube," as shown in Figure 3.

The system is able to adopt differing user-input styles; therefore, it is able to make comparisons between songs on the basis of varying criteria. Each song can be assigned vector values and allocated a coordinate in the cube on the basis of its correlation to a particular criterion, creating a space that intuitively represents the semantic distance between songs, and in which the most relevant piece of music is located very close to the origin, while irrelevant items are more remote. Figures 13 and 14 show typical and effective use cases of this system. A typical scenario in which a user compares songs from multiple aspects is depicted. Figure 13 shows a perspective for detecting the similarity by using the introductive-melody and continued-melody. Figure 14 shows a comparison between the introductive-melody and the bridge-melody. It can be seen that there are obvious differences about the dark green and pink spheres between those two figures. In Figure 13, the two songs represented by these spheres have identical similarities to the blue sphere, whereas they are separated in Figure 14. This means that the two songs are similar in terms of introductive-melody and continued-melody, but have different features in terms of bridge-melody.

VII. Web-Based System Implementation

We implemented a prototype of the Chord-Cube system that calculates the similarity between songs and visualizes them in a 3D cubic space. Screenshots of the prototype, which uses HTML5 Canvas and JavaScript, are shown in Figures 5 through 10. Figure 15 details the architecture of our prototype system, which specifically includes the modern HTML5 technologies WebGL API, Web Storage API, and Web Worker API. The system consists of the following three modules: a query input module, a distance calculation module, and a visualization module. We describe these components in detail below.

The main user interface is the visualization module, which uses the HTML5 WebGL API to render a three-dimensional interactive screen. We implemented this prototype system by utilizing three.js (http://threejs.org/), an open-source WebGL wrapper utility library. The implemented system extends three.js to support interactive music data visualization and real-time rendering of the chord-vector space. This 3D UI enables users to compare songs from any desired perspective. Users can view the rendered cube and spheres representing songs from any angle by rotating the cube, zooming in, and zooming out.

When users input a song as a query, the system invokes the MIDI file analyzer implemented in JavaScript. This MIDI file analyzer is implemented using the HTML5 FileReader and ArrayBuffer objects. On completing the analysis process,

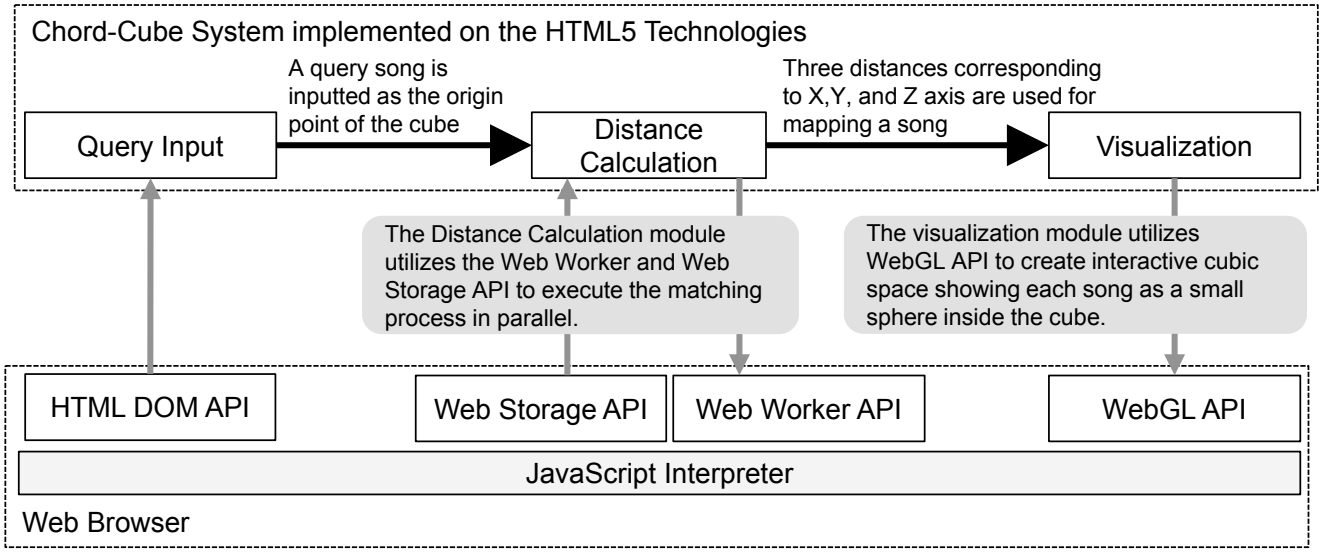


Figure 15. Conceptual view of the query-by-appearance system for style-oriented e-book retrieval using encapsulated editorial design templates for query generation.

the visualization module renders the query song on a vertex of the cube. In addition, the MIDI file analyzer encodes the analysis result into JavaScript Object Notation (JSON) format and passes it to the distance calculation module. This procedure allows our system to share the JSON-encoded figure among multiple web workers to parallelize the execution of distance calculation.

The distance calculation module compares the queries and the database contents. This retrieval process is parallelized by the Web Workers API, and the retrieved songs are presented to the user by the search result visualization engine. This system spawns real OS-level threads from the Web Workers API to parallelize the retrieval process. In this way, modern HTML5 technologies enable us to implement complex processes in web browsers.

After getting a number of users to evaluate the implemented system, we became cognizant of two primary music retrieval use cases. In case-1, the user desires to search for similar songs via the bridge-melody of one song. In this case, the user performs the following music retrieval process:

- Step-1: The user inputs the song that s/he wants to set as the comparison criteria for a bridge-melody.
- Step-2: The system visualizes the similarity calculated based on the input song.
- Step-3: The user rotates the cube on the axis corresponding to the bridge-melody.
- Step-4: The user obtains songs similar in bridge-melody by seeing the visualized results around the axis of the bridge-melody.

In case-2, after the user has found his/her desired song, s/he uses the found song as a query in order to retrieve more songs. This case continues the previous process in case-1.

- Step-5: The user selects a specific song as a new query from the visualized cube.
- Step-6: The system recreates music visualization space based on the new query song.
- Step-7: The user repeats Step-5 and Step-6 until s/he has retrieved enough music items.

VIII. Evaluation

In this section, we discuss several experiments conducted to evaluate the effectiveness of our Chord-Cube system when applied to existing Japanese Pop songs. We conducted the following two experimental studies: Experiment-1, evaluation of the precision of dissimilarity calculations; and Experiment-2, evaluation of the effectiveness of our visualization. We performed the two experiments by comparing the results of similarity measurements between the implemented system and the results of questionnaires submitted to listeners who awarded points based on the level of similarity that they felt. As preprocessing for the two experiments, we asked 10 subjects (three male and seven female) to create a correct set for each query in Experiment-1 and Experiment-2. The correct set is a data set that stores only items that are considered relevant to a query by test subjects.

A. Experiment-1: Outline of Experimental Studies

Experiment-1 was conducted to evaluate the effectiveness of our similarity calculation precision. For this experiment, we chose one query song as a criterion and 10 other songs as comparison targets. Ten listeners used a one-to-five scoring template to evaluate their perceptions of similarity between each comparison song and the criterion by section, after which we aggregated the scoring results from each listener and converted them into reciprocal values defined as the "dissimilarities by survey." We then used these values to calculate the distance within the Chord-Cube of each target song from the criterion point (the query); this process is called "collection of data dissimilarity." To evaluate the effectiveness of our method, we compared the dissimilarities by survey to the dissimilarities as calculated by our method. In Experiment-1-A, we applied our system to measure dissimilarities of introductive-melody for each music item, whereas in Experiment-1-B,

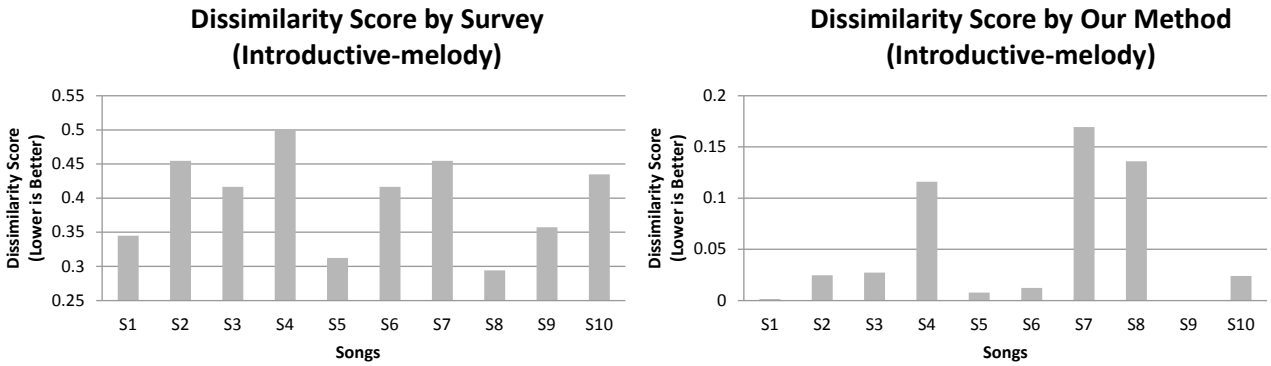


Figure 16. Results of Experiment-1-A: Dissimilarity measurement for introductive-melody.

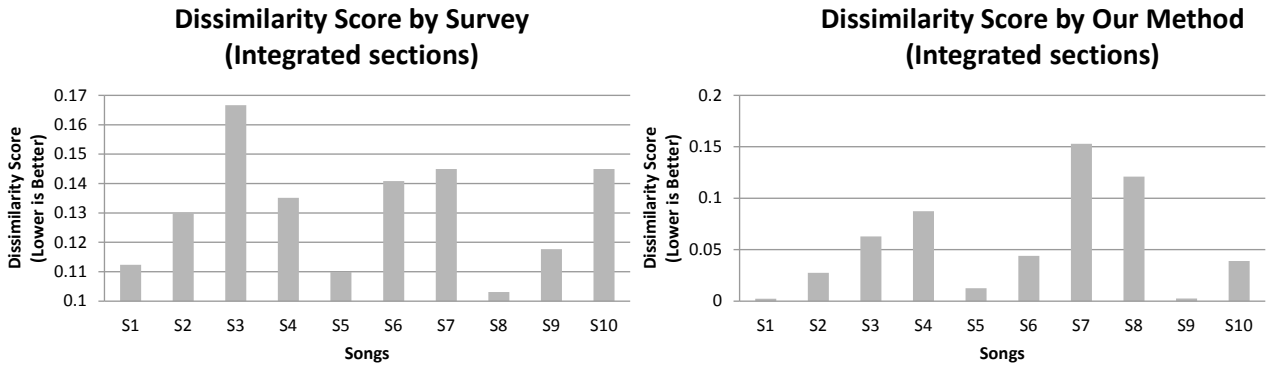


Figure 17. Results of Experiment-1-B: Dissimilarity measurement for integrated sections.

we measured dissimilarities of integration of introductive-melody, continued-melody, and bridge-melody for each music item.

B. Experiment-1: Experimental Results

Figure 16 and TABLE I show the results for Experiment-1-A. The left-hand side of the figure shows the dissimilarity as measured by the manual survey, while the right-hand side shows the dissimilarity as measured by our system. It can be seen in TABLE I that the test subjects judged songs s1, s4, s5, s8, and s9 to be highly similar to the query music, whereas our system retrieved songs s1, s5, s6, s9, and s10 as similar music; thus, the system correctly extracted songs s1, s5, and s9.

Figure 17 and TABLE II show the results for Experiment-1-B. As before, the left-hand side shows dissimilarity measured by manual survey, and the right-hand side shows dissimilarity measured by our system. By comparing Figures 16 and 17, it can be seen that the surveyed dissimilarity of song s3 significantly increases from Experiment-1-A to Experiment-1-B, whereas our system returns identical results for all songs in both experiments. Thus, it can be concluded that our system improves its retrieval precision by integrating a differing evaluation axis into the Chord-Cube visualization space, and thus can effectively display multiple perspectives simultaneously.

The results for song s8, on the other hand, show that some improvements are still necessary. Whereas the survey results judged s8 to be similar to the query music, our system judged it to be dissimilar. We believe that a perceptional gap between the theme melody and the chords progression of song s8 strongly affected the results here, because s8 has a complex chord progression but a very simple melody. However, the experimental results from the other songs closely parallel the results obtained from the dissimilarity by survey, clarifying the overall effectiveness of our method for utilizing chord-metric space and 3D visualization.

C. Experiment-2: Outline of Experimental Studies

In this section, we evaluate the precision of our visualization result by using three types of queries. This experiment clarifies that our approach calculates the appropriate distance between songs. As in Experiment-1, we compared the results of similarity measurements between calculated results and questionnaire survey. For this experiment, we established three query songs as criteria and ten other songs as comparison targets. We have selected three songs from ten JPOP songs randomly. Ten test subjects (three male and seven female) used a one-to-five scoring template to evaluate their perceptions of similarity between each comparison song and the criterion by section. The scoring template is as follows: 0 (completely irrelevant), 1 (irrelevant), 2 (slightly relevant), 3 (relevant), and 4 (very relevant). We consider the ideal ranking as the average of ten results. We then used these scores to

TABLE I. SIMILARITY RANKS OF INTRODUCTIVE-MELODY

Rank	Survey	Score	Our Method	Score
1	s8	0.294118	s9	0.000280
2	s5	0.312500	s1	0.001422
3	s1	0.344828	s5	0.007712
4	s9	0.357143	s6	0.012242
5	s6	0.416667	s2	0.024543

TABLE II. SIMILARITY RANKS OF INTEGRATED SECTIONS

Rank	Survey	Score	Our Method	Score
1	s8	0.103093	s9	0.002429
2	s5	0.109890	s1	0.002381
3	s1	0.112360	s5	0.012566
4	s9	0.117647	s2	0.027525
5	s2	0.129870	s6	0.044053

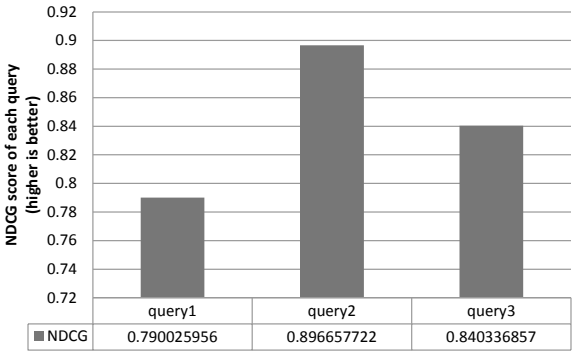


Figure 18. NDCG Scores.

compare with the distance from the origin point in the visualization result.

D. Experiment-2: Experimental Results

To evaluate this experiment, we computed the normalized discounted cumulative gain (NDCG) as follows:

$$DCG = \sum_{i=2}^{11} \frac{rel_i}{log_2 i} \quad (8)$$

$$IDCG = \sum_{i=2}^{11} \frac{rel'_i}{log_2 i} \quad (9)$$

$$NDCG = \frac{DCG}{IDCG} \quad (10)$$

where rel_i are the average survey scores given by the test subjects in order based on ranking of visualized distance, and rel'_i are the average scores in descending order. Figure 18 shows the NDCG of visualization in the Chord-Cube for three queries. A higher score implies a better retrieval precision. From this experimental result, we obtain a value for NDCG that is higher than 0.79 in every query. This result explains the high precision of the visualization result of our system.

IX. Conclusion and Future Work

In this paper, we proposed the Chord-Cube system, a music visualization and navigation system that provides an intuitive visual retrieval method using chord-metric space. The unique feature of this system lies in its construction of a chord-vector space to extract the transition of emotions within a song as a feature vector. We implemented a prototype system utilizing modern HTML5 technologies. The implemented system supports the chord-metric based similarity between songs according to a user's selected criterion song and visualizes that result within a 3D cube constituted by three evaluation axes. We also performed evaluations of the effects of our system applied to existing J-pop songs. Our experimental results indicate that visually represented search results carry out a practical function.

In future work, we plan to improve the chord-metric space by capturing the direction of chord transitions in order to represent the change in emotional energy through the resulting motion on the cycle of fifth. We are also developing an automatic playlist generation function using the spatial analogy for selecting songs in the visualized cube. The most important future work is to apply our system to raw audio signals, such as MPEG Audio Layer-3 (MP3) format. Our system rely on the score data of music, so we are planning to integrate an existing music transcription system into the Chord-Cube system. In order to enhance the query description scope, multiple songs can be used as a query. We have implemented such a query interpretation method for video retrieval in [24].

References

[1] Imai, T. and Kurabayashi, S., "Chord-Cube: multiple aspects visualization & navigation system for music by detecting changes of emotional content," In Proceedings of the Eighth International Conference on Internet and Web Applications and Services (ICIW 2013), pp.129-134, June 23-28, 2013.

[2] Goto, M. and Hirata, K., "Recent studies on music information processing," Acoustical Science and Technology, vol. 25, no. 6, the Acoustical Society of Japan, pp. 419-425, 2004.

[3] Type, R., Wiering, F., and Veltkamp, R.C., "A survey of music information retrieval system," In Prof. of the 6th International Conference on Music Information Retrieval (ISMIR 2005), pp. 153-160, 2005.

[4] Ghias, A., Logan, J., Chamberlin, D., and Smith, B.C., "Query by humming: musical information retrieval in an audio database," In Prof. of the Third ACM International Conference on Multimedia (MM 1995), pp. 231-236, 1995.

[5] Dannenberg, R.B., Birmingham, W.P., Tzanetakis, G., Meek, C., Hu, N., and Pardo, B., "The MUSART testbed for query-by-humming evaluation," In Prof. of 4th International Conference on Music Information Retrieval (ISMIR 2003), pp. 34-48, 2003.

[6] Shifrin, J., Pardo, B., Meek, C., and Birmingham, W., "HMM-based musical query retrieval," In Proc. of the 2nd ACM/IEEE-CS joint conference on digital libraries (JCDL 2002), pp. 295-300, 2002.

[7] Cheng, H.T., Yang, Y.H., Lin, Y.C., Liao, I.B., and Chen, H.H., "Automatic chord recognition for music classification and retrieval," In Proc. of the IEEE International Conference on Multimedia and Expo (ICME2008), pp. 1505-1508, 2008.

[8] Bello, J.P., "Audio-based cover song retrieval using approximate chord sequences: testing shifts, gaps, swaps and beats," In Prof. of the 8th International Conference on Music Information Retrieval (ISMIR 2007), pp. 239-244, 2007.

[9] Cooper, M., Foote, J., Pampalk, E., Tzanetakis, G., "Visualization in audio-based music information retrieval," Computer Music Journal, Vol. 30, No. 2, pp. 42-62, MIT Press, 2006.

[10] Pampalk, E. and Goto, M., "Musicrainbow: A new user interface to discover artists using audio-based similarity and web-based labeling," In Prof. of 7th International Conference on Music Information Retrieval (ISMIR 2006), pp. 367-370, 2006.

[11] Knees, P., Schedl, M., Pohle, T., and Widmer, G., "An innovative three-dimensional user interface for exploring music collections enriched with meta-information from the web," In Proc. of the 14th ACM International Conference on Multimedia (MM 2006), pp. 17-24, 2006.

[12] Stober, S. and Nürnberger, A., "MusicGalaxy: a multi-focus zoomable interface for multi-facet exploration of music collections," In Proc. of the 7th International Symposium on Computer Music Modeling and Retrieval (CMMR 2010), pp. 259-272, Springer, 2010.

[13] Dittmar, C., Großmann, H., Cano, E., Grollmisch, S., Lukashevich, H., and Abeßer, J., "Songs2See and GlobalMusic2One: two applied research projects in music information retrieval at Fraunhofer IDMT," In Proc. of the 7th International Symposium on Computer Music Modeling and Retrieval (CMMR 2010), pp. 259-272, Springer, 2010.

[14] Hijikata, Y., Iwahama, K., and Nishida, S., "Content-based music filtering system with editable user profile," In Proc. of the 2006 ACM Symposium on Applied Computing (SAC 2006), pp. 1050-1057, 2006.

[15] Goussevskaia, O., Kuhn, M., and Wattenhofer, R., "Exploring music collections on mobile devices," In Proc. of the 10th International Conference on Human Computer Interaction with Mobile Devices and Services (MobileHCI 2008), pp. 359-362, 2008.

[16] Gómez, E. and Bonada, J., "Tonality visualization of polyphonic audio," International Computer Music Conference 2005, MPublishing, University of Michigan Library, 2005.

[17] Mardirossian, A. and Chew, E., "Visualizing music: tonal progressions and distributions," In Prof. of the 8th International Conference on Music Information Retrieval (ISMIR2007), pp. 189-194, 2007.

[18] Ciuha, P., Klemenc, B., and Solina, F., "Visualization of concurrent tones in music with colours," In Proc. of the 18th International Conference on Multimedia (MM 2010), pp. 1677-1680, ACM, 2010.

[19] Imai, S., Kurabayashi, S., and Kiyoki, Y., "A music database system with content analysis and visualization mechanisms," In Proc. of the IASTED International Symposium on Distributed and Intelligent Multimedia Systems (DIMS 2008), pp. 455-460, 2008.

[20] Krumhansl, C.L., "Cognitive foundations of musical pitch," Oxford University Press, 1990.

[21] Temperley, D., "The cognition of basic musical structures," MIT Press, ISBN-13: 978-0-262-70105-1, 2001.

[22] Temperley, D. "Music and probability," MIT Press, ISBN-13:978-0-262-20166-7, 2007.

[23] Kurabayashi, S. and Kiyoki, Y., "MediaMatrix: a video stream retrieval system with mechanisms for mining contexts of query examples," In Proc. of the 15th International Conference on Database Systems for Advanced Applications (DASFAA2010), pp. 452-455, Springer, 2010.

[24] Kurabayashi, S. and Kiyoki, Y., "Impression-aware video stream retrieval system with temporal color-sentiment analysis and visualization," In Proc. of the 23rd International Conference on Database and Expert Systems Applications (DEXA 2012), pp.168-182, Springer, 2012.

Rethinking Traditional Web Interaction: Theory and Implementation

Vincent Balat
Univ Paris Diderot – Sorbonne Paris Cité – PPS, UMR 7126 CNRS, Inria – Paris, France
Email: vincent.balat @ univ-paris-diderot.fr

***Abstract*—In recent years, Web sites evolved into ever more complex distributed applications. But current Web programming tools are not fully adapted to this evolution, and force programmers to worry about too many inessential details. We want to define an alternative programming style better fitted to that kind of applications. To do that, we propose an analysis of Web interaction in order to break it down into very elementary notions, based on semantic criteria instead of technological ones. This allows defining a common vernacular language to describe the concepts of current Web programming tools, but also some other new concepts. We propose to use these new concepts to create new frameworks for programming Web applications. This results in a significant gain of expressiveness. The understanding and separation of these notions also makes it possible to get strong static guarantees, that can help a lot during the development of complex applications, for example by making impossible the creation of broken links. We show how most of the ideas we propose have been implemented in the Ocsigen Web programming framework. Ocsigen makes possible to write a client-server Web applications as a single program and the interaction model we propose is fully compatible with this kind of applications.**

Keywords–Typing; Web interaction; Functional Web programming; Continuations

I. INTRODUCTION

Nowadays, Web sites behave more and more like real applications, with a high-level of interactivity on both the server and client sides. For this reason, they deserve well-designed programming tools, with features like high-level code structuring and static typing. These tools must take into account the specificities of that kind of application. One of these specificities is the division of the interface into *pages*, connected to each other by links. These pages are usually associated to *URLs*, which one can bookmark. It is also possible to turn back to one page using the *back button*. This makes the dynamics of the interface completely different from a regular application. Another specificity is that this kind of applications is highly dependent on standards as they will be executed on various platforms.

Web programming covers a wide range of fields, from database to networking. The ambition of this paper is not to address them all, nor to deal with the full generality of *service oriented computing*. We concentrate on what we will call *Web interaction*; that is, the interaction between a user and a Web application, through a browser interface [1]. Web applications communicate with one or several servers, and sometimes need the ability to make part of the computation in the browser. A few similar Web interaction systems have already been described before (for example Links [2], or Hop [3]). The goal of this paper is to improve the way we program Web interaction using one feature that has been very rarely addressed before, namely: *service identification*. That is: how the service to handle a request is chosen. We will show that a good service identification mechanism can help programmers a lot, and that the concepts we present here allow to take into account very concrete needs of Web developpers that are usually not addressed by more theoretical works.

Through an analysis of existing programming frameworks and Web sites, we will give a semantic description of the diverse behaviours we want to have. This language will lead to the definition of a vernacular language for describing Web interaction. This will allow to understand the concepts and to propose new ones to increase the expressiveness of Web frameworks.

Our second goal is the safety of programming. By using well defined concepts and static checking, it is possible to eliminate a lot of programmer's errors and make the application much more reliable thus simplifying by a lot the maintenance work and evolutions of the site.

The concepts we present here have been implemented in the Ocsigen Web programming framework [4], [5], [6], [7] (Eliom project). It allows to program fully in OCaml both the server and client parts of a Web application, with a consistent abstraction of concepts. A compiler to Javascript is used to run the client parts in the browser [8], [9].

A. A common vernacular language for Web interaction

Web development is highly constrained by technologies. First, it relies on the HTTP protocol, which is non-connected and (mainly) stateless (on the applicative layer – even if many solutions have been proposed to circumvent this). Then, Web applications must be executable in various browsers that implement more or less accurately common standards and recommendations.

One of our goals is to remove these constraints and focus on the programming of Web interaction. Obviously, this is also a question of taste. But rather than proposing yet another programming model from scratch, we start by analyzing common Web programming practices, in order to understand the notions they use. Then we decompose them in very elementary

Work partially supported by the French national research agency (ANR), PWD project, grant ANR-09-EMER-009-01, and performed at the IRILL center for Free Software Research and Innovation in Paris, France

notions that can be used to describe the features of Web programming tools, from PHP to JSP or Miscrosoft.NET Web Forms, etc. Some frameworks impose some artificial restrictions, like the shape of URLs. Ideally, we would like to give a generic language, flexible enough to describe all possible behaviours, without imposing any artificial restriction due to one technology.

We will then see how to implement this with current technology, when possible, and how it would be interesting to make technologies evolve.

We place ourselves at a semantic level rather than at a technical one. Moving away from technical details will allow to increase the *expressiveness* of Web programming frameworks. In the domain of programming languages, high-level concepts have been introduced over the years, for example genericity, inductive types, late binding, closures. They make easier the implementation of some complex behaviours. We want to do the same for the Web. For example the notion of *"sending a cookie"* benefits from being abstracted to a more semantic notion like *"opening a session"* (which is already often the case today). Also it is not really important for the programmer to know how URLs are formed. What matters is the service we want to speak about (and optionally the parameters we want to send it).

This abstraction from technology allows two things:

- First, it increases the expressiveness of the language by introducing specific concepts closer to the behaviours we want to describe (and irrespective of the way they are implemented). From a practical point of view, this allows to implement complex behaviours in very few lines of code.
- Having well-designed dedicated concepts also allows to avoid wrong behaviours. We forbid unsafe technical possibilities either by making them inexpressible, or by static checking.

B. Improving the reliability of Web applications

As Web sites are currently evolving very quickly into complex distributed applications, the use of strongly and statically typed programming languages for the Web becomes more and more helpful. Using scripting languages was acceptable when there was very little dynamic behaviour in Web pages, but current Web sites written with such languages are proving to be very difficult to evolve and maintain. Some frameworks are counterbalancing their weaknesses by doing a lot of automatic code generation (for example [10]). In the current state of knowledge, we are able to do much better, and Web programming must benefit from this.

Static validation of pages: One example where static typing revolutionizes Web programming concerns the validation of pages. Respecting W3C recommendations is the best way to ensure portability and accessibility of Web sites. The novelty is that there now exist typing systems sophisticated enough to statically ensure a page's validity [11], [12], [13]. Whereas the usual practice is to check the validity of pages once generated, such typing systems make it possible to be sure that the program that builds the XML data will always generate something valid, even in the most particular cases.

For example, even if a programmer has checked all the pages of his site in a validator, is he sure that the HTML table he creates dynamically will never be empty (which is forbidden)? What if for some reason there is no data? He must be very conscientious to think about all these cases. It is most likely that the evolutions of the program will break the validity of pages. In most cases, problems are discovered much later, by users.

In lots of cases, such errors will even make the generated output unusable, for example for XML data intended to be processed automatically. The best means to be sure that this situation will never happen is to use a typing system that will prevent one from putting the service on-line if there is the slightest risk for something wrong to be generated.

For people not accustomed to such strong typing systems, this may seem to impose too much of a constraint to programmers. Indeed, it increases a bit the initial implementation time (by forcing to take into account all cases). But it also saves such a huge amount of debugging time, that the use of such typing systems really deserves to be generalized. For now, these typing systems for XML are used in very few cases of Web services, and we are not aware of any major Web programming framework. Our experience shows that it is not difficult to use once one get used to the main rules of HTML grammar, if error messages are clear enough.

Validity of Web interaction: Static checking and abstraction of concepts can also benefit in many other ways to Web programming, and especially to Web interaction. Here are a few examples:

- In a link, do the types (and names) of parameters match the types expected by the service it points to?
- Does a form match the service it points to?
- Do we have broken links?

It is not so difficult to have these guarantees, even if almost no Web programming framework are doing so now. All what is needed is a programming language *expressive* enough (in the sense we explained above).

Improving the ergonomics of Web sites: Lots of Web developers are doing implementation errors resulting in reduced ease of use (wrong use of sessions or GET and POST parameters, etc.). Take as example a famous real estate Web site that allows to browse through the results of a search; but if someone sets a bookmark on one of the result pages, he never goes back to the same page, because the URL does not refer to the advertisement itself, but to the rank in the search. We will see that a good understanding of concepts can avoid such common errors.

C. Overview of the paper

Sections II and III are devoted to the definition of our vernacular language for describing the services provided by a Web application. Section II explains the advantage of using an abstract notion of service instead of old-fashioned page-based programming and string URLs. Section III presents a new service identification and selection method. It shows how

powerful this notion of service can be made, by separating it into several kinds. This results in a very new programming style for Web interaction.

Section IV explains how to ensure correct use of these services using static typing. Then, Section V shows how these notions of services interact with sessions. Finally, some hints on the implementations of these concepts are given in Section VI.

II. Abstracting services

As explained above, our goal is to formalize Web interaction, that is, the behaviour of a Web application in reaction to the actions of the user. What happens when somebody clicks on a link or submits a form? A click often means that the user is requesting a new document: for example a new page that will replace the current one (or one part of it). But it can also cause some actions to take place on the server or the client. Let us enumerate the different kinds of reactions. A click (or a key strike) from the user may have the following main effects:

1) Modifying the application interface. That is, changing the page displayed by the browser (or one part of the page), or opening a new window or tab with a new page,
2) Changing the URL displayed by the browser (protocol, server name, path, parameters, etc.),
3) Doing some other action, like the modification of a state (for example changing some database values),
4) Sending hidden data (like form data, or files),
5) Getting some data to be saved on the user's hard disk.

Two important things to notice are that each of these items is optional, and may either involve a distant server, or be processed locally (by the browser).

This decomposition is important, as a formalization of Web interaction should not omit any of these items in order not to restrict the freedom of the programmer. All these items are described semantically, not technically.

A. The role of URLs

The item "Changing the URL" above is a really significant one and is one key to understand the behaviour of Web applications. This section is devoted to the understanding of that notion. URLs are entry points to the Web site. Changing the URL semantically means: giving the possibility to the user to turn back to this point of interaction later, for example through bookmarks.

Note that, unlike many Web sites, a good practice is to keep the URL as readable as possible, because it is an information visible to users that may be typed manually.

1) Forgetting technical details about URLs: The syntax of URLs is described by the Internet standard STD 66 and RFC 3986 and is summarized (a bit simplified) here:

`scheme://user:pwd@host:port/path?query#fragment`

The path traditionally describes a file in the tree structure of a file system. But this view is too restrictive. Actually, the path describes the hierarchical part of the URL. This is a way to divide a Web site into several sections and subsections.

The query string syntax is commonly organized as a sequence of '*key=value*' pairs separated by a semicolon or an ampersand, e.g., `key1=value1&key2=value2&key3=value3`. This is the part of the URL that is not hierarchical.

To a first approximation, the path corresponds to the service to be executed, and the query to parameters for this service. But Web frameworks are sometimes taking a part of the path as parameters. On the contrary, part of the query, or even of the host, may be used to determine the service to call. This will be discussed later in more detail.

The *fragment* part of the URL only concerns the browser and is not sent to the server.

The item "Changing the URL" is then to be decomposed semantically into these sub-tasks:

1) Changing the protocol to use,
2) Changing the server (and port) to which the request must be made,
3) Choosing a hierarchical position (path) in the Web site structure, and specifying non hierarchical information (query) about the page,
4) And optionally: telling who the user is (credentials) and the fragment of the page he wants to display.

2) URL change and service calls: There are two methods to send form data using a browser: either in the URL (GET method) or in the body of the HTTP request (POST method). Even if they are technical variants of the same concept (a function call), their semantics are very different with respect to Web interaction. Having parameters in the URL allows to turn back to the same document later, whereas putting them in the request allows to send one-shot data to a service (for example because they will cause an action to occur).

We propose to focus on this semantical difference rather than on the way it is implemented. Instead of speaking about POST or GET parameters, we prefer the orthogonal notions of *service calls* and *URL change*. It is particularly important to forget the technical details if we want to keep the symmetry between server and client side services. Calling a local (javascript for example) function is similar to sending POST data to a server, if it does not explicitly change the URL displayed by the browser.

Semantically speaking, in modern Web programming tools, changing the URL has no relation with *calling a service*. It is possible to call a service without changing the URL (because it is a local service, or because the call uses POST parameters). On the contrary, changing the URL may be done without calling a service. There is only one reason to change the URL: give the user a new entry point to the Web site, to which he can come back when he wants to ask the same service once again, for example by saving it in a bookmark.

To keep the full generality that is necessary for programming, it is important to understand that the notions of URL change and service call are completely disconnected. It is possible to change the URL by a call to a javascript DOM function, without calling a service. This changes the entry point of the Web site (just the page one will turn back to if a bookmark is registered). On the contrary, it is possible to all a service without changing the URL, either a distant service

through POST parameters, or a local service through a local function call.

B. Services as first class values

The main principle on which is based our work is:

1) *consider services as first class values,*

exactly as functional languages consider functions as first class values. That is: we want to manipulate services as abstract data (that can for example be given as parameter to a function). This has several advantages, among which:

- The programmer does not need to build the syntax of URLs himself. He can rely on some function that will create the syntax of the URL (if it is a distant call) or the function call if it is a client side call. Thus, it is really easy to switch between a local service and a distant one.
- All the information about the service is taken automatically from the data structure representing the service, including the path to which the service is attached and parameter names. This has a very great consequence: if the programmer changes the URL of a service, even the name of one of its parameters, he does not need to change any link or form towards this service, as they are all built automatically. This means that links will never be broken, and parameter names will always be correct (at least for internal services, i.e., services belonging to the Web site, and modulo to some restrictions in current implementation, as we will see later).

Some recent frameworks already have an abstraction of the notion of service. We will show how to take the full benefit of it. Our notion of service must be powerful enough to take into account all the possibilities described above, but without relying on their technical implementation.

A service is some function taking parameters and returning some data, with possibly some side effects (remote function calls). The server is a provider of *services*. Client side function calls can also be seen as calls to certain services. The place where services take place is not so significant. This allows to consider a Web site with two versions of some services, one on server side, the other on client side, depending on the availability of some resources (network connection, or browser plug-ins for example).

The model we strongly advocate for building Web applications is a compiled language with static type checking and dynamic generation of URLs from abstract services. The language must provide some way to define these services, either using a specific keyword or just through a function call.

Once we have this notion, we can completely forget the old "page-based" view of the Web where one URL was supposed to be associated to one file on the hard disk. Thus, it is possible to gain a lot of freedom in the organization and modularity of the code, and also, as we will see later, in the way services are associated to URLs. One of the goals of next section is precisely to discuss service identification and selection, that is, how services are chosen by the server from the hierarchical and non-hierarchical parts of the URL, and hidden parameters.

III. A TAXONOMY OF SERVICES

A. Values returned by services

A first classification of services may be made according to the results they send. In almost all Web programming tools, services send HTML data, written as a string of characters. But as we have seen before, it is much more interesting to build the output as a tree to enable static type checking. To keep full generality, we will consider that a service constructs a result of any type, that is then sent, possibly after some kind of serialization, to the browser which requested it. It is important to give to the programmer the choice of the kind of service he wants.

A reflection on return types of services will provide once again a gain of expressiveness. Besides plain text or typed HTML trees, a service may create for example a redirection. One can also consider using a service to send a file. It is also important to give the possibility to services to choose themselves what they want to send. For example, some service may send a file if it exists, or an HTML page with an error message on the other case. The document is sent together with its *content type*, telling the browser how to display it (it is a dynamic type). But in other cases, for example when a service implements a function to be called from the client side part of the program, one probably want the type of the result to be known statically.

We also introduce a new kind of output called *actions*. Basically sending an action means "no output at all". But the service may perform some action as side effect, like modifying a database, or connecting a user (opening a new session). From a technical point of view, actions implemented server side are usually sending a 204 (No content) HTTP status code. Client side actions are just procedures. We will see some examples of use of actions and how to refine the concept in Section III-D2.

B. Dynamic services, or continuation-based Web programming

1) Dynamic services: Modern Web frameworks propose various solutions to get rid of the lack of flexibility induced by a one-to-one mapping between one URL and one service. But very few take the full benefit of this, as most of them do not allow to dynamically create new services.

For example, if we want to add a feature to a Web site, or even if we occasionally want to create a service depending on previous interaction with one user. For example, if one user wants to book a plane ticket, the system will look in a database for available planes and dynamically create the services corresponding to booking each of them. Then it displays the list of tickets, with, on each of them, a link towards one of these dynamic services. Thus, we will be sure that the user will book the ticket he expects, even if he duplicates his browser window or uses the back button. This behaviour is really simple to implement with dynamic services and rather tricky with traditional Web programming. Witness the huge number of Web sites which do not implement this correctly.

If we want to implement such behaviour without dynamic services, we will need to save somewhere all the data the service depends on. One possibility is to put all this data in the

link, as parameters, or in hidden form data. Another possibility, for example if the amount of data is prohibitive, is to save it on the server (for example in a database table) and send only the key in the link.

With dynamic service creation, all this contextual data is recorded automatically in the *environment* of the closure implementing the service. This closure is created dynamically according to some dynamic data (recording the past of the interaction with the user). It requires a functional language to be implemented easily.

2) Continuations: This feature is equivalent to what is known as *continuation-based Web programming*. This technique was first described independently by Christian Queinnec [14], [15], [16], John Hughes [17] and Paul Graham [18].

There are basically two ways to implement dynamic services. The first (described as continuation-based Web programming) consists of viewing the sending of a Web page by the server as a function call, (a question asked of the user) that will return for example the values of a form or a link pressed. The problem is that we never know in advance to which question the user is answering (because he may have pressed the back button of his browser or he may have duplicated the page). Christian Queinnec solves this problem by using the Scheme control operator `call/cc` that allows to name the current point of execution of the program (called *continuation*) and to go back to this continuation when needed.

The second solution is the one we propose. It is symmetric to the first one, as it consists in viewing a click on a link or a form as a remote function call. Each link or form of the page corresponds to a continuation, and the user chooses the continuation he wants by clicking on the page. This corresponds to a *Continuation Passing programming Style* (CPS), and has the advantage that it no longer needs control operators (no saving of the stack is required). Strangely, this style of programming, usually considered unnatural, is closer to what we are used to doing in traditional Web programming.

The use of dynamic services is a huge step in the understanding of Web interaction, and an huge gain of expressiveness. Until now, very few tools have used these ideas. None of the most widely used Web programming frameworks implement them, but they are used for example in Seaside [19], PLT Scheme [20], Hop [3], Links [2], and obviously Ocsigen.

3) Implementation of dynamic services: The implementation of dynamic services (in CPS) is usually done by registering closures in a table on the server. It associates to an automatically generated key a function and its environment, which contains all the data needed by the service. The cost (in terms of memory or disk space consumption) is about the same as with usual Web programming: no copy of the stack, one instance of the data, plus one pointer to the code of the function.

An alternative approach [2], [21], [22] is to serialize the closures and send them to the client. Either we serialize the environment and a pointer to the function, or even the environment and the full code of the function. This has the advantage that no space is required on the server to save the closures. But serializing functions is not easy and this solution may require sending large amounts of data to the client, with potentially several copies of some state information. Obviously, security issues must be considered, as the server is executing code sent by the client.

C. Finding the right service

The very few experimental frameworks which are proposing some kind of dynamic services impose usually too much rigidity in the way they handle URLs. This section is devoted to showing how it is possible to define a notion of service identification that keeps all the possibilities described in Section II.

The important thing to take care of is: how to do the association between a request and a service? For example if the service is associated to an URL, where, in this URL, is the service to be called encoded?

To make this as powerful as possible, we propose to delegate to the server the task of decoding and verifying parameters, which is traditionally done by the service itself. This has the obvious advantage of reducing a lot the work of the service programmer. Another benefit is that the choice of the service to be called can depend on parameters.

Let us first speak about distant (server side) bookmarkable services, i.e., services called by sending a GET request to a server. We will speak later about client side services, and hidden services.

1) Hierarchical services: One obvious way to associate a service to an URL is by looking at the path (or one part of it). We will call these services *hierarchical services*. These kinds of services are usually the main entry points of a Web site. They may take parameters, in the *query* part of the URL, or in the path. One way to distinguish between several hierarchical services registered on the same path is to look at parameters. For example the first registered service whose expected parameters exactly match the URL will answer.

2) Coservices: Most of the time one probably wants dynamic services to share their path with a hierarchical service, at least those which last for only a short time (result of a search for example). Also one may want two services to share the same hierarchical position on the Web site.

We will call *coservices* services that are not directly associated to a path, but to a special parameter. This is one of the main original features of our service identification mechanism and this has a huge impact on expressiveness, as we will see on example in Section III-D2. From a semantic point of view, the difference is that hierarchical services are the entry points of the site. They must last forever, whereas coservices may have a timeout, and one probably want to use the associated main service as fallback when the coservice has expired.

We will distinguish between *named coservices* and *anonymous coservices*, the difference being the value of the special parameter. Named coservices have a fixed parameter value

(the name of the coservice), whereas this value is generated automatically for anonymous coservice.

Like all other services, coservices may take parameters, that will be added to the URL. There must be a way to distinguish between parameters for this coservice and parameters of the original service. This can be done by adding automatically a prefix to coservice parameters.

3) Attached and non-attached coservices: We will also distinguish between coservices *attached* to a path and *non-attached* coservices. The key for finding an attached coservice is the path completed by a special parameter, whereas non-attached coservices are associated to a parameter, whatever the path in the URL. This feature is not so common and we will see in Section III-D2 how powerful it is.

4) Distant hidden services: A distant service is said to be *hidden* when it depends on POST data sent by the browser. If the user comes back later, for example after having made a bookmark, it will not answer again, but another service, not hidden, will take charge of the request. We will speak about *bookmarkable* services, for services that are not hidden.

Hidden services may induce an URL change. Actually, we can make exactly the same distinction as for bookmarkable services: there are hierarchical hidden services (attached to a path), hidden attached coservices (attached to a path, and a special POST parameter), and hidden non-attached coservices (called by a special POST parameter).

It is important to allow the creation of hidden hierarchical services or coservices only if there is a bookmarkable (co)service registered at the same path. This service will act as a fallback when the user comes back to the URL without POST parameters. This is done by specifying the fallback instead of the path when creating a hidden service. It is a good idea to do the same for bookmarkable coservices.

Registering a hidden service on top of a bookmarkable service with parameters allows to have both GET and POST parameters for the same service. But bear in mind that their roles are very different.

5) Client side services: Client side service calls have the same status as hidden service calls. In a framework that allows to program both the server and client sides using the same language, we would like to see local function calls as non-attached (hidden) coservices. Hierarchical hidden services and (hidden) attached coservices correspond to local functions that would change the URL, without making any request to the server.

6) Non-localized parameters: One additional notion that is interesting in concrete cases is to enable parameters that are not related to any service at all. The server does not take into account their presence to choose the service, and services do not have to declare them, but can access them if they want. This avoids declaring the same optional parameters for each service when you want the whole Web site to be parametrized by the same optional parameters (for example the language the user prefers to display the pages).

D. Taxonomy of services

1) Summary of service kinds: Figure 1 summarizes the full taxonomy of services we propose. This set is obviously *complete* with respect to technical possibilities (as traditional services are part of the table). It is powerful enough for describing in very few lines of code lots of features we want for Web sites, and does not induce any limitations with respect to the needs of Web developers. Current Web programming frameworks usually implement a small subset of these possibilities. For example "page-based" Web programming (like PHP or CGI scripts) does not allow for non-attached coservices at all. Even among "non-page-based" tools, very few allow for dynamic (anonymous) coservice creation. To our knowledge, none (but Ocsigen) is implementing actions on non-attached services as primary notions (even if all the notions can obviously be simulated).

2) Example cases: We have already seen some examples of dynamic service creation: if a user creates a blog in a subdirectory of his or her personal site, one possibility is to add dynamically a hierarchical service to the right path (and it must be recreated every time the serser is relaunched). If we want to display the result of a search, for example plane ticket booking, we will create dynamically a new anonymous coservice (hidden or not), probably with a timeout. Without dynamic services, we would need to save manually the search keyword or the result list in a table.

Coservices are not always dynamic. Suppose we want a link towards the main page of the site, that will close the session. We will use a named hidden attached coservice (named, so that the coservice key is always the same).

We will now give an example where non-attached hidden coservices allow to reduce significantly the number of lines of code. Consider a site with several pages. Each page has a connected version and a non-connected version, and we want a connection box on each non-connected page. But we do not want the connection box to change the URL. We just want to log in and stay on the same URL, in connected version. Without non-attached services (and thus with almost all Web programming tools), we need to create a version with POST parameters of each of our hierarchical services to take into account the fact that each URL may be called with user credentials as POST parameters.

Using our set of services, we just need to define only one non-attached (hidden) coservice for the connection. At first sight, that service only performs an action (as defined in Section III-A): saving user information in a session table. But we probably want to return a new page (connected version of the same page). This can be done easily by returning a redirection to the same URL. Another solution if we do not want to pay the cost of a redirection, is to define a new kind of output: *"action with redisplay"* that will perform the action, then make an internal (server side) request as if the browser had done the redirection. The solution with redirection has one advantage: the browser would not try to resend POST data if the user reloads the page.

Now say for example that we want to implement a wiki, where each editable box may occur on several pages. Clicking on the *edit* button goes to a page with an edit form, and submitting the form must turn back to the original page. One dirty solution would be to send the original URL as hidden parameter in the edit link. But there is now a simpler solution:

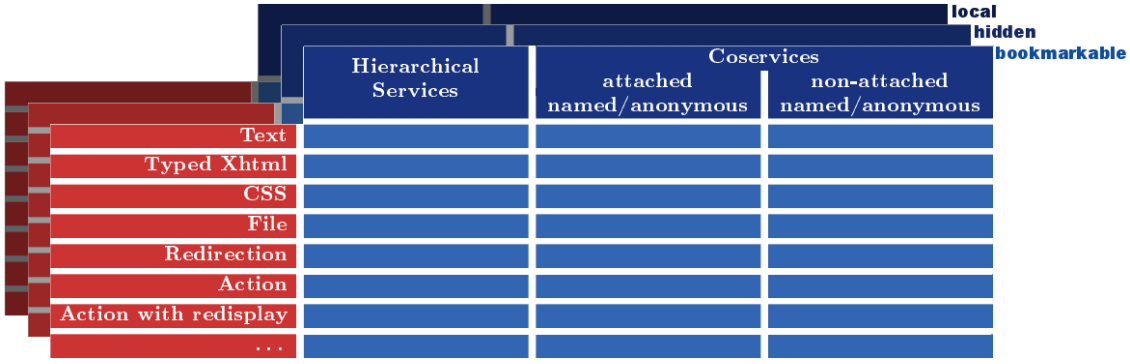


Figure 1: Full taxonomy of services.

just do not change the path. The edit form is just a page registered on a non-attached service.

Our reflexion on services, also allows to express clearly a solution to the real estate site described in Section I-B. Use (for example) one bookmarkable hierarchical service for displaying one piece of advertisement, with additional (hidden or not) parameters to recall the information about the search.

3) Expressiveness: The understanding of these notions and their division into very elementary ones induces a significant gain in expressiveness. This is particularly true for actions with redisplay. They are very particular service return values and seem to be closely related to non-attached coservices at first sight. But the separation of these two concepts introduces a new symmetry to the table, with new cells corresponding to very useful possibilities (see the example of the wiki above). It is noteworthy that all cells introduced in this table have shown to be useful in concrete cases.

4) Priority rules: One may wonder what succeeds when one gets apparently conflicting information from the request, for example both a hidden coservice name and a URL coservice name. In most cases, the solution is obvious. First, hidden services always have priority over URL services.

Secondly, as non-attached coservices parameters are added to the URL, there may be both attached and non-attached coservice parameters in the URL. In that case, the non-attached service must be applied (because its parameters have necessarily been added later).

The last ambiguous case is when we want to use a hidden non-attached coservice when we already have non-attached parameters in the URL. We may want to keep them or not. The programmer must choose the behaviour he wants himself when defining the service.

IV. Typing Web interaction

A. Typing service parameters

Another advantage to our way of building Web sites is that it becomes possible to perform more static checking on the site and thus avoid lots of mistakes. We have already seen that some very strong guarantees (no broken links) are ensured just by the abstraction of the notion of services and links. But static types can help us to make things even safer.

When defining a service, it is easy to add type information to each parameter. This has three advantages:

- The server will be able to dynamically convert the data it receives into the type expected by the service (and also check that the data corresponds to what is expected, which saves a lot of time while writing the service).
- It is possible to check statically the types of parameters given to the links we create towards the services.
- It is also possible to do some static verification of forms (see next section).

It should also be possible to use type inference to avoid declaring manually the types of parameters.

On first sight, the type system for services parameters seems rather poor. But declaring basic types like string, int and float is not enough. We need more complex types. Just think about variable length forms, for example a page displaying a list of people with just a checkbox for each of them. The implementation of this is a tedious work with traditional Web programming tools, because you need to give different names to each parameter (for example by using numbers) to be able to find back on server side the person associated to the checkbox. Obviously, you do not want to worry about such details when programming a Web application. Your service just wants to get an association table from a name to a boolean value, however they are encoded in parameters.

Another example is when you want to send an (unordered) set of values to a service, or optional values.

Unfortunately, most of this is sometimes difficult to implement, due to the way parameters are encoded in the URL, and the way browsers send them. One example of this is the way browsers handle unchecked boxes: they send no parameter at all, instead of sending a *false* value. Thus, there is no way to distinguish between an unchecked box and no parameter at all.

Sets of base types data may be implemented using the same parameter name for each member of the set. But an implementation of sets of more complex data requires more thinking.

In conclusion, the types of parameters for Web pages is highly constrained by current technology.

In the case your Web program has a client side, and you are using the same language for both sides, a solution is to

send directly your language values, after serialisation. This is acceptable and very powerful in the case of hidden (POST) services but will result in unreadable URLs for bookmarkable services.

B. Forms

To be correct relative to the service it leads to, a form must respect these conditions:

- The names of the fields of the form must correspond to the names expected by the service,
- The types of the fields must correspond to the types expected by the service,
- All required fields must be present, and the right number of times.

The first item may be solved by taking the parameter names from the data representing the service in memory. The static verification of the adequacy of types may be done by putting type information in the type of names (instead of using just the *string* type, we use an abstract type with phantom type information [23] on the type of the parameter). The third condition is more difficult to encode in types.

One idea interesting for some particular cases of services is to create dynamically the service that will answer to the form from the form itself (as for example in Seaside, Links [24] or PLT Scheme).

V. Abstracting sessions

Sessions are a way to maintain a state during several interactions with one user. The abstraction of sessions is something more common in usual Web programming tools than the abstraction of services. It is now standard to have a way to save some data for one user and recover it each time the user comes back.

A. Technical remainder

From a technical point of view, sessions are not so easy to implement due to the fact that the HTTP protocol is stateless on the applicative layer (once a request is fulfilled, the connexion is usually closed). There are several ways to get around this limitation.

One possibility is to send all the data to the user. But this does not have exactly the semantics we expect for sessions. Indeed, several browser tabs opened on the same site will have different versions of the state, which is not what we want. The nature of a state being to be unique, it is supposed to be kept in one place, and the only simple solution is to keep it server side. Therefore, there must be one way to identify the user to get back her data.

This is done by asking the browser to send one session identifier at each request. Two techniques are used for that: either you put the session identifier in each link and form of the site, or you ask the browser to send it at each request using cookies. But only the second solution has exactly the semantics we expect for sessions. Using the first solution leads to incompatibilities between tabs (for example if you log in from one tab and return to another tab opened on the same site).

B. Session data and session services

Opening a session on server side may be done transparently when the Web site decides to register some data for one user. A session identifier is then generated automatically and a cookie is set, and sent back by the browser in the header of each request for the site.

This is a common feature in current Web programming tools. But we propose to add the ability to dynamically register coservices or services in a session table, that is, for one user. One obvious reason for this is to allow the use of certain anonymous coservices only for the user who requested them. You probably do not want to give access to these kind of coservices to everybody, especially because they may contain private data in the closure.

We also propose to allow to re-register some existing services in a session table. When the server is receiving a request, it first looks in the session table to see if there is a private version of the service, and if not, looks in the public table. Using this feature, it is possible to save all the session data in the closure of services. From a theoretical point of view, this makes session data tables useless. Basically, when a user logs in you just need to register in the session service table a new version of each service, specialized for this user.

For the sake of flexibility, it is a good idea to allow both session data and session services.

Thus, there is now a new possibility for creating each of our service kinds: registering them in the session table. This corresponds to a fourth dimension in the table of Figure 1.

C. Session duration

When implementing such kinds of session, one must be very careful about the duration of sessions and timeout for services. If you want your sessions to survive a restarting of the server, you need a way to serialize data (for session data) and closures (for session services).

D. Session names and session groups

To make the session system more powerful, we may want to add two more notions, namely *session names* and *session groups*.

Naming sessions allows to use several sessions in the same site. Think for example about one site that allows some features for non connected users, which requires some private coservices to be created. If the user logs in, this opens a new session, but must not close the previous one. To avoid that, we use another session by specifying another session name.

Technically speaking, session naming can be implemented by recording the name of the session in the cookie name.

The idea of session groups is complementary to that of the session name. While session naming allows for a single session to have multiple buckets of data associated with it, session grouping allows multiple sessions to be referenced together. For most uses, the session group is the user name. It allows to implement features like “close all sessions” for one user (even those opened on other browsers), or to limit the number of sessions one user may open at the same time (for security reasons).

As we have session data end session services, it is possible to create notions of group data and group services.

E. Session scopes

In Ocsigen, session data are saved in some kind of (persistent) references whose value depends on the browser that is perfomring the request. This makes very easy to access the data in a way that is very well integrated in language features.

It is possible to make the concept even more powerful by making possible to choose the *scope* of these references:

Session The default scope is *session* and correspond to the usual use of sessions, implemented with regular browser cookies. The data recorded in references with session scope are specific to a browser.

Session group If you create a reference with scope *session group*, the value will be available from all browser sharing the same session group. For example, you can implement a shoping basket just by setting this kind of reference. It will be shared by all your sessions.

Client process Ocsigen makes possible to write both sides of a Web application (server and browser) as a single program. Client side parts are extracted and compiled into Javascript. Among many other things, this makes possible to create server side references with scope *client process*. These reference values are specific to one tab of the browser. Think for example of several instances of the same game running in several tabs. We are also using this to record the communication channels that are specific to one tab. To implement that, we added a cookie mechanism, very similar to the usual one, but at the level of a client process.

Site References with scope *site* have the same value for everyone. They have the same semantics of regular references, but may be persistent across server restarts.

Request References with scope *request* are a simple way to store some data for a single request (that is, for one thread of the server).

VI. Implementation

To implement the features presented above, we had two possible solutions: either we created our own language, which would have given us the full freedom in implementation. Or we needed to choose a language expressive enough to encode most of the features presented here.

We chose the second solution, for two reasons:

- We think that Web programming is not a task for a domain specific language. We need the full power of a general purpose language, because we do not want only to speak about Web interaction and typing of pages, but also for example about database interaction. We also want code structuring and separation, and we need programming environments and libraries. The cost of creating our own new language would have been much too high.
- We knew one language, namely OCaml that has almost all features we want to implement the concepts of this paper, most notably a powerful typing system able to encode most of the properties we wanted to check statically. Moreover, this language now has a strong basis of users, in academia and industry, and a large set of libraries.

We made this choice with the goal in mind to write not only a research prototype but a full framework usable for real applications. We benefited greatly from the free software development model, which enabled us to have a powerful framework very quickly, thanks to the growing community of users.

Our implementation takes the form of a module for the *Ocsigen* [4] Web server, called *Eliom* [25]. More implementation details may be found in [26] (but for an old version of Eliom that was using a more basic model of services).

A. Static type checking of pages

As the return value of services is completely independent of services, it is important to make the creation of new kinds of output types easy. This is realized through the use of OCaml's module language, which allows parametrized modules (called functors). To create a new output module, you apply a functor that will create the registration functions for your output type.

Eliom does not impose one way to write the output, but proposes several predefined modules. One allows text output, as in usual Web programming, if you do not want any type-checking of pages.

Another one is using an extension of OCaml, called OCamlduce [11], that adds XML types. This is really powerful, as the typing is very strict, and corresponds exactly to what you need to take into account all features of DTDs. It also has the advantage of allowing to easily create new output modules for other XML types, just from a DTD. Furthermore, it allows to parse and transform easily incoming XML data. The drawback is that is not part of the standard OCaml compiler.

As an alternative, we are using a second typing method based on OCaml's *polymorphic variants* (see [27]) used as *phantom types* [23].

This technique is not new as we use an implementation due to Thorsten Ohl, which is also distributed as a distinct library. We are aware of another very similar implementation used by Alain Frisch for the Bedouin project [28]. It is less strict than OCamlduce for the validation of pages, as it only checks the correct nesting of XML tags (for example it is not possible to put a `<div>` tag inside a `<em>` tag). There also a way to check the presence of mandatory tags.

See [26] for more information about that technique. Both typing techniques have shown to be very helpful, and relieves the programmer from thinking about validation of her pages.

B. Defining services

1) Creation and registration of services: Creating a new service means two things: first filling a data structure with all

the information about the service (that will be used to create forms and links towards this service), and registering in a table the "handling" function implementing the service.

In Eliom, these two operations have been disconnected. You first create the service, then register a function on it. This may be considered dangerous, as some services may be created and not registered, which can lead to broken links. We were forced to do so because services are usually highly mutually recursive (every page may contain links towards any other one), and it is not easy in OCaml to have mutually recursive data in several different modules.

The solution that is currently implemented consists in a dynamic check of the registration of services, when the server starts. This solves the problem for static services, but not for dynamic ones.

A solution we consider is to use a syntax extension to the language to put back together creation and registration. But the concrete realization of that idea is not straightforward.

2) Types of parameters: One important thing to check statically when registering a function on a service is that the type of this function corresponds to the type expected by the service. This makes the type of the registration function dependent on the value of one of its parameters (the service). This requires very sophisticated type systems. As explained in [26], it can be done either using *functional unparsing* [29] or using *generalized algebraic data types* [30], [31].

Ideally, one would expect to declare page parameters only once when defining both the service and its handling function. As we separate definition and registration, it is not possible to do so. But even without this separation it is probably not possible to avoid this duplication without a syntax extension for the language, for two reasons:

- We need the names of parameters to live both in the world of variable names (static) and in the world of data (dynamic), to be used as parameter names for pages.
- We need dynamic types (that is keeping typing information during the execution) to enable the server to convert received page parameters into the type expected by the handling function.

But in any case, we must keep in mind that the types of service parameters are not OCaml types, and it is difficult to use sophisticated OCaml types for services as mentioned in Section IV-A.

C. Links and forms

We are using functions to create links and forms. Obviously the adequacy of the service to its parameters is checked statically while creating a link.

To implement non-localized parameters, we just needed a way to build a new service data structure by combining together a service and non-localized parameters.

We mentioned in Section IV-B that performing statically all the verification on the form would require a very sophisticated type system, that would be difficult to mix with the already complex typing model we use to check the validity of pages. We decided to restrict the static verifications to the names and types of fields. To do that, we use an abstract type for names, as explained in Section IV-B, and we get parameters names from the service. This is done by giving as parameter to our form building function, a *function* that will build the content of the form from parameters names.

We implemented some sophisticated types for service parameters, like lists or sets. In the case of lists, the names of parameters are generated automatically by an iterator.

VII. Conclusion

A. Related work

A lot of modern Web programming frameworks (for example GWT or Jif/Sif [32]) are trying to propose integrated and high level solutions to make easier the development of Web application. They often provide some abstraction of concepts, but most of them preserve some historical habits related to technical constraints. It is impossible to make a full review of such tools, as there are numerous. We will concentrate on the main novel features presented here. One can try to make a classification of existing Web frameworks with respect to the way they do service identification.

The old method is what we called "page-based Web programming", where one path corresponds to one file. Modern tools are all more flexible and make service identification and selection independent of the physical organization of components in the Web server (for example JSP assigns an URL to a service from a configuration file). But very few belong to the third group, that allows dynamic services. Among them: Seaside [19], Links [2] and Hop [3], Wash/CGI [33]. Their service identification models are more basic, and they do not have a native notion of coservice. Some of them are using an abstraction of forms [33], [24] that is fully compatible with our model.

There have been few attempts to formalize Web interaction. The most closely related work is by Paul T. Graunke, Robert Bruce Findler, Shriram Krishnamurthi and Matthias Felleisen [34], [35]. Their work is more formal but does not take into account all the practical cases we speak about. In particular their service model is much simpler and does not fully take into account the significance of URLs. Peter Thiemann [33] uses monads to create HTML pages, which makes possible an original and interesting way of handling the typing of forms, using Haskell's type system.

We think our approach is compatible with more data driven approaches [10], component based interfaces [36], or even code generation techniques. One interesting work would be to see how they can be mixed together.

B. Evolution of technologies and standards

This reflection about Web programming techniques has shown that Web technologies suffer from some limitations that slow down the evolution towards really dynamic applications. Here are a few examples:

- As mentioned above, the format of page parameters and the way browsers send them from form data does not allow for sophisticated parameters types.
- (X)HTML forms cannot mix GET and POST methods. It is possible to send URLs parameters in the `action`

attribute of a form that is using the POST method, but it is not possible to take them from the form itself. This would open many new possibilities.
- A link from HTTP towards the same site in HTTPS is always absolute. This breaks the discipline we have to use only relative links (for example to behave correctly behind a reverse proxy). We have the same problem with redirections, which have to be absolute URLs according to the protocol.
- There is no means to send POST data through a link, and it is difficult to disguise a form into a link. Links and forms should probably be unified into one notion, that would allow to make a (POST or GET) request from a click on any part of the page. This limitation is not really significant if we have a client side program that does the requests itself when we click on a page element.
- Having the ability to put several `id` attributes for one tag would be very useful for automatically generated dynamic pages.
- Probably one of the main barriers to the evolution of the Web today is the impossibility to run fast code on the browser (without plug-ins), even with recent implementations of Javascript. When thinking about a Web application as a complex application distributed between a server and a client, we would often like to perform computationally intensive parts of the execution on the client, which is not feasible for now. We want to make some experiments with Google Native Client [37].

C. Concluding words and future works

This paper presents a new programming style for Web interaction which simplifies a lot the programming work and reduces the possibilities of semantical errors and bad practices. The principles we advocate are summarized here:

1) Services as first class values
2) Decoding and verification of parameters done by the server
3) Dynamic creation of services
4) Full taxonomy of services for precise service identification
5) Same language on server and client sides
6) Symmetry between local and distant services

One of the main novel feature is the powerful service identification mechanism performed automatically by the server. It introduces the notion of coservice which make the programming of sophisticated Web interaction very easy.

Beyond just presenting a new Web programming model, this paper defines a new vocabulary for describing the behaviour of Web sites, on a semantic basis. It is a first step towards a formalization of Web interaction. We started from an analysis of existing Web sites and we extracted from this observation the underlying concepts, trying to move away as much as possible from non-essential technical details. This allowed a better understanding of the important notions but above all to bring to light some new concepts that were hidden by technical details or historical habits. The main feature that allowed this is the introduction of dynamic services, and also forgetting the traditional page-based Web programming. There exist very few frameworks with these features, and none is going as far as we do, especially in the management of URLs.

Besides the gain in expressiveness, we put the focus on reliability. This is made necessary by the growing complexity of Web applications. The concepts we propose allow for very strong static guarantees, like the absence of broken links. But more static checks can be done, for example the verification of adequacy of links and forms to the service they lead to. These static guarantees have not been developed here because of space limitation. They are summarized by the following additional principles:

7) Static type checking of generated data
8) Static type checking of links and forms

This paper does not present an abstract piece of work: all the concepts we present have been inspired by our experience in programming concrete Web sites, and have been implemented. Please refer to Ocsigen's manual [25] and source code for information about the implementation. Some implementation details may also be found in [26] (describing an old version of Ocsigen that was using a more basic model of services). Ocsigen is now used in industry (for example BeSport [38], Pumgrana [39]). These concrete experiences showed that the programming style we propose is very convenient for Web programmers and reduces a lot the work to be done on Web interaction.

As we have seen, the use of an existing language for the implementation induces some limitations: the typing of forms is not perfect, there is no type inference of service parameters, the need to disconnect creation and registration of services. But despite these limitations, our implementation is very close to the model we presented in the paper and it is a huge step forwards in the implementation of robust Web applications.

This paper is not a full presentation of Ocsigen. Many aspects have been hidden, and especially how we program the client side part of the application [8], [9], [40] using the same language, and with the same strong static guarantees. As we have seen, our notions of services also apply to client side functions. Obviously, we are using the same typing system for services but also for HTML. It is not easy to guarantee that a page will remain valid if it can evolve over time [41]. We did not show how the server can send data to the client at any time, or even call a function on client side. We are currently working on extending our service model to multi-tiers architecture, when more than two pairs are interacting. On a more theoretical point of view, we are working on a formal description of our services.

ACKNOWLEDGMENT

Many acknowledgements are due to Jean-Vincent Loddo, Jérôme Vouillon and all the people who took part in Ocsigen development. I also want to thank Russ Harmer, Boris Yakobowski and Yann Régis-Gianas for their helpful remarks about this paper, and Dario Teixeira for the idea of session groups.

REFERENCES

[1] V. Balat, "Rethinking traditional Web interaction," in International Conference on Internet and Web Applications and Services (ICIW), Jun. 2013, pp. 206–211.

[2] E. Cooper, S. Lindley, P. Wadler, and J. Yallop, "Links: Web programming without tiers," in In 5th International Symposium on Formal Methods for Components and Objects (FMCO). Springer-Verlag, 2006, p. 10.

[3] M. Serrano, E. Gallesio, and F. Loitsch, "Hop, a language for programming the web 2.0," in Dynamic Languages Symposium, Oct. 2006.

[4] "The Ocsigen project," http://www.ocsigen.org [retrieved: 2014-06-16].

[5] V. Balat, J. Vouillon, and B. Yakobowski, "Experience report: ocsigen, a web programming framework," in ICFP '09: Proceedings of the 14th ACM SIGPLAN international conference on Functional programming. Edinburgh, Scotland: ACM, 2009, pp. 311–316.

[6] V. Balat, P. Chambart, and G. Henry, "Client-server Web applications with Ocsigen," in WWW2012 dev track proceedings, Lyon, France, Apr. 2012, p. 59.

[7] V. Balat, "Client-server Web applications widgets," in Proceedings of the 22nd international conference on World Wide Web (WWW 2013 dev track), Rio de Janeiro, Brazil, 2013, pp. 19–22.

[8] B. Canou, E. Chailloux, and J. Vouillon, "How to Run your Favorite Language in Web Browsers," in WWW2012 dev track proceedings, Lyon, France, Apr. 2012, pp. –.

[9] J. Vouillon and V. Balat, "From bytecode to javascript: the js_of_ocaml compiler," Journal of Software: Practice and Experience, 2013.

[10] "Ruby on rails," http://www.rubyonrails.com/ [retrieved: 2014-06-16].

[11] A. Frisch, "Ocaml + xduce," in Proceedings of the international conference on Functional programming (ICFP). ACM, 2006, pp. 192–200.

[12] V. Benzaken, G. Castagna, and A. Frisch, "CDuce: An XML-centric general-purpose language," in Proceedings of the International Conference on Functional Programming (ICFP), 2003, pp. 51–63.

[13] H. Hosoya and B. C. Pierce, "XDuce: A statically typed XML processing language," ACM Transactions on Internet Technology, vol. 3, no. 2, May 2003, pp. 117–148.

[14] C. Queinnec, "The influence of browsers on evaluators or, continuations to program web servers," in International conference on Functional programming (ICFP), 2000

[15] C. Queinnec, "Continuations and web servers," Higher-Order and Symbolic Computation, Dec. 2004.

[16] C. Queinnec, "Inverting back the inversion of control or, continuations versus page-centric programming," ACM SIGPLAN Notices, vol. 38, no. 2, Feb. 2003.

[17] J. Hughes, "Generalising monads to arrows," Science of Computer Programming, vol. 37, no. 1–3, 2000, pp. 67–111.

[18] P. Graham, "Beating the averages" http://www.paulgraham.com/avg.html [retrieved: 2014-06-16].

[19] S. Ducasse, A. Lienhard, and L. Renggli, "Seaside – a multiple control flow web application framework," in Proceedings of ESUG Research Track 2004, 2004.

[20] S. Krishnamurthi, P. W. Hopkins, J. Mccarthy, P. T. Graunke, G. Pettyjohn, and M. Felleisen, "Implementation and use of the plt scheme web server," in Higher-Order and Symbolic Computation, 2007.

[21] J. Matthews, R. B. Findler, P. Graunke, S. Krishnamurthi, and M. Felleisen, "Automatically restructuring programs for the web," Automated Software Engg., vol. 11, no. 4, 2004, pp. 337–364.

[22] J. A. McCarthy, "Automatically restful web applications: marking modular serializable continuations," in ICFP '09: Proceedings of the 14th ACM SIGPLAN international conference on Functional programming. New York, NY, USA: ACM, 2009, pp. 299–310.

[23] D. Leijen and E. Meijer, "Domain specific embedded compilers," in Domain-Specific Languages, 1999, pp. 109–122.

[24] E. Cooper, S. Lindley, P. Wadler, and J. Yallop, "The essence of form abstraction," in Sixth Asian Symposium on Programming Languages and Systems, 2008.

[25] V. Balat, "Eliom programmer's guide," Technical report, Laboratoire PPS, CNRS, université Paris-Diderot, Tech. Rep., 2007. [Online]. Available: http://ocsigen.org/eliom [retrieved: 2014-06-16]

[26] V. Balat, "Ocsigen: Typing web interaction with objective caml," in ML'06: Proceedings of the 2006 ACM SIGPLAN workshop on ML

[27] X. Leroy, D. Doligez, J. Garrigue, D. Rémy, and J. Vouillon, "The objective caml system release 3.10 documentation and user's manual," Inria, Tech. Rep., may 2007.

[28] A. Frisch, "The bedouin project", http://sourceforge.net/projects/bedouin [retrieved: 2014-06-16].

[29] O. Danvy, "Functional unparsing," Journal of Functional Programming, vol. 8, no. 6, 1998, pp. 621–625.

[30] F. Pottier and Y. Régis-Gianas, "Stratified type inference for generalized algebraic data types," in Proceedings of the 33rd ACM Symposium on Principles of Programming Languages (POPL'06), Charleston, South Carolina, Jan. 2006, pp. 232–244.

[31] H. Xi, C. Chen, and G. Chen, "Guarded recursive datatype constructors," in Proceedings of the 30th ACM SIGPLAN Symposium on Principles of Programming Languages, New Orleans, January 2003, pp. 224–235.

[32] S. Chong, J. Liu, A. C. Myers, X. Qi, K. Vikram, L. Zheng, and X. Zheng, "Secure web applications via automatic partitioning," SIGOPS Oper. Syst. Rev., vol. 41, no. 6, 2007, pp. 31–44.

[33] P. Thiemann, "Wash/cgi: Server-side Web scripting with sessions and typed, compositional forms," in Practical Aspects of Declarative Languages (PADL'02), 2002.

[34] P. T. Graunke, R. B. Findler, S. Krishnamurthi, and M. Felleisen, "Modeling Web interactions," in European Symposium on Programming (ESOP), April 2003.

[35] S. Krishnamurthi, R. B. Findler, P. Graunke, and M. Felleisen, "Modeling web interactions and errors," in In Interactive Computation: The New Paradigm. Springer Verlag, 2006.

[36] J. Yu, B. Benatallah, R. Saint-Paul, F. Casati, F. Daniel, and M. Matera, "A framework for rapid integration of presentation components," in WWW '07: Proceedings of the 16th international conference on World Wide Web. New York, NY, USA: ACM, 2007.

[37] "Google native client," http://code.google.com/p/nativeclient/ [retrieved: 2014-06-16].

[38] "Besport," http://www.besport.com/ [retrieved: 2014-06-16].

[39] "Pumgrana," http://www.pumgrana.com/ [retrieved: 2014-06-16].

[40] B. Canou, V. Balat, and E. Chailloux, "O'browser: objective caml on browsers," in ML '08: Proceedings of the 2008 ACM SIGPLAN workshop on ML. New York, NY, USA: ACM, 2008, pp. 69–78.

[41] B. Canou, V. Balat, and E. Chailloux, "A declarative-friendly api for Web document manipulation," in International Symposium on Practical Aspects of Declarative Languages (PADL'13). Springer-Verlag, January 2013.

Enabling Data Collections for Open-Loop Applications in the Internet of Things

Alexander Kröner
Georg Simon Ohm University of Applied Sciences
Nuremberg, Germany
Alexander.Kroener@th-nuernberg.de

Jens Haupert, Matthieu Deru, Simon Bergweiler, Christian Hauck
German Research Center for Artificial Intelligence
Saarbrücken, Germany
{jens.haupert, matthieu.deru, simon.bergweiler, christian.hauck}@dfki.de

***Abstract*—Using label technology, a physical object may be employed to build a continuously growing data collection, which can, for instance, be exploited for product quality monitoring and supply chain management. Along the object's life-cycle, queries to such a collection may stay quite similar, e.g., "get unusual observations". However, expectations to a "good" answer may change, as with time different entities will come into contact with the object. This article reports on work in progress concerning a framework for collecting data about things, which aims at decoupling logic employed for interpreting such a collection from processing hardware and using the collection itself for transporting such logic. Main contributions include an approach to hardware-abstraction of processing logic at the object or remote, an app store for retrieving interpretation and presentation logic, and interaction forms with such memories.**

Keywords-Ubiquitous computing; RFID tags; Distributed information systems; Supply chain management.

I. Introduction

Within the Internet of Things, physical objects may function as a focus for digital data and services concerning the artifact itself as well as associated things, people and processes as presented in [1] at UBICOMM'13. This function enables an object to take a new role as a data collector and provider in a broad range of scenarios, e.g., users may manually associate data with an object in order to socialize and foster discussion in a community [2], tools may automatically collect usage data in support of pay-by-use accounting [3], and products may steer and document their production [4] and transport rules that support reasoning of healthcare applications [5]. Existing applications of such technology are typically deployed in "closed" scenarios, i.e., requirements of users and applications are known before the collection process starts.

This reflects only to some extent a supply chain with continuously changing users and requirements. In order to facilitate communication between stakeholders in such an "open" scenario, a uniform interaction behavior of the collection would be advantageous, e.g., a uniform way to "check integrity" of an object, i.e., compliance to criteria for objects or kinds of objects specified by a third party on an individual base.

In the following, Section II provides an example scenario, where one stakeholder has to employ logics provided by another stakeholder. Section III summarizes requirements that arise from this scenario. Then, Section IV wraps up work accomplished so far concerning so-called Active Digital Object Memories (ADOMe), a framework for processing logic in a way that allows for embracing a broad range of infrastructure approaches common to Internet of Things applications. Section V extends this approach with a concept of an app store supporting distribution of the processing logic. Section VI deals with approaches to support user interaction with access to object memories - by the framework itself as well as by mobile devices and smart objects. Finally, the article concludes with a summary of results and a discussion of future work in Section VII.

II. Scenario

The following logistics scenario deals with integrity control during transportation of a heterogeneous set of goods (see Figure 1). Each good is packaged in a way matching its nature (e.g., fragility) and value. All packages are tagged with some kind of label technology, which allows for automatically identifying the object. Depending on the respective kind of package, this technology may range from passive RFID (Radio Frequency IDentification) to embedded systems with integrated sensing and processing capabilities. At the same time an object memory was created and filled with static product and manufacturer data, as well as criteria to be monitored in the following. Additionally, the memory can contain information what sensor values are interesting to the object and orders how to treat and transport the object.

A retail chain advertises the quality of products sold in its stores, which is subject of the company's own, particular strong quality guideline. In order to leverage compliance to this guideline, the company provides business partners along the supply chain with constraints on parameters that need to be monitored. A supplier uses these parameters in order to configure an integrity test for each package destined for this particular retailer; for accuracy, input parameters should be sensed and processed by the package itself or by IT infrastructure near the object. Performing this configuration task is supported by a hardware-abstraction layer, which allows for assigning tests to packages independent from the kind of label technology provided by the respective package.

During loading a truck (by means of this layer), a dialog between package, truck, and an app store, hosting implementations of tests matching the retailer's parameters, is performed. Result of this dialog is an assignment determining which technical component (truck or package) has to conduct the monitoring task, an assignment, which may differ for each package.

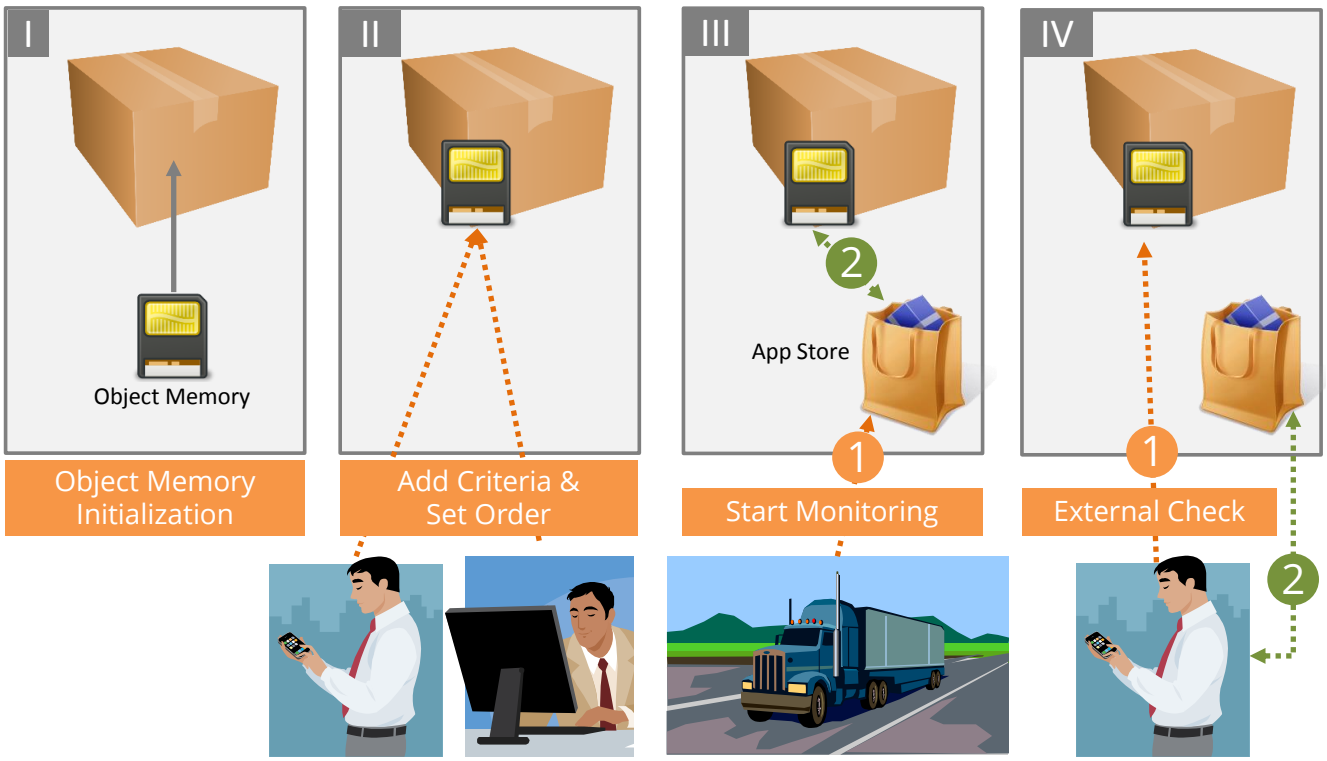


Figure 1. Logistics scenario with memory initialization [I], on-product criteria storage and monitoring orders [II], active logic processing [III], and external checks [IV].

Finally, the packages arrive at the retailer's receiving area. There, an employee uses a mobile device in order to identify the respective object and access its digital records - values sensed during transport as well as testing methods and their results. The access is performed using an app, which downloads from the app store a method suited to visualize the test chosen by the logistics expert. This visual adaptation is performed automatically in the background; even for very different kinds of packages the employee experiences always the same way of interacting with the respective object.

At the retailer's store some products and their new capabilities can be utilized for direct product-to-customer interaction. E.g., a milk carton (called "Milky") instrumented with an embedded controller, sensors and a display is present on a shop shelf (see Figure 3). The milk carton tries to catch the attention of customers with blinking eyes and an acoustic feedback. The proximity sensor and accelerometer sensor detect the fact that the customer is going to take the product from the shelf. Once bought, milky activates a new mode as the product now belongs to the customer. At this moment the sensor data tracking module is activated and all parameters are tracked and stored in the object's memory. In the same manner the customer will be also informed when the best-before day is up. The consumer can also switch between two views, the first one being the face of Milky and the second one showing a more factual view by displaying the sensor's raw data values. Depending on the content of the milk carton a personalization module can be started; strawberry milk could be presented as a pink face on the display. Once the product is no more consumable, e.g., due to an expiry of the best-before day or because the customer has completely used it, Milky goes into the "dead"-state. In this mode three options are displayed on the screen: the first one displays a map with the nearest recycling stations to ensure correct recycling for optimizing the product's carbon footprint. The second option allows the customer to share his "product-experience" over a social network. With the third and last option the product asks the consumer if he would like to buy the same product again.

Extending the mentioned fixed stakeholder chain, a more flexible approach is currently emerging. The so called "open-loop" life-cycle chain is determined by the idea of supporting different successors in each life-cycle step. This approach allows for a flexible "routing" of products by incorporating different stakeholders and delaying the process of choosing which stakeholder is next from design time to runtime (see Figure 2). This approach demands flexible systems that support various hardware platforms, diverse device capabilities and different data content.

Considering such open-loop supply chains, our scenario can be enhanced. Let's assume our known logistics partner equips each individual object with a dedicated embedded system to perform the monitoring tasks. By adding additional providers we also have to support their approach (see Figure 4). E.g., some providers equip the products only with RFID tags with server-based storage and perform the monitoring only on pallet- or container-level, whereas others attach an embedded controller to each product. Such controllers provide storage, processing and sensor capabilities. The complete data set is created and stored locally. And other providers even do not support direct monitoring of individual products during transport at all. Depending on the providers involved, a retailer might receive memories equipped with different hardware

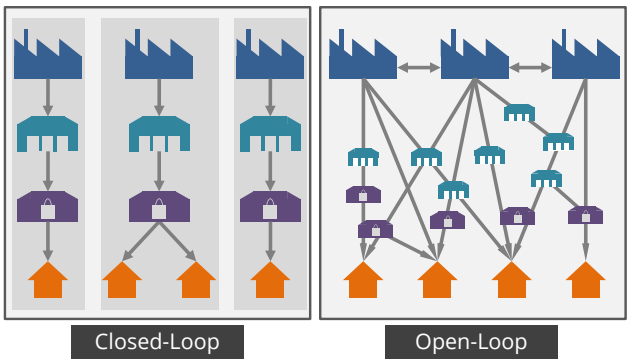


Figure 2. Supply chain: ”closed-loop” (left) and ”open-loop” (right) paradigm.

Figure 3. Smart milk carton ’Milky’ showing object-related recipes (1), in anthropomorphic mode ’happy’ (2), advertising on-product re-buy task (3), and presenting recycling information (4).

platforms and filled with data of different quality and quantity.

Summarizing, the described scenario is characterized by the following key features:

- Varying stakeholders
- Varying capturing technology
- Varying interaction devices
- Varying interaction processes
- Reconfiguration at any time throughout the process

III. Related Work

This work is related to research and development concerning frameworks that leverage collecting and processing data related to physical objects, and as such related to the Internet of Things. Related research comprises embedded systems as well as web-based data stores. So-called Collaborative Business Items (CoBIs) illustrate the benefits of delegating small parts of a well-defined business process - e.g., monitoring and self-assessment tasks - to objects with embedded sensing and processing capabilities [6].

In order to decouple such a service from the employed hardware, SmartProducts [7] seek to dynamically integrate resources - including web-based structures - in the object’s environment into the service realization. Complementary to our proposal, this work puts particular emphasis on semantic device and data descriptions for products with embedded technology.

Other projects also cover innovative solutions for the logistics domain. European projects EURIDICE [8] and iCargo

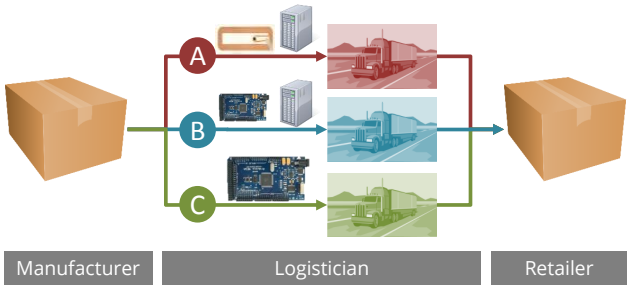


Figure 4. Extended supply chain based with three difference logisticans.

[9] build information services platforms centered around the individual object to increase the interaction, the exchange and the organization of data to increase efficency and to reduce the carbon footprint. The approach introduced in this paper targets partially similar goals, but will utilize techniques that can cover the entire life-cycle chain.

An example of collecting object-related data in a web-based data store is the Tales of Things electronic Memories (TOTeM) system. It seeks to foster communication between humans via personal stories digitally linked with things [2]. Its infrastructure shares aspects of an ADOMe, in particular a unified approach for structuring data concerning a thing, and open web-based information storage. The human-computer-interaction is performed by a web-based application on mobile devices.

Similar applications focus on connecting people with objects (e.g., like Anythinx [10], or creating object-centric data collections for personal use (e.g., like Qipp [11]). They all share a user interface, which allows for accessing collections from a desktop as well as on-the-way from a smartphone; data creation is left to the user.

Going beyond, EVRYTHNG [12] extends this general approach with Active Digital Identities for objects, where services linked with an object employ information collections (concerning the object, or objects of the same kind) in order to adapt to the user. The web-based system can be accessed either by a desktop computer or a mobile device.

The question of how implementation and provision of such services can be supported is addressed by Xively [13]. The web-based service supports not only hosting and sharing object data, but also software products and descriptions concerning devices, which we propose to extend in a way that supports exchanging such components across devices using unified data structures and semantic descriptions.

A data structure for representing object memories in open loop scenarios similar to the one mentioned in this article has to meet particular requirements. These are addressed by the so-called *Object Memory Model* (OMM), created by the W3C Object Memory Model Incubator Group (OMM-XG) and co-developed by the authors. The model partitions the memory content into several blocks, each with content of the same origin or nature (see Figure 5) [14]. Each block consists of two parts: the payload representing the content itself and a set of corresponding meta data defining and describing the payload. This set of machine-readable annotations eases the process of searching data inside object memories that

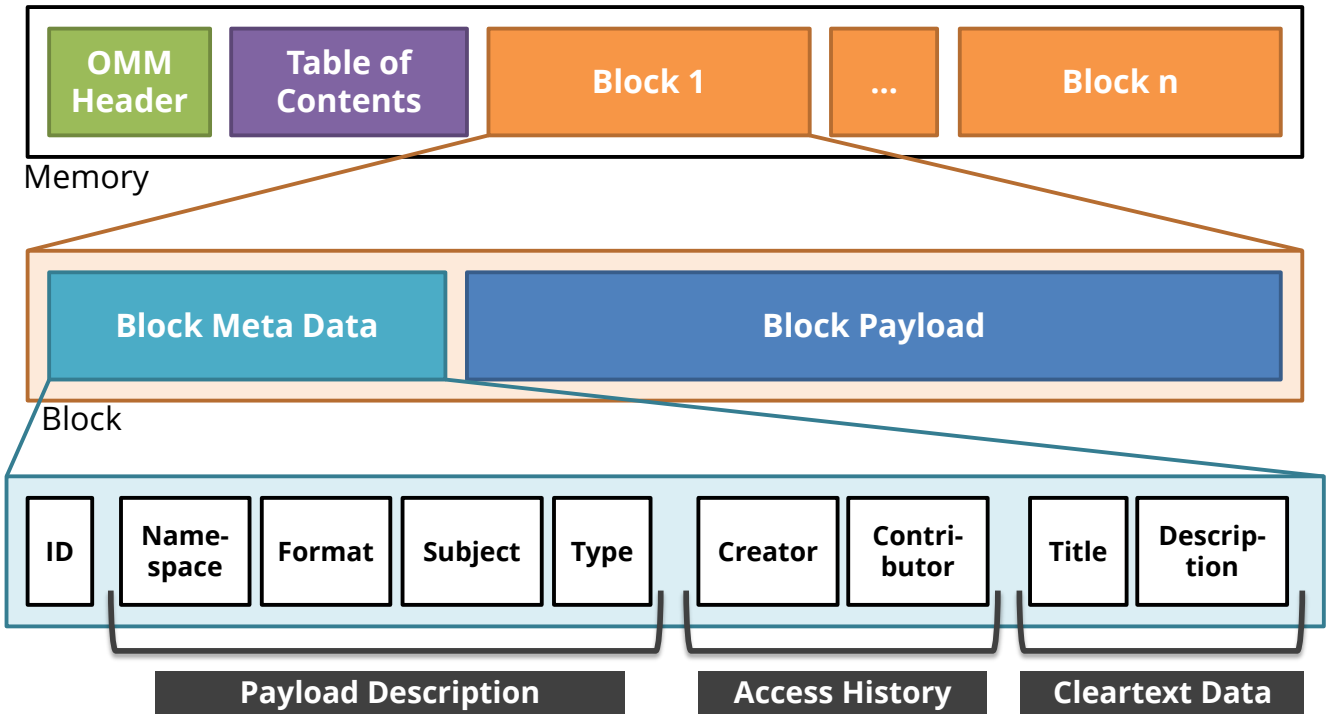


Figure 5. Sample memory based on the Object Memory Model (OMM) with detailed metadata.

contain heterogeneous data and are often not known in advance. Additionally, such memories demand new interaction forms considering machine-to-machine communication (M2M) and human-computer-interaction (HCI) [15].

IV. The ADOMe Framework

The envisioned framework has to balance between the need for flexibility (to support different hardware platforms, open processes, new requirements in the future), for standardized structures that ease data retrieval and communication among different and cross-domain content providers and for normative description of object-related logic. Corresponding goals to the architecture model can be divided in three parts that reflect a 3-tier approach to the realization of such a model, namely the collection model for data storage (1), support for active analysis functionality (2), and a common access architecture and infrastructure support for hardware abstraction with an app-store-like approach (3).

Data storage is the basic functionality of any ADOMe. In order to enable a stakeholder to analyze data added to the memory by another one, a common storage model is achieved, which is shared by all parties along the object's life-cycle. In addition, the model should be flexible enough to cover a large variety of data formats (including encodings and further data types) and should ease the process of data retrieval. Due to the cross-domain usage of such memories involving several partners, a common infrastructure that provides an abstract memory access (including protocol and data exchange specifications) independent from any memory implementation or hardware platform is necessary to ease the task of memory access for existing and newly created applications. This hardware abstraction layer enables a transparent access for clients, which includes a compensation of missing object features by the environment. Setting up activity and analysis support on top of the data storage can extend the functionalities of object memories, by allowing the memory to process data autonomously based on given algorithms. Based on this functionality we want to enable applications to ask common (but pre-defined) semantic questions, rather than processing the entire memory data, which might be a complicated process due to possibly very large and capacious memories and in contrast a slow connection speed. In addition, the memory should pro-actively process data with rules based on expert knowledge, and deploy results to memory storage or return the result on queries. Finally, a mechanism to retrieve machine and platform-compatible logic code for the given use case is needed to support the mentioned hardware abstraction.

A. Object Identification and Data Model

To identify each physical object a unique ID is necessary that goes along with the object during the entire life-cycle chain and represents the corresponding object memory. Our framework uses a unique *Uniform Resource Locator* (URL) [16], [17]. This approach has the advantage that URLs can be used to easily create unique identifiers and to directly indicate the type of memory access (web-based via a http-connection). This URL can be attached to physical objects in different ways. The options range from simple machine-readable 1D- or 2D-codes (e.g., barcodes, DataMatrix Codes or QR Codes) [18]–[20] to more sophisticated wireless solutions like Radio Frequency Identification (RFID) [21] and Near Field Communication (NFC) [22], or the well-known standards like Bluetooth [23] and Wi-Fi [24]. Both approaches can be easily accessed with the help of a smartphone or tablet.

The foundation of the framework is the storage of object-related data. To structure this data a data model covering the requirements of open-loop scenarios is necessary. In our approach we use the aforementioned Object Memory Model (OMM) for structuring the memory content in blocks composed of the payload and additional meta data (see Figure 5). In the following, we will describe the set of meta data in detail.

The first attribute, called *ID*, is a unique identification for each block represented as string. The next four attributes are intended for machine-to-machine (M2M) communication purposes. The *Namespace* is represented as URN and can be used to indicate that the payload has a defined content and a standardized type (e.g., "`urn:objectdata:pml`"). This allows for direct access to the payload, if the reader supports the namespace. *Format* is just the MIME-Type of the payload. *Subject* contains a tag cloud like structure to annotate the block payload with free text tags (e.g., "manual"), hierarchical text tags with a point as delimiter (e.g., "norms.din.a4") and ontology concepts (e.g., "http://s.org/o.owl#Color"). The *Type* attribute is a Dublin Core DCMI Type Vocabulary Type [25]. The following two attributes create a modification history for this block. *Creator* is a tuple of the entity that created the block and a corresponding timestamp. In addition, *Contributor* is a list of tuples with entities that change the block and the corresponding timestamps. Finally, the last two attributes are rather intended for human-computer-interactions (HCI). *Title* contains a human readable short title for this block and *Description* contains a longer textual description, both with support for multiple languages.

In case that the payload has a very large size and does not fit into the memory (e.g., located in an embedded system) or the data is redundant and used in many similar memories, the payload can be out-sourced. This is done by an additional *Link* attribute that indicated the source of the payload (e.g., in the World Wide Web).

Furthermore, each memory contains a header that includes the unique ID of this memory and an optional list of links to additional blocks (e.g., that are out-sourced due to space restrictions) and a table of contents (ToC) that provides the meta data information from all blocks to enable applications to get an overview of the memory content without the need to download and access all blocks.

Generally, the OMM does not provide a set of regulations for the block payload, so the users are free to store the information in the way they want or in the way they have defined with other partners. However, the model includes three pre-defined blocks useful in several scenarios. The *OMM-ID Block* carries a list of identification codes combined with the corresponding string type and a validity timestamp or time-span. The *OMM-Structure Block* indicates relations of the physical object to other objects, without the need of additional formats. The user can use one or several of the following fixed relations: isConnectedWith, isPartOf, hasPart and isStoredIn. Each relation can also be combined with a validity time span. The *OMM-Key-Value Template* provides a container for an arbitrary amount of key-value-pairs that can be used in many use cases.

In addition, we added a proposal for additional blocks (called OMM+) extending the OMM meta model. The *OMM+ Semantic Block* is an extension of the OMM-Structure Block

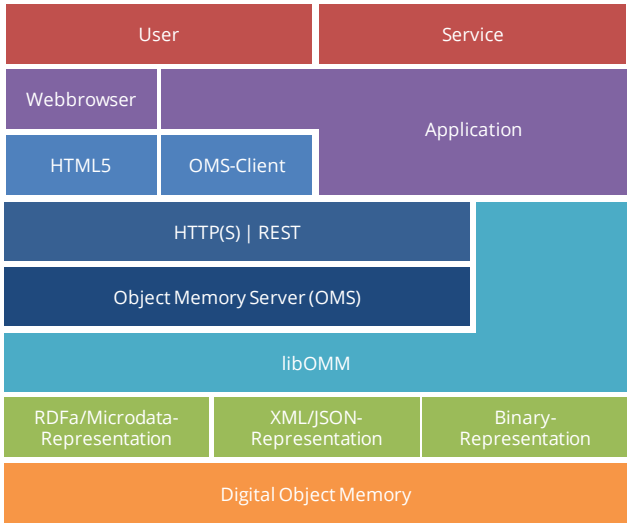


Figure 6. Hierarchical view of OMM-related components.

and allows the definition of arbitrary relations similar to ontology relations (e.g., provided by RDF and OWL) relative to the physical object. Each relation consists of a triple (subject, predicate, and object) represented by uniform resource identifiers (URIs) combined with a validity statement. To indicate the physical object itself the URI `urn:omm:this` is used. The predicate can be use case specific or use common RDF/OWL object relations. It is also possible to include the OMM-Structure Block relations, e.g., the isConnectedWith relation by using the URI `urn:omm:structure:isConnectedWith`. This block allows the definition of simple semantic statements that can be processed semantically (e.g., with a graph reasoner) or without a reasoner just compare the strings of the relation triple.

The *OMM+-Embedded Block* is meant to integrate an entire OMM-based memory into a specific block. In use cases where objects are physically combined to a compound object, or if access is physically hindered by arrangement of objects, it might be the case that the attached ID or the embedded system cannot be reached any longer. The problem can be solved by copying the object's memory, e.g., to the memory of the factory, so its memory can be accessed even if the object's label can no longer be reached. A specific subject meta data attribute `primaryID.<ID of integrated object>` is used to indicate the embedded memory's ID without the need of extracting the memory itself.

B. Storage Infrastructure and Communication Interfaces

The storage infrastructure built on top of this data model is divided in two components. Firstly, a generic software library (libOMM) was developed to integrate the object memory model into Java- and C#-applications and can handle local XML-based OMM-representations. Secondly, we extended this library to a dedicated object server system called *Object Memory Server* (OMS) [26]. Figure 6 shows a hierarchical view of the OMM-related components. This server is divided into several modules that can be compiled depending on the intended application (see Figure 7).

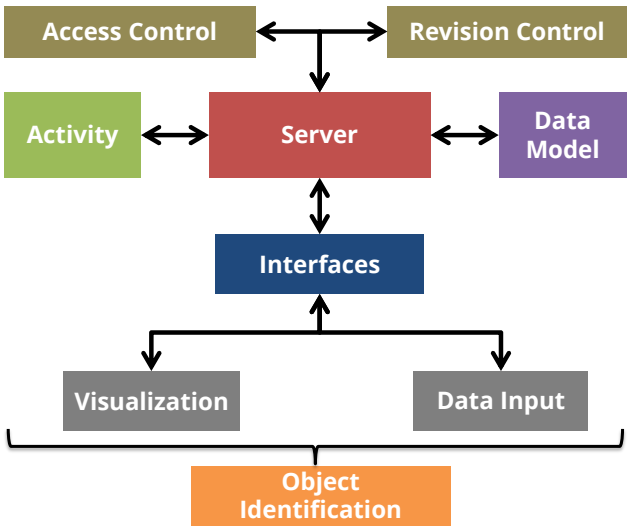


Figure 7. Server-based OMM Architecture.

To foster the usage of digital object memories, the access to memories is not restricted in general. However, the OMS is equipped with a role-based security module. The owner of a memory can restrict read and write operations for specific blocks or the entire memory. Three approaches are available to grant access to memories: passwords, certificates, and electronic ID cards. The simple approach uses a username and a password stored in a white-list containing all entities with access permissions. The more sophisticated approach utilizes digital certificates based on the ITU standard X.509 [27]. With certificates an additional way of restriction is possible: the certificate chain mode. This mode demands that an accessing entity uses a valid certificate and this certificate must provide a valid certificate chain with respect to the root certificate of the memory. The owner can select between two options. First the certificate must be directly signed by the owner or secondly a valid chain to the owner is sufficient. The latter option allows for certificate users to create their own child certificates and deploy them to other users of the memory. This approach lacks the risk that the user pool can be extended in an uncontrolled manner, but allows to distinguish between different ”subcontractors” (each using individual certificates inherited from a accepted authority) without any administrative costs by the memory owner. Finally, we created a prototype application based on the new German ID card (nPA). This card is issued to German citizens by the local registration offices and provides an electronic identification (eID) mechanism to create a unique but anonymous and application dependent ID for each card. This can be utilized to restrict memory access to such eIDs by using an DOMe infrastructure with access control [28], [29].

To increase the level of security the system also prevents the possibility of creating plagiarism by duplicating memories. If a server-based approach is used, the given API allows only block-based memory access, so each block has to be copied to another fake memory, but these ”new” blocks contain different creator information than the origin. If a solution based on RFID-tags is used, the framework can add an additional hash value incorporating the entire memory and the tag ID. A simple copy of the tag data breaks this hash value, due to a different tag ID. For more complex scenarios a more sophisticated solution providing a secure provenance change is described in the next section.

For memory interaction tasks two different interfaces are available (see Figure 6). Applications can use a RESTful HTTP-interface to access, to supplement and to modify object memories. End users can alter memories with the built-in web-based HTML5 user interface that can be utilized within a standard browser on multiple different devices. As mentioned before each memory can be accessed with a unique URL that serves simultaneously as access point and as the object’s primary ID. This URL begins with the DNS name or IP address of the OMS and ends with the name of the memory. In between the caller indicates which module of the OMS should be triggered. This module can be set to ’st’ for the RESTful storage interface or to ’web’ for the HTML5 user interface. Further actions and commands deployed to this module are added to this generic URL part. In the following, we use an exemplary memory that is stored on an OMS and accessible at the domain ’sampleoms.org’ and the sample memory named ’s_memory’:

```
http://sampleoms.org/st/s_mem
http://sampleoms.org/web/s_mem
```

The RESTful interface, represented throughout the ’st’ path, maps the functions and operations of the mentioned libOMM. The URL path `.../st/s_mem/block/_ids/` retrieves the list of all blocks with their IDs that allows applications to access the entire memory. For data retrieval located in a block with unknown ID the function `.../st/s_mem/toc/` can be utilized to get the table of contents of this memory that is a compressed set of meta data from all blocks (e.g., large meta data like clear text description and tags are excluded). Finally, a direct access to block meta data is possible, e.g., to access the creator of a block an application can use URL part `.../st/s_mem/<blockID>/meta/creator/` or `.../st/s_mem/<blockID>/payload/` to access or to change the block payload.

C. Smart Binary Encoding and Secure Provenance

The proposed Object Memory Model and its XML representation can be easily used on a server-based infrastructure with virtually unlimited storage space. Introducing technologies with smaller storage spaces like RFID-tags [21], [30], [31] or cheap embedded controllers do not provide enough capacity to store all memory related metadata, not even by using compressed binary representations. To foster the usage of OMM-based memories even on such technologies, we introduce a context-aware dynamic schema-based mapping approach (see Figure 8) to reduce the overhead of OMM metadata [32]. The mapping schema is utilized to define the mapping process between OMS-interfaces and the corresponding byte stream. The advantage of this approach is to be format-independent by concentrating the mapping logic to the schema rather than using inflexible format-dependent code. The schema is created by an author, delivered to each stakeholder of this object memory and combined with the corresponding OMM access code. The storage space contains only the raw binary memory content.

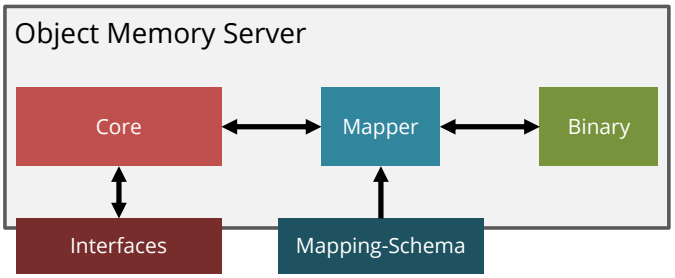


Figure 8. Binary mapping for Object Memory Models.

The schema itself contains rules to map the binary content to code interfaces. In this sample schema, the namespace metadata of a block is defined. The `<composition>`-section defines the structure of the corresponding binary part. Due to the variable length of a namespace, the binary representation consists of a length indicator (one byte) with a restriction to a maximum of 32 bytes and a value (the namespace string itself) as UTF-8 string. The `<interface>`-part defines a code interface including methods that enable the mapper to retrieve the data to be written and to transfer the loaded binary data to the application logic. The definition allows for specifying methods (e.g., for Java or C++ implementations) as well as properties (e.g., for C# or PHP implementations).

The mapping process supports different modes. First, it simply streams the given object memory data to a byte array and vice versa based on the schema definitions. The more advanced modes allow the schema author to limit specific information, e.g., the length of the *title* block metadata or the number of blocks of a specific type. In addition, priorities can be defined for each part of the metadata model to allow the mapper to dynamically drop metadata and blocks with low priority. This process can be applied in two different ways: only to newly written or changed blocks or retroactively to all existing blocks in the memory. The quality and quantity of this loss of information is transparently retrieved by the respective application.

The framework presented in the previous chapters is meant to leverage the process of free access to object memories, to encourage different stakeholders to contribute available data to such memories. However, other use-cases demand a more controlled and secured memory structure. E.g., in a pharma setting where drugs are filled in boxes, then are transported to a pharmacy and finally are sold to customers a consistently and verifiable documentation for these life-cycle steps is needed to ensure the necessary quality requested by the customer. For such purposes the object memory model can be extended, to guarantee the integrity and the authenticity of the memory and to prevent subsequent modifications and amendments of the stored data, as well as the illegal erasing of unwanted information similar to EPC Pedigree approach [33].

This goal is achieved with the help of an extended model by adding the following information to each block (see Figure 9). First, a hash-value of the block content (the metadata and the payload) is generated by concatenating the content and using a SHA-512 [34] hash function. Second, the ID of the predecessor block to connect each block with its predecessor is added. Third, a cryptographic signature is created by using standard

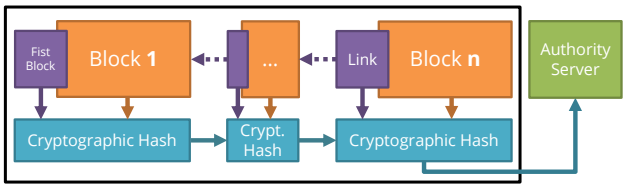


Figure 9. Secure Object Memory with linked Blocks.

X.509-certificates [27] and a private key. The aforementioned two values are integrated as certificate extensions. Finally, the signature of the last (meaning the most recent block) is stored on an additional *authority server*. This server contains the certificate of the most recent block for several memories.

These modifications can prevent an attacker to add additional blocks or to change existing blocks because he needs a private key to generate valid certificates. In addition, an attacker can no longer remove unwanted blocks from a memory because the integrity chain (linking each block with its predecessor) would be broken in this case. Finally, the authority server shows the certificate of the most recent block for each memory such that it is not possible to remove this block without leaving a broken integrity chain. However, since such techniques cannot be used directly by users, e.g., within a web browser, applications have to be extended (by using the mentioned libOMM or the RESTful interfaces) to benefit from this approach.

V. Hardware Abstraction

The aforementioned scenario involves a heterogeneous set of goods equipped with different techniques, ranging from simple barcodes to embedded systems with storage and processing capabilities. In our sample scenario, a user does not have to care about processing power and storage capabilities of the hardware used to implement the object memory. In order to achieve this goal, the ADOMe framework includes a data access interface, which provides abstraction that allows accessing users and the objects themselves to complement their missing functionalities on their own, or at least to inform the outer environment about requested but not available functionalities.

This concept utilizes the aforementioned ADOMe framework that is built on modular software components. It allows the framework to run on a large variety of platforms raging from high-end servers to small embedded systems (see Figure 10). The framework's RESTful interface decouples communication with a memory from its concrete hardware and software implementation. In addition, it enables systems lacking embedded ADOMe functions (e.g., for logic processing) to outsource these to server-based ADOMe solutions by passing-through incoming requests.

In fact, we distinguish three kinds of instrumentation types and interaction approaches, respectively (see Figure 10). These solutions range from a packaging instrumented minimalistically with a barcode or an RFID-tag that can be read during the production and by any mobile compatible device, or instrumented with a fully integrated intelligent embedded system, incorporated into the packaging during the production process.

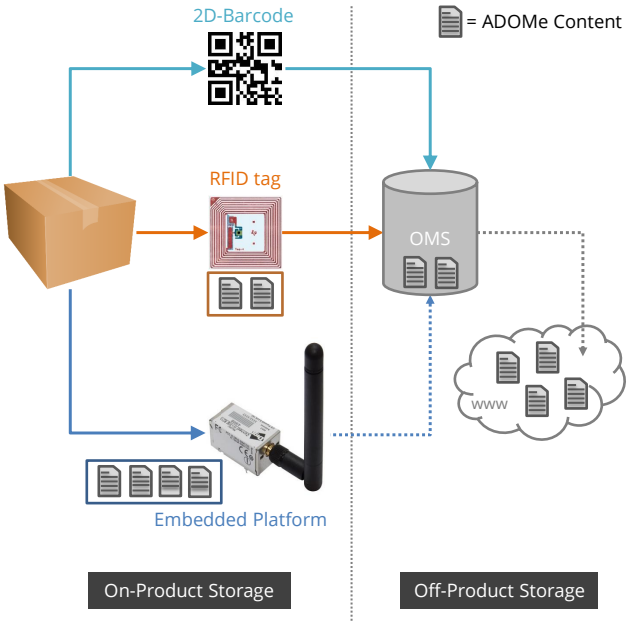


Figure 10. ADOMe hardware abstraction ranging from ”off-product” memories linked via barcodes to ”on-product” embedded systems.

The first approach, called *off-product*, is focused on a lightweight object instrumentation with a barcode or RFID-tag. An external device with internet connection is necessary to access the data stored on server-based solution (e.g., Object Memory Server). The reading and visualization of the data is done with the consumer's own internet compatible device. The drawback of this approach is that it heavily relies on the instrumented environments providing services like sensor measurements and data connections. An access to memory data is only possible with available internet connection. On the other hand, the centralized storage allows an access to memory data without access to the physical object.

Whereas the off-product scenario relies heavily on a dedicated infrastructure, the *on-product* scenario uses the full power of the inbuilt embedded system. Such systems (like Gadgeteer [35], Arduino [36] or other embedded controllers) are directly included within the packaging and are constantly tracking the physical properties of the product like temperature, humidity, air pressure, geo-localization over GPS position, or even shock detection, and thus enabling a precise autonomous tracking. Integrated visualization (ranging from simple multicolor LEDs to touch displays) allow users to get a full overview of the status of the intelligent product and to inspect the object's memory (see Figure 11) without the need of an external device (e.g., like a smartphone or tablet). The drawback of this solution is that the costs are much higher than simple barcodes or RFID stickers, due to the electronic components that need to be embedded into the packaging to realize the sensor value tracking. In addition, the memory data is only accessible as long as the physical device is in range if there is no distributed backup server kept in sync.

A hybrid solution set between the on-product and off-product approach is the so called *incycling* approach [37], [38] that might be a solution for the following cases: the cost-benefit ratio of the product concerning the price of an on-

Figure 11. Performing built-in and on-product integrity checks on a smart package equipped with ADOMe and accessed by a mobile handheld device.

product instrumentation and the given surplus value of a local object memory is below 1, or the on-product instrumentation is only meaningful for a short term or a short life-cycle phase. Hence, incycling proposes the following scenario: a product is instrumented at a specific point of time (e.g., a new lify-cycle phase). As soon as the next phase is applied (e.g., the product is consumed or local sensors are not needed anymore) the intelligent board is removed from the package and is attached to a new object that is just entering this phase. Due to the fact that such an on-product memory can be reused in a closed-loop and the costs are distributed over several carriers of the memory, the value of benefit is increased significantly. Sample incycling applications range from attaching an embedded controller to a solid product during transport to workpiece carriers that provide memory functionalities for each workpiece that is carried through the production line.

The next evolution is the integration and processing of logic code directly inside such active object memories. This approach has the advantage that there is no need for an external device to execute the logic code, no need to download the entire memory content to this device and no need to keep code ready for each platform and for every object type. The code fragments (called *snippets*) are stored in blocks inside the memory. This allows manufacturers to deliver their products with build-in quality check or monitoring scripts. The snippets can be addressed with a unique ID or a semantic concept (e.g., ”#QualityCheck”) stored in the block meta data. The latter one can be deemed as definition of semantic questions and accessed from external sources that trigger such concepts as asking a question (trigger of ”#QualityCheck” equals asking the question ”Are you ok?”). To complement the activity module, snippets can also be triggered event-based or by a so called heartbeat timer. The event-based mechanism allows users to define a set of monitored blocks for each snippet and the activity module triggers the snippet each time these blocks are changed. The heartbeat represents a timer, that allows for snippet execution in a defined period (e.g., every 30 minutes).

The processing and execution location (either in embedded systems ”on-product” or on server-based solutions ”off-product”) is transparent for the access application. In case of no fitting software module available in memory, an access to a so called app store is possible based on a semantically

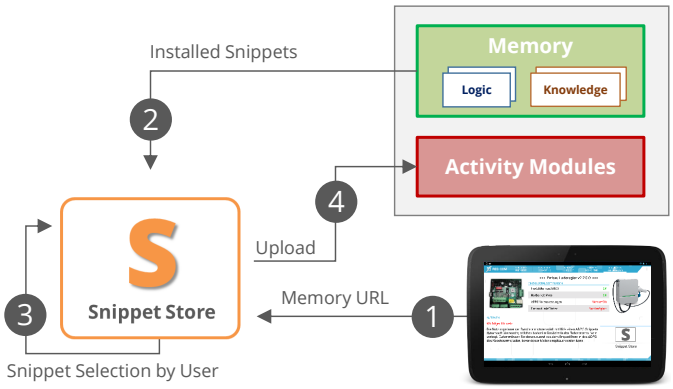


Figure 12. Dataflow of Activity Update with external Snippetstore.

Figure 13. Client application displaying ADOMe-based measurements created by a software module downloaded from an external appstore.

defined logic description. This store contains a large set of logic modules for different applications and platforms, ranging from fixed domain-specific code to generic modules capable of parametrization. If available the framework downloads, installs and executes the downloaded module from the store (see Figure 12). Client applications performing operations based on memory data can be extended with in-memory or app store modules the same way.

VI. User Interaction with Object Memories

In the following section, we describe the interaction forms with smart objects within the already mentioned logistic process starting at the warehouse and ending at the customer's house. The kind of interaction used within these scenarios is closely linked to the hardware type and chosen packaging type.

During the entire logistics chain, an intelligent product will need to go through several checks and validation processes until reaching the end consumer. In the first stage, the product leaves the production phase, in which an object memory was created and filled with static product and manufacturer data. This is done by machine-to-machine (M2M) communication without any user involvement. The special monitoring or processing capabilities of some products can be achieved with the already mentioned snippets that can perform such tasks. The manufacturer adds tailored snippets to the object memory and starts execution. If the product is equipped with an on-board memory the snippets are inside the embedded controller otherwise (in the off-product case) the execution is done within the object memory server.

During transport sensor values are generated, e.g., by monitoring temperature, humidity, acceleration, and position either by the object itself (on-product) or by the environment (off-product). Since memories may contain hierarchical information about their environment, sensor data measured at higher levels (e.g., on container or palette level) can be passed to and stored in lower levels (e.g., a transport box or the object itself). The continuously updated memory can be accessed (pulled) with a simple web-browser or control station directly (off-product, via OMS) or by relaying information to higher levels (on-product), e.g., palettes relay data to containers and these containers sync their data to web-based OMS. Additional important data can also be pushed by active memories with the help of output channels triggered by snippets (e.g., a product can send an email as soon as a temperature threshold is exceeded).

Some products may require special treatment or have to comply with special customer demands. In our scenario, compliance is verified by a worker by means of a mobile device (smartphone or tablet). Once the worker has scanned the barcode or RFID-tag attached to the object, an app running on the mobile device connects to the on-product memory or the off-product object memory server. In case of an on-product memory the app can directly access the memory, e.g., with Wi-Fi Direct [39] or Bluetooth [23]. The app retrieves memories nearby with the UPnP [40] standard (see also Figure 11). The snippet store is available as smartphone or tablet app and is structured and handled the same way as known app stores on such devices. A check for available snippets regarding the specific object is performed and displayed to the user. He or she can select the favored one and automatic upload and activation process is initiated (see Figure 12). Such snippets ease the process of adding and adjusting additional monitoring and processing logic.

The identical concepts and hardware extended with a display can be integrated in products of everyday life. An example is the already mentioned milk carton "Milky" that supports direct product-to-consumer interaction within three life-cycle steps [41]. Due to the highly instrumented product no further infrastructure or tools are necessary to interact with the carton. In the advertising phase (intended to raise the customer's attention) product-related data is presented via the built-in display. A generic framework gets all related data from the object memory. Information is provided by the manufacturer and can be extended by the retailer. Once bought, milky continues monitoring conditions like temperature and humidity without any user interaction if the customer allows such tasks in his or her preferences. Otherwise, milky asks the customer. Any condition violations are displayed. At home milky informs about best-before dates coming closer based on a memory data set by the manufacturer and possibly adjusted by sensor readings (e.g., higher temperatures are measured and the best-before date is set to an earlier date) Once the content is consumed recycling information are displayed regarding the current position. In addition to these presentations based on raw data, the object can also display the 'face' of milky (see Figure 14). This anthropomorphic style presents the 'mood' of the milk carton. The system provides four different moods: happiness, sadness, amorousness and annoyance reflecting the

Figure 14. Smart milk carton 'Milky' in anthropomorphic mode 'happy'.

current state of the milk inside the carton by addressing the customer's feelings. Happiness is the default mood representing the milk to be in a good condition. Sadness occurs shortly before the carton is empty. Amorousness is used to display that other products or customers are detected in range. And finally annoyance (addressing the fact that in German language the states annoyance and sour are expressed by the same word 'sauer') is displayed once the milk has turned sour.

VII. Conclusion and Outlook

This article summarized work in progress concerning a framework for setting up so-called Active Digital Object Memories. Its contribution is twofold: In order to leverage the application of such data collections in open scenarios, this framework seeks to 1) embrace different ways of deploying answering logic in a collection, and 2) provide abstraction from the technical diversity of existing infrastructures for collecting object-related data, which employ technology embedded into physical objects, virtual data stores located in the Web, and combinations of both approaches. Future work will address in the very first place the proposed method of distributing processing logic within this framework: the app store implementation. A first prototype illustrates the feasibility of this approach in a manufacturing scenario involving passive RFID, Android tablets, and embedded devices; however, more efforts are needed to verify that concept for broader range of embedded system platforms as well as logic hosted for deployment.

Acknowledgment

This research was funded in part by the German Federal Ministry of Education and Research under grant number 01IA11001 (project RES-COM) and 01IS12050 (project OMM++). The responsibility for this publication lies with the authors.

References

[1] A. Kröner, J. Haupert, C. Hauck, M. Deru, and S. Bergweiler, "Fostering access to data collections in the internet of things," in UBICOMM 2013, The Seventh International Conference on Mobile Ubiquitous Computing, Systems, Services and Technologies located at NexTech 2013, September 29 - October 3, Porto, Portugal, W. Narzt and A. Gordon-Ross, Eds., IARIA. IARIA, 9 2013, pp. 65–68.

[2] R. Barthel, K. Leder Mackley, A. Hudson-Smith, A. Karpovich, M. de Jode, and C. Speed, "An internet of old things as an augmented memory system," in Personal and Ubiquitous Computing, vol. 17. Springer London, 2011, pp. 321–333.

[3] D. Fitton, F. Kawsar, and G. Kortuem, "Exploring the design of a memory model for smart objects," in Ambient Intelligence and Smart Environments, vol. 4: Workshops Proceedings of the 5th International Conference on Intelligent Environments. IOS Press, 2009, pp. 33–38.

[4] P. Stephan, G. Meixner, H. Kling, F. Flörchinger, and L. Ollinger, "Product-mediated communication through digital object memories in heterogeneous value chains," in Proceedings of the 8th IEEE International Conference on Pervasive Computing and Communications. IEEE, 2010, pp. 199–207.

[5] M. Schneider, M. Velten, and J. Haupert, "The objectrules framework - providing ad hoc context-dependent assistance in dynamic environments," in Proceedings of the Sixth International Conference on Intelligent Environments. July 19-21, Kuala Lumpur, Malaysia. IEEE Computer Society CPS, 2010, pp. 122–127.

[6] C. Decker, T. Riedel, M. Beigl, L. Moreira, S. Souza, P. Spiess, and S. Haller, "Collaborative business items," in Proceedings of IE 07: 3rd International Conference on Intelligent Environments, Ulm, Germany, 2007, pp. 40–47.

[7] M. Mühlhäuser, "Smart Products: An introduction," in Constructing Ambient Intelligence: AmI 2007 Workshops. Springer Berlin / Heidelberg, 2007, pp. 158–164.

[8] EURIDICE, "EURopean Inter-Disciplinary research on Intelligent Cargo for Efficient, safe and environment-friendly logistics," 2011, [last accessed: 05-16-14]. [Online]. Available: http://www.euridice-project.eu/index.php/web/pubdocs/58

[9] iCargo, "Intelligent Cargo in Efficient and Suitable Global Logistics Operations," 2014, [last accessed: 05-16-14]. [Online]. Available: http://i-cargo.eu/type/publications

[10] stuffl UG, "Anythinx," 2014, [last accessed: 05-16-14]. [Online]. Available: http://www.anythinx.de/

[11] qipp AG, "qipp," 2014, [last accessed: 05-16-14]. [Online]. Available: https://www.qipp.com/en

[12] EVRYTHNG Ltd., "EVRYTHNG Every Thing Connected," http://evrythng.com/, [last accessed: 05-16-2014].

[13] Xively (by LogMein Inc.), "Xively - Internet of Things Platform Connecting Devices and Apps for Real-Time Control and Data Storage," http://xively.com/, [last accessed: 05-16-2014].

[14] A. Kröner, J. Haupert, M. Seißler, B. Kiesel, B. Schennerlein, S. Horn, D. Schreiber, and R. Barthel, "Object memory modeling - W3C incubator group report," 2011, [last accessed: 05-16-2014]. [Online]. Available: http://www.w3.org/2005/Incubator/omm/XGR-omm-20111026/

[15] R. Barthel, A. Kröner, and J. Haupert, "Mobile interactions with digital object memories," Pervasive and Mobile Computing, vol. 9, Issue 2, 2013, pp. 281–294.

[16] M. Mealling and R. Denenberg, "RFC 3305: Uniform resource identifiers (uris), urls, and uniform resource names (urns): Clarifications and recommendations," 2005.

[17] T. Berners-Lee, R. Fielding, and L. Masinter, "RFC 3986: Uniform resource identifier (uri): Generic syntax," 2005.

[18] O. Rosenbaum, Das Barcode-Lexikon, ser. Edition advanced. BHV-Verlag, 1997.

[19] International Organization for Standardization, "Information technology automatic identification and data capture techniques data matrix bar code symbology specification," Geneva, Switzerland, 2006.

[20] ——, "Information technology — automatic identification and data capture techniques — qr code 2005 bar code symbology specification," ISO/IEC 18004:2006, 2006.

[21] ——, "Identification cards - contactless integrated circuit(s) cards - proximity cards," ISO/IEC 14443:2000, 2000.

[22] ——, "Near field communication interface and protocol-2," ISO/IEC 21481 / ECMA-352, 2010.

[23] Bluetooth Special Interest Group, "Bluetooth Specification ," 2013, [last accessed: 05-16-14]. [Online]. Available: https://www.bluetooth.org/en-us/specification/adopted-specifications

[24] IEEE, "802.11: Wireless LANs," 2014, [last accessed: 05-16-14]. [Online]. Available: http://standards.ieee.org/about/get/802/802.11.html

[25] DublinCore, "DCMI Type Vocabulary," 2014, [last accessed: 05-16-14]. [Online]. Available: http://dublincore.org/documents/dcmi-type-vocabulary/#H7

[26] J. Haupert, "DOMeMan: A framework for representation, management, and utilization of digital object memories," in 9th International Conference on Intelligent Environments (IE) 2013, July 18-19, Athens, Greece. IEEE, 7 2013, pp. 84–91.

[27] ITU-T Recommendation X.509 Version 3, "Information technology - open systems interconnection - the directory: Authentication framework," 1997.

[28] B. Brandherm, J. Haupert, A. Kröner, M. Schmitz, and F. Lehmann, "Demo: Authorized access on and interaction with digital product memories," in 8th Annual IEEE International Conference on Pervasive Computing and Communications, March 29 - April 2, Mannheim, Germany. IEEE Computer Society, 2010, pp. 838–840.

[29] ——, "Roles and rights management concept with identification by electronic identity card," in 8th Annual IEEE International Conference on Pervasive Computing and Communications, March 29 - April 2, Mannheim, Germany. IEEE Computer Society, 2010, pp. 768–771.

[30] G. Reinhard, P. Engelhardt, E. Genc, T. Irrenhauser, M. Niehues, M. Ostgathe, and K. Reisen, "Einsatz von RFID in der Wertschöpfungskette - Konsortium entwickelt RFID-basierte hybride Steuerungsarchitektur und Bewertungsmethode für Wertschöpfungsketten," RFID im Blick - Sonderausgabe RFID in der Region München, 3 2011.

[31] N. Schlitter, F. Kähne, S. T. Schilz, and H. Mattke, Operations and Technology Management, ser. Innovative Logistics Management. Erich Schmidt Verlag, 2007, vol. 4, ch. Potential and Problems of RFID-Based Cooperation in a Supply Chain, pp. 147–164.

[32] A. Höh, "Smart Binary Representation For Digital Object Memories," Bachelorthesis, Saarland University, 2014.

[33] EPCglobal, "The Pedigree Ratified Standard Version 1.0," 2007, [last accessed: 05-16-14]. [Online]. Available: http://www.epcglobalinc.org

[34] D. Eastlake 3rd and T. Hansen, "RFC 6234: US secure hash algorithms (SHA and SHA-based HMAC and HKDF)," 2011.

[35] Microsoft Corporation, ".NET Gadgeteer," 2014, [last accessed: 05-16-14]. [Online]. Available: http://www.netmf.com/gadgeteer/

[36] Arduino, "Arduino Project," 2014, [last accessed: 05-16-14]. [Online]. Available: http://arduino.cc/

[37] B. Brandherm, A. Kröner, and J. Haupert, "Incycling - sustainable concept for instrumenting everyday commodities," in Proceedings of the International Workshop on Networking and Object Memories for the Internet of Things. Workshop on Digital Object Memories (DOME-11), located at UbiComp 2011, September 17-21, Peking, China. ACM, 9 2011, pp. 27–28.

[38] B. Brandherm, A. Kröner, J. Haupert, M. Schmitz, F. Lehmann, and R. Gampfer, "Sustainable instrumentation of everyday commodities - concepts and tools," in 10th Annual IEEE International Conference on Pervasive Computing and Communications, 10th, March 19-23, Lugano, Switzerland. IEEE Computer Society, 2012, pp. 467 – 470.

[39] WiFi Alliance, "WiFi Direct," 2010, [last accessed: 05-16-14]. [Online]. Available: http://www.wi-fi.org/discover-wi-fi/wi-fi-direct

[40] International Organization for Standardization, "Information technology - UPnP device architecture - part 1: UPnP device architecture version 1.0," ISO/IEC 29341-1:2011, 2011.

[41] M. Deru and S. Bergweiler, "Milky: On-product app for emotional product to human interactions," in Proceedings of the 15th International Conference on Human Computer Interaction with Mobile Devices and Services, 15th, August 27-30, Munich, Germany. ACM, 2013.

Virtualization as a Driver for the Evolution of the Internet of Things: Remaining Challenges and Opportunities Towards Smart Cities

Andreas Merentitis[1], Vangelis Gazis[1], Eleni Patouni[2], Florian Zeiger[1], Marco Huber[1], Nick Frangiadakis[1], and Kostas Mathioudakis[1]
[1]AGT International, Darmstadt, Germany
[2] Department of Informatics & Telecommunications, University of Athens, Athens, Greece
{amerentitis, vgazis, fzeiger, mhuber, nfrangiadakis, kmathioudakis}@agtinternational.com
{elenip}@di.uoa.gr

***Abstract*— Fueled by advances in microelectronics, wireless communications and the availability of affordable mobile connectivity, the last decade has seen an unprecedented proliferation in the number of interconnected devices. This evolution is part of the transition to the Internet of Things (IoT), which envisions connecting anything at any time and place. While it can be argued we are already living in the IoT era, the next paradigm shift is already emerging on the horizon, targeting yet another order of magnitude increase in the number of interconnected devices and promising to bring people and processes in the equation. This is particularly important towards the vision of Smart Cities, where physical infrastructure is complemented by the availability of intellectual and social capital, increasing both urban competitiveness and quality of life. However, before such a paradigm shift can be realized, significant challenges with respect to scalability, cooperative communications, energy consumption, as well as convergence of sensor and analytics trends have to be resolved. In this paper we elaborate on the different trends, as well as the remaining open problems and we show how Sensor Virtualization Technology, capturing both the Virtual Sensors and Virtual Sensors Networks aspects, promises to alleviate or resolve these challenges, and pave the way towards the evolution of the Internet of Things.**

Keywords- Sensor Networks, Sensor Virtualization; Machine to Machine Communications; Internet of Things; Future Internet.

I. INTRODUCTION

Technological advances in the fields of sensor technology, low power microelectronics, and low energy wireless communications paved the way for the emergence of Wireless Sensor Networks (WSNs). These networks are currently used in a wide range of industrial, civilian and military applications, including healthcare applications, home automation, earthquake warning, traffic control and industrial process monitoring. A WSN is a system composed of small, wireless nodes that cooperate on a common distributed application under strict energy, cost, noise and maintenance constraints [1], [2]. Although many interesting applications have been implemented/developed for WSNs, further work is required for realizing their full potential as "the next big thing" that will revolutionize the way we interact with our environment.

Such promises are particularly important when viewed in the context of the global urbanization trend and the challenges that accompany it. With 60% of the world population projected to live in urban cities by 2025, the efficient use of resources becomes a topic of paramount importance. Such efficiency calls for situational awareness of the Smart City across multiple domains in an unprecedented level.

As a promising step in this direction, during the last decade there has been a growing research interest in the Internet of Things (IoT), ranked as a disruptive technology, according to the US National Intelligence Council [3]. An early definition for the IoT envisioned a world where computers would relieve humans of the Sisyphean burden of data entry, by automatically recording, storing and processing all the information relevant to the things involved in human activities, while also providing "anytime, anyplace [...] connectivity for anything" [4].

Beyond offering pervasive connectivity, the IoT ecosystem is composed of smart things, objects, and applications. This notion of smartness is taking different forms in the literature. For example, the user experience of a mediated context-aware mobile system which is enabled by modern smart phones and is focusing on urban environments is presented in [5]. Approaches that support the exploitation of semantic technologies in context aware smart space applications are described in [6]. The presented technologies enable the creation of pervasive computing systems. A new flow-based programming paradigm for smart objects and the IoT is introduced in [7]. New workflow models suitable for embedded devices have been proposed, as well as orchestration techniques for the ad-hoc combination of smart objects. Smart spaces are discussed in [8] as a way to meet challenges such as interoperability, information processing, security and privacy towards the deployment of IoT.

Combining the notions of pervasive connectivity and smartness, different understandings and definitions have been reported in the literature [9]-[11] regarding what the Internet of Things is about. However, while it is possible to argue that the IoT is already here [12], the next (r)evolutions are already on the horizon, ranging from the open effort to the Future Internet and the rapidly spawning Smart City projects around the world up to industry driven initiatives. The latter include efforts such as the National Instruments Data Acquisition Technology Outlook [13], the General Electric concept of "Industrial Internet" [14], and the

CISCO initiated "Internet of Everything" [12], [15]. Such initiatives have differences in flavor and focus; yet, it is possible to distil the general trends and enablers that need to be in place for successfully realizing the shift to the next networking paradigm, whichever form it might take.

In this paper, we argue that, among these enablers, Sensor Network Virtualization is a technology that has the potential to augment and unlock advances in several other fronts (e.g., scalability, cooperation, low energy solutions and convergence of Sensor Network and Data Analytics trends) that will pave the way towards this paradigm shift. Smart Cities are going to be at the forefront of this paradigm shift, therefore a lot of the examples and use cases discussed in following Sections are coming from the domain of Smart Cities.

The rest of the paper is organized as follows: Section II highlights the main challenges of Smart Cities and the costs associated to the lack of data integration across multiple verticals. The lack of such data integration capability can be seen as a driver for some of the key networking trends that are commonly captured in several independent views for the next networking paradigm evolution. It finishes with a selection of four core areas where significant challenges remain unresolved. Section III introduces the Virtualization layers and the main functionality that each layer is responsible for. It gives also a broad overview of which virtualization types promise to address each of the core areas. The selected areas and the nature of the challenges in each of them are then discussed in more detail in Sections IV-VII. Section VIII elaborates on the different aspects of sensor infrastructure virtualization. Their advantages are captured and the potential of using different virtualization flavors to address the challenges described earlier is explained. Finally, Section IX concludes the paper.

II. TOWARDS SMART CITIES: IDENTIFICATION OF RELEVANT NETWORKING TRENDS

Amassing large numbers of people, urban environments have long exhibited high population densities and now account for more than 50% of the world's population [16]. With 60% of the world population projected to live in urban cities by 2025, the number of megacities (i.e., cities with at least 10 million people in population) is expected to increase also. It is estimated that, by 2023, there will be 30 megacities globally. Considering that cities currently occupy 2% of global land area, consume 75% of global energy resources and produce 80% of global carbon emissions, the benefit of even marginally better efficiency in their operation will be substantial [16]. For instance, the Confederation of British Industries (CBI) estimates that the cost of road congestion in the UK is GBP 20 billion (i.e., USD 38 billion) annually. In London alone, introduction of an integrated ICT solution for traffic management resulted in a 20% reduction of street traffic, 150 thousand tons of CO_2 less emissions per year and a 37% acceleration in traffic flow [17].

Being unprecedentedly dense venues for the interactions (economic, social and of other kind) between people, goods and services, megacities also entail significant challenges. These relate to the efficient use of resources across multiple domains (e.g., energy supply and demand, building and site management, public and private transportation, healthcare, safety and security, etc.). To address these challenges, a more intelligent approach in managing assets and coordinating the use of resources is envisioned, based on the embodiment of sensor and actuator technologies throughout the city fabric in a pervasive manner. This ubiquitous fabric will be supported by flexible communication networks and the ample processing capacity of data centers.

By aggregating data feeds and applying data processing algorithms to reveal the main relationships in the data, the situational awareness of the Smart City across multiple domains (e.g., transportation, safety, health, energy, etc.) at the executive level is greatly facilitated. For instance, by leveraging its open data initiative, the city of London provides a dashboard application demonstrating the kind of high-level overview and insight achievable by cross-silo data integration and innovative analytic applications [18]. However, this vision entails significant challenges on the design of the sensory fabric and the application model through which sensory data are discovered, accessed and consumed. It is currently understood that an intermediary layer of abstraction between the actual sensors and the applications utilizing them will be necessary [19].

The role of such a layer is to abstract the peculiarities of the sensor hardware from the applications, thus facilitating interoperability; to provide opportunities for forming shared resource pools, therefore increasing the efficiency and scalability of the system; and to allow creation of sandboxed islands that enforce the least privilege principle, thus enabling privacy protection (e.g., particularly important for a lot of healthcare applications in Smart Cities). Related activities towards such goals have been in the scope of various initiatives, focusing both on the scalable interconnection part, as well as on efficiency and privacy topics. All of these objectives have to be supported in a transparent way through well-established and standardized discovery and negotiation protocols, so that the devices can autonomously perform them with only minimal or no human intervention.

In parallel with the efforts towards efficiently and transparently interconnecting a myriad of smart devices according to the IoT vision, the Future Internet stands as a general term for research activities and communication paradigms towards a more up to date and efficient Internet architecture. Approaches towards the "Future Internet" cover the full range from small, incremental evolutionary steps up to complete redesigns (clean slate) of the core architecture and the underlying mechanisms, where the applied technologies are not to be limited by existing standards or paradigms (e.g., the client server networking model might evolve into co-operative peer structures). In

general, most of the work in this area is summarized by the Future Internet Assembly (FIA) [20], where it is underlined that whatever form the Future Internet may take, a set of core principles need to be preserved:

- *Heterogeneity support principle*, refers to supporting a plethora of devices and nodes, scheduling algorithms and queue management mechanisms, routing protocols, levels of multiplexing, protocol versions, underlying link layers or even administrative domains and pricing structures.
- *Scalability and Amplification principle*, describing the ability of a computational system to continue operating under well specified bounds when its input is increased in size or volume.
- *Robustness principle*, ensuring that each protocol implementation must transparently interoperate with other implementations.
- *Loose Coupling principle*, describing a method of interconnecting architectural components of a system so that those components depend on each other to the least extent practicable.
- *Locality principle*, which in the computer science domain focuses on the design of thrashing-proof, self-regulating, and robust logical systems.

However, apart from these principles that should only undergo small incremental changes (if any) a list of additional principles that need to be significantly adapted/relaxed or augmented is also provided. Here, we focus on a subset of this list that is related or overlapping to the IoT evolution:

- *Keep it simple, but not "stupid" principle* [20], which refers to the fact that in current Internet design, the complexity belongs always at the edges, while in a more flexible architecture inherently supporting heterogeneous "Things" this might not always be the case.
- *Polymorphism principle*, which refers to the ability to manipulate objects of various classes, and invoke methods on an object without knowing that object's type. The idea is to extend this principle to allow the same abstract components exhibiting different functional and non-functional behavior in case of changing environments or circumstances [20].
- *Unambiguous naming and addressing principle*, establishing that protocols are independent of the hardware medium and hardware addressing scheme. The proposal of the FIA initiative is to extend this principle in order to also capture the data and services.

Even more recently than the FIA initiative, CISCO has evangelized the Internet of Everything (IoE) as the next wave in the evolution of the networking paradigms [12]. With a clear all-IP focus, building on the same principles as Machine to Machine Communications (M2M) and the Internet of Things but extending them, the IoE envisions to increase the number of connections by yet another order of magnitude (from ~10 billion currently connected "Things"). However, arguably the biggest innovation is that it targets to include processes and people in the loop, facilitating and enabling communications that are more relevant in order to offer new capabilities, richer experiences and unprecedented economic opportunities.

In all the previous activities, as well as in various independent research efforts, it has already being identified that in future large-scale heterogeneous networks, the adoption of mechanisms achieving scalable, predictable and self-adaptive network behavior ("more relevant" in CISCO IoE terminology, "pushing the boundaries" in the GE Industrial Internet notion) will be a key enabler [12], [14], [15], [21], [22]. At the same time, with systems becoming continuously more complex in terms of scale and functionality, reliability and interoperability are getting increasingly important. Therefore, techniques for achieving dependable system operation under cost and energy constraints will be an important evolutionary step [2], [21], [22].

In the majority of cases, wireless network development is guided by horizontal mass-markets ("one size fits all"). On the other hand, typically different verticals and niche markets require dedicated applications [22]. Consequently, the deployment or evolution of a wireless network in these areas often demands for expensive infrastructure replacement. Moreover, extending system and network capabilities, switching services or adopting the purpose of an operational network consisting of heterogeneous "Things" usually calls for costly (manual) reconfigurations and upgrades, while it often results in temporary unavailability of system services. Both of these properties are not attractive in a Smart City environment, while the second one is strictly unacceptable for a large number of relative vertical areas that form the backbone of the city infrastructure, such as water and electricity supply networks, Intelligent Transportation Systems, etc.

On the other hand, dynamic changes during operation typically allow for only a limited subset or scope of updates, which may not be sufficient for example if the goals of the network have to be radically changed in order to support a mega-event or provide emergency services in case of a catastrophic event such as an earthquake or flood. Even in normal operation, the ability to evolve significantly the objectives of the networking infrastructure over a period of time might provide opportunities for cutting costs, making it easier to integrate new systems as they become available or change the scope of a network to a secondary objective, while still being able to provide backup capacity to the new primary network in case it is required. Solutions for such problems require capabilities for spontaneous ad-hoc cooperation between objects, self-adaptive behavior, exploitation of dynamic information, predictability of non-functional properties (e.g., energy consumption), and on-the-fly reconfiguration [21], [22], [23].

Summarizing, first and foremost, **scalability** is the key enabler for facilitating the (r)evolution of the Future Internet as the number of interconnected devices is expected to rise by yet another order of magnitude. The vast majority of these devices will be smart sensors with relatively limited computation resources. Thus, key challenges lie in efficient **cooperation** of heterogeneous network elements in order to realize advanced capabilities and services. Furthermore, innovations to **low energy solutions** create an attractive business case by offering benefits in terms of operational cost, long-term product reliability and increased lifetime of wireless and mobile elements (especially relevant for a significant portion of the myriad of electronic "Things" that will be battery powered in the Smart City environment). Last but not least, as the number of interconnected devices will increase a **convergence of the Sensor Network and Data Analytics trends** is required for effectively bringing processes and people into the equation. Following a short description of the different virtualization levels, an overview of the respective trends and key open issues is provided in the sequel of this section.

III. VIRTUALISATION LEVELS

The challenges identified in Section II for the evolution of Internet of Things require solutions for the scalability, data isolation and generation of relevant information at the end-user side. The latter will inevitably trigger changes at the network level, to handle performance issues as well as network/resource management technical challenges related to the vast number of interconnected devices and huge amount of generated data. Thus, this analysis addresses the benefits of virtualization at the end-user level, complemented by related requirements at the network side.

Several types of virtualization can be distinguished at both the network and the end user side, including Virtual Machines and OS Virtualization, Sensor Virtualization, and Sensor Network Virtualization [24]. While the first two types have found their way into mainstream applications and are arguably the driving forces behind the cloud computing paradigm, the other two types are still in their infancy. In this work, we investigate sensor virtualization from the perspective of extracting relevant information from a large network of heterogeneous sensors, in a secure, efficient, and device-agnostic way.

The end-user side addresses the interconnection of the different user hardware appliances/things (e.g., sensor or embedded devices) and is closely related to the evolution of the Internet of Things. However, the biggest breakthrough envisioned in this part is to include processes and people in the loop, enabling communications that are more relevant in order to offer new capabilities, richer experiences and unprecedented economic opportunities. To pave the way for this vision, sensor virtualization will play an important role towards: (1) addressing scalability challenges in the interconnection, control and management of a plethora of heterogeneous smart things, (2) promoting cooperation between the different elements in an energy efficient way, and (3) providing a basis over which the data analytics and sensor network trends can evolve and converge, independent of manufacturer-specific hardware or software perks [1].

At the network side, virtualisation implements the abstraction of network elements and transport resources, as well as their combination into a common pool, possibly distributed among different network locations. When a static network location is considered, the physical resources of a single network element are partitioned to form virtual resources. The distributed case is realised through the relocation of specific network functions to standard hardware servers that can be placed anywhere in the network; in addition, the separation between physical resources and logical services of network elements is possible [25].

In order to realize this separation, the Network Infrastructure Virtualization layer supports resource reusability and flexible resource pooling at the PHY and MAC layers. Its main purpose is to facilitate efficient usage of the network resources and not to abstract and aggregate their management from a central point. Thus, it facilitates the virtualization at the end-user side.

In the end-user side we introduce the Thin Software Virtualization layer, to support dynamic formulation, merging and splitting of sensor network subsets that serve different applications and are possibly administered by different entities. This software is embedded in the end-user devices. It caters for (1) interoperability of heterogeneous sensors from different vendors, (2) exposure of the sensor basic functionality to the data consumer and sensor assignment to tasks, (3) data isolation and enforcement of the least privilege properties, and (4) collaboration with other sensors and/or consideration of analytic models that connect the underlying phenomena so that the sensed data can be transformed to relevant information, produced and transmitted on demand.

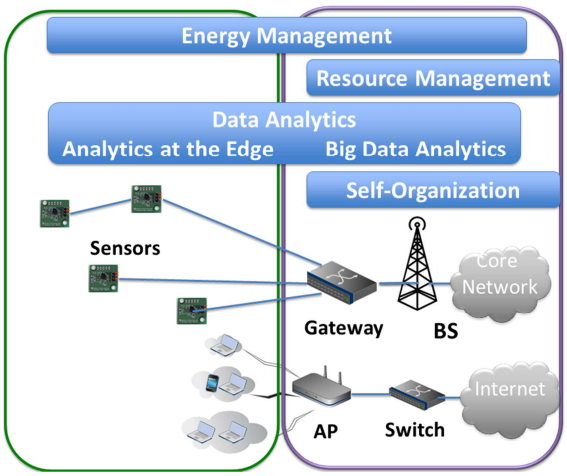


Figure 1: Virtualization layers and supporting functions

In addition, we propose the introduction of the following functionality within the layers (Figure 1): a) the Energy Management function, which spans across both the end-user and the network side - at the end-user side, an example of

such functionality are the various LEACH variants or similar protocols that can be part of the node operating system; b) the Resource Management function, which realizes the fair dynamic resources allocation to the end-user devices; c) the Data Analytics function, which is responsible for making sense of the collected information and extracting value from it, and d) the Self-Organization Function, residing at the network side to support the dynamic sensor collaboration.

IV. SCALABILITY OF COMMUNICATION AND MANAGEMENT

In order to realize the vision of ~50 billion devices connected to the Internet by 2020 [12], several scalability enablers need to be in place. One can argue that some of them are already here and they have driven the evolution towards the estimated ~10 billion interconnected devices that we have currently reached [12], [15]. Hardware node miniaturization, node capability enrichment and cost reduction, all fueled by Moore's law, are a good example of such enablers. Processing and storage availability are also improving thanks to the cloud computing paradigm. On the network protocol naming and addressing part, the transition to IPv6 has to take place sooner than later in order to facilitate the next jump in number of interconnected devices.

However, apart from the hardware node and protocol/ communication part, efficient management of this huge number of heterogeneous devices is also a big challenge. The concept of network management traditionally captures the methods and tools that are related to the operation, administration, maintenance, and provisioning of networked systems. In this context, operation is related to keeping the network working according to the specifications; administration is dealing with resource tracking and utilization; maintenance is concerned with changes and upgrades to the network infrastructure; and finally provisioning addresses dynamic, service-based resource allocation. However, catering for heterogeneous sensors and actuators deployed in Smart Cities, each with different requirements and operational properties calls for a paradigm shift; higher layers need to efficiently capture the changing dynamics of the systems and the lower layers need to transform this information into appropriate action, in an autonomous and scalable fashion.

In recent years, several extensions have been proposed to the traditional definition of network management that are specifically designed to address the topic of ever increasing network management complexity. The Self-Organizing Network (SON) notion was introduced by the 3rd Generation Partnership Project (3GPP) and targets to constitute future radio access networks easier to plan, configure, manage, optimize and heal compared to current state of the art. In similar direction, Autonomic Networking, inspired by the IBM initiated vision for Autonomic Computing [26], has been proposed as a means to create self-managing networks able to address the rapidly growing complexity of modern large scale networks and to enable their further growth, far beyond the size of today. The four main pillars of Autonomic Networking are self-configuration, self-healing, self-optimization, and self-protection, known also as self-CHOP features. However, the related technologies have so far found their way mostly in cellular networks or in smaller scale ah-hoc sensor networks. Frameworks for configurable and, to some extent, reusable deployment of SON functionality would be an important evolutionary step in the direction of scalable network management and lower maintenance cost.

V. COOPERATIVE COMMUNICATIONS AND NETWORKING

Close cooperation between network elements is increasingly seen as an important driver for further evolution. In the FIA recommendations, it is referenced, for example, that the traditional client-server model will at least partially evolve into co-operative structures between peer entities. Cooperation frameworks cover the full range from information exchange, actions coordination and decision making. Moreover, such aspects are expected to be utilized in different context, thus spanning different communication layers and capabilities. A taxonomy of cooperative and collaborative frameworks was presented in [21].

In order to achieve cooperation between networks in multi-stakeholder networking environments, proper incentives need to be in place. Such incentives formulate the expected networking benefits that a single network can derive from its cooperation with another. Networks are only motivated to cooperate with other networks when this cooperation improves their performance according to such incentives [21]. However, in order to be effective and support generalization in a large scale dynamic environment, the incentives should not express low-level performance metrics, but instead indicate high level functional and network requirements. An incentive formulates a reason for cooperation between networks (i.e., if cooperation with another network can improve this high level objective, cooperation might be viable). Example incentives are (i) increasing coverage (to reach more clients), (ii) reduce energy consumption (to increase battery life), and (iii) increasing QoS guarantees (higher throughput, higher reliability, lower delay, etc.), among others [21].

Deciding, however, on the most beneficial cooperation settings requires mechanisms such as negotiation [21], [27]. During negotiations, independent devices or complete networks with the required capabilities are identified and the utility of the cooperation is derived also as part of the cooperation incentive [28], [29], [30]. While significant research efforts have been invested in this area, large scale commercial application is still limited. Variations in the realization of the cooperation mechanisms and compatibility problems between the early products of different vendors are among the more important inhibitors; therefore ways to alleviate them will be particularly beneficial.

VI. Low Energy Solutions

Energy efficiency is commonly perceived as one of the most important design and performance factors of a Wireless Sensor Network (WSN). This fact is only expected to increase in relevance as a myriad of additional mobile and portable devices will be connected to the Future Internet. The desired low energy behavior can be achieved by optimizing the sensor node as well as the communication protocol [31]. The goal is to reduce energy consumption and, consequently, increase the lifetime of the system.

At the level of the independent nodes, the fundamental limit of the energy requirements is calculated by taking into account the energy consumption of every hardware (HW) component on a WSN node like sensors and conditioning electronic circuitry, processing and storage, radio, etc. The components selected in the final node architecture will have a significant impact on the nodes' capabilities and lifetime. Thus, a holistic low-power system design should be pursued from the very beginning, creating the correct HW infrastructure base for further network, protocol, software and algorithmic energy efficiency optimization.

This holistic low-power system approach can further incorporate methods for energy harvesting from the environment in order to utilize ambient energy sources (e.g., mechanical, thermal, radiant and chemical) that will allow extending lifetime and minimizing or possibly removing the need for battery replacement. Such a scenario would enable the development of autonomous wireless sensor networks with theoretically unlimited lifetime. Still focused on the sensor node level, but on the algorithmic part, ongoing efforts are targeting to design the sensor nodes in an inherent power-aware approach. The goal is to develop an adaptable system that is able to prioritize either system lifetime or output quality at the user's request.

Optimizations for low energy are a relatively mature field that has been (in different forms) around for a long time. For example, the radio communication and network protocol part is a major source of energy consumption that is often targeted for optimization. However, most of the available solutions are not directly transferable across different verticals and application domains.

Optimizing the network protocol is typically done with respect to a specific application domain, usually to favor bursts of transmission followed by cycles of low or no activity. As the range of transmission is also a very important parameter, low energy operation of a specific protocol version is often achieved only for a selected range, whereas other protocols are more efficient beyond that range. Thus, a certain low energy protocol is typically "optimal" only with respect to a specific communication range and bandwidth, while other solutions might be preferable outside of this area. This implies that making the best selection usually requires a thorough understanding of the specific requirements and peculiarities of the targeted application domain and environment, so that the energy optimization can be appropriately tailored to these parameters. Therefore, a more transparent on-the-fly mechanism for node reconfigurations between different Pareto-optimal states is required to enhance sensor node reusability in the context of different vertical applications.

VII. Convergence of the Sensor Network and Data Analytics Trends

In order to efficiently bring together "Things" with processes and people as envisioned by the Internet of Everything, connected "Things" will need to share higher-level information with distributed peer entities, as well as with centralized processing units or people for further evaluation and decision making. This transformation from data sharing to information sharing is considered as particularly important in the IoE notion because it will facilitate faster, more intelligent decisions, as well as more effective control of our environment [12]. Similarly, in the field of industrial automation, there is clear movement towards keeping the pace with the rapidly increasing data footprint by a paradigm shift in data acquisition and processing [13].

In parallel with these activities, a significant evolution is taking place in the data analytics domain. In this case, the trend is to evolve from "descriptive analytics" that capture what is happening to "predictive analytics" that describe what is likely to happen. Similarly, a little further down the road is the progress from "diagnostic analytics" that describe why something is happening to "prescriptive analytics" that describe what should happen, i.e., what is the optimal response. Fusion of "hard" data coming from sensors with "soft" data from, e.g., social networks (often called also soft fusion or social fusion) is another important trend in this domain, which is already going in the direction of bringing humans into the equation. "Pervasive analytics" (in some cases even referenced as "butler analytics") are envisioning to bring the power of analytics in an ever increasing range of day-to-day applications and make them available to non-experts. The relation between sensor and analytic trends is depicted in Figure 2.

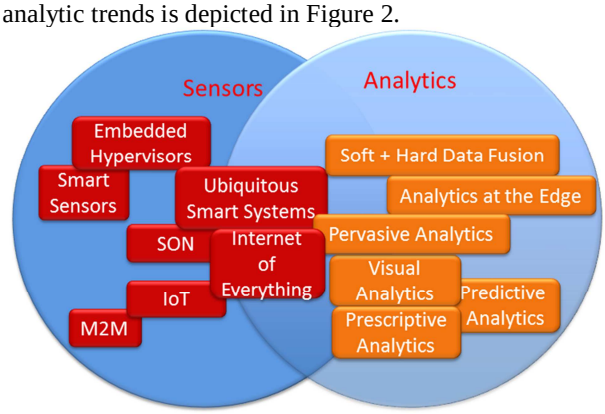


Figure 2: Convergence between sensor and analytic trends

At the same time, as the amount of data generated by this ever increasing number of sensors (augmented also by

the social fusion trend) reaches new heights, the defining 3Vs of Big Data (Volume, Variety and Velocity) require revisiting in order to cope with the new requirements. In this direction, IBM has added Veracity as a fourth dimension that captures the uncertainty of the data. And while Volume and Velocity are to some extent infrastructure planning issues, a fundamental paradigm shift might be needed in order to address Variety and Veracity in a generic framework that is able to handle the requirements of all the data types without the need to develop from scratch algorithms for each of them. Deep Learning is a novel idea in machine learning that promises to do exactly that, extracting the relevant information (features) from different types of raw data, without the need for (expensive and time consuming) manual feature engineering by human experts [32].

Although data sharing and access to sensor information enables a number of new and innovative applications beneficial for users, a major effort is needed to ensure that data protection and privacy policies are met. In order to leverage the full potential of IoT, work needs to be done beyond identity and access management – trust and reputation systems need to be introduced which can serve the needs of widely distributed and highly scalable mobile networks, while offering mechanisms to preserve privacy for the users.

Whenever users are accessing Smart City services in the IoT enabled world, identity related data must be handled according to existing regulations and principles. In order for the system to work efficiently at full capacity, sensitive data need to be exchanged between multiple devices. The challenges in the future IoE environment are even more complex as protecting privacy is evolving to a continuous effort. For example, privacy protection cannot stop with the end of the users' session as the focus is not only on protecting the identity on short term. Location of users, content of queries, as well as the footprint everybody is creating by using services in IoT is of interest [33]. Unless proper precautions are taken, aspects such as people location, previously considered very hard to trace, will become traceable. At the same time, an adversary employing an IoT enabled attack will have a vast capacity for data collection and thus a large attack surface. The research community is faced with new challenges that have yet to be fully addressed [34].

A nice example of a future Smart City / IoT service is participatory sensing enabled environmental monitoring. In this scenario people are encouraged to provide data on pollution throughout the city using measurements from personal mobile sensors. Even this simple example shows how easy the users' location together with a measurement timestamp can give more information than originally intended. The success of numerous Internet of things applications of similar nature will depend on the ability of contributors to preserve their privacy while maintaining accountability [35]. Despite the numerous challenges, some important steps in the required direction have already been made. New techniques combining anonymization, pseudonyms, and statistical disclosure control, will allow users to keep track of their privacy footprint [36], including also the information they are disclosing indirectly.

Having processed the IoT generated information by some advanced data analytics algorithm, one scenario is that certain actions are then automatically realized without human intervention. However, there are cases that the final decision process might still be desirable to be done by a human expert, especially in the context of Smart Cities in the IoE vision where people are also an important part of the equation. In the latter case, Visual Analytics are coming into play in order to make the information perceptible to humans. Visual Analytics are a combination of machine learning tools and advanced information visualization methods with the goal of facilitating analytical reasoning. Such techniques might be for example of particular interest in the detection of trends and their possible causes inside an ocean of unstructured sensor data, so that informed decisions that combine human judgment and relevant data evidence can be made.

Nevertheless, in order to apply all these advanced Data and Visual Analytics algorithms major impediments such as the limitations in bandwidth and storage (for example when dealing with devices generating a large data footprint, such as video camera streams) have to be tackled. To overcome these limitations arising from the current systems for M2M applications, novel approaches have been proposed which are based on the following principle: storing and processing the data as close as possible, both in space and time, to where they are generated and consumed, hence enabling the so-called *analytics at the edge* [37].

It is worth mentioning that developments in pervasive analytics and analytics at the edge go hand in hand, as both are aiming for migrating analytics capability to the "Things", i.e., towards the edge of the network. An indicative realization of those proposals will be defined by a content-centric platform distributed over a local cloud, hosted by the gateways or advanced edge devices with process and storage capabilities. This approach will not only alleviate the big-data problem as data is processed where it is created, but also will reduce network traffic and communication costs and can facilitate faster reactions when an event or an alarm is generated.

The desired destination in the convergence of IoT and Data Analytics is a framework of abundant sensor information taping at the "anytime, anyplace […] connectivity for anything" notion of the IoT combined with advanced analytic models that can provide real insight (in the form of human-consumable prediction and recommendation) for any situation and usable by everyone.

However, significant steps need to be taken before this vision is realized. "Analytics" is a very broad and varying field, and while wrapping them in a user friendly package is easy, using them in an irresponsive way without knowledge

or respect for possible limitations or model constraints, can be the recipe for disaster [38]. Frameworks that can provide different tradeoffs of accuracy, execution time and easiness to interpret, enforce privacy policies, and at least make the users aware of model limitations and constraints would be an important driver towards approaching this vision.

VIII. SENSOR INFRASTRUCTURE VIRTUALIZATION AS A DRIVER TOWARDS THE FUTURE INTERNET

Achieving a significant progress in the four open challenges identified in the previous sections calls for frameworks that either facilitate innovation or minimize the cost/risk for each of the four pillars identified previously (scalability, cooperation, low energy solutions and convergence of Sensor Network and Data Analytics trends). It is also important to underline that these pillars are not completely autonomous, but are mutually dependent. For example, one of the objectives of cooperation might be low energy operation, while the cooperation process by itself has to be scalable. Therefore, an important constraint is that possible solutions for each challenge are as transparent as possible to the other topics, to avoid setbacks in other fronts. A promising paradigm for addressing challenges in terms of decreasing the cost/risk as well as facilitating innovation in some of the topics identified previously is virtualization, as discussed in the Virtualization Levels Section.

Virtual Sensor Networks (VSNs) are emerging as a novel form of collaborative wireless sensor networks [39] that can establish the basis over which the evolution from connecting "Things" to the efficient interaction of the "Things" with processes and people can be realized [1]. A VSN can be formed by supporting logical connectivity among collaborative sensors [24], [39], [40]. Nodes are grouped into different VSNs based on the phenomenon they track (e.g., number of cars vs. NO_2 concentration) or the task they perform (e.g., environmental monitoring vs. traffic control). VSNs are expected to provide the protocol support for formation, usage, adaptation, and maintenance of the subset of sensors collaborating on a specific task(s).

Even nodes that do not sense the particular event/phenomenon (directly or indirectly by the notion of Virtual Sensor - VS) could be part of a VSN if they permit sensing nodes to communicate through them. Thus, VSNs can utilize intermediate nodes, networks, or other VSNs to deliver messages across VSN members. The same physical infrastructure can be reused for multiple applications, promoting scalability and resource efficiency. In addition, VSN at the end user side allows for devices sharing among several virtual networks serving different purposes/applications. This concept builds upon the Service Oriented Architecture (SOA) paradigm, which provides a flexible infrastructure and processing environment for service-based software design. SOA lays its foundation in service provision to end-user applications/other services distributed in a network and comprises functionality for describing, publishing and discovering services as well as service composition and management [41], [42]. Using SOA, each end-user device may use one or more of the available services independent of the other devices. In a similar manner, respective functionality will be supported by the VSN at the end user side for the mapping of the devices to the virtual networks, the aggregation of the application based on the functions available in each node and the over VSN management. All of these architectural considerations are relevant for creating a unified situational awareness picture of the Smart City, as discussed in Section II.

The VSNs may also evolve into a dynamically varying subset of sensor nodes (e.g., when a phenomenon develops in the spatial domain, the sensors that can detect it change over time). Similarly, the subset of the users or processes having access rights to different subsets of the VSN can vary (e.g., the people that have access to the network change with time or specific operations on a sensor network subset are only available to specific groups of people based on their role, etc.). This node grouping, merging and splitting property makes it easier to define, apply, and update policies (e.g., least privilege access) based on conceptual models rather than by configuring each of the myriad nodes independently.

Having alleviated part of the scalability and information protection/privacy requirements through the VSN concept is a good starting point for progressing on an even more ambitious front: going from data exchange between sensors to the sharing of relevant information, produced on the spot as and when required, so that it can be consumed on demand by processes and people. This paradigm is also promising to address the transmission and processing challenges that traditional large scale sensor installations face. The latter include various Big Data scalability issues with respect to the centralized gathering, logging and processing of the sensor data. The Virtual Sensor notion is instrumental in this effort.

In this paper, we use the term Virtual Sensor (VS) to refer to a software entity that can serve as an aggregation point for multiple sensors, using physical sensor entities and a computational model to combine their measurements [1]. The VS can be a thin layer of virtualization software that is executed on physical sensors (often referred as embedded hypervisor) or it can be a mathematical model for aggregating information residing in a sensor management platform similar to [41].

These different realizations of the VS notion face different types of challenges. For example, centralized or hybrid semi-centralized solutions based on analytic engines have to address the challenges of data fusion from heterogeneous sources, both functional (different credibility levels of the sensors, co-dependent sensor observations, difficulty to link human information needs to sensor control, etc.) and non-functional (scalability and performance problems, security and privacy requirements, etc.). It should be noted at this point that the non-functional requirements

such as scalability and privacy can prove critical in a Smart City context were a multitude of private and public devices need to interoperate and exchange potentially sensitive information.

At the same time, the embedded hypervisors have to cope with the integrated nature of embedded systems, and the related need for isolated functional blocks within the system to communicate rapidly, to achieve real-time/deterministic performance, and to meet a wide range of security and reliability requirements [2], [44]. In this context, bringing more processing capacity and intelligence in the end devices is both inevitable and necessary in order to cope with scalability challenges [21]. The related analytics at the edge effort (see Section VII) is focusing on realizing this transition from centralized to (semi)distributed analytics.

However, despite the different types of challenges, both embedded hypervisors and analytically/computationally realized virtual sensors share a number of key properties. First and foremost, the VS is doing more than interpolating values of physical sensors measuring the same phenomenon, as translation between different types of physical sensors is a far more interesting topic when models for the relations between the underlying phenomena are available. Furthermore, such models can even be learned by data over time in a (partially) unsupervised manner, according to the Deep Learning paradigm [32] as indicated in Section VII.

An interesting use case for this translation process in an urban setting is the estimation of car pollution based on a model that combines car counting (e.g., by induction loops or cameras) and weather conditions, while possibly utilizing also the information from the few available pollution sensors [1]. In this case, the VS can be configured to report periodically the estimated pollution value and give a warning if the pollution is above certain regulations (relevant information), instead of continuously reporting all the data.

Another example that is applicable in smart grid scenarios is the calculation of electric grid parameters (e.g., the load on given points in the transmission network, or the sag of transmission lines). Such information can be deduced by the Virtual Sensor indirectly from correlated values and a model of the related phenomena, even with a sparse network of different sensors (e.g., voltage and temperature sensors for the sag case, coupled with measurements of wind speed from a nearby weather station). Again the VS can issue warnings or alerts when some dynamic threshold values are exceeded instead of producing and transmitting all the information continuously. It is important to note that both the embedded hypervisor and the platform-based realizations of Virtual Sensors can employ state of the art signal processing techniques such as compressive sensing (for efficiently reconstructing a signal from relatively few measurements taking advantage of sparseness properties) or robust statistics (for copying with outliers, impulsive interference, etc.).

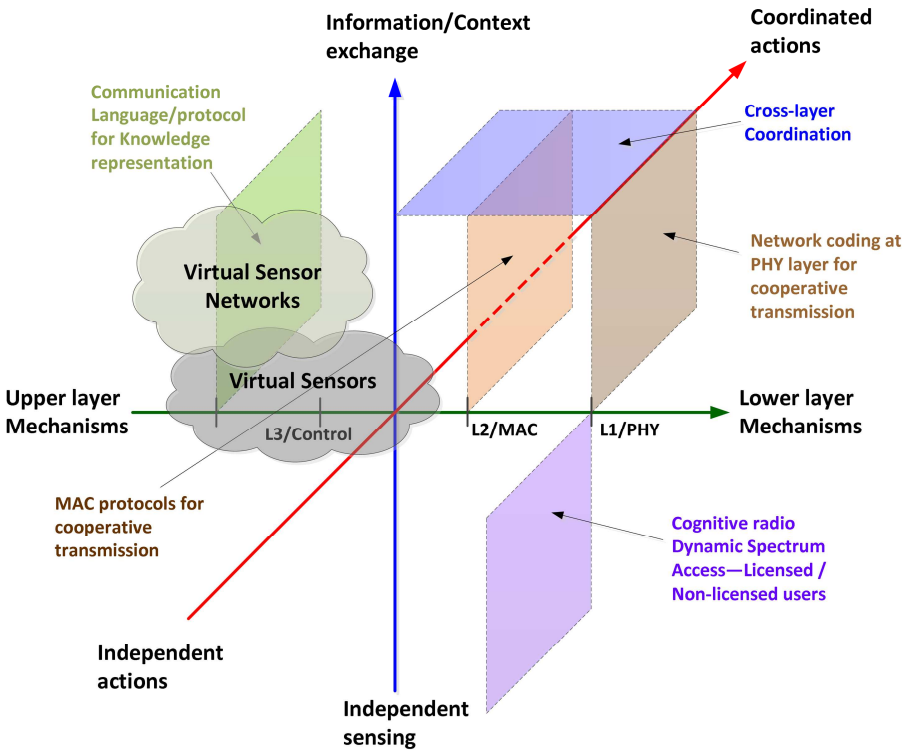


Figure 3: Sensor Infrastructure Virtulization depicted over the various dimensions of cooperative decision making and control.

At their core, VSNs and VS are building on and/or extending existing collaborative networking paradigms, therefore classifying them with respect to the ways that cooperation is realized in more conventional cooperative communication schemes is of great value. Taking into consideration the properties of Virtual Sensors and VSNs discussed previously, an updated model of the 3D cooperative methods taxonomy introduced in [21] that also captures the different sensor-level virtualization aspects is provided below. Figure 3 depicts the scope of the cooperation as planes in a 3D space. Specifically, the 3 axis are: 1) information exchange, with the extreme values being independent sensing and full context exchange, 2) decision and configuration control with the extreme values being independent actions and fully coordinated actions, and 3) layer mechanisms, with the extreme values being upper layer and lower layer mechanisms.

Each of these dimensions is being associated to a set of enablers and technical areas [1]. For example, cross-layer coordination spans the range of medium and low layer mechanisms, it requires a high information exchange level, and the level of coordination varies from medium/high to very high. Similarly, Virtual Sensors are depicted in the representation as a 3D cloud that spans medium to upper layer mechanisms. This cloud covers low/medium to high information exchange (because a VS can be either realized on the nodes as thin virtualization software or implemented as an aggregation software component running in a centralized platform). Finally, the cloud is mostly touching the area around medium action coordination since the state of the art efforts are mainly focusing more on the sensing rather than the actuation. The cloud that represents a VS can therefore expand to cover more of the axis that represents actions, in case virtualized actuation becomes more relevant in the future.

IX. CONCLUSION

The rapid proliferation in the number of devices connected to the Internet that occurred during the last decade is expected to continue, targeting yet another order of magnitude increase and promising to bring people and processes in the equation. However, in order to realize this paradigm shift, important challenges with respect to scalability, cooperative communications, energy consumption, as well convergence of sensor and analytics trends need to be addressed. In this paper, we have elaborated on the different flavors of Sensor Infrastructure Virtualization as a powerful enabler that can pave the way towards the next evolution of the IoT. The latter is expected to trigger disruptive innovation across different domains, laying the foundation for the Smart Cities of the future.

ACKNOWLEDGMENT

Part of this work has been performed under the research project FutureNET. The research project is implemented within the framework of the Action "Supporting Postdoctoral Researchers" of the Operational Program "Education and Lifelong Learning" (Action's Beneficiary: General Secretariat for Research and Technology), and is co-financed by the European Social Fund (ESF) and the Greek State.

REFERENCES

[1] A. Merentitis, et al., "WSN Trends: sensor infrastructure virtualization as a driver towards the evolution of the Internet of Things," International Conference on Mobile Ubiquitous Computing, Systems, Services and Technologies (UBICOMM), pp. 113-118, September 29 - October 3, 2013, Porto, Portugal.

[2] A. Merentitis, N. Kranitis, A. Paschalis, and D. Gizopoulos, "Low energy on-line self-test of embedded processors in dependable WSN nodes," IEEE Transactions on Dependable and Secure Computing, 2011, vol. 9, issue 1, pp. 86-100.

[3] National Intelligence Council, "Disruptive civil technologies: six technologies with potential impacts on US Interests out to 2025," Apr. 2008. [Online]. http://www.fas.org/irp/nic/disruptive.pdf, last access: May 2014.

[4] International Telecommunication Union "The Internet of Things" 2005.

[5] J. Kjeldskov, M. Skov, G. Nielsen, S. Thorup, and M. Vestergaard, "Digital urban ambience: mediating context on mobile devices in a city," Pervasive and Mobile Computing, vol. 9, issue 5, 2013, pp. 738–749.

[6] J. Kiljander, A. Ylisaukko-oja, J. Takalo-Mattila, M. Eteläperä, and J.-P. Soininen, "Enabling semantic technology empowered smart spaces," Journal of Computer Networks and Communications, vol. 2012, 2012, pp. 14.

[7] G. Kortuem, F. Kawsar, V. Sundramoorthy, and D. Fitton, "Smart objects as building blocks for the internet of things," IEEE Internet Computing, vol. 14, no. 1, pp. 44–51, Jan. 2010.

[8] D. Korzun, S. Balandin, A. Gurtov, "Deployment of Smart Spaces in Internet of Things: Overview of the Design Challenges, Internet of Things, Smart Spaces, and Next Generation Networking, 13th International Conference," NEW2AN 2013, and 5th Conference, ruSMART 2013, Springer, Lecture Notes in Computer Science, vol. 8121, 2013, pp 48-59.

[9] L. Atzori, A. Iera, and G. Morabito, "The Internet of Things: a survey," Computer Networks, vol. 54, no. 15, 2010, pp. 2787–2805. [Online]. http://www.sciencedirect.com/science/article/pii/S1389128610001568, last access: May 2014.

[10] D. Miorandi, S. Sicari, F. D. Pellegrini, and I. Chlamtac, "Internet of Things: vision, applications and research challenges," Ad Hoc Networks, vol. 10, no. 7, 2012, pp. 1497–1516. [Online]. http://www.sciencedirect.com/science/article/pii/S1570870512000674, last access: May 2014.

[11] J. Gubbi, R. Buyya, S. Marusic, and M. Palaniswami, "Internet of Things (IoT): A vision, architectural elements, and future directions," Future Generation Computer Systems, 2013, vol. 29, issue 7, pp. 1645-1660 [Online]. http://www.sciencedirect.com/science/article/pii/S0167739X13000241, last access: May 2014.

[12] D. Evans, "The Internet of Everything: how more relevant and valuable connections will change the world", CISCO White Paper [Online]. http://www.cisco.com/web/about/ac79/docs/innov/IoE.pdf, last access: May 2014.

[13] National Instrumnets, Data Acquisition Technology Outlook 2013, [Online]. http://www.ni.com/daq-trends/, last access: May 2014.

[14] P. C. Evans and M. Annunziata, "Industrial Internet: pushing the boundaries of minds and machines," GE White Paper, Nov. 26, 2012, [Online]. http://www.ge.com/docs/chapters/Industrial_Internet.pdf, last access: July 2013.

[15] J. Bradley, J. Barbier, and D. Handler, CISCO White Paper, "Embracing the Internet of Everything to capture your share of $14.4 trillion," [Online].http://www.cisco.com/web/about/ac79/docs/innov/IoE_Economy.pdf, last access: May 2014.

[16] United Nations, "World urbanization prospects 2011 revision," [Online] http://esa.un.org/unup/, last access May 2014.

[17] The Economist, "Running out of road", [Online] http://www.economist.com/node/8355114, last access May 2014.

[18] The London Dahsboard, [Online] http://data.london.gov.uk/london-dashboard, last access: May 2014.

[19] ETSI technical report, "Machine-to-Machine communications (M2M); study on semantic support for M2M data," ETSI TR 101 584, http://www.etsi.org/deliver/etsi_tr/101500_101599/101584/02.01.01_60/tr_101584v020101p.pdf, last access: May 2014.

[20] Future Internet Architecture (FIArch) Group, "Future Internet design principles," January 2012, [Online]. http://www.future-internet.eu/uploads/media/FIArch_Design_Principles_V1.0.pdf, last access: May 2014.

[21] G. Koudouridis, et al., "Enablers for energy-aware cooperative decision and control in wireless networks," Vehicular Technology Conference C2POWER Workshop, May 2011, pp.1-5, Budapest, Hungary.

[22] N. Alonistioti, et al., "Towards self-adaptable, scalable, dependable and energy efficient networks: the self-growing concept," UBICOMM, 25-30 October 2010, pp. 324-327, Florence, Italy.

[23] E. Patouni, A. Lilis, A. Merentitis, N. Alonistioti, C. Beaujean, D. Bourse, and E. Nicollet, "Protocol reconfiguration schemes for policy-based equipment management," Vehicular Technology Conference (VTC), September 2006, pp.1-5, Montréal, Canada.

[24] H. M. N. D. Bandara, A. P. Jayasumana, and T. H. Illangasekare, "Cluster tree based self organization of virtual sensor networks," GLOBECOM Workshops Nov. 2008, pp.c 1-6, New Orleans USA.

[25] NGMN Technical Document, "Suggestions on potentil solutions to C-RAN by NGMN alliance," January 2013, version 4.0.

[26] B. Jacob, R. Lanyon-Hogg, D.K. Nadgir, and A.F. Yassin, "A practical guide to the IBM autonomic computing toolkit," IBM, International Technical Support Organization, 2004.

[27] P. Magdalinos, et al., "A proof of concept architecture for self-configuring autonomic systems," ICT Mobile and Wireless Communications Summit, June 2008, Stockholm, Sweden.

[28] K. Chatzikokolakis, R. Arapoglou, A. Merentitis, and N. Alonistioti, "Fair power control in cooperative systems based on evolutionary techniques," UBICOMM, September 2012, pp. 111-116 Barcelona, Spain.

[29] A. Merentitis and D. Triantafyllopoulou, "Transmission power regulation in cooperative cognitive radio systems under uncertainties," International Symposium on Wireless Pervasive Computing, (ISWPC), pp. 537-568, 5-7 May 2010, Modena, Italy.

[30] E. Patouni, B. Fuentes, and N. Alonistioti, "A network and service governance framework: case study for efficient load balancing," in the Proceedings of the IEEE 17th International Workshop on Computer Aided Modeling and Design of Communication Links and Networks 2012 (CAMAD 2012), 17-19 September, Barcelona, Spain.

[31] G. Anastasi, M. Conti, M. D. Francesco, and A. Passarella, "Energy Conservation in Wireless Sensor Networks: a Survey," Journal of Ad Hoc Networks, vol. 7, issue 3, May, 2009, pp. 537-568.

[32] G. E. Hinton, "Learning multiple layers ofrepresentation," Elsevier Trends in Cognitive Sciences, 2007, vol. 11, no. 10.

[33] J. Braun, J. Buchmann, C. Mullan, and A. Wiesmaier, "Long term confidentiality: a survey," Designs, Codes and Cryptography, pp. 1–20, September 2012.

[34] I. Barreira, T. Gustavsson, A. Wiesmaier, C. Galan, and S. Gorniak, "ENISA Guidelines for trust services providers," ENISA Guidelines, December 2013.

[35] F. Zeiger and C. Gorecki, "Method and system for preserving privacy and accountability," Application for patent at the European Patent Office, EP2592805 .

[36] R. Steele and A. Clarke, "The Internet of Things and Next-generation Public Health Information Systems," Communications and Network, vol. 5 No. 3B, August 2013, doi=10.4236/cn.2013.53B1002.

[37] Distributed Data Mining and Big Data, Intel White Paper [Online]. http://www.intel.com/content/dam/www/public/us/en/documents/white-papers/distributed-data-mining-paper.pdf, last access: July 2013.

[38] N.N. Taleb, "The Black Swan: the impact of the highly improbable," Random House Digital, Inc., 2010, ISBN 978-1400063512, doi=10.1007/s00362-009-0226-8.

[39] L. Sarakis, T. Zahariadis, H. Leligou, and M. Dohler, "A framework for service provisioning in virtual sensor networks," EURASIP Journal on Wireless Communications and Networking 2012, pp. 1-19.

[40] N. M. M. K. Chowdhury and R. Boutaba, "A survey of network virtualization," Computer Networks, vol. 54, no 5, Apr. 2010, pp. 862–876.

[41] R. Berbner, T. Grollius, N. Repp, J. Eckert, O. Heckmann, E. Ortner and R. Steinmetz R, "An approach for the Management of Service-oriented Architecture (SoA)-based Application Systems," In: Proceedings of the Workshop Enter-prise Modelling and Information Systems Architectures, EMISA 2005, pp 208-221.

[42] M.P. Papazoglou, and W-J van den Heuvel. "Service-Oriented Computing: State-of-the-Art and Open Research Issues," IEEE Computer. v40 i11 (2003).

[43] V. Gazis, K. Sasloglou, N. Frangiadakis, P. Kikiras, A. Merentitis, K. Mathioudakis, and G. Mazarakis, "Architectural Blueprints of a Unified Sensing Platform for the Internet of Things," International Conference on Computer Communications and Networks (ICCCN), 30 July – 2 August, 2013, Nassau, Bahamas.

[44] A. Merentitis, G. Theodorou, M. Georgaras, and N. Kranitis, "Directed Random SBST Generation for On-Line Testing of Pipelined Processors," International On-Line Testing Symposium (IOLTS), 6-9 July 2008, Rhodes, Greece.

A Lightweight Distributed Software Agent for Automatic Demand—Supply Calculation in Smart Grids

Eric MSP Veith[1], Bernd Steinbach[2], and Johannes Windeln[3]

[1,3]Institute of Computer Science
Wilhelm Büchner Hochschule
Pfungstadt, Germany
e-mail: eric.veith@wb-fernstudium.de

[1,2]Institute of Computer Science
Freiberg University of Mining and Technology
Freiberg, Germany
e-mail: veith@informatik.tu-freiberg.de

***Abstract*—The number of renewable energy sources participating in the world-wide energy mix is increasing. However, they come with different characteristics than the traditional power sources. Energy generation happens on a smaller scale and is more distributed, often because the location of such a power generator cannot be freely chosen. Also, some sources like wind or solar power depend on the weather, which is not controllable. This poses more difficult challenges on every grid management. We propose a distributed, self-adjusting agent-based solution for smart grids. Based on a lightweight protocol, this distributed software will dynamically and pro-actively calculate supply and demand within the smart grid.**

***Keywords*—smart grid; messaging; protocol description; agent design; renewable energy sources.**

I. Introduction

Two major parts contribute to the success of a distributed agent software. First, the software itself, which must work and act correctly. Second, a proper method of communication must exist between any two agents. This does not only include the information interchange itself in terms of encoding, but also the correct behavior when sending or upon reception of a message.

Therefore, the ground work for any distributed software is the communication between the instances that are formed by deploying the software. In [1], we have outlined a protocol that focuses on the problem at hand: A distributed, i.e., non-centralized, supply-demand calculation.

This completely distributed supply-demand calculation is the primary goal of the architecture we propose in this article. In his article "integration is key to smart grid management" [2], J. Roncero shows how different technologies are integrated in the rather abstract smart grid concept. Including the customer via smart metering is typically considered one of the cornerstones of the smart grid. However, the increasing number of renewable energy sources with either a lower power output than a traditional power plant or a not even completely controllable output (e.g., a wind farm) will also introduce more control logic at the producer side.

Considering a country such as Germany, an already high number of 3841 wind farms [3] are controlled from only a few control centers, which oversee a part of the transmission net. Figure 1 shows control centers in Germany. Including smaller, also distributed energy-generation appliances along with photovoltaic and other renewable energy sources puts an increasing management strain on these control centers since along with the number of small generators the data volume also increases.

Figure 1. Control Centers in the power transmission system

Distributing control logic along with distributed energy generation is often proposed as a solution to this problem. Several architectures exist, such as the one described by Lu and Chen [4]. In these designs, the concept of microgrids often plays an important role. Aggregating small distributed energy generator along with consumers in a microgrid that acts as one unit to the rest of the grid is considered necessary [5]. This still views energy generators as singular blocks within the grid that yield a more or less constant behavior or can even work in island mode, completely disconnected from the rest of the power grid. While this accommodates the "central

TABLE I. Share of renewable energy sources in Germany's energy mix in 2012 [7]

Type	Power Produced [GWh]	Percentage of Total Production [%]
Water	21,200	3.6
Wind (on- and offshore)	46,000	7.7
Photovoltaic	28,000	4.7
Biogas	20,500	3.4
Geothermal Energy	25.4	0.004

control"-approach, it is also an argument for a less flexible management of the power grid.

We propose an architecture that enables every consumer and producer node in the smart grid to communicate. The primary goal is to create a self-organized smart grid allowing for greater flexibility and a more efficient usage of dynamic power sources such as wind power or photovoltaic while reducing the information load for central control facilities.

II. Motivation

The number of renewable energy sources increases steadily. For the European Union, a goal of 20%, 30% and 50% for the years 2020, 2030, and 2050, respectively has been fixed [6]. Since wind turbines and photovoltaic panels are relatively easy to set up compared to other renewable energy sources, they already contribute the biggest part of power generated from renewable energy sources (see Table I for Germany). However, they are dependent on a source of energy that is not controllable by mankind, i.e., the weather.

Having a wind farm permanently connected to the power grid means that it feeds in a greatly variable amount of energy, as can be seen in Figure 2. This contrasts with the demand of a stable power supply. In order to integrate renewable energy sources more tightly, the grid needs to accommodate this dynamic energy generation characteristic. Also, wind farms are raised at positions providing strong and steady wind currents, which is typically not ideal regarding the spread of the power grid. This and smaller, local power generators lead to a distributed generation, a paradigm shift considering traditional energy generation.

One approach is to aggregate distributed generators and nearby consumers into microgrids [4] or to compensate variations in power generation by using buffers, i.e., batteries. In [8], Vasirani *et al.* use electric vehicles as buffers that form a virtual power plant (VPP) together with the wind farm.

These approaches still assume homogeneous behavior as central attribute of the power grid and its connected devices and thus favor the traditional, simplified "base load"-view. If, however, both consumers and producers act together in a grid-wide planing phase of a short period's load profile, we assume that a more dynamic profile will emerge that allows a more efficient inclusion of renewable power sources.

In the past, the initial approach has been to place smart meters at the customers' side in order to exercise indirect control of their energy consumption. Providing dynamic rates and information feedback to the customer should contribute to

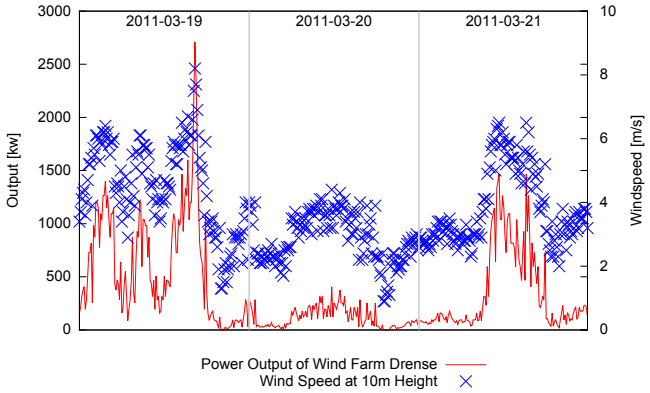


Figure 2. Modelled Output of Wind Farm *Drense* in the County of Brandenburg, Germany, and the corresponding Wind Speed during the same time

a more efficient use of energy [9]. Field studies have offered mixed results, with some even suggesting that few consumers change their behavior [10], [11]. One can also argue that electricity should be a "when you need it" resource instead of something that must be conserved.

Still, industrial consumers, i.e., factories, could accommodate to changes in the availability of energy. Also, the behavior of a large or a group of customers such as a neighborhood can yield valuable information. If a wind farm also becomes a smart node within the power grid, the additional information can be used to plan power supply more effectively.

However, this requires to introduce forecasting since we cannot control a wind farm or photovoltaic panels the same way as an operator is able to control a traditional power plant. Specifically, there is no possibility to increase power output when there is no wind blowing or sun shining. With an increasing numbers of electricity producers based on renewable energy sources, we rely more and more on a power source completely outside of our control.

Planning beforehand will therefore help to integrate these renewable energy sources better since the smart grid will be able to match a partly controllable power generation with customer behavior beforehand. This also includes a grid-wide, distributed calculation of energy storage.

To this end, we propose a protocol that defines the ground rules for such a distributed system to work. The protocol's information fields are defined by the necessity to follow these rules instead of the wish to query information. This provides us with a lightweight core allowing software agents in a smart grid to create short-lived contracts on the fly. Thus, all agents participating in this protocol act pro-actively; a consumer node is no longer a database that is queried from remote. Instead, it signals an increase or decrease in power consumption beforehand. This, in turn, triggers the mentioned supply-demand calculation.

The reminder of this article is structured as follows. In Section III we survey related work, especially paying attention to technologies useful to reaching our goal. Section IV outlines the design considerations shaping the actual protocol. The

then following Section V specifies the basic rules every node implementing our protocol must follow in order to properly do so. We then detail the actual message types in Section VI. In order to test the rules we define as inherent part of the protocol, we propose a modular agent architecture in Section VII. Although it is only a high-level view of the architecture, it identifies the important parts of an agent implementing our protocol and serves as a implementation-agnostic test case notation that we introduce in Section VIII along with a test driver proposition for an actual implementation. Selected test cases are then outlined in Section IX. We discuss all presented parts in Section X before concluding and outlining planned future work in Section XI.

III. Related Work

Several protocol proposals have been designated usable or even specifically developed for use in a Smart Grid scenario.

The *Scalable and Secure Transport Protocol* (SSTP) [12] by Kim *et al.* uses the existing IP-networking [13] infrastructure. It addresses security and durability against attacks as well as resource usage. The latter is especially important in the case of sensor hardware, where SCTP [14] alone is already too heavyweight and IPSec/TLS for encryption add to this burden. SSTP is also state- and connectionless for the same reason.

Although the authors of SSTP designate it as a "protocol for Smart Grid data collection", they do not explicitly specify data structures specific to devices or device groups. SSTP resides in layer 4 of the ISO/OSI stack just as SCTP or TCP [15] do.

The Open Smart Grid Protocol (OSGP) [16] offers a bit-by-bit protocol design that also includes transmission security. It allows remote querying of devices, mostly smart meters, which offer a virtual table-based interface. The OSGP does not utilize existing communication infrastructures such as IP networks.

The ISO/IEC 61850 standard for substation automation has also been successfully used in a smart grid scenario by Zhabelova and Vyatkin [17], where a substation shows self-healing capabilities due to a multi-agent approach that allows to detect errors and work around them.

The multi-agent approach has been chosen for several problems in the context of the smart grid. Pipattanasomporn *et al.* design and implement a multi-agent system for microgrids in [18]. In their design, the agents perform different tasks based on their roles, thereby breaking the complex problem of centralized management of a microgrid into smaller problems. Their system uses the IP. A similar approach is chosen by Oliviera *et. al* in MASGriP [19]. The decision-finding process among agents in the latter case is based on market competition.

In fact, agents competing in a market scenario is often chosen as a way to motivate an agent's behavior. The market will, so the premise, be the basis for demands and supply, and market competition will allow agents to have a scale for evaluating offers in order to select the "best" one. In fact, a price forms a simple yet effective "fitness value" for a goal-oriented behavior of agents. Hommelberg *et al.* go as far as to call automatic markets an "indispensable feature of smart power grids" [20]. For reasons we discuss in Section X, we advise against this.

Many distributed agent approaches are based on the Contract Net Protocol proposed by Smith [21]. The semantics, however, differ in the understanding of how work packages should be handled. For the contract net, a work item *can* be awarded to another node; however, each node is free to offer its services as it deems fit. This can not prevail in the power grid, where shortages *must* be handled by each agent. Considering any problem as essential for each agent is essential for all nodes to survive. Also, we do not propose any pre-selection of nodes since all agents should be able to participate in the global solution-finding process equally.

It is also important to note that many multi-agent approaches that are—directly or indirectly—based on the original Contract Net Protocol have the notion of one atomar work item that can be awarded. In a smart grid, situations will arise where a node cannot fulfill the whole contract, but only a part of it. Breaking the work package into smaller sub-packages, however, is the responsibility of the offering node. In the smart grid, this would lead to an increased amount of negotiation for the "right" work package size.

This detail excludes most service discovery protocols per se, as they are not designed offer "half a printer".

IV. Protocol Design Criteria

In striving to be as simple as possible, our protocol uses already existing technologies. This has let us to choose the ISO/OSI stack model as basis for our design, where it can be placed on the application layer (layer 7) of the stack model. This design choice allows us to draw upon the strengths of already existing infrastructure used for transport via the Internet Protocol (IP). Utilizing IPv6 [13], we gain an address space large enough for our needs.

Choosing the ISO/OSI protocol stack model also helps to integrate other technologies. We can choose between TCP/IP with IPsec for security, or SSTP, which has been specifically outlined in Section III for this purpose.

Integrating hardware in our system is possible as long as it can be attached to an IP network. Thus, we do not need to accommodate to vendor specifics and have access to a wide range of hardware through layer 2–3 protocols. For example, remote locations can use GSM or UMTS links [22] to exchange information with other nodes within our distributed system.

Nodes within the grid exchange data via *Connections*. We explicitly introduce the connection concept since it is a virtual concept not directly given by IP networks unless utilizing TCP or another connection-oriented transport protocol is explicitly chosen. Since we have already offered SSTP as possible transport protocol, we introduce the connection concept.

Here, a connection means that two nodes are known to each other; as long as message boundaries can be preserved and a loss reduction algorithm is in place, the choice of the transport protocol is up to the implementor. SSTP is therefore suitable as layer 4 protocol for our own. Connections must

be established only between two adjacent nodes and are bi-directional communication channels between exactly these two nodes. Concepts such as multicast must be realized on top of this. There is no explicit connection between two distant nodes, i.e., there is no end-to-end connection concept that crosses several hops such as TCP offers on top of IP nodes.

A connection serves three purposes. First, it identifies the two endpoints. Second, by establishing a (largely virtual) network of nodes and connections, this protocol creates a communications structure that resembles the actual power grid, recreating it on top of any other networking structure, such as an IP-based wide-area network (WAN). This way, the power grid and the telecommunications infrastructure do not have to match in their layout. The layout recreation algorithm must be implemented by the actual connection facilities, which, e.g., map to an IP network. Third, since one connection always concerns exactly two nodes, it allows us to set individual connection parameters such as compression that do not interfere with other data links to other nodes.

Having those virtual connections represent the actual physical power supply line also enables us to model "dumb" cables, which have no other properties than a maximum capacity and a line loss. Taking these attributes into account, the actual power transfer becomes part of the protocol. Smart power supply lines that are equipped with, e.g., metering devices, become nodes of their own. The simple power line–connection unit then evolves into a connection–power line–connection building block, which also adheres to the protocol semantics described in the following section.

Messages can travel further than the node–connection–node boundary. To enable nodes to answer to requests that do not originate from their immediate neighbors, each node must be uniquely identifiable. The *Sender ID* of a node must be unique at any given time. It is an opaque bit array of arbitrary length and must not contain any additionally information about the node itself or anything else. Generating an Universally-Unique Identifier (UUID) [13] whenever the node's software boots is one way to get such an identifier.

Each message must contain an unique identifier (*ID*). This is important since messages fall into two distinct categories: requests and answers. A request is sent actively by a node because of an event that lies outside the protocol reaction semantics, such as a changed forecast. Answers are reactions that occur because of the protocol semantics as described below. Since any reaction pertains to an original action, it needs to identify this action, which is the reason for the unique identifier of each message. Reactions must carry a new, unique identifier, too, since they are messages of their own.

Identifying individual messages is also important in order to identify duplicates. All messages within our design must be idempotent, i.e., they must yield the same result every time they are received. For example, a request for energy from one node must always lead to an increase in energy production by exactly the amount requested; if duplicates were not identified as such, twice or even many times the amount requested could be fed into the grid. Complex grid structures will eventually lead to duplicates, and so it is essential to identify those.

The type of the message must be denoted by a *Message Type* field. The mapping is outlined in Table II. These numbers are simple integer values with no coded meaning whatsoever. We do not distinguish between message classes or priorities here: The goal of the protocol is to remain simple, and we believe that the message types outlined here suffice in reaching the primary goal of the protocol, i.e., energy supply-demand mediation.

A message must also contain a *Timestamp Sent* field denoting the time and day when the message was initially sent as an Unix Timestamp (see [23] for the definition of the Unix Timestamp).

To prevent messages from circulating endlessly, a Time-To-Live (*TTL*) field is introduced. This TTL has the same semantics as the IP TTL [24] field: It starts at a number greater than 0. Whenever a message is forwarded or sent, the TTL is decremented by 1. If the TTL reaches 0, the packet must not be forwarded or otherwise sent but must be discarded. Messages with a TTL value of 0 may be processed.

The TTL field exists in addition to the IP TTL mechanic: First, because there is no vertical integration of our protocol with the lower-layer protocols. Even though it might seem unusual, other protocols can be used instead of IPv6 that do not offer a TTL field in the same way IP does. Second, our protocol creates an overlay network where a hop from one node to another translates to any number of hops in the IP network, which is even variable depending on different paths chosen by IP-level routers. Thus, we need a separate TTL field in order to preserve the semantics of this protocol.

Additionally, an *Hop Count* is introduced. The Hop Count is the reverse of the TTL: It starts with 0 and must be incremented upon sending a message. It allows to measure the distance between two nodes in the form of hops.

A message must carry an *Is Response* flag to distinguish original requests from responses. If the *Is Response* flag is set, the ID of the original message is contained in the *Reply To* field. If *Is Response* is not set, the *Reply To* field must not be evaluated; however, if a response is indicated, *Reply To* must contain a value that must be evaluated by the receiving system.

An answer must also contain the original message's *Timestamp Sent* field (in addition to its own), and the *Timestamp Received* denoting the time when the original message was received.

To summarize, each message must contain at least the fields of the following enumeration. In parentheses, we give the identifier used in the actual implementation.

1) message ID (`ID`)
2) message type (`type`), see Table II above
3) original sender ID (`sender`)
4) timestamp sent (`sent`)
5) TTL (`TTL`)
6) hop count (`hops`)
7) is response (`isResponse`)

TABLE II. Message Types

Value	Type
0	Null Message
1	Echo Request
2	Echo Reply
3	Online Notification
4	Offline Notification
5	Demand Notification
6	Offer Notification
7	Offer Accepted Notification
8	Offer Acceptance Acknowledgement
9	Offer Withdrawal Notification

The message type defines what additional values a message carries; these message types are described in Section VI. The message type itself is a simple integer value field with type-to-number mapping shown in Table II.

If *Is Response* is true, the following fields must be added:

1) *Reply To* (i.e., original message ID) (`replyTo`)
2) *Timestamp Received* (`received`)

V. Common Protocol Semantics

The following rules must be applied to each message, regardless of their type.

First, a message must not be ignored ("no-ignores" rule). This might seem trivial and obvious, but it is actually an essential rule: If a request for electricity or for electricity consumption would be silently ignored, other nodes would not be able to react since they probably would not even receive the message. This, in turn, would again lead to a "dumb grid" behavior. For this reason, we do not propose a mechanism for resending messages. Here, we differ from the behavior implemented in the internet. The IP routing's best effort approach is dictated by scarce buffer space. If a router's buffer is full, a packet is simply discarded; the original sender usually is not noticed about that. This behavior implicitly leads to a conservation of the node itself. However, the primary goal of a smart grid agent is to preserve the grid and not the agent itself. If an agent fails due to overload, traditional grid protection mechanism will still be available.

All messages except the *Null Message*, the *Echo Request Message* and the *Echo Reply Message* must be forwarded, partially answered and forwarded, or answered. This is the "match-or-forward" rule. It becomes important with requests and offers and it is further specified in Subsections VI-F and VI-G.

Forwarding denotes the general process of receiving a message and resending it. The message may be modified in this process, for example, the requested energy level must be lowered when a node can fulfill a portion of the request (see below).

When forwarding, message must be sent to all connected nodes except to the node from which the original message was received. If the message is an answer, the node may limit the number of outgoing connections to those via which it can reach the addressee. This prevents message amount amplification: Would the receiver also send the message on the connection on which it was originally received, it would be useless since the original sender already knows about its offer or request. It would thus only lead to additional processing and unnecessary use of bandwidth ("forwarding" rule).

Each node must keep a cache of recently received messages. If a message is received again, it must not be answered or forwarded ("no-duplicates" rule). This cache should also be used to forward answers to requests recorded in the cache only on the originally receiving connection.

If a two nodes are connected via more than one connection, only one may be used to send or forward a message. If the sending node can determine the best connection in order to reach its partner, it should choose it. However, what constitutes this "best connection" is hard to define. Many properties a connection between two nodes may have can be attributed to the lower layers of the ISO/OSI stack and, therefore, should not be known to the agent software. If, however, there were two distinct connections between two agents, with one encrypting traffic and the other one featuring low latency, these properties are not comparable in an automatic and quantified way.

Additionally, if an administrator wanted to achieve rendundancy in order to implement a resistance against failures, he can (and should) resort to proven algorithms on the ISO/OSI layers 2, 3, and 4.

We therefore advise that the administrator establishes only one connection between two nodes. As noted above, the Connection concept serves to create an overlay network and to define two connected endpoints (i.e., the agents). If there still remains a wish for redundant connections, a connection priority should be applied, and a lower-priority connection should only be used when the connection with a higher priority is disconnected.

VI. Message Types

These nine available message types constitute the minimum set that is required for the distributed supply-demand calculation. Except for the *Null*, the *Echo Request* and *Echo Reply* messages, all directly contribute to this task.

Any node must be able to signal its online or offline state. This is not only important for scheduled maintenance, but also allows a node to make itself known it its other endpoints.

During normal operation, two main cases must obviously be handled: First, a demand for energy, and second, a possible over-production that is advertised. The latter reason especially applies to renewable energy sources. A wind farm, for example, would signal an increasing power output due to increasing wind speeds.

For an power request, the corresponding *Demand Notification* must be used. It travels through the grid until either its TTL reaches 0, or it is received and answered by another node using a *Offer Notification*. In this case, the *Is Answer* flag is true. However, as stated above, an over-production can also occur, in which case the node will send an *Offer Notification* without having received a message indicating

```
{
    ID: "...eb9e90335495",
    type: 0,
    sender: "...e4d9b83d2bb7",
    TTL: 42,
    sent: 1367846889,
    hops: 23,
    isResponse: false
}
```

Figure 3. An example for a null message, encoded as JSON. The UUID strings have been shorted for clarity.

demand beforehand. The *Is Answer* flag is consequently set to `false` in this case.

What follows in both cases is the actual calculation. The requesting node—whether it requests power or requests the usage of power does not matter—receives offers or demand notifications and now has to decide which offers it takes on. Since there is no limit in how many messages related to the original request may arrive, this explicit "contract-making" needs to take place. This is the reason for the *Offer Accepted Notification*.

An explicit acknowledgment is also introduced in form of the *Offer Acceptance Acknowledgment*. The same offer could have been made to different nodes, requiring the offering node to withdraw all other offers as soon as one node takes it. This is possible using the *Offer Withdrawal Notification*.

In the following subsections, we will describe these message types in more detail, including the additional information fields they introduce along with the corresponding concepts.

A. Null Message

The *Null* message is the simplest message available in the protocol. It contains no additional information besides the basic protocol fields each message carries.

Null messages can be used as a form of heartbeat information. This is especially useful on weak links, for example for a remote wind farm, which might only have a mobile phone (GSM) connection. It thus can be sent at regular intervals to keep the line open.

A Null message in JSON representation is shown as an example in Figure 3. Please note that the message is formatted to be easy to read. In an actual transmission, the JSON string would be compressed, with unnecessary whitespace characters removed.

B. Echo Request Message

An *Echo Request* can be sent on any connection to see if the endpoint is still alive and reachable. It must be answered. An Echo Request must not be an answer, and it also must not contain any additional information.

C. Echo Reply Message

An *Echo Reply* is the answer to an *Echo Request*. It must always be an answer and thus cannot be sent independently. This message type also does not contain any additional information; the proposed common fields (*Timestamp Sent* and *Timestamp Received*) are sufficient for Round-Trip-Time measurements.

D. Online Notification

Using this message, a node in the grid can notify its neighbors that it is going online or will be online at a certain point in the future.

To actually be able to carry the second kind of information, i.e., going online at a certain point in the future, this message contains two additional fields: *Valid From* (`validFrom`) and *Valid Until* (`validUntil`). A message using validity dates must use the *Valid From* field and may optionally make use of the *Valid Until* field.

This concept of validity dates is used by other message types, too. It denotes a timespan between the time indicated by *Valid From* and *Valid To*, both inclusive. Both fields are Unix timestamps like, e.g., the *Timestamp Sent*. Whenever a node wants to indicate that a message is valid immediately, it places the current time and date in the *Valid From* field. A "valid until further notice" semantic can be achieved by omitting the *Valid Until* field entirely.

Any protocol implementor, however, must take care to adjust his implementation whenever the Unix Timestamp data type changes. As the time of writing, a Unix Timestamp of 64 bit width is typically used in modern operating systems, which provides enough seconds since midnight 1.01.1970 (UTC) for the whole lifetime of this protocol. Previously, the `time_t` C type was specified as having 32 bits, which meant that an overflow would happen on 19.01.2038, the so-called "year 2038 problem".

Note that the Unix Timestamp also allows for negative values to represent times before 1.01.1970. Although this would not be a necessary feature in the terms of this protocol, we advise against choosing an unsigned type as it would introduce the need for additional programing quirks for implementors.

An *Online Notification* may be forwarded, but can also be discarded. This type of message is important for all directly connected nodes, because it has influence on the wires connecting the originating and its neighbor nodes. Any change in power levels, however, will be communicated using demand/supply messages, which will be described later.

E. Offline Notification

The *Offline Notification* is the counterpart of the aforementioned *Online Notification*. It notifies the neighboring nodes that the originating node will be offline (i.e., possibly disconnected from the grid), utilizing the same *Valid From* and *Valid To* Timestamp fields. For all purposes of the protocol, especially for complying with the "match-or-forward" rule, the Connection to the node originally sending the Offline Notification must be considered as inactive.

Unlike the *Online Notification*, this type of message must be forwarded. It provides additional information to the energy supply/demand solving algorithms of other nodes, which get a chance to re-calculate their supply plans. It is assumed that a demand or supply message that reaches the node sending

the *Offline Notification* means that the *Offline Notification* will also be received by the original sender of the demand/supply message since the Hop Count is the same both ways.

However, since the *Offline Notification* message does not contain a field supplying the change in the grid energy level when the shutdown happens, an additional supply/demand message must be sent if the node has influence on the grid's energy level.

F. Demand Notification

A *Demand Notification* message indicates the need for energy of a particular node. It carries the *Valid From* and *Valid To* fields.

Primarily, it carries the quantified demand for energy in watts in the *Power* (`power`) field. Fractions of watts are not supported, i.e., the lowest amount that can be requested is 1 W. The field must not be 0, as this would make the message itself superfluous. This field must not carry negative values; those would mean an offer, which has its own message type.

This message type additionally features the *Answer Until* (`answerUntil`) field, which holds the date and time the requester can, at the latest, meaningfully incorporate an answer into its internal planning phase. This bit of information accommodates a wide range of different constraints that apply to a demand/supply calculation such as the time it takes to spin up turbines or scale down production in case a request is not answered as desired. This field must be present, and it must contain a time that is before the one contained in the *Valid From* field. At the time and date *Answer Until* indicates, the internal process of solution finding may begin; definite answer must not be sent until this time has arrived.

An answer arriving after the given date must not be considered by the requester. If a successful solution to the original request can not be found by offers from other nodes alone, the passing of this date and time indicates the need for an internal solution, for example, the deactivation of some turbines within a wind farm.

Demand Notification messages must be forwarded if they cannot be (completely) fulfilled. Each node must try to react to a demand message, i.e., try to match it and supply the energy requested. This is called the "match-or-forward" rule as described above. If it cannot fulfill the demand, it must at least forward it under the semantics outlined in Section IV.

If the node can supply the requested amount of energy completely, it must notify the requester using an *Offer Notification* message. It must not forward the original *Demand Notification* then.

If, however, the demand can only be partially fulfilled, the node must send an *Offer Notification* indicating the amount of energy that can be offered. It must then subtract this value from the original value indicated in the request and forward the thusly modified message. It must not change the message's ID or the message's sender ID ("same-ID" rule). The partial matching described in this paragraph is depicted in Figure 4.

A *Demand Notification* message must not be an answer.

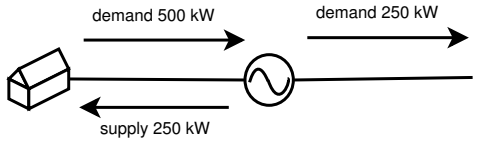


Figure 4. A *Demand Notification* having the "match-or-forward" rule applied

```
{
    ID: "deadbeef",
    type: 6,
    sender: "2d60a262",
    TTL: 42,
    sent: 1367846889,
    hops: 23,
    isResponse: false,
    validFrom: 1367846889,
    validUntil: null,
    answerUntil: 1367846289,
    power: 500000,
    cost: 12
}
```

Figure 5. An example for an *Offer Notification* message that is sent as a request to consume power in order to accommodate to an over-supply of energy. Note the `isResponse` field, which is set to `false` to express this circumstance. UUID strings are shortened for clarity.

G. Offer Notification Message

This type of message indicates an offer to the grid. It carries the fields *Valid From*, *Valid Until*, and *Answer Until* as they are described in Subsection VI-D and the amount of energy offered in the field *Power*. This number is an unsigned integer and is expressed in units of watts with no fractions possible.

Additionally, the offer includes a field *Cost*, which carries the cost of this offer in cents per kilowatt hour (ct/kWh). This allows for implementing cost-based policies, such as accepting energy only if it is cheap.

An *Offer Notification* may be an answer. If so, it is an answer to a previous *Demand Notification*, as described in the above subsection. A node receiving multiple offers must prefer offers of lower hop count over those with higher hop count. This favors micro-grids and reflects the actual flow of energy.

However, *Offer Notification* messages may also be sent as a request. This is the case whenever the agent estimates that it will output more power than it currently does. Consider, for example a wind park, which is dependent on the weather. If the agent's forecasting module predicts an increased wind speed in an hour and therefore an increased energy output, it may send an *Offer Notification* instead of pitching or stalling the wind turbines. This could allow a factories to increase its demand by powering up machines. Figure 5 shows an example of such a message.

Just like a *Demand Notification*, such an original offer must be matched by nodes in the grid. The difference between an original offer and one that is an answer to a request is the

value of the *Is Answer* field: If set to `false`, a node must try to match the offer.

For matching and forwarding, the same mechanics as for the *Offer Notification* message type applies, especially if it can only be partly fulfilled.

H. Offer Accepted Notification

Whenever a request for energy is made and the offers have been received, there may arise a situation when more energy is offered by all nodes than originally requested. For example, if a wind park, a solar park and a traditional power plant send *Offer Notification* messages after a request has been sent, the sum of energy offered is likely to exceed the original amount requested.

For this reason, a node must indicate which offer it accepts. Otherwise, all offers would be fulfilled, leading to an oversupply of energy in the grid, which would be fatal.

As soon as the node finishes its demand/supply calculation, it must send *Offer Accepted Notification* messages to all nodes that were offering energy. In the body of the message, it must list the IDs of those nodes whose offer it takes, using the *Accepted Offers* field (`acceptedOffers`). All other nodes will notice that their ID is missing from the notification and thus not actually deliver the energy they offered.

An *Offer Accepted Notification* must be an answer. It must also be sent by the node that is taking on an original offer (as indicated above). In that case, the *Offer Accepted Notification* must be addressed to the offering node only, while the original offer must be forwarded if it cannot be completely fulfilled as described in Subsection VI-G.

I. Offer Acceptance Acknowledgement

After an offering node has received an *Offer Accepted Notification*, it must reply with an *Offer Acceptance Acknowledgement* to indicate that the offer is still valid. This message type must always be an answer.

J. Offer Withdrawal Notification

If a node has offered a certain amount of energy, be it as an answer or as an original offer, and it can no longer stand up to the offer, it must withdraw it. This type of message is always an answer, carrying the ID of the original offer (in case of an original offer that was withdrawn) or the ID of the original request in the *Reply To* field.

If a node can still offer energy, but the amount has changed, the original offer must be withdrawn using this message type, and the new amount must be separately announced.

VII. Implementing the Distributed Agent

The protocol itself defines the ground rules of a distributed supply/demand calculation. In order to actually test it, however, an implementation adhering to the rules is necessary. Our own implementation consists of several modules, each contributing to a part of the agent's behaviour. We will use the route of an incoming message as common theme for describing all parts of our design, although, of course, information can flow in any direction. We will come to other reasons for action later.

Each agent has exactly one *Messaging Module* that is its interface to the rest of the world. It maintains connections to other agents, and is responsible for receiving and correctly sending messages.

The Messaging Module contains the Duplicate Message Cache. Each message received is first checked against this cache; only if it is not yet stored in this cache will it reach the other modules. Otherwise, it will be silently discarded. It is important to keep this filter in mind during the following paragraphs as every notion of a message will mean an unique sending from another agent.

This message cache is regularly cleared of old messages. For our tests, we have chosen a message retention period of 15 minutes. An implementor can, of course, choose another value. However, he must keep in mind that the Duplicate Message Cache with its retention period is vital to preserve the idempotence of all messages. A period that is too short will harm the grid. The only reason to lower this period is to preserve memory.

Instead of simply "throwing more hardware at the problem" and setting a higher, but fixed time period for message retention, a semi-dynamic cleanup should be implemented. The Messaging Module can infer from Offer Acceptance Acknowledged messages that a particular planning phase has ended and clean up its cache accordingly. Of course, a maximum retention period should still exists as a fall back.

When sending messages, the Messaging Module also takes care to use the correct agent connection, which particularly includes the application of the "forwarding" rule.

An incoming, unique message is then routed to the *Governor*. It has two main tasks. First, during startup, it initializes all other modules, including the Messaging Module, monitors them throughout the lifetime of the agent instance and is finally responsible for resource deallocation on shutdown. Second, it contains the business logic that allows it to act on incoming messages, machine sensor readings or forecasts. The latter one is, obviously, essential to the actual agent behaviour.

Upon receiving a new message, the Governor creates a `Requirement` class instance. Requirements are the building blocks the agent uses internally for bookkeeping and the actual demand/supply calculation. Therefore, it contains two attributes: The actual power delta and the immediately associated messages. The power delta is a deviation from a balanced state of the grid, i.e., it is a relative value. Since it can attain both positive and negative values, the `Requirement` class allows us to treat both an anticipated excess in energy offered as well as an anticipated demand uniformly. Consequently, Demand Notifications and Offer Notifications constitute two different Requirement instances that are matched against each other in the agent's demand/supply calculation.

The requirement class can store power deltas as IEEE 754 [25] *binary32* single precision floating point variables. Variances as they typically occur when using this data type and prompt developers to use `x + 1.0 == 1.0` when

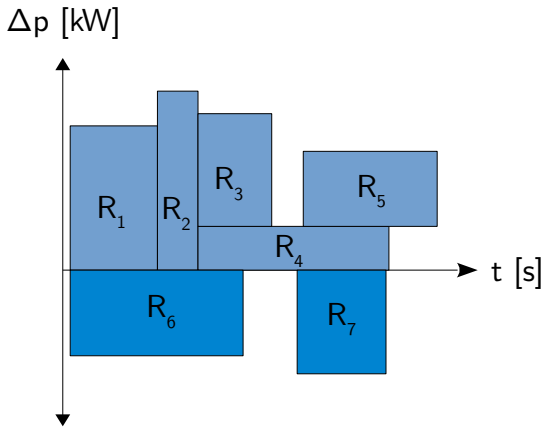


Figure 6. A simplified illustration of the timeline concept

checking whether `x` is 0 or not are small enough as to not harm the grid. However, an implementor should incorporate proper safeguards against over- and underflows.

This is the main reason for the introduction of the classes in the `Winzent::Unit` namespace, such as `KiloWatt`. History has shown that simple floating point or integer variables can lead to a mixing of units in calculations with possibly horrible results, such as the loss of the Mars Climate Orbiter [26].

This calculation is done in the *Supply/Demand Module*. Requirements are inserted into a timeline, which can be viewed as a graph representing the power variances over time. For the module, $\Delta p = 0$ means a balanced grid, but not a power level of 0.0—hence the delta. It might be noteworthy that the Supply/Demand Module at no point has any knowledge of an absolute energy level, but only needs relative levels in order to work.

Figure 6 shows a very simplified, symbolic illustration of the timeline. Requirements form blocks that are inserted into the timeline. The module tries to find the optimal solution within the search room of all blocks. Positive and negative deltas cancel each other until the grid is, from the perspective of the agent, balanced again. From an electric engineer's point of view, a total balance of $\Delta p = 0$ will hardly ever be achieved or even be desirable; we retain this formula for the sake of simplicity but point out that, when deploying, a shift will have to be taken into account.

This search is triggered by the `answerUntil` field of the Offer Notification and Demand Notification messages. Typically, the module has no simple 1-to-1 matching of offers and demands, but can choose from a number of blocks, including the possibility of the agent to adjust itself. For example, a wind farm signalling an offer pro-actively originating from a forecasted increase of wind speed can either try to literally collect Demand Notification messages until the excess energy is used completely, or it can throttle itself and pitch or stall turbines.

This exemplary case also introduces another module within the agent: The *Forecast Module*. Incoming requirements must be matched—or, at least, the agent must try—, and thus the basic question is: "Can we scale up (or down) in order to accommodate the new situation?" The Forecast Module therefore contains the logic of the node's ability to change its production or consumption.

How this is done, depends on the actual node. A traditional power plant will start its own planning phase in order to spin up or down turbines, whereas a wind farm will try to forecast weather conditions. Such a local forecast could be done using Artificial Neural Nets, which have already been proposed and successfully used for weather forecasting, for example in [27], or even in connection to load forecasting [28]. An incorporation of weather forecasting in our agent is still future work as we detail in Section XI.

As we previously noted, the Supply/Demand Module does not have knowledge about absolute numbers. In an ideal world, this is not a problem as the agent's foremost goal is to provide a stable power supply. However, constraints limit the solution space. Such a constraint is hardware-based, e.g., in the form of transformers, which have a limited capacity. All solutions are therefore first checked against the output of the *Constraints Module*. This module also allows administrator interaction, which gives us the possibility to set policies, for example, to not accept a power offering exceeding a certain cost.

As stated, this largely forms the way the agent behaves upon incoming messages. However, this is obviously not the only source of activity for a node—that initial request has to come from somewhere. Within the agent, the Forecast Module continuously creates new projections for the node. Once this forecast has a variance that exceeds a certain limit, it creates a Requirement of its own. The Governor then prompts a new demand/supply calculation, which will, in many cases, yield a deficit, i.e., no solution to the current situation. This, in turn, prompts the Governor to create a request of its own, i.e., a Demand Notification or Offer Notification message, which is sent to other agents.

Although this might seem an obvious course of action, it is not negligible. The continuous forecasting and adjustment of forecast constitutes the very source of our agent's pro-active behavior. Thus, it is not a finite state machine at its heart, but a long-running, stateful software agent.

All parts described live in the `Winzent::Agent` namespace as depicted in Figure 7.

VIII. Testing the Protocol

A. Notation

To ensure that our protocol implementation, or, in fact, any implementation of our protocol adheres to the rules defined in the previous sections, we have created a test suite. This test suite consists of two parts:

1) A written definition for test cases, initial situation and expected results
2) A software implementation of the unit tests

The latter is tied to the implementation that is being tested. We therefore provide the definition of our test cases in order to document how we assert that the behavioral rules defined

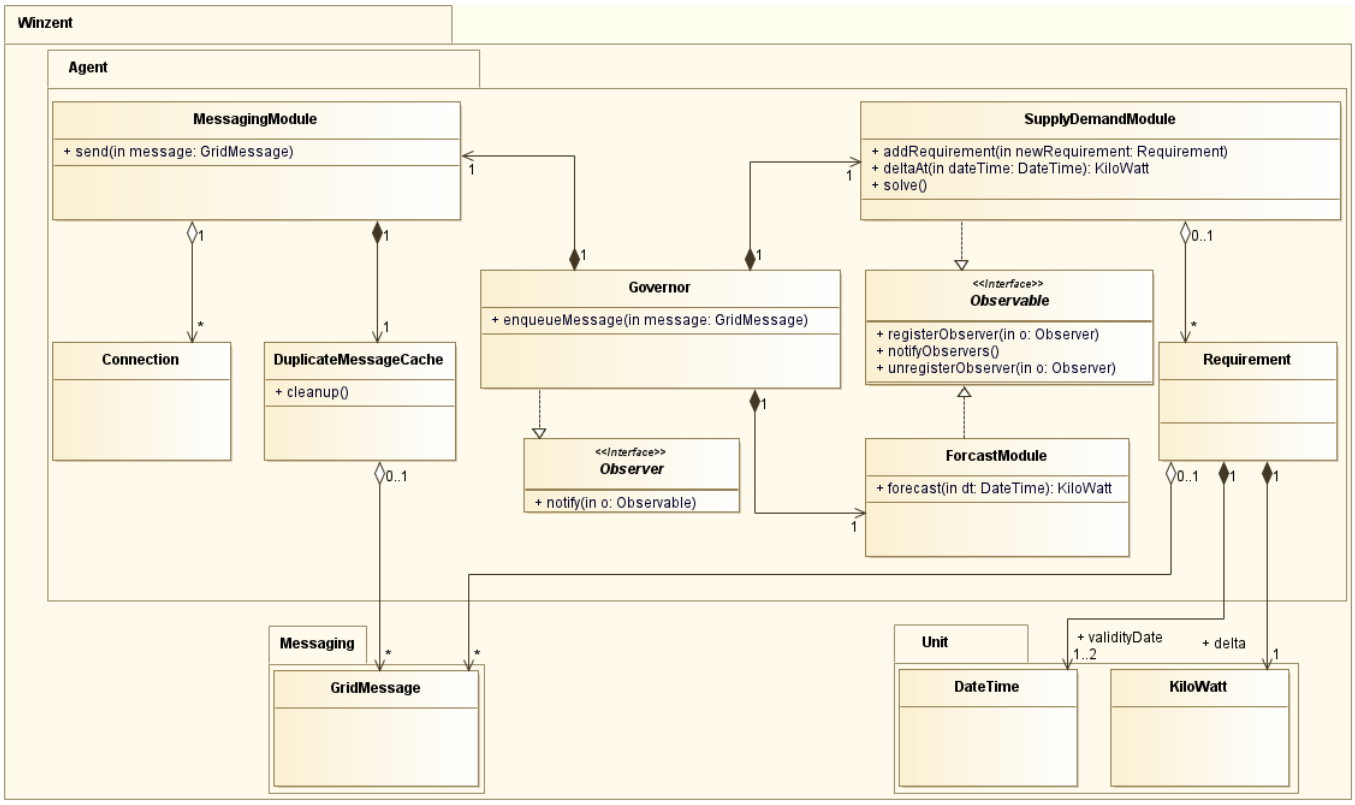


Figure 7. The Winzent Agent design

through the protocol itself ensure that the system as a whole works correctly if every agent adheres to the rules.

In our definition, we denote agents with upper case letters, starting from A. Connection between agents are written down as tuples. Since all connections are bi-directional, we simply order the letters by the alphabet. For example, (A, C) defines a data link from agent *A* to agent *C* and, at the same time, a link from *C* to *A*.

We further define the inner state of any agent also by the following quadruplet:

$$Agent = (Messaging, Forecast, Demand/Supply, Constraints)$$

This can be shorted to (M, F, D, C) for brevity. A module of a specific agent has the agent's letter as subscript, e.g., F_M denotes the Messaging Module of agent *F*. A subscript x denotes a do-not-care, i.e., "any node" or "any value".

Each module, finally, also has a state. Initial as well as final module states are sets of items specific for the particular module. All noted sets are interpreted as subsets of the actual state. For initialization, this allows for local or implementation-specific extra values, while for the final state, it defines the way a successful or failed test run is determined. The final state set must be a real subset of the actual state set of an agent: $Required \subset Actual$. This is especially necessary for the Messaging Module, where a correctly working agent implementation can send Echo Request messages at any time, which would lead to test case failures without this definition.

The state of the Messaging Module is defined by a list of messages; we simply note the JSON text representation as it is already quite easy to read. Message fields that do not matter for the final result are omitted; in this regard, it follows the real subset rule already employed for the module states.

The `id` field is never originally considered for subset matching since it is opaque and implementation-specific to begin with. However, we use it on a meta level to identify individual messages in our notation. This way, we can track messages on their way. The same technique is applied for senders and receivers, where the actual ID of any agent is similarly opaque. For example, `{ answerTo: "m1", sender: "A" }` would correctly describe a message that is an answer to another message identified as m_1 in our notation coming from agent *A*. In reality, not only would a full message travel across the line, but also would it contain ID strings like `4c23a34fab0`.

In order to keep definitions clean, we refer to individual messages with lower-case m letter with number subscript, $m_i | i = 1, 2, 3, \cdots, n$. The combination of these specifics allows us a more efficient notation: We can identify the original message through the m_i notation and refer to it using the `id` field in our notation while leaving out all fields that do not change. Also we do only note those messages that were received since they are already stored in the duplicate message cache. Otherwise, duplicate messages on forwarding

would clutter the notation too much.

Please note that an ASCII-based JSON text does not allow subscripts and thus m_1 becomes `m1`, but otherwise remains the same.

The Forecast Module is defined by a list of forecasts notes as tuples in the form of $(Timestamp, Power)$. Alternatively, for the initial state, an alternate form containing only the forecast is also used; in this case, these values are emitted when queried.

The Demand/Supply Module features a similar tuple form, i.e., $(Timestamp, Delta)$.

Constraints typically do not change over time. Aside from the fact that the Constraints Module is also defined as a possibly empty list of single constraints, they must be understandable by a human. For example, the maximum power that can be forwarded by a node could be written as $C_x = ((P_{max} = 1000kW))$.

In cases where the state of a module does not change from the initial setup, we simply note the upper case letter in the final state definition. If the state is of no interest, we use the subscript x notation, meaning "any value". E.g., $A = (\cdots, F_x, \cdots)$ would mean that, for the results, any state of the Forecast Module is allowed for Agent *A*.

B. The Test Driver

Although our notation abstracts the actual tests from the details of an agent implementation, it leaves an implementor with the task of carefully reading a written definition and creating actual unit tests out of it. This will again create tests that only check whether an implementation works with a set of tests specific to exactly this implementation. The only advantage the written definition provides here is that it forms a common ground to agree upon when it comes to the scenarios that deserve testing.

Also, most implementation-specific unit test suites will be exactly that: Code written in an imperative style. To that end, a developer has translated the set form of the original definition in source code. If two implementations yield different results, the unit test code will have to be re-translated, at least implicitly in the developer's mind, to the set definition in order to find out where things go wrong. This is obviously an error-prone process.

For this reason, we have defined a test driver that reads a JSON representation of the notation we introduced in the previous paragraph and sets up a test bed for the agents. The adapter an implementor has to create comes in the form of interfaces or abstract classes.

The test driver is initialized using the `Manager` class, which sets up the necessary environment. It creates `TestCase` objects, with one object representing one distinct test case. This class reads and parses the JSON notation of the test case itself and is responsible for setting up the test, running it and cleaning up afterwards. Success or failure is indicated by the return value of the `run()` method, which is a simple Boolean value indicating success on true or failure on false.

During setup, agents are created and initialized according to the initial state description. This is the responsibility of the `AgentFactory` class. This factory, along with the `Agent` class instances it creates, is an abstract class: The concrete factory as well as the concrete agent must be implemented by the vendor wishing to test his product. Along with the interfaces representing the modules, i.e., `MessagingModule`, `ForecastModule`, `DemandSupplyModule`, and `ConstraintsModule`, this forms the API an implementor must use when attaching his own agent code.

Although this API carries the spirit of the design we propose in this article, it can be understood as nothing more than a mere wrapper; the concrete classes implementing the module interfaces can be shallow wrapper classes.

All these module interfaces simply provide a getter and a setter for a set of objects. The setter is used during initialization to establish the initial state, while the getter allows us to retrieve the final state. The `Set` class finally implements two set primitives: `equals()` and `isSubsetOf()`. These two are necessary for testing the success of a test case. Since the final state consists of sub sets of the actual state, the success or failure of any test case boils down to a number of subset checks. If, and only if, all succeed, the test case itself succeeds.

The test driver architecture itself does not need many classes in order to provide the necessary API. Figure 8 shows an overview in form of an UML class diagram.

In order to work, this architecture needs test case definitions in a computer-parsable format. We have again chosen JSON for this for the same reasons we use it for the message format: It is easy to read and write for a human and likewise easy to parse for a computer. Also, reliable parsers exist, i.e., it is no obscure, exotic format.

A test case is a JSON object consisting of three root attributes: a list of agents, a list of connections, a list of initial states, and a list of final states. Basically, it is a transcription of the formal definition in JSON that adjusts the written notation to the idiosyncrasies of JSON's syntax.

The list of agents, `agents`, simply introduces the agent IDs in much the same way the formal notation does. The same is true for the connection list, `connections`, that contains tuples in the form of JSON arrays with two elements.

The last two attributes, `initialStates` and `finalStates`, contain the state sets of the agents. All agents are listed here along with their modules: `messagingModule`, `forecastModule`, `demandSupplyModule`, and `constraintsModule`. Each of them contains a list with items as defined in our formal representation. As variables with subscripts do not exist in JSON, references and do-not-cares, e.g., F_x or M_A are strings without subscripts in the same way as we refer to messages: m_1 becomes `"m1"`, and subsequently M_A becomes `"MA"`, D_x is written as `"Dx"`.

Figure 9 has an exemplary test case definition where two agents, *A* and *B* exist. For the test case to succeed, *A* is required to send an Echo Request message to *B*, which the latter one

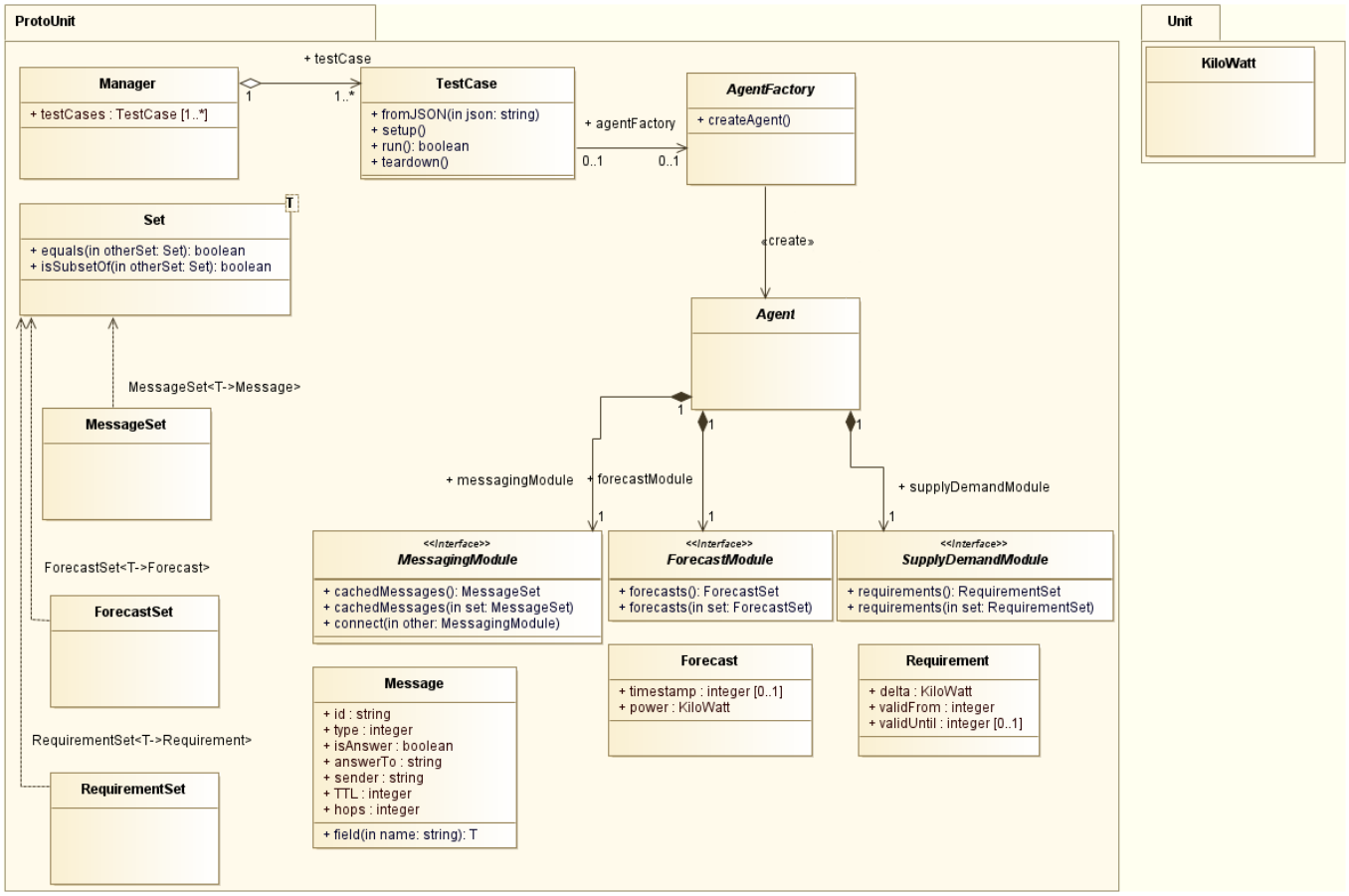


Figure 8. The Test Driver implementing the Test Case notation

answers with an Echo Reply message.

IX. Selected Test Cases of the Protocol Test Suite

This chapter illustrates how we test the semantics of the protocol and the correct functioning of our implementation using selected test cases. We refrain from publishing the full source code of each test case for the sake of readability. Also, this does not constitute a complete suite of test cases. Its intent is to illustrate not only the application of the implementation-agnostic description, but also to show how nodes that correctly implement the protocol behave.

Therefore, we illustrate the important parts using the test case notation introduced in the previous Section VIII.

A. Test Case: "Forward" Rule

Three agents are connected in a linear fashion, i.e., *A*—*B*—*C*. The correct notation of this layout is:

Agents (A, B, C)
Connections $((A, B), (B, C))$

A sends a Demand Notification message for $500kW$, which *B* and *C* cannot answer. The test case ends when *C* receives the message that *B* has to forward. This test case proves the correct working of the "forwarding" rule.

Thus, for the initial state, we need only to define the Demand/Supply Module of all three agents, all other state tuples remain empty. The final state then has to show that a message has travelled to both nodes *B* and *C*, but no answer has been transmitted. Agent *A*'s Demand/Supply tuples therefore remains the same, while the Messaging Module tuple of *A* and *B* each have to list one message.

The complete source code must contain a definition of this message. Both its *Type* and *Power* fields have to be defined in order to indicate that *A* and *B* have received a Demand Notification message. We thus note:

Initial State

$$A = ((), (0, -500000), (), ())$$
$$B = ((), (0), (), ())$$
$$C = ((), (0), (), ())$$

Final State

$$A = ((), F_x, (0, -500000), C)$$
$$B = ((m_1), F, D, C)$$
$$C = ((m_1), F, D, C)$$

Together with the definition of the message m_1, this test case completely defines initial and final state of the simulated

```
{
    agents: ["A", "B"],
    connections: [ ["A", "B"] ],
    initialStates: {
        A: {
            messagingModule: [],
            forecastModule: [],
            demandSupplyModule: [],
            constaintsModule: []
        }, B: {
            messagingModule: [],
            forecastModule: [],
            demandSupplyModule: [],
            constaintsModule: []
        }
    },
    finalStates: {
        A: {
            messagingModule: [{
                    type: 2,
                    hops: 1,
                    sender: "B"
                }],
            forecastModule: [],
            demandSupplyModule: [],
            constaintsModule: []
        },         B: {
            messagingModule: [{
                    type: 1,
                    hops: 1,
                    sender: "A"
                }],
            forecastModule: [],
            demandSupplyModule: [],
            constaintsModule: []
        }
    }
}
```

Figure 9. A Test Case Definition in JSON notation for a "ping" example: *A* sends an Echo Request message to *B* that the latter one answers.

system concerning the transmission of messages and the agent's reactions to it.

B. Test Case: TTL

The setup is similar to the previous test case, however, an additional node *D* with connection (C, D) is introduced. The message TTL of *A*'s Demand Notification is 2. Thus, *C* may not forward the message on the connection (C, D) and the final state of $M_D = ()$ must exist.

Also, the notation of messages needs to change. *A* and *B* have now received messages which have changed regarding the value of the *TTL* field. Thus, we note the TTL field with the changed value for both messages and therefore need to assign different indexes to them:

Final State

$$A = ((), F_x, (0, -500000), C)$$
$$B = ((m_1), F, D, C)$$
$$C = ((m_2), F, D, C)$$
$$D = (M, F, D, C)$$

Even though we need to distinguish the two messages in the test case definition by writing m_1 and m_2, we can still show that the message carries the same ID. We do so by indicating the *ID* field of m_1 and setting it to the identifier m_2. Thus, the message can be traced while still showing that a part of it has changed its value.

C. Test Case: Simple Demand and Supply

This test case will, again, use the topology of the first test case. But now, agent *C* is able to completely answer *A*'s request for energy.

This seems to be the simplest of all test cases, but, in fact, more messages than for the previous ones are required. Here, we need to explicitly confirm the offer using an Offer Accepted Notification.

This test case serves to check an implementation for all defined rules of behavior. It applies the "forwarding" rule and will modify a message's *TTL* field in the same way as the other two test cases, combined.

Additionally, it shows the application of the "match-or-forward" rule. The agent *C* is required to answer the Demand Notification, which can be checked using *B*'s message cache. We discuss this imperative in Section X. Also, the correct formation of the implicit contract can be monitored.

The creator of the test case therefore needs to track a number of messages: First, the forwarding of the initial Demand Notification with modified TTL Values. Second, the Offer Notification message which travels back to the requester and must be recorded in both *B*'s and *A*'s Message Module's duplicate request cache. In the same way, the Offer Accepted Notification is sent by *A* and reaches *C* via *B*, being recorded in the same way as the previous two messages. Finally, an Offer Acceptance Acknowledgement is sent by *C* to *A*.

With the reception of the final message, the Message Module of *A* and *C* must contain two received messages, while that of *B* holds copies of all four. The reception of the Offer Acceptance Acknowledgement also marks the formation of the contract between *A* and *C*. Thus, the Supply/Demand Module of the requester must appear as balanced in the final state definition.

D. Test Case: Circular Demand/Supply with Partial Offers

This test case offers a more complex topology featuring one requester and three suppliers. Since the setup is harder to imagine than the previous ones, instead of a written prose description or the formal connections notation, it is depicted in Figure 10.

This test case serves to test the protocol-implementing nodes' behavior on a more complex topology. It also uses

partial offers, which increases the amount of messages that need to be sent in order to arrive at a working contract.

Initially, the requesting node *A* sends a Demand Notification message for the same amount used in the previous two test cases. However, now three other agents answer with an offer.

First, the agents *C* and *F* will offer one half of the requested power, while agent *H* is able to supply the complete power needed. This tests not only the "forwarding" rule, but also "match-or-forward". The agents *C* and *F* not only have to send their offer, but also forward a modified version of the initial request. This forwarded version can be found in *G*'s Messaging Module's cache.

G and *H* also serve to test the correct implementation of the "no-duplicates" rule. *G* will, via *E* and *D*, receive two modified Demand Notification messages. However, it may forward only one of the two. This means that the cache of *H* may contain exactly one Demand Notification message which can easily be identified using its *ID* field as the one originating from *A*. The transmission of the Demand Notification message therefore stops at *G* and also *B*, effectively preventing an endless circulation.

This original requester will finally have three Offer Notifications received. Since we advise preferring messages with a lower hop count, it will accept the two partial offers. This Offer Acceptance Notification must be recorded by all three offering agents. This is vitally important since all parties have now knowledge of whether they need to provide the advertised power or not.

Applying the same rules, the final Offer Acceptance Acknowledgement messages will then travel through the network back to *A*.

Please note that it is not necessary to list all transmitted messages in order to show that the actual demand/supply scenario is solved. In fact, the Message Module tuple of *A*'s final state alone suffices to show that three agents have sent their offers. Listing its Demand/Supply Module definition serves to verify that it has accepted the two partial offers.

However, if all described messages are recorded in the test case definition, we can also show that an endless message circulation is prevented by the duplicate message cache. Therefore, this additional information asserts an important part of the behavioral rules of this protocol and are included.

X. Discussion

A. Comparison with SIP and RSVP

Our protocol seems to share some features with already deployed, well-known protocols such as the Session Initiation Protocol (SIP) [29], or the Resource Reservation Protocol (RSVP) [30]. SIP is especially designed to to be usable on exiting protocols in Layers 1–6, thus being not vertically integrated similar to our protocol. However, there are several differences that justify the creation of this Layer 7 protocol.

Traditional SIP relies on proxy servers. Here, individuals register in order to be locatable. The proxy server typically serves a domain and makes up the domain part of a SIP URI, e.g., `sip:bob@biloxi.com`. These proxy servers create a layer of indirection in performing their duty. In our approach, there could theoretically be any number of proxy servers from 1 to *n*, with *n* being the number of nodes in the grid. These proxy servers create points of failures while also partially obscuring the direct mesh that is used for routing the offers and demand messages.

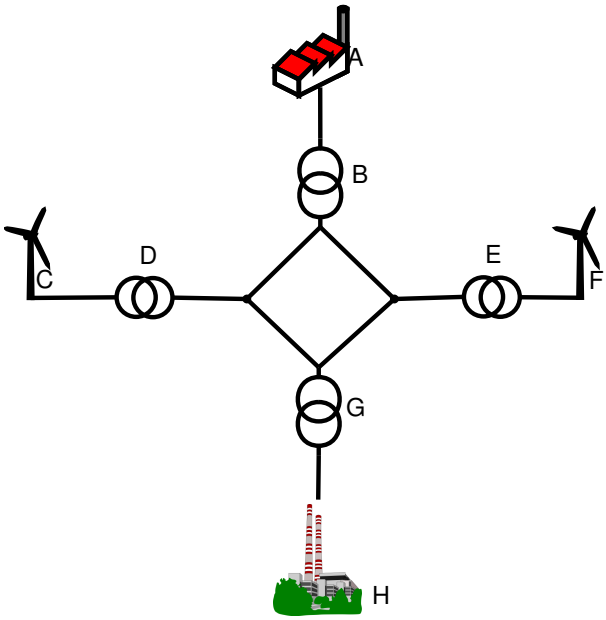


Figure 10. Topology of a circular demand/supply test case

Using a Peer-to-Peer architecture (P2P) seems to be the next logical step in order to rely on already existing architectures. *SOSIMPLE* by Bryan, Lowerkamp, and Jennings [31] is a "serverless, standards-based, P2P SIP communication system". The routing of requests still requires an a-priori knowledge of the (potential) location of the callee in the grid: "Node A is not responsible for that Resource-ID, so it sends a SIP 302 Moved Temporarily reply, including the node it thinks is closest [...] in the headers [...]."

Both SIP and RSVP are based on the premise that the initiating client knows its counterpart, i.e., the caller knows its callee as well as the video-requesting client knows which server offers the desired video. In our case, however, there is no a-priori knowledge about potential contract partners. Each node has the same chance to match an offer, and therefore, our protocol relies on and uses the meshed architecture of the power grid that is re-created in the communication network.

In that regard it also becomes apparent that there is no explicit session that is created and maintained by the protocol. Sessions require a setup, potentially keep-alive and a teardown. However, our protocol does not need the explicit notion of sessions that are maintained. The power grid itself provides the "session" since nodes act and react based on the state of the power grid itself.

It can also be noted that we do not establish an end-to-end connection in the protocol. It is the basis for a negotiation, but the actual connection—if one would call it so—is done in the power grid itself by initiating the flow of energy based on the

negotiation that took place earlier and was facilitated by the protocol.

In summary, intermediate nodes are not required to keep track of contracts made in order to reserve a specific capacity once the flow of power has been initiated. We assume that a node that requests a certain amount of power for a certain amount of time does so truthfully, i.e., that it actually consumes or delivers the power. In contrast, traffic on an Internet channel such as one created by RSVP does not necessarily flow continuously.

B. Message Transmission Volume

The test cases show clearly that there is room for improvement regarding efficiency: A lot of messages travel the information network in order to establish a contract. Many of those are redundant and are caused by the "forward" rule that, in its current, simple form, resembles a routing by flooding algorithm [32].

In fact, in our testcases no actual routing in the sense of selecting a single path is performed at all. For large-scale deployment, this behavior must change into a better version that reduces the number of messages transmitted. Ideally, a node would transmit the message only on sensible connections instead of all but the receiving ones. The intial broadcasting of requests is necessary in order to reach all potential contract partners as there is no central registry of potential suppliers and consumers. However, when those have made their offers, the multicast character of messages can transform into an unicast nature, thus using the existing communication facilities in a more efficient way.

We believe that we can use the already existing Duplicate Message Cache in order to identify a minimal set of outgoing connections for answers. Using this local piece of knowledge, the routing can be optimized for replies. This assumes that no cache expiry timer on a node along the path the answer takes has yet led to the removal of the necessary piece of information.

In comparison to other protocols in the smart grid area, the raw data volume of our proposal seems rather high. The example Offer Notification message in Figure 5 compromises 170 Bytes if all unnecessary line breaks and whitespace characters are removed. If transmitted via standard TCP/IPv6, protocol headers add another 32 Bytes (TCP) and 40 Bytes (IPv6) for a grand total of $170 + 32 + 40 = 242$ Bytes. Including the TCP three-way handshake and the connection termination increases the number to 762 Bytes for a single message. Adding in IPsec raises the grand total even more.

Of course, our Connection concept does not force a permanent setup and teardown of a TCP connection for every message. Using SSTP instead of a solution based on IPsec and TCP will also reduce protocol overhead. And since all agent Connections are end-to-end connections, two nodes can employ compression, e.g., using simple GZIP [33].

Currently, we compute the overall data volume for a contracting process using the following formulae. All variables are summarized in Table III.

First, the transmission volume of a singular message m on an established connection c can be calculated by:

$$v_u(m,c) = s(m)f(m,c) + V_c \quad \text{Bytes}$$

The message's size is given by calculating $s(m)$, i.e., the size of the message. $f(m,c)$ denotes a dynamic cost factor of the connection like compression. The constant V denotes the constant costs of the connection c, such as TCP/IP headers that get added to each transmission.

Forwarding a request generates costs on all connections of a node n except the receiving one c_0, i.e.,

$$v_m(n,m) = \sum_{i=1}^{|C_N|-1} v_u(M,c_i) \quad \text{Bytes}$$

A request is forwarded at most TTL hops, i.e., the request's initial TTL limits the number of hops it can be forwarded. Through the duplicate request cache we ensure that the message will not pass any node twice. Therefore, the maximum cost of transmitting a request message m from an initial node n_0 is given as

$$v_r(m,n_0) \leq \sum_{i=0}^{TTL_M} v_m(n_i,m) \quad \text{Bytes}$$

Since answers can be transmitted in an unicast fashion thanks to the duplicate message cache, the volume of an answer equals the transmission volume of any message, with c_0 being a connection on which the initial request was received. Choosing the "right" connection is the responsibility of the node; using the first receiving one will typically suffice.

Thus, each answer produces at least

$$v_a(m_r,n_r,m_a) = \sum_{i=0}^{h_{m_r}} v_u(m_a,c_0) \quad \text{Bytes}$$

The total volume of bytes each request generates is therefore the sum of the volume a request produces plus the sum of all answers that are sent by other nodes. If the Duplicate Message Cache is not used as a means to optimize the path an answer takes, v_a equals v_r on all nodes. This effect is apparent in the test cases we showed in Section IX.

Although optimization techniques such as using the Duplicate Message Cache where possible are employed, our protocol uses a substantially higher volume in comparison with bit-by-bit defined protocols such as OSGP. However, we approach a different problem. OSGP or the IEC protocols access highly integrated devices with low data rate links in order to query sensor, usage and billing information. However, the protocol we propose acts on the level of a whole neighborhood, a wind farm, factory or traditional power plant in order to enable an efficient, on-the-fly demand/supply calculation.

C. Choice of Offers

In cases where several solutions to the demand/supply calculation emerge, we do not enforce any priorities. We do, however, recommend to prefer messages with a lower Hop

TABLE III. Variables and functions used in the message volume calculation

c	An established connection
C_N	Set of all established connections on node n: $\{c_0, c_1, \ldots, c_n\}$
c_i	ith established connection on a node
n	A node
m	A message
m_r	A request message
m_a	An answer message
h_m	Hop count of message m
TTL_m	TTL of message m
$s(m)$	The size of a message m, in bytes
$f(m, c)$	Dynamic factor by which the message m will be modified when transmitted via connection c
V_c	Static additional costs of a connection
$v_u(m, c)$	Total volume to transmit message m on connection c
$v_m(m, n)$	Total volume to forward a message m using multicast on node n

Count, even if this leads to a solution compromised of more and "smaller" building blocks. In fact, the last test case we describe in this article explicitly checks that a decision is made on exactly this basis.

A lower Hop Count means that a node nearby has sent the message. Our network architecture practically constitutes an overlay network over existing IP-based ones and effectively re-models the existing power grid. As a result, the power that flows on grounds of a message with lower Hop Count bridges less meters of the grid than that flowing due to a message with higher Hop Count. This leads to a lower (transmission) grid load and smaller overall line loesses since it automatically prefers micro grids. Additionally, the Constraints Module can be used to implement cost-based policies.

However, a focus on costs should be avoided. A pure "cents per kilowatt hour" metric can be influenced to an amount that diminishes or even destroys its value as an objective criterion. For example, government subsidies can lead to huge distortions. A study by the German Federal Environmental Agency [34] shows that subsidies of coal and nuclear power plants greatly influence the price per kilowatt hour.

It is obvious that this does not yield to the technically best solution, or a solution that is the best from a grid-wide supply point of view. Instead, a price-based fitness metric as acting basis can even lead agents to hold back offers because the price is too low. However, we see the preservation of the power grid itself as the highest priority, whereas a profit margin is an optimization problem that arises once more than one solution can be considered.

But in comparison, it is clear that the Hop Count metric is very abstract. Power may travel many kilometers with just one hop, depending on the grid layout. Other metrics may be better applicable. Takeru Inoue *et al.* use actual physical metrics in their article [35], which will be a better application for a decision-making algorithm in the future.

Both the *Demand Notification* and *Offer Notification* message types include timestamps. Especially the *Answer Until* field is noteworthy, because it takes part in timing the start of a demand/supply calculation on a node. Any node can set a meaningful time considering its own characteristics that it knows about, like hardware constraints that require a certain time buffer in order to employ a fall-back solution, or to have a search room populated enough to arrive at a meaningful result in a local demand/supply calculation.

It is tempting to include information network constraints in the time buffer the *Answer Until* field provides. However, we advise against it for two reasons:

First, a node does not have knowledge about the latency to other nodes when it broadcasts its initial request. When answering another node directly, e.g., using a *Offer Notification*, it would have to measure the latency beforehand to include a meaningful value. In internet communication, this latency would reside in the area of milliseconds, which are not included in the Unix Timestamp that makes up the `answerUntil` field.

Second, even when packets are routed in an extremely inefficient way [36], a high delay means values of $< 300ms$, while even fast gas turbine power plants react in a matter of minutes ("Boosting times of a few minutes including synchronization to the grid are possible", translated from [37]).

XI. Conclusion and Future Work

In this article, we have defined a lightweight protocol based on behavioral rules that enable a distributed supply/demand calculation for smart grids. It allows nodes to act pro-actively based on their local energy situation and to propagate a future demand or over-supply of power. This, in turn, initiates a distributed, automatic search for a solution to this problem. This way, renewable energy sources can be used more efficiently since consumers can make use of an increased supply or scale down dynamically based on the local knowledge of individual nodes.

We have also proposed an architecture for an agent implementing this protocol and the rules related to it. This architecture serves as basis for our test cases, which are not just unit tests tied to a specific software, but also provide a written-down definition of a successful execution of a test case.

However, the way we define tests today is based on a textual representation. While this is good for parsing and to create a definite collection of implementation-independent test cases, it is clear that all topologies that are not extremely simple are best visualized. That is why we also created a graphical simulation environment, which we will publish in a separate paper.

Currently, we employ a very simple routing algorithm for requests that resembles a classical "routing by flooding" approach, as noted in Section X. We plan to employ all agents to be more aware of their immediate neighbors in order to relay requests more efficiently without using separate registry servers.

In this article, we have not touched the problem of how connections are initially created, but have simply assumed that they already exist. Connections can be a tool to negotiate individual data link parameters such as encryption for low data

rate links and can even be used to identify potentially malevolent nodes by implementing a credibility-based algorithm in the same manner as it is already done in current peer-to-peer networks. However, the actual algorithm how nodes could automatically connect to their immediate neighbours, negotiate parameters and shield themselves against attacks is still future work.

XII. Acknowledgments

This article has been created as part of a cooperative doctorate program between the TU Bergakademie Freiberg and Wilhelm Büchner Hochschule, Pfungstadt.

References

[1] E. M. Veith, B. Steinbach, and J. Windeln, "A lightweight messaging protocol for Smart Grids," in *EMERGING 2013, The Fifth International Conference on Emerging Network Intelligence*. IARIA XPS Press, 2013, pp. 6–12.

[2] J. R. Roncero, "Integration is key to smart grid management," *CIRED Seminar 2008 SmartGrids for Distribution*, no. 9, pp. 25–25, 2008, retrieved 2013-02-11. [Online]. Available: http://link.aip.org/link/IEESEM/v2008/i12380/p25/s1&Agg=doi

[3] M. Pierrot, "Wind energy data for germany - country windfarms," retrieved 2013-12-10. [Online]. Available: http://www.thewindpower.net/country_windfarms_en_2_germany.php

[4] M. Z. Lu and C. L. P. Chen, "The design of multi-agent based distributed energy system," *2009 IEEE International Conference on Systems, Man and Cybernetics (SMC 2009), Vols 1-9*, pp. 2001–2006, 2009.

[5] B. Lasseter, "Role of distributed generation in eeinforcing the critical electric power infrastructure," pp. 146–149, 2001.

[6] European Parliament, Council, "Directive 2009/28/ec of the european parliament and of the council of 23 april 2009 on the promotion of the use of energy from renewable sources and amending and subsequently repealing directives 2001/77/ec and 2003/30/ec (text with eea relevance)," Official Journal of the European Union, 04 2009, date of effect: 25/06/2009; Entry into force Date pub. + 20 See Art 28.

[7] Bundesministerium für Umwelt, Naturschutz und Reaktorsicherheit, "Datenreihen zur Entwicklung der erneuerbaren Energien in Deutschland," p. 41, February 2013.

[8] M. Vasirani, R. Kota, R. L. G. Cavalcante, S. Ossowski, and N. R. Jennings, "An agent-based approach to virtual power plants of wind power generators and electric vehicles," *IEEE Transactions on Smart Grid*, vol. 4, no. 3, pp. 1314–1322, 2013.

[9] A. Faruqui, S. Sergici, and A. Sharif, "The impact of informational feedback on energy consumption—a survey of the experimental evidence," *Energy*, vol. 35, no. 4, pp. 1598–1608, 2010.

[10] P. Merrion, "Pilot test of comed's smart grid shows few consumers power down to save money," *Crain's Chicago Business*, May 2011.

[11] B. M. Buchholz, V. Bühner, U. Berninger, B. Fenn, and Z. A. Styczynski, "Intelligentes lastmanagement — erfahrungen aus der praxis," in *VDE-Kongress 2012*. VDE VERLAG GmbH, 2012.

[12] Y.-J. Kim, V. Kolesnikov, H. Kim, and M. Thottan, "SSTP: a scalable and secure transport protocol for smart grid data collection," in *IEEE International Conference on Smart Grid Communications (SmartGridComm)*. IEEE, 2011, pp. 161–166.

[13] P. Leach, M. Mealling, and R. Salz, "An universally unique identifier (UUID) URN namespace," July 2005, retrieved 2013-05-25. [Online]. Available: http://tools.ietf.org/html/rfc4122

[14] S. Kent and K. Seo, "Security architecture for the internet protocol," IETF RFC 4301. [Online]. Available: http://www.ietf.org/rfc/rfc4301.txt

[15] R. Stewart, "Stream control transmission protocol," RFC 4960, Sep. 2007. [Online]. Available: http://www.ietf.org/rfc/rfc4960.txt [Retrieved 2013-05-13]

[16] European Telecommunications Standards Institute, "Open smart grid protocol," European Telecommunications Standards Institute, Tech. Rep., 2012.

[17] G. Zhabelova and V. Vyatkin, "Multi-agent smart grid automation architecture based on iec 61850/61499 intelligent logical nodes," *IEEE Transactions on Industrial Electronics*, vol. 59, no. 5, pp. 2351–2362, 2011, retrieved 2013-04-03. [Online]. Available: http://ieeexplore.ieee.org/lpdocs/epic03/wrapper.htm?arnumber=6018303

[18] M. Pipattanasomporn, H. Feroze, and S. Rahman, "Multi-agent systems in a distributed smart grid: design and implementation," *2009 IEEE/PES Power Systems Conference and Exposition*, pp. 1–8, March 2009, retrieved 2013-06-01. [Online]. Available: http://ieeexplore.ieee.org/lpdocs/epic03/wrapper.htm?arnumber=4840087

[19] P. Oliveira, T. Pinto, H. Morais, and Z. Vale, "MASGriP — a multi-agent smart grid simulation platform," in *IEEE Power and Energy Society General Meeting*. IEEE, 2012, pp. 1–8.

[20] M. Hommelberg, C. Warmer, I. Kamphuis, J. Kok, and G. Schaeffer, "Distributed control concepts using multi-agent technology and automatic markets: An indispensable feature of smart power grids," in *IEEE Power Engineering Society General Meeting*. IEEE, 2007, pp. 1–7.

[21] R. G. Smith, "The contract net protocol: high-level communication and control in a distributed problem solver," in *IEEE Transactions on Computers*, vol. C, no. 12. IEEE, December 1980, pp. 1104–1113.

[22] V. C. Gungor, D. Sahin, T. Kocak, S. Ergut, C. Buccella, C. Cecati, and G. P. Hancke, "Smart grid technologies: communication technologies and standards," *IEEE Transactions on Industrial informatics*, vol. 7, no. 4, pp. 529–539, 2011.

[23] K. Thompson and D. M. Ritchie, *UNIX programmer's manual*. Bell Telephone Laboratories, 1975.

[24] S. Deering and R. Hinden, "Internet protocol," 1998, retrieved 2013-05-14. [Online]. Available: http://tools.ietf.org/html/rfc2460

[25] "IEEE standard for floating-point arithmetic," *IEEE Std 754-2008*, pp. 1–70, 2008.

[26] E. Euler, "The failures of the mars climate orbiter and mars polar lander—a perspective from the people involved," in *Guidance and Control*, 2001, pp. 635–655.

[27] I. Maqsood, M. Khan, and A. Abraham, "An ensemble of neural networks for weather forecasting," *Neural Computing and Applications*, vol. 13, no. 2, pp. 112–122, May 2004. [Online]. Available: http://link.springer.com/10.1007/s00521-004-0413-4

[28] S.-T. Chen, D. C. Yu, and A. R. Moghaddamjo, "Weather sensitive short-term load forecasting using nonfully connected artificial neural network," *IEEE Transactions on Power Systems*, vol. 7, no. 3, pp. 1098–1105, 1992.

[29] J. Rosenberg, H. Schulzrinne, and G. Camarillo, "SIP: Session initiation protocol," *Vasa*, pp. 1–269, 2002, retrieved: 2014-05-10. [Online]. Available: http://www.hjp.at/doc/rfc/rfc3261.html

[30] R. Braden, L. Zhang, S. Berson, S. Herzog, and S. Jamin, *Request for Comments*.

[31] D. A. Bryan, B. B. Lowekamp, and C. Jennings, "SOSIMPLE: A serverless, standards-based, p2p SIP communication system," *First International Workshop on Advanced Architectures and Algorithms for Internet Delivery and Applications (AAA-IDEA'05)*, pp. 42–49, 2005. [Online]. Available: http://ieeexplore.ieee.org/lpdocs/epic03/wrapper.htm?arnumber=1652335

[32] M. Abolhasan, T. Wysocki, and E. Dutkiewicz, "A review of routing protocols for mobile ad hoc networks," *Ad hoc networks*, vol. 2, no. 1, pp. 1–22, 2004.

[33] P. Deutsch, "GZIP file format specification version 4.3," IETF RFC 1952, 1996, retrived 2013-12-08. [Online]. Available: http://www.ietf.org/rfc/rfc1952.txt

[34] A. Schrode, A. Burger, F. Eckermann, H. Berg, and K. Thiele, "Environmentally harmful subsidies in Germany," Federal Environment Agency, Germany, Dessau-Roßlau, Tech. Rep., 2011.

[35] T. Inoue, K. Takano, T. Watanabe, J. Kawahara, R. Yoshinaka, A. Kishimoto, K. Tsuda, S.-i. Minato, and Y. Hayashi, "Distribution loss minimization with guaranteed error bound," *IEEE Transactions on Smart Grid*, vol. 5, no. 1, pp. 102–111, January 2014, retrieved 2014-01-15. [Online]. Available: http://ieeexplore.ieee.org/lpdocs/epic03/wrapper.htm?arnumber=6693788

[36] Renesys, "The new thread: targeted traffic redirection," November 2013, retrieved 2014-01-01. [Online]. Available: http://www.renesys.com/2013/11/mitm-internet-hijacking/

[37] K. Heuck, K.-D. Dettmann, and D. Schulz, *Elektrische Energieversorgung*. Wiesbaden: Vieweg + Teubner, 2010.

Redundancy Method for Highly Available OpenFlow Controller

Keisuke Kuroki, Masaki Fukushima, and Michiaki Hayashi
Integrated Core Network Control and Management Laboratory
KDDI R&D Laboratories, Inc.
Saitama, Japan
e-mail:{ke-kuroki, fukusima, mc-hayashi}@kddilabs.jp

Nobutaka Matsumoto
Department of Evolved Packet Core Network Development
KDDI Corporation
Tokyo, Japan
e-mail:nb-matsumoto@kddi.com

***Abstract*—OpenFlow is an important element for achieving Software Defined Networking (SDN) and is expected to be an enabler that solves the problems of today's network. Thanks to the centralized management with OpenFlow, agile network operation can be achieved with flexible programmability; however, the centralized management implies a significant impact of any outages of the OpenFlow controller. Hence, a high availability technology is indispensable for building the OpenFlow controller. To achieve the highly available system, we have to consider extraordinary events (e.g., power outage) affecting the entire data center as well as anticipated server failures within a local system. In this paper, we review the issue in using the conventional redundancy method for OpenFlow controllers. Based on this observation, we propose a redundancy method considering both local and global (i.e., inter data-center) recoveries using the multiple-controllers capability that is defined in OpenFlow switch specification version 1.2 and later. The proposed redundancy scheme eliminates virtual IP address-based redundancy and frontend server causing limitation of performance scalability, while it achieves competitive role change and failover times.**

Keywords-OpenFlow; controller; redundancy.

I. INTRODUCTION

This paper is an extended version of our previous work [1]. Towards future telecom services, the programmability of the network is expected to shorten the service delivery time and to enhance the flexibility of service deployment meeting diversified and complex user requirements on various applications (e.g., real-time and non real-time applications). Software Defined Networking (SDN) is an important concept for achieving a programmable network and OpenFlow [2] is an important factor for achieving the concept. OpenFlow is an enabler of the centralized management solution, which enables management and control of several OpenFlow switches, which allows the network operators to configure the switches easily and speedily. However, we have to solve some issues of OpenFlow (i.e., scalability, reliability and so forth) to deploy the OpenFlow technique in carrier grade networks. Many researches have addressed the issues of the OpenFlow-based solution.

Fernandez evaluates several OpenFlow controllers from the viewpoint of scalability in centralized management and control [3]. Message processing performances of two operation modes (i.e., proactive and reactive) of the OpenFlow controller are evaluated using several existent implementations (e.g., Floodlight, NOX, Trema). Pries et al. analyze the scalability of the OpenFlow solution for a data center environment to show an implementation guideline [4]. The paper concludes that, to achieve lossless and low delay performance in the data center application, the number of OpenFlow switches managed by one controller should be limited to eight. To leverage the advantage of centralized management, the OpenFlow controller should not be a simple flow switching policy server. OpenQoS [5] architecture delivers end-to-end quality of service (QoS) with OpenFlow-based traffic control. The OpenFlow controller with OpenQoS plays the role of collecting the network state to perform dynamic QoS routing, i.e., the controller has a route calculation function just like the Path Computation Element (PCE). Indeed, in the Internet Engineering Task Force (IETF), PCE architecture is growing as a stateful operation supporting the enforcement of path provisioning in addition to its original path computation role. Hence, the importance of the OpenFlow controller is growing with the broader concept of SDN, and thus the high availability of the controller system must be discussed.

There are two approaches to achieve high availability of the OpenFlow controllers. One approach is to reduce their load. The OpenFlow controller exchanges many messages with the OpenFlow switches especially in reactive mode. As a result, the OpenFlow controller could be overloaded and thus become unable to process incoming messages. In such a case, some processing is required to handle failover. If the OpenFlow Controller uses Link-Layer Discovery Protocol (LLDP) [6] messages to discover link and node failures and manages and monitors several switches, the monitoring model has serious scalability limitations. Kempf et al. [7] propose a monitoring function for OpenFlow switches that achieves a fast recovery in a scalable manner. Dixit et al. [8] propose a new OpenFlow switch migration algorithm for enabling load shifting among the OpenFlow Controllers. This algorithm improves the response time for the Packet-in messages by shifting the controlled switch. Thus, there are some researches on reducing the load of the OpenFlow controller for protection of the data-plane.

The other approach is to replace a single controller with redundant controllers. However, there is little research on the redundancy of the OpenFlow controller, which must play an important role in SDN.

In this paper, we investigate the issue of achieving redundancy for the OpenFlow controller with a conventional method, and we propose a method to improve the availabil-

ity of the OpenFlow controllers. In the proposed redundant method, "global" recovery (i.e., inter data-center redundancy) as well as local recovery (i.e., redundancy within a local network) are considered. The proposal achieves a competitive failover time compared with existing redundant schemes (e.g., server clustering), while the proposal does not require any frontend server limiting performance scalability of the OpenFlow controller.

The organization of this paper is as follows: In Section II, we review related works and the capability of multiple-controllers as defined in OpenFlow switch specification 1.2 [9] and also explain its applicability to achieving redundancy of the OpenFlow controller. In Section III, we describe a conventional method to achieve the redundancy of the OpenFlow controller by using the Virtual Router Redundancy Protocol (VRRP) and its limitation. In Sections IV-A and B, we propose the redundancy method using multiple-controllers in a single domain and evaluate its performance. In Sections IV-C and D, we propose the redundancy method using multiple-controllers in multiple domains and evaluate its performance. Finally, concluding remarks are given in Section V.

II. Background and Related Work

Typical implementation of OpenFlow allocates a controller separating the control plane from the data plane, and an OpenFlow switch playing the role of data plane communicates with an OpenFlow controller using the OpenFlow protocol over a Transport Layer Security (TLS) [10] or a Transmission Control Protocol (TCP) connection [11] defined as an "OpenFlow channel." The switch tries to forward a packet by looking up flow entries populated in advance by the controller. If the packet does not match the current flow entries, the switch sends a packet-in message over the OpenFlow channel to the controller in order to retrieve a direction on how to treat the packet.

One method of handling data plane failure is to implement a monitoring function on the OpenFlow switch; however, only the monitoring function in a data plane is not sufficient for achieving high availability in an OpenFlow network. We cannot achieve a highly available OpenFlow network without achieving the redundancy of the OpenFlow controller. In the case of controller outages, the OpenFlow channel is lost accordingly, and then the controller cannot successfully process the packet-in message. Hence, new packets that are not matched with the flow entry are simply dropped or allowed to fall in a default operation (e.g., forwarding to a neighbor anyway) that does not provide desirable services until the ultimate recovery of the controller. To achieve a high availability in the OpenFlow network, we have to achieve recovery methods in both global and local networks that exploit the redundancy of the OpenFlow controllers.

The HyperFlow [12] approach improves the performance of the OpenFlow control plane and achieves redundancy of the controllers. HyperFlow introduces a distributed inter-controller synchronization protocol forming a distributed file system. HyperFlow is implemented as a NOX-C++ application and synchronizes all events between controllers by messaging advertisements. In the case of controller failures, HyperFlow requires overwriting of the controller registry in all relevant switches or simply forming hot-standby using servers in the vicinity of the failed controller. Thus, this approach assumes re-establishment of the OpenFlow channel, and does not assume the multiple-controllers capability defined in OpenFlow 1.2. Therefore, the time duration of the failover operation may increase with the growth of the number of switches managed by the failed controller. Since the failover process of HyperFlow does not consider any server resource, overload of CPU utilization is a potential risk in the event of migrating switches to a new controller especially in the global recovery scenario.

There are several methods of general server redundancy, and such methods may also be effective for OpenFlow controllers. For example, one possible server redundancy can use one virtual IP address aggregating hot-standby or several servers. Koch and Hansen [13] evaluate a failover time in the case of using the virtual IP address-based implementation with the Common Address Redundancy Protocol (CARP), which is similar to VRRP [14]. According to the analysis, the average time to change the role between master and backup is 15.7 milliseconds. However, the virtual IP address-based approach may take a longer failover time in the case of applying this approach on the OpenFlow network because this approach involves the re-establishment process of the OpenFlow channels. We discuss this issue in Sections III and Sections IV-A. Although the virtual IP-based scheme is straightforward if it is applied within single LAN, it cannot simply be applied to multiple locations (e.g., data centers) managed under different addressing schemes. This means that the virtual IP-based scheme alone is not sufficient to tackle global recovery. Zhang et al. [15] propose a server clustering method with a mechanism to seamlessly handover the TCP connection between backend servers. While each TCP connection is visible to only one backend server in a normal clustering scheme, the proposal [15] makes the connection visible to at least two back-ends using proprietary backup TCP (BTCP) protocol within a backend network. The connection migrates to a backup, and then the backup is able to resume the connection transparently before the client TCP connection is lost. Using this scheme, the connections are recovered by the backup server within 0.9 seconds including a failure detecting time of 0.5 seconds. This approach is expected to be applicable also for global recovery involving multiple locations. However, from the viewpoint of the performance scalability of the OpenFlow controller as analyzed in [3, 4], a common frontend server required in the clustering system can be a serious bottleneck of message processing in the control plane (e.g., if the frontend server is broken, all TCP connections are lost). The high availability scheme should avoid such single frontend server to ensure the performance scalability of OpenFlow controllers. In addition, when we tackle global recovery with many switches, the migration process should also consider the server utilization. However, conventional approaches do not consider utilization of the server resources (e.g., CPU).

OpenFlow specification 1.2 introduced the capability of multiple-controllers by defining three states (i.e., MASTER, SLAVE, and EQUAL) of a controller. A controller plays its own role by using the multiple-controllers capability, and the state itself is owned by the switch. In the three states, MASTER and EQUAL have full access to the switch and can receive all asynchronous messages (e.g., packet-in) from the switch. A switch can make OpenFlow channel connections to multiple EQUAL controllers, but the switch is allowed to access only one MASTER controller. In the SLAVE state, a controller has read-only access to switches and cannot receive asynchronous messages apart from a port-status message from the switches. A controller can change its own state by sending an OFPT_ROLE_REQUEST message to switches. On receipt of the message, the switch sends back an OFPT_ROLE_REPLY message to the controller. If the switch receives a message indicating the controller's intent to change its state to MASTER, all the other controllers' states owned by the switch are changed to SLAVE. This function enables a switch to have multiple OpenFlow channels, and thus the switch is not required to re-establish new OpenFlow channels in the event of controller outages. In the multiple-controllers capability, the role-change mechanism is entirely driven by the controllers, while the switches act passively only to retain the role. Therefore, it is important to investigate the implementation of the controller side to achieve the redundancy; however, that has yet to be proposed. We use the capability of multiple-controllers to achieve high availability of the control plane.

III. CONVENTIONAL METHOD

In this section, we describe a conventional method of a redundant OpenFlow controller (OFC) using VRRP. We investigate the issue in the case of using VRRP. Table I shows the parameters common to all experiments (i.e., Section III, Section IV-A, and Section IV-C) in this paper, and Table II shows the parameters specific to the VRRP experiment in Section III.

We implement OpenFlow-1.2-compliant controllers and switches on Linux by extending an existing implementation [16], which consists of a NOX-based controller [17] and Ericsson TrafficLab 1.1 software switch [18]. In addition, we use Keepalived [19] to run VRRP between the controllers.

We conducted an experiment on our testbed as shown in Fig. 1. There are two controllers (i.e., OFC01 and OFC02). To achieve redundancy between the two controllers, VRRP is used. Initially, the state of OFC01 is set to *Master* and thus OFC01 has a virtual IP address. The state of OFC02 is set to *Backup*. An OpenFlow Switch (OFS01) is connected to OFC01 through an OpenFlow channel since OFC01 has a virtual IP address. OFS01 sends a packet-in message to the controller when it receives a new packet undefined in the flow entry because OFS01 is operated under the reactive mode. A traffic generator sends packets at the rate of 100 packets per second (pps).

Fig. 2 shows an operational sequence that indicates the state transition in the case of OFC01's going down. Initially,

TABLE I. PARAMETERS COMMON TO ALL EXPERIMENTS.

Node	Parameter	Value
OpenFlow controller	operating system	Ubuntu12.04
	openflow implementation	NOX-based [16]
	network interface	Gigabit Ethernet
OpenFlow switch	operating system	Ubuntu12.04
	openflow implementation	TrafficLab 1.1 software-based [16]
	network interface	Gigabit Ethernet
Traffic generator	sending rate	100 packets/s

TABLE II. PARAMETERS SPECIFIC TO VRRP EXPERIMENT.

Node	Parameter	Value
OpenFlow controller	vrrp implementation	Keepalived [19]
	vrrp advertisement interval	1000 ms
	vrrp master down interval	3004 ms

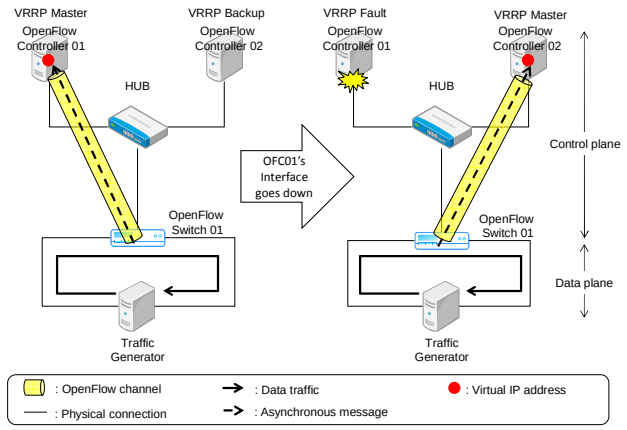


Figure 1. Experimental scenario using VRRP.

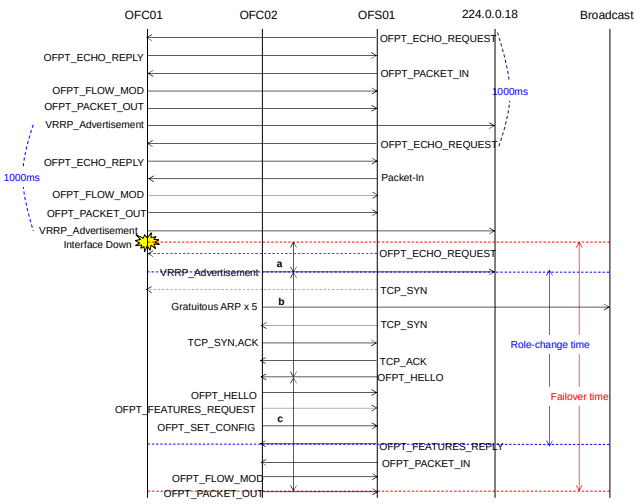


Figure 2. Operational sequence of the recovery using VRRP

the OFS01 sends an asynchronous message to OFC01 through the OpenFlow channel. Since the OFCs are running VRRP, OFC01 sends a VRRP advertisement message to OFC02 every 1000 ms. When OFC01 goes down, OFC02 sends a VRRP advertisement message to take over the virtu-

al IP address and change its own state to Master from Back-up after the master down interval, which is a guard timer for Backup to judge the failed condition of Master. Master down interval is defined by

3 * Advertisement_Interval + ((256 – Priority) / 256).

The master down interval for OFC02 is 3004 ms because the priority of OFC02 is set to 255 (i.e., the highest priority) to detect the failure of OFC01 as soon as possible.

OFC02 sends five gratuitous ARP packets to inform that the MAC address of the virtual IP address is changed in one second after the Keepalived sends the VRRP advertisement message. If OFS01 sends a SYN packet to reconnect to the virtual IP address before OFC02's sending gratuitous ARP packets, OFS01 cannot connect to OFC02 because the destination MAC address of the SYN packet is set to the OFC01's MAC address. If the OFS is successfully reconnected to OFC02, in other words, if TCP connection is reestablished, OFS01 starts sending a Hello message to OFC02 to establish an OpenFlow protocol connection. Then, OFS01 sends a packet-in message to and receives a packet-out message from OFC02. Thus, the failover is completed. In Fig. 2, the *failover time* is defined as the duration time from the failure event of OFC01 to the first packet-out message sent by OFC02. Also, the *role-change time* is defined as the duration time from OFS02's sending the VRRP advertisement message to the receipt of OFPT_FEATURE_REPLY by the OFS. Intervals *a*, *b* and *c* shown in Fig. 2 are defined as follows. Interval *a* is *advertisement delay* that is the time from OFC01's going down to OFC02's sending the advertisement message. Interval *b* is *TCP-recovery delay* that is the time from OFC02's sending an advertisement message to OFS01's sending the OPFT_HELLO message. Interval *c* is *OpenFlow-recovery delay* that is the time from OFS01's sending the successful OPFT_HELLO message to OFC02's sending OFPT_PACKET_OUT message.

We measured the failover time and role-change time 10 times respectively. The results are shown in Table III. We

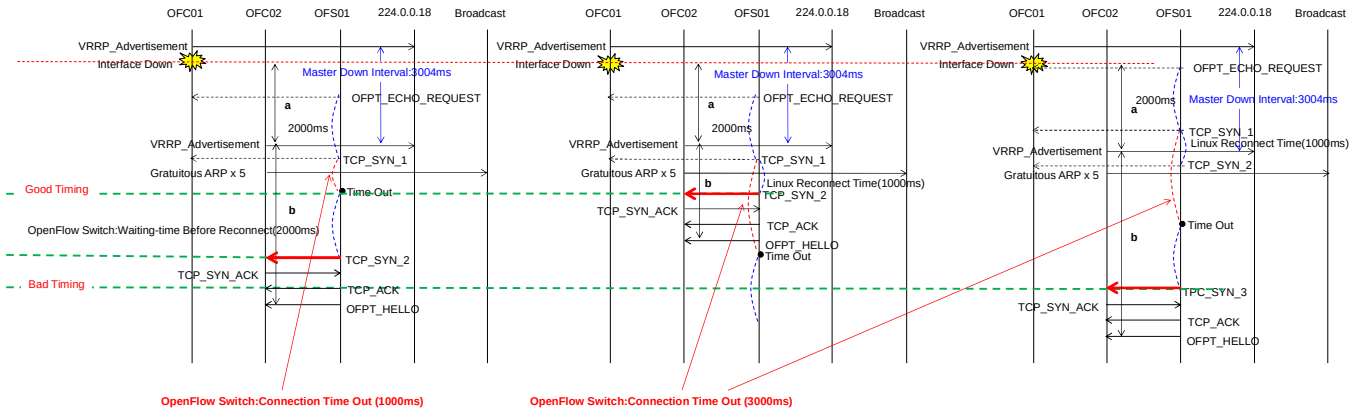
TABLE III. ROLE-CHANGE TIME AND FAILOVER TIME IN THE CASE OF DEFAULT PARAMETER.

	Minimum [ms]	Average [ms]	Maximum [ms]
Role-change time	3058	3365	3613
Failover time	5307	5653	5958

can improve these times by tuning some parameters.

Fig. 3 shows three operational sequence patterns of VRRP and Fig.3-(a) shows the sequence in the case of the default parameter. In VRRP, it is difficult to shorten the time to detect a failure because the minimum value of the master down interval is 3004 ms. To shorten the failover time in VRRP, we should reconnect the OFS to the OFC as soon as possible. To this end, OFS01 should send a SYN packet as soon as OFC02 sends the gratuitous ARP packets. The OFS01 first sends the SYN packet in two seconds after the failure of sending the OFPT_ECHO_REQUEST message. Since OFC01 is down, the OFS cannot receive the SYN_ACK packet. In our OFS implementation, the channel-establishment timer of OpenFlow is expired if both of the TCP connection and OpenFlow connection are not established within one second. And then OFS01 retries to connect to OFC02 after two seconds.

We changed the channel-establishment timer of OpenFlow to three seconds from one second. In that case, the SYN packet is retransmitted in one second after OFS01's sending the first SYN packet because the initial value of the TCP retransmission timer of Linux is one second. So, we can shorten the failover time as shown in Fig. 3-(b). However, the failover time depends on the timing of the failure of OFC01. If OFS01 sends an OFPT_REQUEST message to OFC immediately after OFC01 fails, the second SYN packet is sent before OFC02's sending the gratuitous ARP packets. As a result, the failover time increases as shown in Fig. 3-(c). Table IV shows the result in the case of changing the connection-establish timer to three seconds. According to Table IV, we can shorten the minimum and average times by changing the channel-establishment timer of OpenFlow.



(a) Channel-establishment timer is set to 1000ms

(b) Channel-establishment timer is set to 3000ms in good timing.

(c) Channel-establishment timer is set to 3000ms in bad timing.

Figure 3. VRRP-based switchover operations for three conditions of the channel-establishment timer.

TABLE IV. ROLE-CHANGE TIME AND FAILOVER TIME IN THE CASE OF CHANGING THE CHANNEL-ESTABLISHMENT TIMER.

	Minimum [ms]	Average [ms]	Maximum [ms]
Role-change time	1040	2621	4597
Failover time	3663	5161	7336

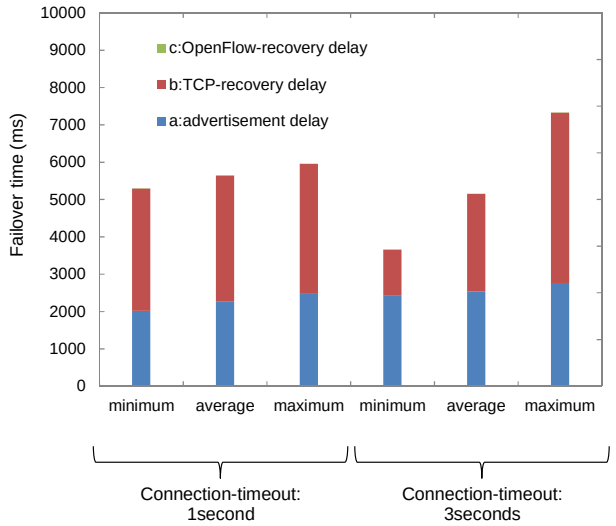


Figure 4. Breakdown of failover time.

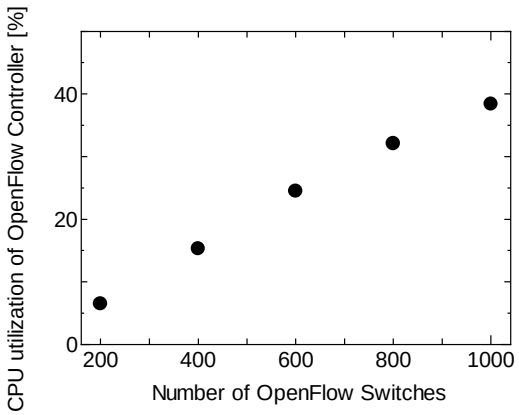


Figure 5. CPU utilization of NOX controller process during VRRP-based failover operation.

However, the maximum time is longer than the case with the original timer of one second.

Fig. 4 shows a breakdown of the failover time in both cases (i.e., the channel-establishment timer of OpenFlow is one second or three seconds). Intervals *a*, *b* and *c* set in Fig. 4 correspond to the markers shown in Fig. 2. Since Interval *c* is a very small value compared with Intervals *a* and *b*, the value is hardly visible in Fig. 4. According to Fig. 4, we can shorten Interval *b* by changing the channel-establishment timer of OpenFlow in the minimum and average value. However, considering that Interval *b* varies depending on the timing of OFC01's failure, it is difficult to adjust the parameter to the optimal values. In addition, it is also costly for network operators to set the optimal value to each switch if the OFC controls many OFSes provided by various vendors on various operating systems.

In the redundancy method using VRRP for the OFC, we measure the CPU utilization of the NOX controller process when the Master controller is changed and the OFS is reconnected to the controller. The experimental testbed is almost the same as shown in Fig. 1, except for the following two points. First, we use Open vSwitch [20] as the OFS to connect several switches to the OFC. Second, the traffic generator does not generate the data packet to measure only CPU utilization due to the failover of the NOX process. We measure the CPU utilization by a *top* command of Linux at one-second intervals. The maximum CPU utilization of the NOX process due to the failover is evaluated as a function of the number of OFSes. Fig. 5 shows the average of 10 measurements. According to Fig. 5, the CPU utilization of the NOX process increases with the growth of the number of OFSes. The CPU utilization is approximately 40% with 1000 OFSes.

In summary, using VRRP for redundancy of OFCs has two issues. First, it requires a long failover time. The failover time of VRRP has lower bound depending on its implementation. For example, Keepalived needs at least three seconds as the failover time since the minimum advertisement delay is two seconds and minimum TCP-recovery delay is one second. Also, it is difficult to shorten the failover time by changing parameters. Second, considering that the CPU utilization due to the failover process is high, VRRP is not suitable for a large OpenFlow network.

IV. PROPOSAL AND EVALUATION

In this section, we propose an architecture that uses multiple-controllers capability for local and global recoveries. We also evaluate recovery operation in two scenarios (i.e., local and global). To avoid the re-establishment of both the TCP connection and the OpenFlow channel, which is inevitable in conventional virtual IP address-based redundancy, we apply the multiple-controllers capability [9] to both local and global scenarios. Through the evaluation of the two scenarios, we use OpenFlow-1.2-compliant controllers and switches on Linux by extending an existing implementation [16] as shown in Section III.

A. Proposed Design of Local Recovery

First, we explain the redundant method in a single domain, which is typically a data-center hosting OpenFlow controllers. Table V shows parameters specific to a local-recovery experiment.

Fig. 6 shows a reference model for describing and evaluating the proposed scheme designed for the local recovery. OFC01 is connected to two controllers through two OpenFlow channels. In a normal operation, the role of OFC01 is set to MASTER and that of OFC02 is set to SLAVE. OFC01 and 02 have the same flow entry information mirrored between the two OFCs. OFS01 and OFS02 are

TABLE V. PARAMETERS SPECIFIC TO LOCAL-RECOVERY EXPERIMENT

Node	Parameter	Value
OpenFlow controller	keep-alive interval	50 ms
	keep-alive timeout	50 ms

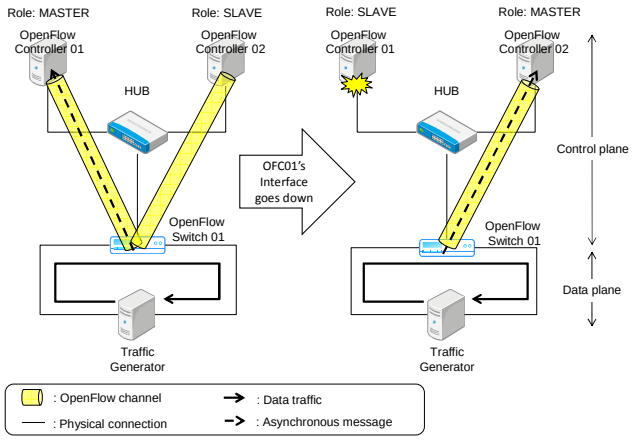


Figure 6. Experimental scenario using multiple-controllers capability in local environment

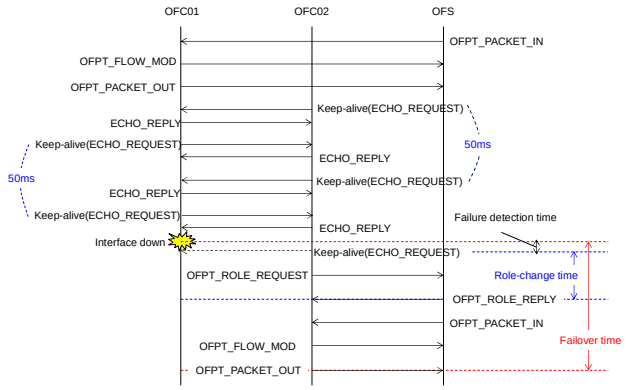


Figure 7. Design of a control procedure for a local recovery

operated under the reactive mode, and send a packet-in message to OFC01 when it receives a new packet undefined in the flow entry. To evaluate the performance influence in the data plane, a traffic generator continuously generates data packets with 100 packets per second (pps) where every packet has unique flow identifiers for stressing the reactive operation of the controller.

Fig. 7 shows an operational sequence of the proposed redundant scheme utilizing the multiple-controllers capability. In the proposed scheme, controllers send keep-alive messages (e.g., ICMP echo) to each other every 50 milliseconds. In a normal operation, OFS01 sends an asynchronous message such as packet-in to OFC01, since the switch recognizes the role of OFC01 as MASTER and that of OFC02 as SLAVE. OFC01 sends a flow-modification message and packet-out message to respond to the packet-in message from the switch. If the keep-alive message is lost, a controller (i.e., OFC01) is assumed to have failed. Due to the failure of OFC01, OFS01 cannot send any packet-in messages, and then the data plane cannot continue successful packet forwarding for any new incoming flows. Upon detecting the failure of OFC01, OFC02 sends an OFPT_ROLE_REQUEST message to OFS for changing its own role to MASTER. Then, OFS replies the OFPT_ROLE_REPLY message, and starts sending asynchronous messages to OFC02 after the completion of the role-change process. To respond to the asynchronous messages, OFC02 starts sending flow-modification and packet-out messages, and finally, the packet forwarding in the data plane is restored. As represented in Fig. 2, failover time is defined as the duration time from the failure event of OFC01 to the first packet-out message sent by OFC02. Failover time is measured using a traffic generator to obtain the data plane outage time. The role-change time is defined as the duration time from the detection of OFC01 failure to the receipt of OFPT_ROLE_REPLY by OFC02. Role-change time is measured by retrieving the event log of each controller to observe the control message process.

B. *Evaluation of Local Recovery*

The failover time and role-change time are evaluated by increasing flow entries in order to investigate the influence of the entry size. Fig. 8 shows the average of 10 measurements of the failover time and role-change time. Failover time is around 60-90 milliseconds and role-change time is about 15 milliseconds. Since the failure detection included in the failover time has a timing offset within the keep-alive interval, the observed failover time has some fluctuation range. Although the role-change time of the proposal is comparable with that of the virtual address-based redundancy, the failover time of the proposal shows a significant advantage thanks to the seamless handover between multiple OpenFlow channels. Fig. 8 also shows that entry size on OFCs does not affect the local recovery operation both for role-change time and failover time.

In the redundancy method that uses the multiple-controllers capability in the local recovery, we measure the CPU utilization of the NOX process due to failover. We use Open vSwitch as OFS instead of Ericsson TrafficLab 1.1 software switch. The traffic generator does not generate any data packet to measure only CPU utilization of the NOX process due to failover. Fig. 9 shows the average of 10 measurements of the maximum CPU utilization. According to Fig. 9, the CPU utilization of the NOX process increases with the growth of the number of OFSes. However, the utilization is smaller than that of using VRRP. This is because there is no process of OFS01's reconnecting (i.e., TCP reconnecting and OpenFlow reconnecting) to OFC02 in the proposed method of using the multiple-controllers capability. Thus, the proposed redundancy method of using the multiple-controllers capability has two advantages compared with the conventional method of using VRRP. First, its failover time is short because the process of failure detection is independent of the process of handover. Consequently we can combine the fast detection method (e.g., BFD [21]) with the

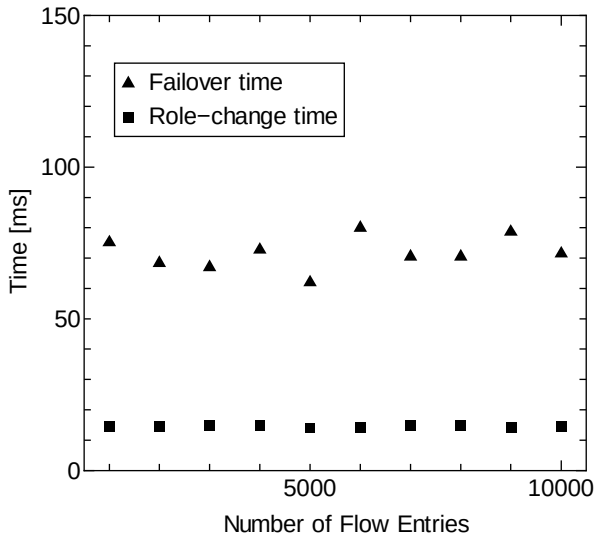


Figure 8. Result of failover and role-change time in a single domain.

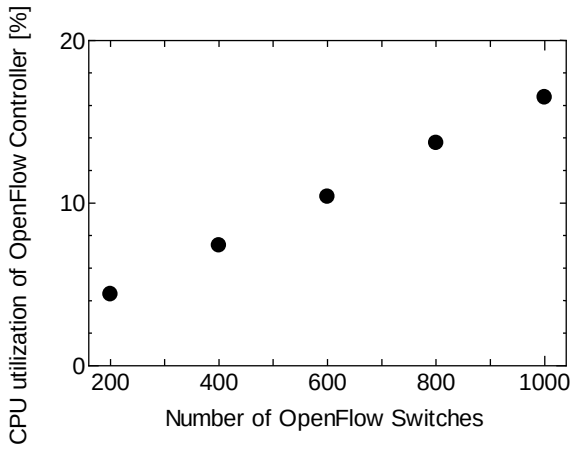


Figure 9. CPU Utilization of NOX process during multiple-controllers-based failover operation.

process of handover and we can achieve the short failover time. Second, considering that the CPU utilization due to the failover is low compared with the method of using VRRP, the proposed redundancy method of using multiple-controllers is suitable especially for a large OpenFlow network.

C. *Proposed Design of Global Recovery*

In this section, we explain the redundant method of multiple domains. Table VI shows the parameters specific to the global-recovery experiment. Fig. 10 shows a reference model of the controller redundancy for the global recovery scenario. The global recovery should consider tackling extraordinary events affecting, for example, the entire data center. We assume that a controller is installed in each domain to retain its scalability and performance. The controller manages OFSes belonging to the same domain as the MASTER, and the controller manages the other OFSes in the other domains as the SLAVE. The respective roles of the controllers are depicted in the upper side of Fig. 10. For example, OFS-A (i.e., some switches belonging to domain-A) recognizes the role of OFC-A (i.e., the controller belonging to domain-A) is MASTER and the role of the other controllers is SLAVE. Similarly, OFS-B and OFS-C also recognize the role of the controller that belongs to its same domain is MASTER and the roles of the other controllers are SLAVE. The controller has flow entry information for only OFSs recognizing the controller as MASTER. Thus, the controller does not need to have an excessive configuration or receive an excessive message. Additionally, one characteristic of our proposal is the existence of a Role Management Server (RMS). RMS monitors all controllers to manage their role, and RMS has some data such as CPU utilization, role information, configurations of all controllers and domain information of all switches. RMS determines which controller should take over the role of MASTER and relevant configuration data, if a controller has failed. In this regard, we have to be careful to prevent second failures. If OFC-B takes over the role of MASTER for broken OFC-A and places OFS-A under management besides OFS-B, there is the possibility of CPU utilization overload of OFC-B and then OFC-B may fail consequently. Thus, we should consider that one failure would induce subsequent failures. That is why RMS monitors CPU utilization and judges multiple-controllers should take over the role of MASTER from one controller, if RMS judges that taking over with a single controller raises overload of CPU utilization.

TABLE VI. PARAMETERS SPECIFIC TO GLOBAL-RECOVERY EXPERIMENT

Node	Parameter	Value
Role management system	operating system	Ubuntu12.04
	network interface	Gigabit Ethernet
	snmp monitoring interval	50 ms

The role of the controller

	OFC A	OFC B	OFC C
OFS A	Master	Slave	Slave
OFS B	Slave	Master	Slave
OFS C	Slave	Slave	Master

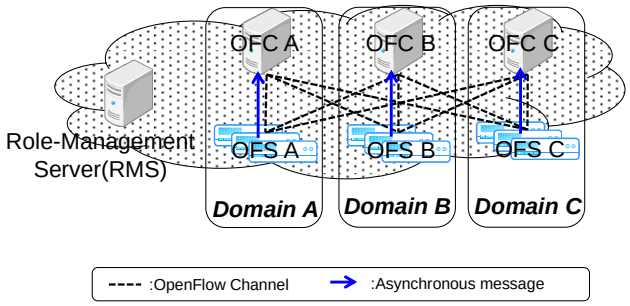
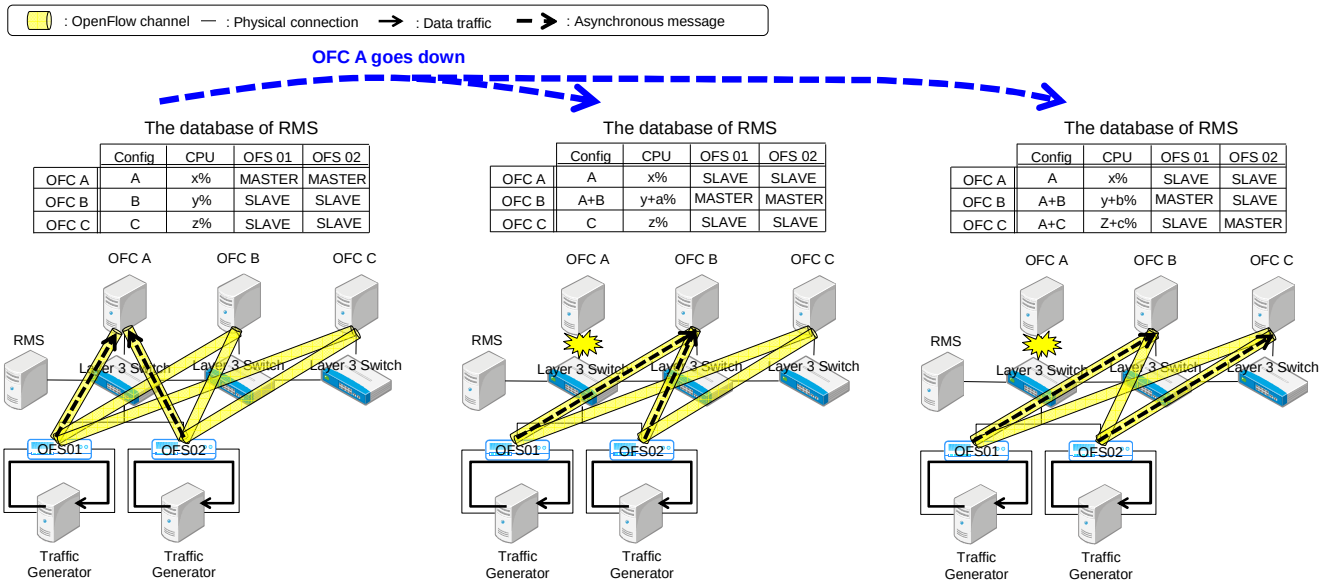


Figure 10. A network model for global recovery.



(a) Normal state.

(b) Two switches are migrated to a single controller.

(c) Two switches are migrated to two controllers.

Figure 11. Role-change transition in the global controller recovery

Fig. 11 shows the role-change transition for the global controller recovery. Fig. 11-(a) shows the initial state, and two switches are connected to three controllers through three OpenFlow channels. In the normal operation, both switches recognize that the role of OFC-A is MASTER and the other controllers are SLAVE. So only OFC-A receives some asynchronous messages such as packet-in messages. In this case, the three controllers have different configurations respectively and the information is reflected in the database of RMS. Also RMS has CPU utilization, the role information of each controller and the cognition haven by switch regarding the role of the controller in its database. The traffic generator connects OFS01 and OFS02 respectively and the data transfer rate is 100 pps. The two switches receive a new packet and send a packet-in message to the controller at all times as well as the measurement of a single domain.

If OFC-A fails and RMS judges there is no problem of a single controller taking over the MASTER role, the initial state (i.e., Fig. 11-(a)) is changed to Fig. 11-(b) where only OFC-B takes over the role of MASTER. The RMS database is updated accordingly, and both switches start sending asynchronous messages to OFC-B.

In contrast, if OFC-A fails and RMS judges that a single controller cannot take over the Master role but two controllers can, the initial state is changed to Fig. 11-(c) where two controllers take over the role of MASTER. The database of RMS is updated accordingly, and then OFS01 starts sending asynchronous messages to OFC-B. OFS02 sends asynchronous messages to OFC-C.

Fig. 12 shows a global recovery scheme in the case of Fig. 11-(b). RMS monitors the CPU utilization of all controllers every 50 milliseconds with Simple Network Management Protocol (SNMP) [22]. Since Fig. 5-(b) has three controllers, each controller is monitored every 150 milliseconds. The proposed recovery process consists of a judge-phase and a takeover-phase. If RMS is unable to retrieve the information about CPU utilization from OFC-A, RMS does not immediately assume that OFC-A has failed to avoid false positive. To ensure the failure detection, RMS requests that the ICMP echo be sent from the other controllers (OFC-B and OFC-C) to OFC-A. If more than half of the results indicate the failure of OFC-A, RMS determines that OFC-A has failed and starts calculating a new MASTER controller migrating OFC-A's configuration and OFSs under OFC-A. The process from failure detection to the determination of a failed controller is defined as the judge phase as indicated in Fig.12.

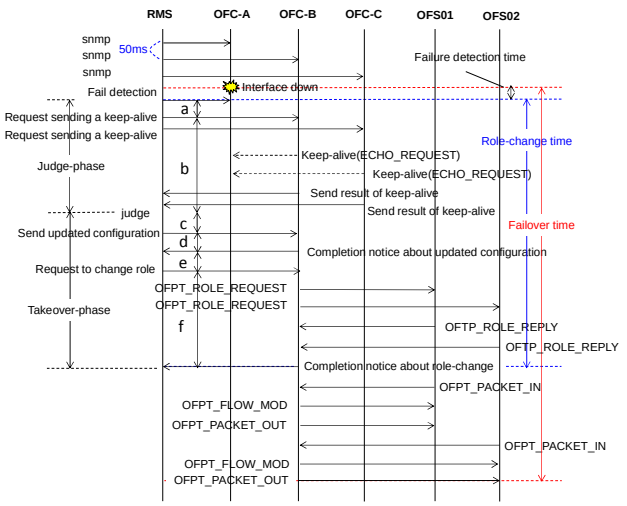


Figure 12. Proposed operational sequence for Figure 5 (b) scenario.

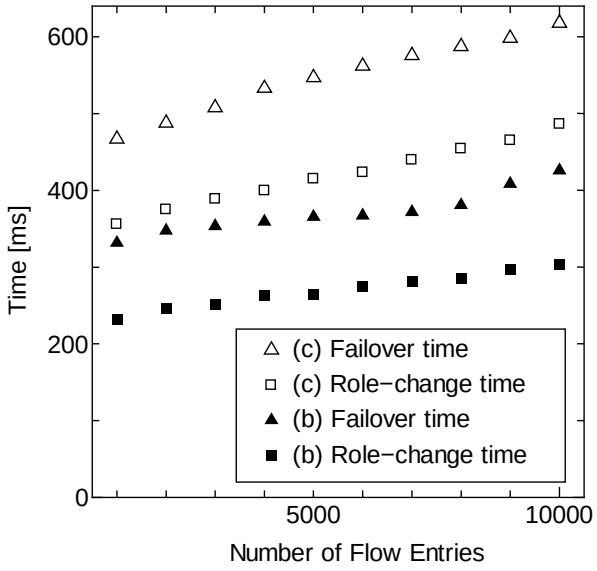


Figure 13. Result of failover time and role-change time in global recovery.

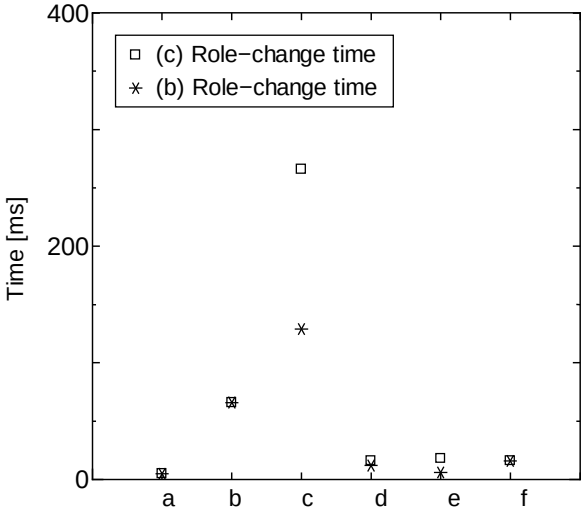


Figure 14. Breakdown of role-change time observed for scenario Fig. 11-(b) and (c)

After the judge-phase, RMS moves to the takeover-phase. In the takeover-phase, RMS firstly calculates whether it is no problem for a single controller to take over all switches connected to OFC-A by considering CPU utilization of OFC-A as well as OFC-B and C. If two or more controllers are required to take over all switches of OFC-A, RMS separates the switches based on the ratio of the available CPU resources of new MASTER controllers. If RMS decides that OFC-B is adequate to become a new single MASTER as shown in Fig. 11-(b), RMS integrates OFC-A's configuration into OFC-B's and registers the integrated configuration into OFC-B. Upon receiving the integrated configuration, OFC-B updates its own configuration and then reports the completion of the integration process. Then, RMS requests OFC-B to send the OFPT_ROLE_REQUEST to the switches for updating the role of OFC-A to SLAVE and OFC-B as MASTER. The switches send the OFPT_ROLE_-REPLY after updating the role change process. Then, OFC-B reports the completion of the role-change process to RMS. The process from completion of the judge-phase to completion of the role-change is defined as the takeover-phase. After the take–over phase, the switches OFC01 and 02 start sending asynchronous messages to OFC-B.

D. Evalution of Global Recovery

Fig. 13 shows the average of 10 measurements of role-change time and failover time in both cases of Fig. 11-(b) and (c). Role-change time and failover time increase with the growth of flow entry size. This result shows the difference in behavior compared with the result of a local recovery shown in Fig. 8. The major reason for this increase of failover time is that RMS needs integration of multiple configurations of failed OFC and registration of the configuration during the takeover-phase. As different scenarios of the global recovery, RMS selects multiple-controllers as the new MASTER as shown in Fig. 11-(c), and the scenario takes a longer role-change time and failover time as shown in Fig. 13. This reason is analyzed using the result of Fig. 14 that shows a breakdown of the role-change time under 1000 entries in both cases (i.e., Fig. 11-(b) and (c)). The characters ("*a*" to "*f*") placed on the x-axis of Fig. 14 correspond to the marker shown in Fig. 12. As shown in Fig. 14, the major performance difference comes from *c* that is the time to integrate configuration in RMS and register it to OFC. Current implementation suffers from the serial processing of the registration of integrated data. This means introducing parallel processing of the registration resolves the delay of role-change for the scenario shown in Fig. 11-(c).

According to Fig. 13, the role-change time is about 300 milliseconds and failover time is 420 milliseconds in 10000 flow entries, in the case of the scenario in Fig. 11-(b). In the case of the Fig. 11-(c) scenario, the role-change time is about 500 milliseconds and failover time is about 620 milliseconds. These results indicate that, for both scenarios, our proposal achieves a competitive role-change time and faster failover time compared with existing redundant mechanisms [13, 15]. We consider the proposed implementation of multiple-controllers achieves high availability controllers for both intra and inter data-center recoveries.

In this paper, we do not evaluate the redundancy of RMS itself. Although conventional server redundancy mechanisms accompanying a relatively longer failover time may be applied to RMS redundancy, RMS cannot be a critical bottleneck of processing asynchronous messages. This is because RMS failure itself does not affect any OpenFlow channel sessions and thus the data plane is not affected, accordingly.

V. CONCLUSION AND FUTURE WORK

In OpenFlow architecture, the controller is an important element for achieving reliable SDN. In this paper, we evalu-

ated the redundant method for the OpenFlow controller by using a conventional method (i.e., VRRP) and we verified the existence of the issue from the viewpoint of failover time and CPU utilization. And then we proposed a redundant scheme to tackle both a single domain ("local") and multiple domain ("global") recovery scenarios, which cannot be resolved with conventional redundant schemes. To avoid a long failover time and heavy CPU load due to conventional virtual IP address-based schemes, our scheme used the multiple-controllers capability for seamless handover. To avoid performance scale-limit due to conventional clustering schemes, our scheme eliminates any frontend server from the redundant system. The evaluation shows that the proposed scheme involves lower CPU utilization and competitive role-change and failover times compared with conventional schemes. In our scheme, the CPU utilization due to the process of failover is half or less compared with the virtual IP address-based scheme in the case of 1000 units of OFSes. Our scheme is more suitable for a large OpenFlow network. The role-change time observed in a local recovery scenario is about 15 milliseconds regardless of entry size, and that in a global scenario ranges from 200 to 400 milliseconds. CPU resource-aware migration of managed OpenFlow switches in the failover process was successfully achieved by our scheme. The proposal is expected to be an effective high availability scheme necessary for deploying reliable and scalable SDN.

In future work, we will shorten the failover time for the scenario of some OpenFlow switches migrated to some OpenFlow controllers. In RMS, we will separate the current redundancy process that is sequential migration into every controller, and we will establish CPU-based controller resource modeling to accurately handover many OpenFlow switches in the event of, especially, global recovery where massive nodes may need to be protected.

ACKNOWLEDGMENT

We are grateful to Yasunori Maruyama for our productive discussions, support for our experiments and programming assistance.

REFERENCES

[1] K. Kuroki, N. Matsumoto, and M. Hayashi, "Scalable OpenFlow controller redundancy tackling local and global recoveries," Proc. International Conference on Advances in Future Internet (AFIN2013), August 2013, pp. 61-66.

[2] N. McKeown et al., "OpenFlow: enabling innovation in campus networks," ACM SIGCOMM Computer Communication Review, vol. 38, isssue 2, April 2008, pp. 69-74.

[3] M. P. Fernandez, "Evaluating OpenFlow controller paradigms," Proc. International Conference on Networks (ICN2013), January 2013, pp. 151-157.

[4] R. Pries, M. Jarschel, and S. Goll, "On the usability of OpenFlow in data center environments," Proc. IEEE International Conference on Communications (ICC2012), June 2012, pp. 5533-5537.

[5] H. E. Egilmez, S. T. Dane, K. T. Bagci, and A. M. Tekalp, "OpenQoS: an OpenFlow controller design for multimedia delivery with end-to-end quality of service over software-defined networks," Proc. Signal & Information Processing Association Annual Summit and Conference (APSIPA ASC 2012), December 2012, pp. 1-8.

[6] "IEEE standard for local and metropolitan area networks – station and media access control connectivity discovery," IEEE Std 802.1AB, September 2009.

[7] J. Kempf, E. Bellagamba, A. Kern, D. Jocha, A. Takacs, and P. Skoldstrom, "Scalable fault management for OpenFlow," Proc. IEEE International Conference on Communications (ICC2012), June 2012, pp. 6606-6610.

[8] A. Dixit et al., "Towards an elastic distributed SDN controller," Proc. ACM SIGCOMM Computer Communication Review, vol. 43, October 2013, pp. 7-12.

[9] "OpenFlow switch specification version 1.2," Open Networking Foundation, December 2011.

[10] "The transport layer security (TLS) protocol version 1.2," IETF RFC5246, August 2008.

[11] "Transmission control protocol," IETF RFC793, September 1981.

[12] A. Tootoonchian and Y. Ganjali, "HyperFlow: a distributed control plane for OpenFlow," Proc. the 2010 Internet Network Management Conference on Research on Enterprise Networking (INM/WREN'10), 2010.

[13] F. Koch and K. T. Hansen, "Redundancy performance of virtual network solutions," Proc. IEEE Conference on Emerging Technologies and Factory Automation (ETFA'06), September 2006, pp. 328-332.

[14] "Virtual router redundancy protocol (VRRP)," IETF RFC3768, April 2004.

[15] R. Zhang, T. F. Abdelzaher, and J. A. Stankovic, "Efficient TCP connection failover in web server clusters," Proc. IEEE International Conference on Computer Communications (INFOCOM'04), vol. 2, March 2004, pp. 1219-1228.

[16] CPqD/OpenFlow-1.2-Tutorial - GitHub, https://github.com/CPqD/OpenFlow-1.2-Tutorial [retrieved: April, 2013].

[17] NOXRepo, http://www.noxrepo.org [retrieved: April, 2013].

[18] TraffcLab/of11softswitch – GitHub, https://github.com/TrafficLab/of11softswitch [retrieved: April, 2013].

[19] Keepalived for Linux, http://www.keepalived.org [retrieved: October, 2013].

[20] Open vSwitch, http://openvswitch.org [retrieved: April, 2013].

[21] "Bidirectional forwarding detection (BFD)," IETF RFC5880, June 2010.

[22] "A simple network management protocol (SNMP)," IETF RFC1157, May 1990.

A Mobile Mashup for Accessing Distributed Recycling Knowledge

Sönke Knoch
German Research Center for Artificial Intelligence
Research Department Intelligent User Interfaces
Saarbrücken, Germany
Soenke.Knoch@dfki.de

Alexander Kröner
Georg Simon Ohm University of Applied Sciences
Department of Computer Science
Nuremberg, Germany
Alexander.Kroener@th-nuernberg.de

***Abstract*—From the consumer perspective, classifying a product regarding its environmental impact is a difficult task because relevant knowledge is usually not only diverse, but also distributed over several information sources. In this work, an analysis of mobile "green" applications formed the basis of a mobile application, which aims at providing all recycling-related information in-situ. Its domain model integrates recycling knowledge from several information sources and is capable of disassembling a product into its elementary parts. An information extraction approach allows the automatic integration of relevant content from new Web sources, which were suggested by the user. The mobile application enables the user to initiate interaction with this model over three different ways of describing a product. Beside insights concerning information access and user interaction, a first evaluation of the prototype indicates that the employed fused domain model may outperform results achieved with a traditional approach to web-based information search concerning recycling information. Based on the outcomes of the evaluation, a revised user interface is presented.**

***Keywords*-Sustainability, decision support, domain model, mobile mashup, mobile computing, case study.**

I. Introduction

Limitation of natural resources affects everyday decision making in diverse ways: indirectly through increasing costs for products, e.g., based on oil, or directly due rationale insight and ecological awareness. Unfortunately, such *sustainable decision making* is a non-trivial task for various reasons. For instance, a product has to be chosen that is "easy" to recycle. From the viewpoint of sustainability, recycling is affected by materials the product is consisting of, the recycling process for disassemble the product, the extent such disassembly is possible, and even the (potentially future) context that determines efforts needed to insert the product into the recycling process.

In order to make an informed decision, a human decision maker has to acquire all of that knowledge—and to fuse it. Information technology may support the user in this task in various ways (cf. [1]). This is reflected by related research and development activities ranging from integrating sustainability-related information along the supply chain (e.g., [2]) to community-driven information hubs for recycling tips (see e.g., [3]).

This complexity partially explains why expert advice in-situ may increase people's will to do such decisions [4]. Information has to become more available [5], and be explained to the user [6]. Thus, it is little surprising that there exists a considerable amount of "green" mobile applications, which seek to support their user in-situ in solving tasks related to sustainability.

This article extends previous work (see [1]) concerning a mobile application and a linked information service, which aim at supporting decisions concerning consumable products based on recycling-related information.

The following Section II reviews typical characteristics of such mobile applications. Then, Section III reports on a data mashup, which fuses different kinds of recycling-related knowledge from distributed sources in a single domain model. Section IV describes a mobile information service, which employs that domain model in order to combine services of various previously reviewed applications. Section IV summarizes the underlying system architecture, and provides further details concerning back end and mobile application. Afterwards, Section V summarizes feedback obtained in a comparative experiment, in which participants acquired recycling-related information with the new service as well with traditional information offers. That feedback affected the redesign of the system's user interface, which is presented in Section VI. Finally, the article closes in Section VII with a summary of achieved results and an outlook on future work.

II. Related Work

In 2011, a preparatory internal study addressed the state-of-the-art of mobile applications supporting sustainable decision making. The survey comprised mobile applications offered at the Android Market and the Apple App Store. Search terms were "energy consumption", "energy efficiency", and "green life" and led to a result of 23 relevant mobile applications in the Android Market and 25 mobile applications in the Apple App Store. The result was sorted into four categories *promotion*, *education and information*, *calculators*, and *monitoring and controlling*. Figure 1 shows the amount of matches for each category in the respective marketplace. The detailed result for each category is described in the following:

Promotion (4 mobile applications). Mobile applications in this category, typically, promote energy saving technologies, such as solar energy systems, low-energy devices of certain product classes (e.g., fridges, air conditioning systems, etc.),

or energy saving techniques (e.g., monitoring tools and programmable thermostats). For example, the mobile application Lennox [7] calculates the energy savings achievable by a new air conditioning system, provides product information and directs the user to the next local dealer.

Education and Information (20 mobile applications). References, encyclopedia, decision support systems, and games form a category on its own. The majority of such mobile applications provide information in form of references, tips, or links and news collections. For example, the mobile application "this is green" [8] offers information that is thematically organized by a picture of a layout of a common one family house. If the user tabs on the garage he will find information on fuel consumption of the car, if he tabs on the bathroom information on how to save water is provided. The application "low carbon life" [9] is a collection of little games that tries to teach the user, e.g., how to use the washing machine in an efficient way and how to recycle trash that occurs in a common household.

Calculators (9 mobile applications). Other mobile applications support the user in calculating balances concerning sustainability-related factors. They can be distinguished in mobile applications meant for the private and for the business domain. The former ones focus on an individual's habits and objects, e.g., flights and TV. The latter ones focus on business branches such as architecture or lamp industry. In general, the user has to enter data manually into the respective mobile application, which is a major difference to mobile applications classified as "monitoring and controlling". For example, the "green footprint calculator" [10] is filled manually with data such as monthly bills (oil, gas, and electricity), number of flights, and recycling behavior. Once filled with this data, the mobile application calculates the yearly carbon footprint and visualizes it with a maximum of six green trees if the carbon footprint is very good/small. The application "MeterRead" [11] captures energy consumption. The number of kilo watts is synchronized manually with the electrical meter over a graphical meter that looks similar to the one that can be found in households. After data gathering, the mobile application provides a prediction for the consumption over the next 30 days.

Monitoring and Controlling (15 mobile applications). Finally, there are mobile applications, which connect to energy consuming devices in the private and the business domain. In the private domain, they focus on devices common for an individual's environment, e.g., house, car, and mobile phone. In the business domain, such mobile applications focus on branches, e.g., IT, manufacturing industry, and facility management. For example in the private domain, the "power tutor" [12] analyzes system and power usage of the mobile device and provides chart views, e.g., for the consumption of the LCD, CPU, and Wi-Fi. The "green gas saver 1.0" [13] shows the greenest way of acceleration in a car. A lot of mobile applications visualize energy consumption (electricity, oil, and gas) and provide remote control features (e.g., switch on/off, timer configuration, etc.). Alarms are set off when consumption exceeds a defined threshold. One example from the business domain is "GSH ienergy" [14]. "DONG Energy eFlex" [15] controls home environments in the private domain. Community features are included in some mobile applications, where the user's green performance can be compared to the performance of the user's friends.

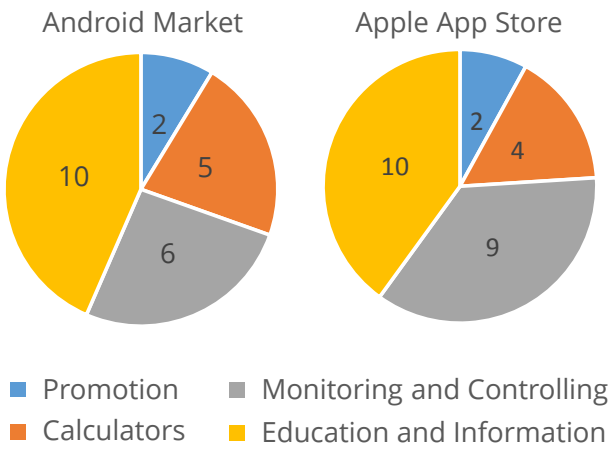


Figure 1. Related Mobile Applications.

General observations included that mobile applications for sustainable decision making were either highly specialized (focus on product advertisement or industrial applications) or generalized (dictionaries, household / lifestyle consulting). Furthermore, the reviewed applications rely on data from a single information source, which does not reflect diverse and distributed character of such information mentioned in the beginning. Finally, despite the mobile platform, there was little use of the mobile sensing capabilities.

In May 2014, both marketplaces were revisited in order to extract changes in categories identified in the previous study. In both cases, the top 10 applications returned in response to the query ("recycling") were briefly reviewed. In 2011, the same query led to irrelevant results, e.g., desktop recycling bins. Compared to the search in 2011, an higher amount of "green" recycling applications (60-70%) was registered in 2014. Of the overall 20 applications, about 30% now provide location-based recycling recommendations for products that are scanned via barcode. For instance, "RecyclingScanner" recommends a trash can in the vicinity or a supermarket for a given product. The application, developed for the German market, was tested and delivered good results. Also of interest and different to the previous study, there were now applications (10%) offering recommendations about creative ways of recycling. Finally, new game applications aim at informing and teaching people the proper way of recycling certain packaging. Nevertheless, these solutions share the narrow application focus observed in the 2011 study. This suggests that the mashup concept proposed in this article is still relevant and can provide a benefit for both user and environment.

This article reports on how these still existing gaps could be addressed for a specific application scenario: an "Eco-Advisor" should support consumers in ranking products according to their environmental impact, and in making informed decisions concerning recycling options regarding a product at hand using information from distributed recycling knowledge.

III. Fusion of Recycling Knowledge

According to the previously introduced classification of related work, the Eco-Advisor could be categorized in the first place as an *information and education* service, which includes aspects of a *calculator*. While the service as such could be employed also for user support in non-mobile scenarios, its particular focus is on decision support concerning a product "at hand".

Therefore, the service has to support the user in establishing a link between the subject of interest—a physical product instance—and relevant information concerning this individual artifact. This information may originate from distributed sources, and may differ in format and semantics. It may describe aspects of the artifact, this kind of artifacts, resources used for creating the artifact, and related services. Efforts needed in performing this task strongly depend on the way data are organized and structured by the service—its domain model.

A. Requirements

As the mobile application is meant to provide information for products, its domain model has to be capable to represent a product's most important properties. The model is kept as generic as possible because it is a storage for all kinds of data, structured and unstructured.

A product is defined in an economic sense as the result of a transformation that was initiated by humans. This transformation consumes scarce resources, such as materials and energy. In this article, we will focus on physical products and exclude virtual products, such as information or services.

The information about a product, its components, and resources that is necessary to provide decision support before or after product usage is distributed and hard to find. This challenge lead to three core requirements for the model, which are explained in the following:

- **Requirement 1:** The domain model has to carry information in form of various data patterns from distributed sources on an abstract and a concrete level and is open for extensions.
- **Requirement 2:** The domain model has to enable a disassembly of products in terms of kind and amount of materials included in the product's (current) physical form.
- **Requirement 3:** The domain model has to support the interaction implemented by the mobile application.

Requirement 1 asks for a domain model, which supports the mapping of a product at hand to recycling-related information. As recycling information is not provided by all manufacturers, such information can be found on the abstract level in the absence of manufacture specific information. If product specific information is available, it is stored on the instance level. Additionally, the model has to ensure a degree of extensibility that allows an adaption for specific needs. The last criterion is related to the open/close design principle from object-oriented programming. To integrate data from distributed sources, the model has to be able to carry data in heterogeneous patterns, and to make information available in a unified format.

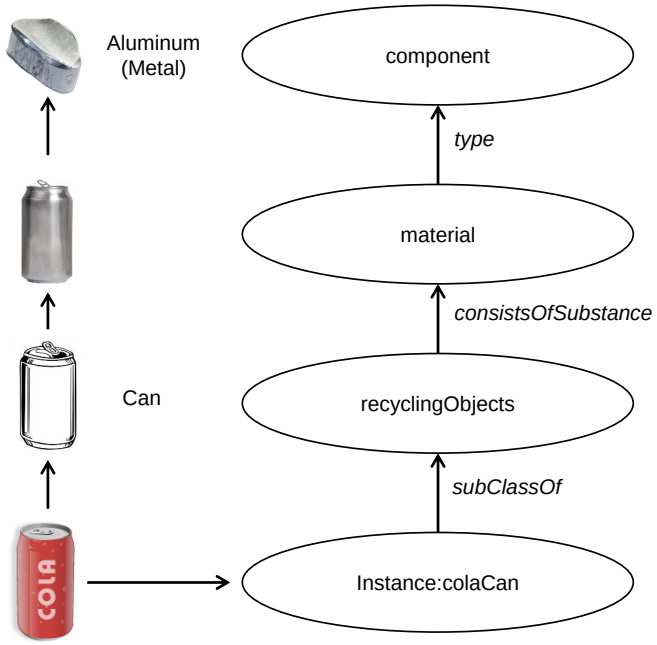


Figure 2. Ontology representation of the product structure.

Requirement 2 demands a domain model able to reveal product's components and materials down to an elementary resource level. For example, a beverage can consist of aluminum, which is a chemical element in the boron group with the symbol Al, the third most common element, and most abundant metal in the Earth's crust. Such information can be employed by the service in order to perform calculations involving a product's durability, kind of resources used, and recycling potential. Thus, while a resource used within a product may be scarce, this may be less crucial if the resource can be extracted with limited efforts during recycling for later reuse.

Requirement 3 demands that the domain model supports the particular kind of user-product-service interaction that forms the background of the envisioned kind of support. The quality of recommendations expressed by the service strongly depends on knowledge about the product the user is interested in. Ideally, this object is at hand and capable to describe itself, e.g., on the basis of data linked by identification (RFID) referenced as ISO 14443 or Quick Response (QR) Code referenced as ISO 18004 describing the individual product instance. However, other situations may require the user to describe the product with less precise means. In order to support the user in this task, the system's user interface provides diverse ways of describing products. The domain model has to reflect this diversity with an organization, which facilitates information retrieval starting from unique identifiers, visual features, keywords and product categories.

B. Domain Model

The assembly information on a product was modeled in the Ontology Web Language (OWL) [16]. In the model shown in Figure 2, a product is an instance of a sub class of recycling objects, which consist of one or multiple substances of a certain type.

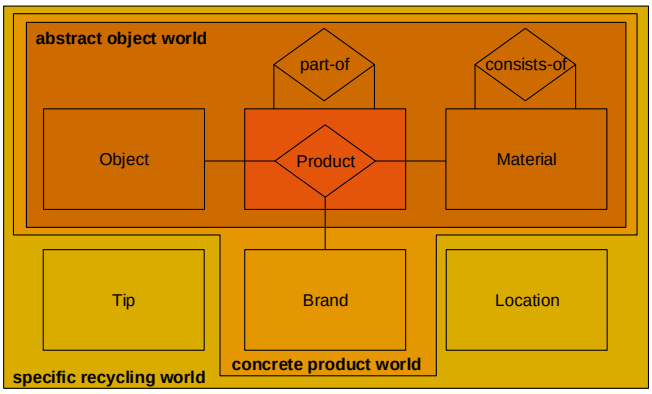


Figure 3. Entity Relationship Model (ERM) of the domain model (most relations and attributes are faded out).

According to *Requirement 1*, the final domain model is open for extensions; it was developed as an onion layered architecture. In the innermost layer lies the core, most abstract model, which is the nucleus of the model that is visualized in Figure 3, the "abstract object world". Objects consist of different Materials, the bill of materials, and have thereby a certain composition (*Requirement 2*). This kind of product assembly is discussed for electromechanical products by Rachuri et al. [17], an extension of the Core Product Model 2 (cf. Fenves et al. [18]) that covers a product's function, form, and behavior. The entities in the next layer, the "concrete product world", form the world of products and contain all entities from the object world. Objects are manufactured differently by different companies under different Brands. The combination of the entities Brand, Object, and Material forms a Product. These two worlds, the object and the product world, represented by the two innermost layers can be transferred on numerous use cases where product data is involved. Two kinds of products are allowed: products with a structure of certain materials and products that provide a structure under a certain brand. All products can contain sub-modules. This hierarchical modeling approach, indicated by the part-of relation, allows the subordination of sub-products, which are produced under a different brand by a certain supplier. A similar classification hierarchy was provided by Pels [19], which distinguishes between product instances, classes, and types to reduce the complexity of product models. In a similar way substances, contained in a material are modeled, which allows the disassembly of a product in its most atomic elements. In the outermost layer, the most specific one ("specific recycling world"), the entities for the use case at hand are modeled and set in relation to the entities in the other layers. The entity Tip contains creative recycling tips, the transformation of old objects into something new, for Products, Objects, and Materials. Location contains recycling points where Products, Objects, and Materials can be recycled. The specific (recycling) world is open for more extensions to extend the Object and Product worlds according to specific needs. The decision for an onion layered design of the domain model supports extension of the model: it is possible to add layers for specializing the model and to remove layers for generalizing the model. A similar way of abstraction was provided by Lee et al. [20], which proposed a generic and independent multilevel product model that is divided into data, model, and metamodel level.

To support the interaction (*Requirement 3*), textual definitions from WordNet [21] are used to identify the entities Object, Material, and Brand that are denoted as things following the notion "Internet of Things". This kind of identification allows text searches on the IDs and users to find the Object, Brand, or Material of interest. The relation among those three entities allows the presentation of related Materials and Brands when an Object is searched, the presentation of related Brands and Objects when a Material is searched, and the presentation of related Objects and Materials when a Brand is searched. Related products from the overlapping of all three entities can be presented. Additionally to the concept of definitions, word forms—a set of synonyms—are assigned to Objects, Materials, and Brands, respectively. These synonyms support a query expansion mechanism that guarantees search results for a set of valid search terms. For example, "Al" leads to the same result as "aluminum", "aluminium", or "atomic number 13". Recycling Tips are assigned to Objects and Materials. A product taxonomy is used to categorize Products, which allows a search for products by category. Products have additional attributes that are amount and unit. This allows for storing information on the quantity of materials, which are obstructed in one object. Locations own the additional fields latitude and longitude to store the GPS position.

IV. Access to Recycling Knowledge

The system is divided in two parts: the mobile application that makes information available to the user and the back end that provides an interface to the Web and pre-processes data for fast information access. Overall, the system implements a *mashup* [22] of tools and resources in order to realize one particular service. The client forms a *mobile mashup* because it combines contextual information provided by mobile devices with a mashup's capability to integrate web resources and process data (cf. [23], [24]). The back end alone is denoted as a *data mashup*. An overview of related work in the area of mobile mashups is provided in previous work (cf. [25]).

Figure 4 provides an overview of the system's main components: The data mashup on the *back end* side, its *information extraction* component, and the *mobile application*.

A. Back End

The data mashup combines the contents of multiple heterogeneous and distributed *Information Sources* that can be seen on top of Figure 4. It integrates these sources in one database in order to speed up query processing. Responsible for this integration is the *Information Integration* component. The latter one is responsible for processing semi-structured data obtained from *Information Extraction* components, which wrap the actual *Information Sources*. The *Information Integration* stores its result in the *Domain Model* database and translates the *Information Management*'s requests into database queries. The *Domain Model* database contains the ontology model depicted in Figure 2 that was transferred to a relational database according to the ERM in Figure 3 for performance reasons. In the database, per default, each entry consists of the 4-tuple $< ID, Name, Description, Image >$. The ID is

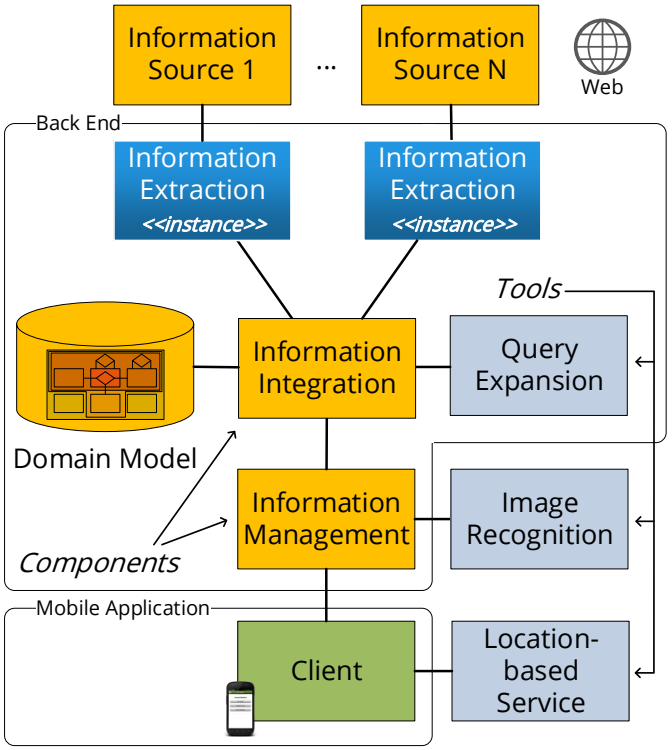


Figure 4. Components of the system architecture.

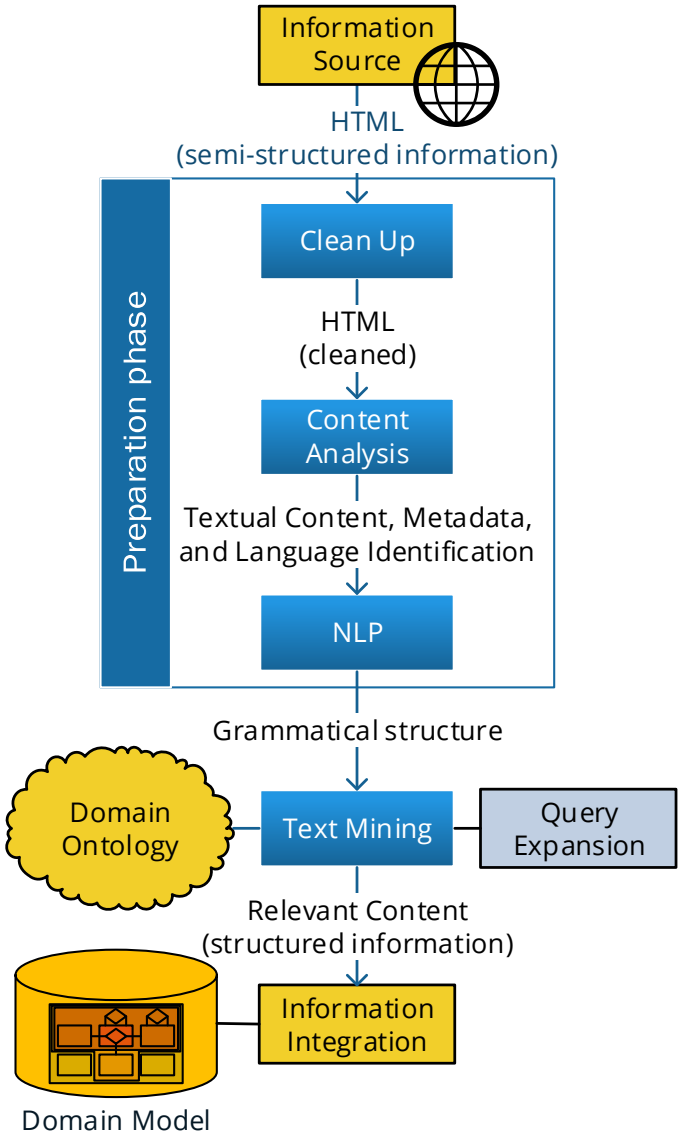


Figure 5. Extracting information from user defined Web sources.

a unique identifier, $Name$ represents the designation of the data entry, and $Description$ contains a long text that helps to characterize the thing. An $Image$ visualizes the entity and can be stored in form of a file path. Each entity is expandable by additional attributes that might be appended to the 4-tuple. Additional attributes concerning an entity may be appended to the tuple. For instance, GPS coordinates are added to the location entity.

The *Query Expansion* tool is used to increase the hit rate of search terms received from the *Client* side. These client requests are handled by the *Information Management* that receives HTTP requests over a REST interface. To process image data, an *Image Recognition* component is connected and delivers describing strings via Web hook (cf. Figure 6), as the *Information Integration* component processes only textual data.

The *Query Expansion* tool expands search terms from all three ways of interaction (search by text, search by category, and search by image) by synonyms from the WordNet [21] dictionary to match additional entries in the database. The *Image Recognition* component was realized by using the IQEngines API, which delivered acceptable results (in most cases the labels and not the things are recognized) that can be improved by training the image recognition algorithm. Since IQEngines was acquired by Yahoo! in 2013, its service is no longer available. Instead, we will use the visual search engine Macroglossa [26].

The system's modular architecture seeks to support adding and removing *Information Sources* as well as exchanging back end components. Technical details concerning the integration and adaptation of information in this framework (e.g., the way how recycling tips from World.org [3] are fused with other recycling information) have been subject of previous work (see [25]).

B. Information Extraction

In the first prototype, the information source was made available by a wrapper module. The source-specific wrapper parsed the content of the respective Web page, structured the information, and delivered the data to the domain model to receive a program-friendly structure. JAXB was used to make the data from the database available at the REST interface. It autogenerates class representations of the database entities and of the corresponding schema files. The Information Management component used this meta information to generate XML structured data and delivered it to the mobile device.

After the first version was realized and the case study that is described in Section V was conducted, it was recognized that it would be useful to have a feature that allows the user to

add and share new information sources by simply providing the URL. In order to achieve this goal, an approach to information extraction without specific wrappers needs to be added to the aforementioned *Information Extraction* (IE) component. In the following, a concept for the realization of such an IE mechanism that represents work in progress is suggested.

The process of IE is outlined in Figure 5 and was in parts inspired by the system suggested by Germesin and Romanelli [27]. The task of the process is the extraction of relevant information and the transformation of semi-structured into structured information that can be added to the database and is merged with existing content. On top of the process, the *Information Source* is defined by an URL that points to semi-structured information usually encoded in HTML. Then, the preparation phase that contains the process steps *Clean Up*, *Content Analysis*, and *NLP* (Natural Language Processing) starts. During the *Clean Up* phase the main textual content of the Web page is extracted and surplus ”clutter” is removed.

For the realization, it is planned to use the Readability API or the boilerpipe JAVA library. The *Content Analysis* process determines the document type, in most cases HTML, and extracts the textual content and metadata. Additionally, the language of the document is identified. It is planned to use Apache Tika for this task. Afterwards, it has to be distinguished between structured information, such as HTML tables and unstructured information, such as free text. Structured information is directly passed to the *Text Mining* process while free text is parsed by the *NLP* process. We plan to use the Stanford Parser for this task, which works out the grammatical structure of sentences that is used in the *Text Mining* process. Finally, the data preparation phase is finished and the *Text Mining* starts. It uses the knowledge of the Domain Ontology whose concepts were shown in Figure 2. The ontology describes the content that is relevant for the IE process and provides the domain knowledge that is compared to the *Information Source*.

The existing *Query Expansion* component is used to provide synonyms for the entities in the ontology. These synonyms are added to search patterns that are formulated based on the ontology containing the materials, products, and brands of interest. It is planned to use Apache Lucene to solve the search task. The grammatical structures from the *NLP* process help to discover relations between multiple search terms. For instance, the sentence ”A cola can has a carbon footprint of 170g” sets the pattern ”cola can” and ”carbon footprint” into relation to each other. The low distance of both patterns indicates a match. When such a relation is discovered, it is passed over to the *Information Integration* component, which stores it in the *Domain Model* database. Multiple relations for the same entities can be stored. The source of the information is added to the tuple to be able to provide the origin of the information on the user interface.

C. Mobile Application

The mobile mashup was realized as a mobile application that presents the contents provided by the data mashup that is encapsulated by the back end and adds additional information from the *Location-based Service*. It provides the user interface components with an interface for data search and retrieval that provides abstraction from the underlying actual data sources.

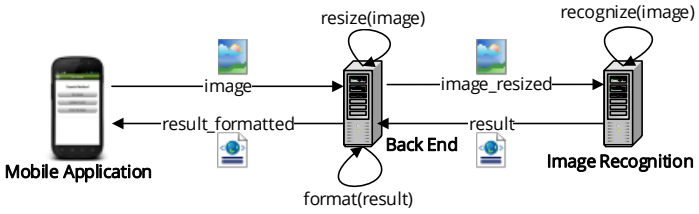


Figure 6. Client–server communication.

The mobile application runs on a mobile device with Internet connection. It communicates via a REST interface with the back end, which is implemented as a Web service. The user interacts with the mobile device and things—in our scenario for the experiment an aluminum can, a plastic or a glass bottle. Three ways of interaction were realized, search by text, search by category, and search by image. A navigation tree containing screen shots from the mobile application is presented in Figure 7 and shows the search result for an aluminum can manufactured by a certain brand.

When the user starts the application he sees the home screen that is labeled with A. The three buttons trigger the three interaction methods. *Search by text* leads to screen B, which provides a text field and a search button. A drop down list shows a list of recent search terms. *Search by image* starts the camera application of the phone and allows the user to take a picture from an object or to choose a picture from the phone’s file system. Once an image is selected, it is sent to the back end. The back end resizes the image and forwards it to the associated image recognition API that is invisible to the user, and waits for a response (Web hook). When the response is arrived it is immediately passed over to the mobile device. The whole process is outlined in Figure 6. The processing time strongly depends on quality of image and Internet connection (in our setup 2-5 seconds). For a given object, search by text and search by image may result in a broad range of search terms, since users may follow different approaches to describing or photographing objects. The search result is visualized on screen E and shows a definition of the object in the headline. If multiple definitions are matching the query the user has to choose a definition from a list of definitions that the system considers as relevant. The object’s composition is viewed in four categories on screen E. The drop down list Products contains products in the database for a given object. Aluminum can is an example for a product related to the object can. The Objects list contains the objects, in this case a can. Materials lists all materials that are contained in the listed products. Finally, all brands that are selling the products under the selected definition are listed in the Brands list. Another search mechanism is the *Search by category* that was built using the Google product taxonomy and can be seen on screen D. It allows to browse for products by category. Screen F shows the results for the category soda pops. Aluminum cans, glass bottles, and plastic bottles are listed.

In the next step, the user can select one entry: a product, an object, a material, or a brand. If the user selects an object, the application displays a description of the object and creative recycling tips on screen G. Recycling tips are structured in categories. If the user selects a category, then the application responds with a list of tips. The list is sorted by relevance. If

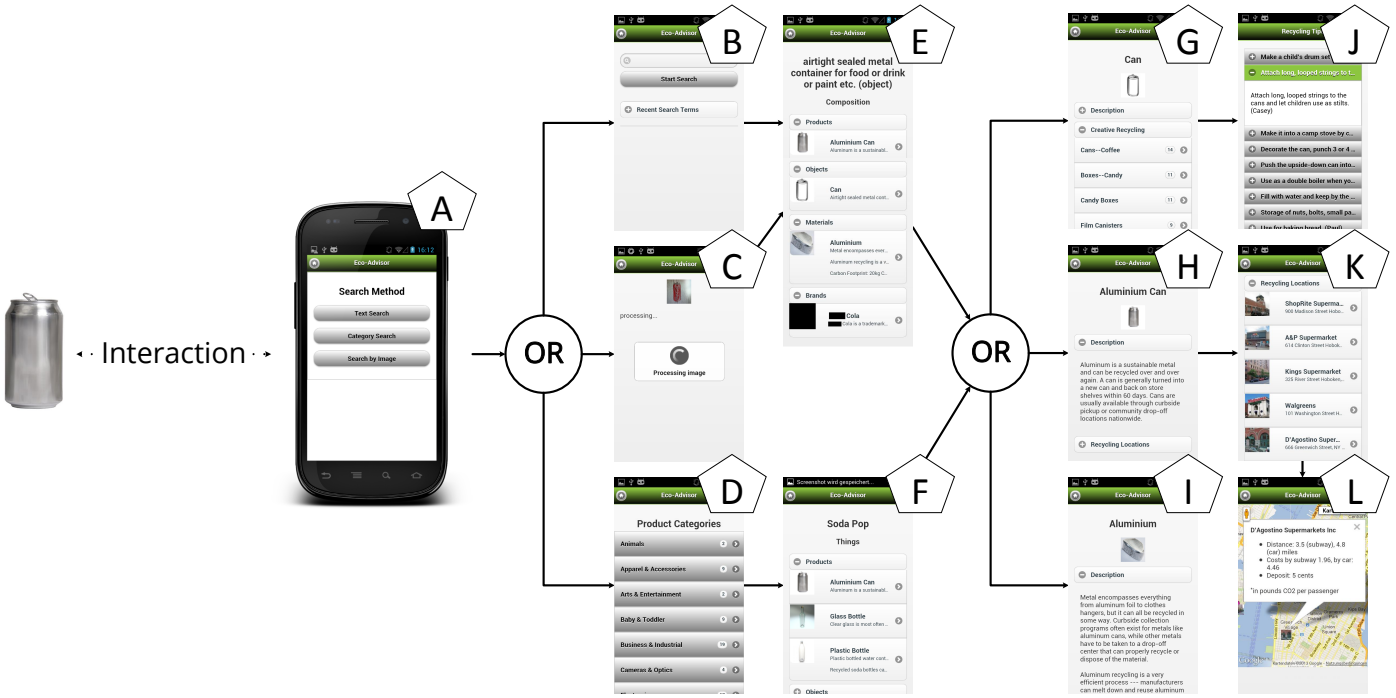


Figure 7. Screens and navigation.

the user selects a product, screen H and K show the product's description and recycling locations nearby. When the user selects a certain location, the application loads the Google Maps view (screen L). For each location the distance to the user's current position, an estimation about the emissions associated with the trip to the location, and the deposit to receive for this product is presented in a bubble. If the user selects a material, the application presents a detailed description about the material and its recycling behavior (screen I). Selecting brands, the user receives a list about associated product's carbon emissions provided by the respective company.

The mobile application was implemented platform-independently using HTML5 on top of the frameworks jQuery mobile and PhoneGap. A prototype running on an Android device was publicly demonstrated [25]. It is subject of the study described in the following.

V. Case Study

In the following, a survey is presented that evaluates the mobile mashup and its underlying data mashup built on top of the domain modeling terms of usability and usefulness. First, the user interface is evaluated to check if the navigation and interaction method is easy to handle for the user. Second, the data mashup stored in the aforementioned domain model is evaluated to find out if the integrated information sources are helpful (1) in the way they are presented, (2) while the user has to solve different tasks from the recycling domain.

A. Research Question and Experimental Design

When the first running prototype of the Eco-Advisor mobile application was finished, feedback was gathered by involving a small probe of people in order to validate concept and basic design decisions. The main question the experiment sought to answer was the following one:

Do the mobile mashup and the domain model help a user to achieve recycling goals more efficiently compared to a stationary Web browser?

Here, "efficiency" comprises various facets of the original task, including quality of result (subjective measures such as user satisfaction, objective quality of recycling), efforts required to perform this kind of recycling, as well as efforts needed to deal with the application (time, interaction steps). In addition, the experiment aimed at gathering information concerning the preferred way of interaction with such a service. Acquiring information from such a service can be realized in quite different ways of interaction ranging from search by text, category, to image taken from the subject of interest.

In order to address these questions, three experimental tasks were defined, which had to be executed by participants of an experiment. These tasks had to be solved with the mobile application on a mobile device ("app variant"), and with a regular web browser ("browser variant"), respectively. The web browser was installed on a regular desktop PC in order to remove effects from potential issues specific to the interaction with mobile web browsers from the experiment (e.g., entering URLs, need for zooming gestures). Furthermore, the web browser was pre-configured in order to support participants in the requested tasks. This setup was chosen based on the assumption that users interested in recycling would have created bookmarks and other pointers to knowledge relevant for performing such tasks. Thus, the browser configuration seeks to reduce search for information sources as such, and instead to leverage search for information using these sources.

During **Task1 (Conventional Recycling)**, the participant

is confronted with an object that has to be recycled in a conventional way in the vicinity. In the browser variant, the participant will find his or her location in an opened Google Maps tab and additional tabs with websites about recycling. The offer of opened websites on a workstation instead of an empty browser on the mobile phone makes the comparison between browser and app variant fairer and prevents the occurrence of a bias. During the study of results, the reader should keep in mind the difference between the two settings.

During **Task2 (Environmental Impact)**, the participant is confronted with a set of objects and is asked to choose the most environmental-friendly one among them. During task execution in the browser variant, the participant can continue his or her Web browser session from Task1.

During **Task3 (Creative Recycling)**, the participant is confronted with one of the objects from Task2. For this object, the participant should search a creative way of recycling, which stands in contrast to conventional ways of recycling in Task1.

During the three tasks, the main factor is the Search for Information regarding the domain of sustainability. Every participant interacts on both levels Web browser and mobile application. Each task is related to one particular hypothesis:

- **H1:** *The mobile application supports a more efficient search for conventional ways of recycling than a common stationary Web browser.*
- **H2:** *The mobile application supports the user in judging an object's environmental impact more efficiently than a common stationary Web browser.*
- **H3:** *The mobile application supports a more efficient search for creative recycling methods than a common stationary Web browser.*

For measuring support of these hypotheses in the respective tasks, the study relies on several parameters: one measurement is time. The time a participant takes to accomplish one task is measured and allows for comparing, which kind of search method (stationary browser/mobile application) leads faster to results. Another measurement is the satisfaction of the user concerning search result and interaction comfort. The participants are asked to rank their opinion in both categories (satisfaction and comfort) on a five point Likert scale (ranging from 1 (disagree) over 3 (neutral) to 5 (fully agree). To check a user's preference, the participant has to select the preferred search variant per task (stationary browser/mobile application). To check if the domain model and the information it provided was helpful, each participant specified the criteria taken into consideration for the decision eventually made at the end of each task.

To receive feedback on usability related aspects, a user rating in the dimensions usefulness, readability, navigation, and visualization is gathered on a 5 point Likert scale, respectively.

Questions about the preferred search mechanism (by text / by category / by image) and ideas for improvement are meant to provide the developer some feedback for further improvements.

The (potential) persuasive nature of the mobile application is tested by asking about the influence of the mobile application on the participant's current recycling behavior: if the information offered by the mobile application would be available during decision making, would people expect a change in their behavior?

Finally, at the end of the study, an overall preference (stationary browser versus mobile application) is asked for.

B. Setup

The experiment was conducted in-lab under the supervision of one instructor. The participants sat at a table in front of a common PC workstation. On the workstation, participants filled out questionnaires and solved the tasks in the browser variant. The instructor guided through the experimental procedure, explained the tasks, and answered questions. For the mobile setting the mobile device Google Nexus S by Samsung was used. The objects during task execution contain three objects from the category soda pop beverages. It was decided to use beverages from one well-known brand, to allow a brand specific search and to avoid that an unknown product will confuse a user. As questions of the survey are answered on the workstation, it can be profited by the advantage of fast result analysis and automated time measurements during the experiment. Most of the questions were of closed nature, while in some cases open questions were asked where the participant had to fill in an answer into the text field, for example the result of each task. All questions were mandatory, except the questions for problems during execution and ideas for improvement. During operations in the browser variant, the browser's history was used to log visited pages and used search terms. During operations on the mobile phone, search terms and navigation paths were logged on server-site.

C. Procedure

The experiment was divided into three phases: In the first phase, the participant had to answer a set of questions on his or her demographical background, the experience level concerning computer, mobile phone, and Internet usage, and the knowledge about recycling. In the second phase, all participants had to solve three tasks. To solve these tasks two tools were provided: a Web browser on the workstation and a mobile phone with an application. For each task the participant had to use the Web browser in the first run and the mobile application in the second. After each run the participant had to answer a set of questions. In order to balance competition of mobile application and browser variant, in the latter one, 7 Web pages were already open in the browser's tabs once a session started. Those pages contained the same content that is integrated in the data mashup behind the mobile application. However, during task execution the participants were allowed to open new tabs and to start an own free search.

In the third phase, the study concluded with questions about the preferred search method, problems during task execution, and ideas for improvement. Additionally, it was asked if the presented mobile application could influence the participants recycling behavior, and if the mobile application would be preferred over the stationary browser.

D. Result

The study lists 22 records, 2 experts and 20 non-experts. The average participant was 26 (median) years old. In the following presentation of the results percentages are rounded to integers. 13 female (59%) and 9 male people (41%) took part. Regarding the occupation, among the participants were 2 pupils (9%), 18 students (82%), and 2 professionals (9%, one software engineer and one researcher). Areas of work are wide spread and include linguistics and translation, computer science and IT, literature and culture, business administration and economics, and education.

The technical experience level regarding the usage of stationary and moveable computers was relatively high. 22 (100%) use a computer that is connected to the Internet, 16 (73%) use a mobile phone with Internet. On the stationary computer 8 (36%) surf more than 20 hours per week and 8 (36%) less or equal than 10 hours per week. On the mobile, only 4 (25%) spent more than 10 hours per week in the internet, while 8 (50%) are only between 0 and 2 hours online. While browsing the Web on the mobile, 4 out of 16 (25%) use predominantly applications. 4 (25%) additionally search for information about products during a shopping trip.

The participants' recycling knowledge was diverse. 19 (86.36%) are recycling their trash, 13 (68%) self-motivated, and 11 (58%) through regulation (multiple selections possible). 13 (68%) consider a product's environmental impact while coming to a decision during a shopping trip. Those who do, consider all different kinds of factors, energy consumption during operation as well as production and packaging. Those who do not, do not have time, are not informed enough, or have other reasons. Additionally, 8 (36%) knew what a carbon footprint is and were able to explain it, in most cases precisely.

Task1: Browser. All participants except one (the participant was not really motivated to spend some minutes on a location search) found a location for the glass bottle. The average distance to the user location was 0.71 miles. Two locations (9%) were subtracted out, one location was a container service and the other a junk hauling service. 4 (19%) identified trash cans, 5 (24%) chose supermarkets, and 10 (48%) identified a recycling center as point of disposal. Decision criteria were distance in most cases (15 / 71%), deposit value in 4 cases (19%), the "fastest result" in 2 cases (9%), and missing information on trash cans in 1 case.

Task1: Mobile application. All participants found a location for the glass bottle. The average distance to the user location was 0.36 miles, 0.35 miles lower compared to the results from the browser search. Distance was the most frequently mentioned decision criteria. Only one participant named carbon emissions associated with the trip as a decision criterion.

The preferred search method for Task1 was the mobile application (15 votes out of 22 / 68%).

Task2: Browser. All participants except one were able to identify one product out of three (glass bottle/plastic bottle/aluminum can) as the most environmental friendly one. 12 (57%) decided for the glass bottle, 6 (29%) for the plastic bottle, and 3 (14%) for the aluminum can. The decision criteria were carbon footprint (17 / 77%), the product's composition into materials (6 / 27%), and studies found through a search engine (1 / 5%). One participant said: "glass bottle is re-usable and I am safe from molecules from the plastic bottle entering my drink".

TABLE I. AVERAGE EXECUTION TIME IN MINUTES

	Browser	***Application***
Task1	8:17 min.	7:10 min.
Task2	7:09 min.	5:17 min.
Taks3	6:26 min.	5:25 min.

Task2: Mobile application. All participants were able to identify one product out of three (glass bottle/plastic bottle/aluminum can) as the most environmental friendly one. 9 (41%) decided for the glass bottle, 10 (45%) for the plastic bottle, and 3 (14%) for the aluminum can. While 43% of the participants changed their mind, 57% kept the decision from the browser variant.

The preferred search method for Task2 was the mobile application (16 votes out of 22 / 73%).

Task3: Browser. All participants except one (95%) found a creative way of recycling for the aluminum can. Several creative ways of recycling were discovered: potting plants, lanterns, aluminum boat, pen and pencil holder, build a children's telephone, tinker decorative items, sculptures, art, camping cooker, solar furnace, ashtray, money box, and so on. Asked, if the knowledge about reusing a product would influence the participant's buying decision was approved by 5 out of 21 / 24%).

Task3: Mobile application. All participants identified a creative way of recycling for the aluminum can. Additional results were a children's drum set, a candy box, a seed storage, a picture frame, gift wrapping, hooks, and film canisters. All participants except 3 (86%) found a new creative way of recycling different from the one they found in the browser variant. Knowledge about reusing the product could influence the participant's buying decision in 9 (41%) out of 22 cases, 17% more compared to the browser variant.

The preferred search method was the mobile application (14 votes out of 22 / 64 %).

Satisfaction and Comfort during the tasks is shown in Figure 8. The time measurement during the tasks resulted in the values that are presented in Table I.

The concluding questions showed that most participants preferred the traditional search mechanisms "search by text" (13 / 59%) to the "search by category" (4 / 18%) and the uncommon "search by image" (5 / 23%). In the four categories usefulness, readability, navigation, and visualization the lowest average rating received the navigation (3.27) on a scale between 1 (worst) and 5 (best). Visualization was rated with 3.36, usefulness with 4.05, and readability with 4.14. Many participants experienced problems to find information placed at the leaf level of the navigation tree although a legend with hints on the underlying content was given on the screen. Room for improvement was seen in the navigation ("too complicated", "less clicking"). One participant suggested placing favorites on the home screen. Another one suggested integrating more pictures to improve the visualization, e.g., to visualize the creative ways of recycling. Asked if the mobile

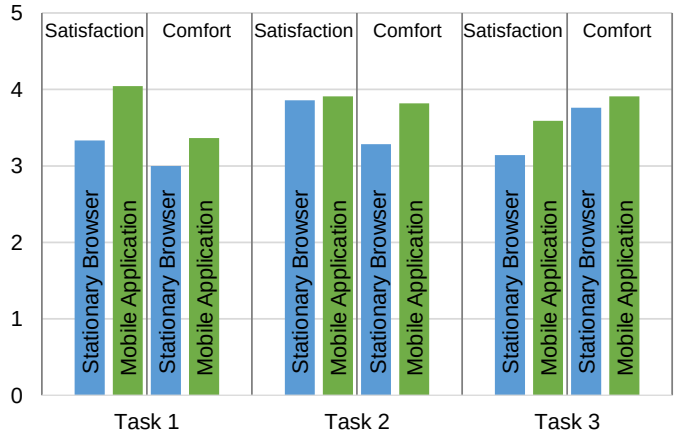


Figure 8. Satisfaction and comfort during task execution (satisfaction: 1=not satisfied, 2=satisfied in parts, 3=indifferent, 4=satisfied, 5=very satisfied; comfort: 1=not comfortable, 2=comfortable in parts, 3=indifferent, 4=comfortable, 5=very comfortable).

application could influence the participants recycling behavior, 73% responded with ”yes”. After all, the mobile application was mentioned as the preferred method of acquiring recycling information (15:7 / 68% : 32%).

E. Findings and Discussion

Feedback obtained in the categories navigation and visualization indicates that potential for improving the mobile application lies in the optimization of navigation concept and the presentation of content. For example, some participants had difficulties to find the content that was necessary to solve the task. Especially pieces of information on recycling locations, which is provided in bubbles on the map, for example information on carbon emissions associated with a trip from the user location to the recycling location, are hard to discover. This information lays 5 navigation steps away from the start screen and hidden behind a 4 categories menu, which is too far. Especially users not familiar with mobile applications in general became frustrated very fast, as they did not understand the mobile application’s concept.

An interesting phenomenon is the development of time that was necessary to solve the tasks (cf. Table I). The first task took in average 7:10 minutes on the mobile application. For Task2 and 3 the duration lowered by about 2 minutes. This fact supports the statement of one participant who said, ”after I was used to the mobile application I found it very helpful”. However, since a mobile application might be installed right before a situation where its support is needed, it should be usable with little to no training. Therefore, this barrier has to be overcome. It has to be mentioned that in this experimental setting only a brief introduction to the mobile application was given. Usually, the user reads a description from the app store and may have a better understanding of the mobile application in advance. Thus, further experiments should start with an informing page about the mobile application as it is common in the big mobile application portals. Nevertheless, having a look on the average task execution times in the stationary browser and the app variant, the app variant outperforms the browser variant in all three tasks. This result underlines that, after understanding the mobile application, the participants were able to find

TABLE II. HYPOTHESE MEASUREMENTS APPLICATION VS BROWSER

	Time	*Satisfaction*	*Comfort*	*Preference*
H1	-1:07 min.	+0.71	+0.36	+36%
H2	-1:52 min.	+0.05	+0.53	+46%
H3	-1:01 min.	+0.45	+0.15	+28%
Avg.	***-1:20 min.***	***+0.40***	***+0.35***	***+37%***

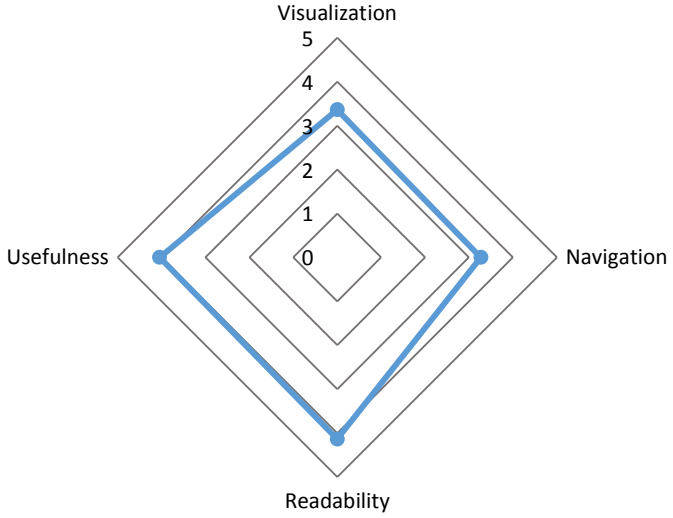


Figure 9. Usability.

information faster using the mobile application than using the Web browser. Having a look at the level of satisfaction concerning the investigated result in Figure 8, the level of satisfaction was higher for the mobile application in all tasks. The perceived comfort during task execution was also higher when searching with the mobile application. The fact that the average distance to the identified recycling location during Task1 was about 0.35 miles lower in the app variant, while distance was the most important criterion for the participants shows that the implemented map visualization was easy to understand. These aforementioned results show that the three task-related hypotheses are supported in all categories, time, satisfaction, comfort, and user preference. Table II depicts the ”delta”, Measurement(Browser) - Measurement(Application), in all categories that were used to measure hypotheses support. Only some users used the uncommon search method ”search by image”. People with a great interest in technics found this search variant ”very nice”.

16 out of 22 (73%) participants reported that the mobile application could influence their recycling behavior. 15 (68%) participants reported that the mobile application is the preferred method of research for the tasks given. Both facts together support the appropriateness of the provided kind of support and indirectly of the employed domain model.

VI. USER INTERFACE IMPROVEMENTS

Since the usability dimensions visualization and navigation performed not so well in the case study (cf. Figure 9), a revised version of the user interface (UI) is provided in Figure 10. The design was inspired by the Google Play Store. In the figure, only the most relevant screens are presented. The remaining screens are designed using the same style. Welcome screen and the three search methods (search by text, search by image,

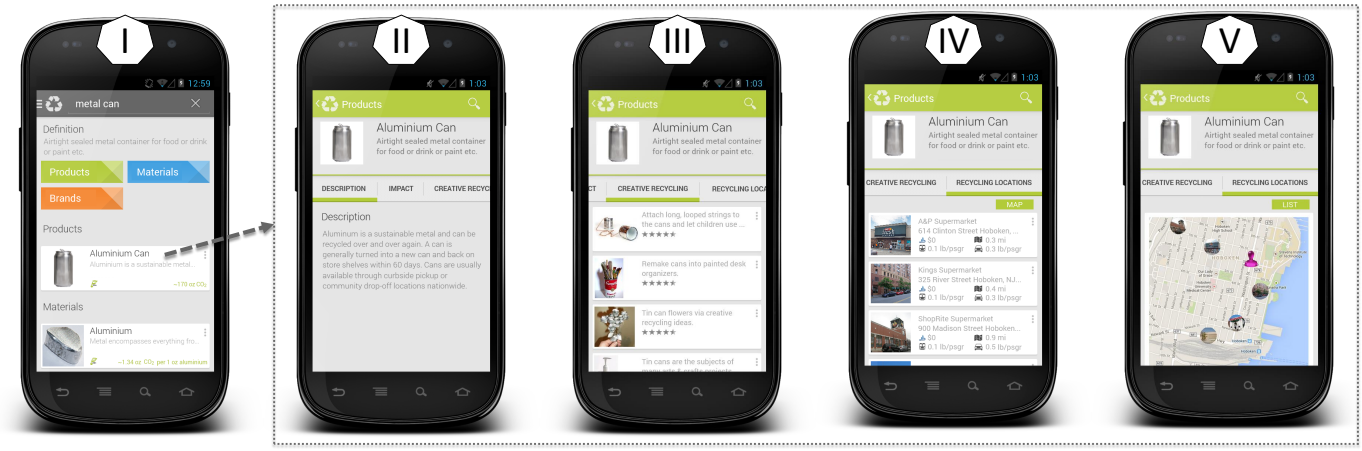


Figure 10. Revision of the user interface based on feedback obtained in the experiment and on metaphors from state-of-the-art mobile applications.

search by category) are left out. They all lead to the new screen I. On screen I, the categories Products, Materials, and Brands are visualized by colors to allow a better orientation. The category Objects was removed because many users in the case study did not understand the concept of abstract objects, which lead to confusion by many participants. The text search field is embedded in the bar on top of the screen (a loupe indicates the search function) to allow an edit. It enables initiating a new search at any time, which directs back to screen I. When a product is selected (in the figure an aluminum can) a screen with four different tabs is presented (screens II–V). Some participants were confused by organization of information behind the categories Products, Materials, and Brands. For them, it was not intuitive that an Object leads to recycling tips and a product to recyclinglocations, for instance. The new interface addresses this issue with a display of all search categories for each category. The first two tabs, description and impact, contain a short textual description of the selected object and its environmental impact. The tab Creative Recycling contains a list of recycling tips. New are the pictures that visualize the tips and a five star user rating that allows users to rate a recycling tip and to see how other users rated it. The tab Recycling Locations contains a list of recycling locations nearby. Deposit value to receive, distance to the location, and estimated carbon emissions associated with the trip to this location by public transportation and by car are presented for each list entry. This information was hard to find on the old UI, as it was hidden in the bubbles on the map visualization. Now, the user has the possibility switch between list and map view (screens IV and V). Compared to the UI in Figure 7, the depth of the navigation tree was reduced by one, which can be ascribed to the fact that the user now simply switches between list and map view of recycling locations.

VII. Conclusion and Future Work

Sustainable behavior requires people to take a considerable amount of diverse information from distributed sources into account for decision making. This article reported on a domain model for a mobile mashup, which integrates such sources automatically. In order to gain feedback concerning the appropriateness of model and system architecture, a case study was conducted. In an experimental setup, participants had to perform recycling-related tasks with a mobile application implementing the mobile mashup approach, and with a browser-based solution on a desktop PC providing similar, but non-integrated features. Findings include that participants were able to find faster more accurate results when using the mobile application. Beyond, they were more satisfied with the mobile application's results and with the way of interaction provided by the mobile application.

Thus, the mobile mashup concept turned out to be of value for supporting people in making recycling-related decisions. However, this conclusion is limited in some ways. For instance, the user group shares certain demographic aspects, and the experiment did not involve true real-world interaction, where time pressure, interruption, and cognitive load might influence the results. Consequently, potential directions of future research should include a revision of the proposed interaction method in order to support new users in getting familiar with the mobile application. Furthermore, positive feedback obtained during the experiment indicates that persuasive technics might combine well with the mobile application concept. A context model could help to involve more user related constraints during decision support.

Acknowledgment

This research was funded in part by the German Federal Ministry of Education and Research under grant number 01IA11001 (project RES-COM). The responsibility for this publication lies with the authors.

References

[1] S. Knoch and A. Kröner, "Enabling mobile access to distributed recycling knowledge," in The Seventh International Conference on Mobile Ubiquitous Computing, Systems, Services and Technologies, UBICOMM '13, W. Narzt and A. Gordon-Ross, Eds. IARIA, Sept. 2013, pp. 30–37.

[2] A. Schiffleitner, T. Bley, R. Schneider, and D. Wimpff, "Stakeholder perspectives on business model requirements for a sustainability data exchange platform across supply chains," in Electronics Goes Green 2012+ (EGG), Sept. 2012, pp. 1–5.

[3] WORLD Environmental Organization, "Recycling database reduce, reuse, recycle," July 2013. [Online]. Available: http://www.world.org/weo/recycle

[4] S. Burn and S. Oskamp, "Increasing community recycling with persuasive communication and public commitment," Journal of Applied Social Psychology, vol. 16, no. 1, Feb. 1986, pp. 29–41.

[5] E. M. Huang and K. N. Truong, "Sustainably ours: Situated sustainability for mobile phones," ACM Interactions, vol. 15, no. 2, Mar. 2008, pp. 16–19.

[6] V. Margolin, "The waste manifesto," ACM Interactions, vol. 16, no. 4, Nov. 2009, pp. 12–15.

[7] Lennox Industries, "Lennox," June 2013. [Online]. Available: https://play.google.com/store/apps/details?id=com.lennox.len

[8] This Is Green, Inc., "This is green," June 2013. [Online]. Available: https://itunes.apple.com/us/app/this-is-green/id337495391

[9] Adbrownies Advertising Ltd., "Low carbon life," June 2013. [Online]. Available: http://www.appleapp.com/low-carbon-life.html

[10] SampathG, "Green footprint calculator," June 2013. [Online]. Available: https://play.google.com/store/apps/details?id=com.virdea.mobile.android.carbonfp

[11] M. Barton, "MeterRead," June 2013. [Online]. Available: http://www.macworld.com/product/71548/meterread-sale-gogreen-save-money-save-earth-.html

[12] PowerTutor.org, "PowerTutor," June 2013. [Online]. Available: https://play.google.com/store/apps/details?id=edu.umich.PowerTutor

[13] The Apple Seed Store, "Green gas saver 1.0," June 2013. [Online]. Available: http://iphone-apps.toptenreviews.com/green/3015-green-gassaver-screenshot4.html

[14] GSH Group, "GSH ienergy," June 2013. [Online]. Available: https://itunes.apple.com/en/app/gsh-ienergy/id377909195

[15] GreenWave Reality, Inc., "DONG energy eFlex," June 2013. [Online]. Available: http://dong-energy-eflex.topapp.net

[16] S. Bechhofer, F. van Harmelen, J. Hendler, I. Horrocks, D. L. McGuinness, P. F. Patel-Schneider, and L. A. Stein, "OWL web ontology language," Dec. 2013. [Online]. Available: http://www.w3.org/TR/owl-ref/

[17] S. Rachuri, Y.-h. Han, S. Foufou, S. C. Feng, U. Roy, F. Wang, R. D. Sriram, and K. W. Lyons, "A model for capturing product assembly information," Journal of Computing and Information Science in Engineering, vol. 6, Mar. 2006, pp. 11–21.

[18] S. Fenves, S. Foufou, C. Bock, and R. Sriram, "CPM2: a core model for product data," Journal of Computing and Information Science in Engineering, vol. 8, no. 1, 2008, p. 014501.

[19] H. J. Pels, "Classification hierarchies for product data modelling," Production Planning & Control, vol. 17, no. 4, 2006, pp. 367–377.

[20] J. H. Lee, S. J. Fenves, C. Bock, H.-W. Suh, S. Rachuri, X. Fiorentini, and R. D. Sriram, "A semantic product modeling framework and its application to behavior evaluation," IEEE Transactions on Automation Science and Engineering, vol. 9, no. 1, 2011.

[21] G. A. Miller, "WordNet: a lexical database for english," Communications of the ACM, vol. 38, no. 11, 1995, pp. 39–41.

[22] D. Merrill, "Mashups: The new breed of web app," in developerWorks. IBM Corporation, July 2009.

[23] E. Maximilien, "Mobile mashups: Thoughts, directions, and challenges," in IEEE International Conference on Semantic Computing, ICSC '08, Aug. 2008, pp. 597–600.

[24] K. Xu, X. Zhang, M. Song, and J. Song, "Mobile mashup: Architecture, challenges and suggestions," in International Conference on Management and Service Science, MASS '09, Sept. 2009, pp. 1–4.

[25] S. Knoch and A. Kröner, "A mashup supporting sustainable decision making," in 9th International Conference on Intelligent Environments, IE '13, 2013, pp. 274–277.

[26] Macroglossa, "Macroglossa," Feb. 2014. [Online]. Available: http://www.macroglossa.com

[27] S. Germesin and M. Romanelli, "terkait: semantic interactions with web content," in 8th International Conference on Semantic Systems, I-SEMANTICS '12, V. Presutti and H. S. Pinto, Eds. ACM, 2012, pp. 169–172.

Query-Based Static Analysis of Web Services in Service-Oriented Architectures

Michael Gebhart
Gebhart Quality Analysis (QA) 82 GmbH
Karlsruhe, Germany
michael.gebhart@qa82.de

***Abstract*—The switch to a service-oriented architecture is often associated with strategic goals, such as an increased flexibility and maintainability of the IT architecture. The design of the services as building blocks directly influences the achievement of these goals. For that reason, in recent years best practices and patterns have evolved that describe how to design services and how to implement them by means of web service technologies. However, the best practices and patterns that focus on the architectural issues are often too abstract to be verified on concrete web service artifacts. Previous work describes how these best practices and patterns can be broken down into measurable quality indicators. This article shows a query-based approach for a static analysis to measure these quality indicators on implemented web services. To illustrate the approach, services of an automotive scenario are developed using a product that realizes the introduced concepts.**

Keywords-soa; web service; design; quality; metrics

I. INTRODUCTION

This article is an extended version of [1]. When companies switch to a service-oriented architecture (SOA) as paradigm to structure their IT architecture, in most cases strategic goals are the main drivers. Typical strategic goals are to increase the flexibility and maintainability as the ability to realize new business requirements within shortest time has become a critical success factor for companies [2][3]. In the past, experiences have shown that the success of SOA projects is influenced by the design of the architecture especially its service layer [4]. On a service layer the architecture focuses on the design of service interfaces, service components, and their dependencies. Decisions, such as the grouping of operations to services and their granularity, impact the achievement of the previously described goals.

For that reason several best practices and patterns from a conceptual point of view have evolved that describe how to design services in a way that they support the achievement of these strategic goals. These best practices include hints, such as how to group operations to services and what is important to consider regarding their names. When starting with the implementation, further guidelines provide information about how to implement services using a certain technology. While SOA does not dictate any technology usage, in most cases web services are applied as their standardization supports the flexibility and maintainability of the architecture from a technical point of view [5]. In this case, the web services are described using the World Wide Web Consortium (W3C) standards Web Services Description Language (WSDL) [6] and XML Schema Definition (XSD) [7]. Furthermore, in some projects the Service Component Architecture (SCA) [8] standardized by the Organization for the Advancement of Structured Information Standards (OASIS) is applied to describe the component model.

Though both the best practices and patterns from a conceptual point of view and the guidelines from a technical point of view provide valuable information, there is a gap between both approaches. On the one hand, best practices that focus on architectural issues from a conceptual point of view are too abstract to be verified on concrete web service artifacts. On the other hand, the technology-specific guidelines that describe how to implement services using a certain technology are not related to strategic goals which hampers their motivation. As result, for architects and developers who want to design and implement web services that consider existing architectural best practices it is hard to verify that they have done everything correct.

In previous work, we have shown how to close this gap: In [9], we have described how to break architectural best practices down into measurable quality indicators that can be verified on concrete artifacts, such as web services. The quality indicators can be formalized using metrics that enable an objective and repeatable quality analysis. The metrics already provide the first step to evaluate web services systematically. However, for an efficient application in development processes the metrics have to be measured automatically on concrete technology elements of web services, such as WSDL documents and SCA artifacts. For that reason, this article introduces a query-based static analysis (QSA) approach that includes the mapping of metrics and their constituents onto elements of web services implementation artifacts for an automatic execution.

The concept is illustrated using a scenario in the context of automotive manufacturing. In this case, the usage of formalized metrics helps to systematically design web services and to coordinate several developers. Furthermore, the concepts are integrated into the QA82 Analyzer as product for analyzing software and data. The product enables the automatic measurement of the design quality of the created web services, thus increases the efficiency.

The article is organized as follows: Section II introduces existing best practices and patterns for web services, their formalizations, and their automatic measurement. The scenario is introduced in Section III. In Section IV, the services for the scenario are developed using a quality model, the QSA approach, and our product. Section V concludes this article and introduces future research work.

II. BACKGROUND

This section describes best practices for the design of services in service-oriented architectures. Furthermore, this work is examined regarding its possibility to be efficiently measured on web services using tools. In addition, work in the context of evaluating software regarding best practices is considered. The technologies of web services, such as WSDL, XSD, and SCA are not further introduced in this article. They are assumed to be well known.

The service design phase is an essential ingredient of software service engineering that can be defined as the "discipline for development and maintenance of SOA-enabled applications" [10]. The service design phase includes design decisions about the interface of a certain service, such as its grouping of operations, and its internal behavior. As services constitute the building blocks of an SOA, they determine the design of the entire architecture. In the last years, for services several best practices and patterns have evolved.

In [4] and [11], Erl describes numerous patterns for services in particular web services. They have been derived from experiences in real-world projects and provide valuable hints for architects and developers. Nevertheless, all guidelines are only textually describes. This results in ambiguities and requires interpretation before using it in concrete projects. This again may result in faulty applications.

Similar to Erl, also Cohen [12] and Josuttis [13] focus on patterns from a similar point of view. While the guidelines are clearly motivated, their usage in projects similarly requires interpretation. Furthermore, due to the textual description concrete artifacts cannot be checked against these guidelines without manual effort.

A more academic approach is chosen in [14] and [15]. Perepletchikov et al. introduce metrics for quality attributes, such as loose couplings. These metrics consider formalized service designs independent from concrete technologies. The essential benefit of this work is its ability to perform an automatic measurement. However, the motivation of the introduced metrics is not obvious. Work as introduced by Erl and Josuttis that is derived from real-world projects is not reflected by the metrics. This is even not possible as Perepletchikov et al. consider an abstract formalization of services. Most of the best practices introduced by Erl and Josuttis refer to elements that are not part of the formalization used by Perepletchikov et al. Furthermore, the abstract formalization is not mapped onto concrete technologies, such as web services. This hampers the measurement of the introduced metrics in real-world projects and requires additional effort.

Similarly to Perepletchikov et al. [14][15], Hirzalla et al. [16] and Choi et al. [17] introduce metrics for services. Also in this work, the metrics are very abstract and cannot be directly applied in projects. Even though they are formalized which reduces interpretation effort, they do not represent best practices as introduced by Erl and Josuttis which hampers their motivation. These metrics should be associated with best practices of real-world projects. A mapping onto concrete web service technologies would enable their application in concrete projects.

To fill this gap, in previous work [9] we created a quality model that combines best practices as introduced by Erl et al. [4][11] with a formalization as used by Perepletchikov et al. [14][15]. The quality model was aligned with the Service oriented architecture Modeling Language (SoaML) [18] as profile for the Unified Modeling Language (UML) [19] that is meant to replace proprietary UML profiles for services, such as the one developed by IBM [20][21][22]. As result of this work, an SOA formalized using SoaML can be checked against wide-spread best practices. The usage of SoaML is explained in [23][24] and a case study that applies the metrics is presented in [25]. However, in most cases web services are created or are already existent without a formalization based on SoaML. Furthermore, some best practices refer to elements that are not part of a SoaML-based description. Thus, an approach is necessary that is applicable on web services directly.

In [26], it is shown how service designs based on SoaML can be transformed into web services using WSDL, XSD, and SCA. This work was not necessarily created with quality analysis in mind. However, it can be applied to transfer the service design metrics based on SoaML to web services.

The summary of existing work in the context of best practices for web services shows that a lot of good work exists, which focuses either on the description of best practices, patterns, design guidelines etc. for web services or on a formalization of academic metrics. Whilst the former are too imprecise to be efficiently measured as they are only textually described, the latter are too academic to be comprehensible understandable and motivated. For that reason, we use the metrics introduced in [9] that on the one hand represent best practices and on the other hand are formalized so that they can be automatically measured. They are transformed so that they can be applied on web services using the mapping rules described in [26]. As result, metrics are available that can be directly be measured on web services and their development artifacts. However, there is also a mechanism for the measurement itself necessary.

Next, existing work to evaluate software artifacts, such as the described web service artifacts, regarding best practices and patterns is examined.

A typical approach to evaluate implementation artifacts regarding a certain architecture specification is the usage of software reflexion models as shown by Murphy et al. in [27]. This approach is helpful to find differences between two models, mostly a specification and a source code model. By this means, inconsistencies between an architecture specification and its implementation can be identified. However, this approach is not applicable to analyze an

architecture regarding best practices as the compared models have to be on the same level. Best practices describe rules that refer to elements of the metamodel. Thus, they are not described on the same level as the source code model.

A more applicable approach is shown by Giesecke et al. in [28]. In this work, architecture styles represent the basis for architecture evaluations. Even though this is the only work of the authors in this context and there is no example described, it can be recognized that the basis for the evaluation is an architecture model that is derived from the source code and has to be described in a certain language. The essential disadvantage of this approach is the limiting metamodel the architecture model bases on. Best practices can refer to many different aspects of an architecture that go beyond components and their dependencies. When creating an architecture model from the source code, all these specifics the best practices refer to have to be available in the architecture metamodel and have to be considered when mapping the source code to the architecture model. Furthermore, especially when considering best practices that are more technology-specific, either the metamodel has to be extended in a way that it represents all these technology specifics or information gets lost and the best practices cannot be verified. Another approach could be to check some best practices on a general architecture model and some other best practices on the source code directly. However, our experience is that this results in further complexity: First, again a mapping mechanism is required to get the architecture model from the source code. And second, to check the technology-specific best practices two approaches are necessary: One to verify the architecture model and one to verify the source code.

We suggest to unify the evaluation methodology to reduce complexity. The consideration of the entire wide range of best practices would result in complex mapping rules and a very complex architecture metamodel. This is exactly the reason why we propose not to derive an abstract architecture model from the source artifacts, such as source code, but directly work on the source artifacts using a query-based approach.

III. SCENARIO

To illustrate the query-based static analysis approach for the evaluation of web services, a scenario from automotive manufacturing is chosen. A service landscape has to be created that supports the manufacturing of cars. A new service has to be provided that offers functionality to initialize the manufacturing of a new automobiles. Meta data about the manufactured automobile are expected to be stored in external systems. Furthermore, the construction system has to be triggered.

The project team consists of two developers and one product and quality manager who coordinates the developers and delivers reports to the management and the customer. In some cases, the role of the product and quality manager might also be fulfilled by an architect, who is responsible for the design of the architecture and its quality. Fig. 1 illustrates the participants and their relationships.

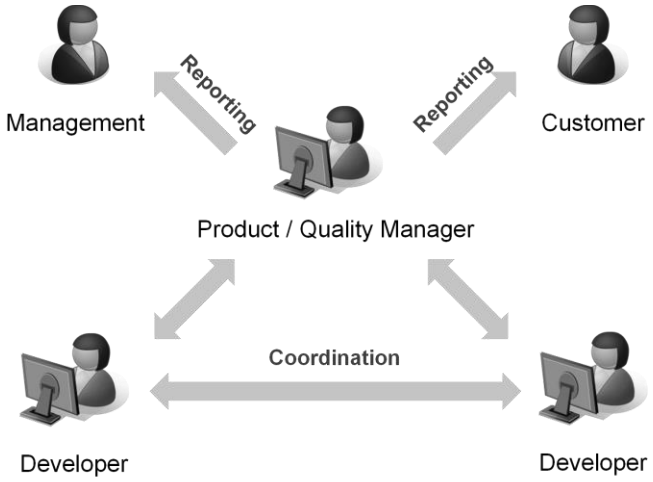


Figure 1. Participants and their relationships.

According to this figure, the product and quality manager has an interest in proving the high quality of the created software. In this scenario, besides functional requirements especially the architectural design is of interest. So it is necessary that developers consider best practices and patterns that support the achievement of a flexible and maintainable architecture. Furthermore, the product and quality manager is required to analyze software artifacts regarding these quality requirements. To support this quality assurance, this article shows how to analyze artifacts, such as web service interfaces, regarding wide-spread best practices and guidelines for services.

The scenario begins with the development of a service for the manufacturing of automobiles by the first developer. An SCA Composite is created, which combines a service for manufacturing automobiles and a service for filing manufactured automobiles in the database. Furthermore, for all services appropriate web services interfaces using WSDL are developed. The artifacts are filed in a shared Git repository. Fig. 2 illustrates the composite using the graphical representation introduced in the official SCA standard. In the scenario, originally a proprietary tool is applied that uses a different visualization.

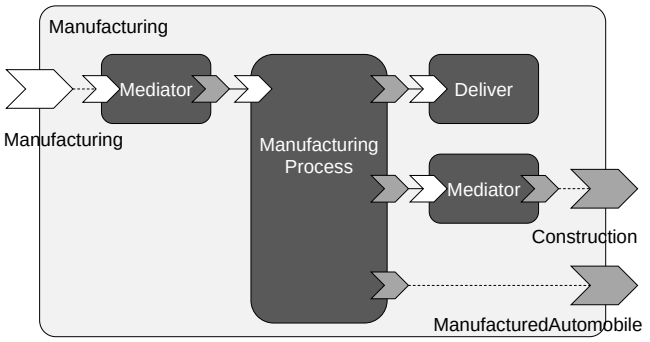


Figure 2. Created SCA composite.

Starting with this SCA composite the product and quality manager determines the quality of the architecture using the approach introduced in the following section. The used WSDL documents are shown later in this article. The results of the quality analysis will help both the product and quality manager and the developers to revise the architecture in a quality-oriented way.

IV. Query-based Static Analysis of Web Services in Service-oriented Architectures

In this section, the query-based static analysis approach is applied to automatically analyze web services in service-oriented architectures regarding best practices. For that purpose, first the applied quality model for web services in service-oriented architectures is shown. As this quality model describes the quality of web services only on a conceptual lever, a mapping onto web service artifacts, such as WSDL documents and SCA artifacts, is described. Finally, based on this mapping the analysis is automated using the query-based static analysis approach.

A. Quality Model for Web Services

To determine the quality of software, one approach is to refine the term quality until it can be measured. A wide-spread quality model methodology is Factor, Criteria, Metric (FCM) introduced by McCall et al. in [29]. According to this methodology a factor is refined into more fine-grained criteria that again are refined into quantifiable metrics. Similar approaches use the equivalent terms quality characteristics, quality sub-characteristics, and quality indicators.

Correspondingly, applied on the design of web services in service-oriented architectures the term quality from a design perspective has to be broken down into measurable aspects that can be formalized by means of metrics. In [9], a quality model has been created that enables the measurement or at least systematic evaluation of services regarding best practices and patterns that have evolved as important for service-oriented architectures. The quality model is shown in Fig. 3 in a tree structure.

In recent work, the quality model has been formalized on basis of Service oriented architecture Modeling Language (SoaML) as language to formalize the architecture. When the product and quality manager of the scenario in Section III tries to apply this quality model, the usage of SoaML hampers the direct application. As in the scenario other technologies, in particular WSDL, XSD, and SCA are used, the metrics introduced in [9] cannot be applied without additional effort. However, in [26], a mapping between SoaML and web service technologies is described. The combination of this work enables the mapping of metrics onto web services so that they can be directly applied. This mapping is shown next and constitutes the basis to implement the query-based analysis approach.

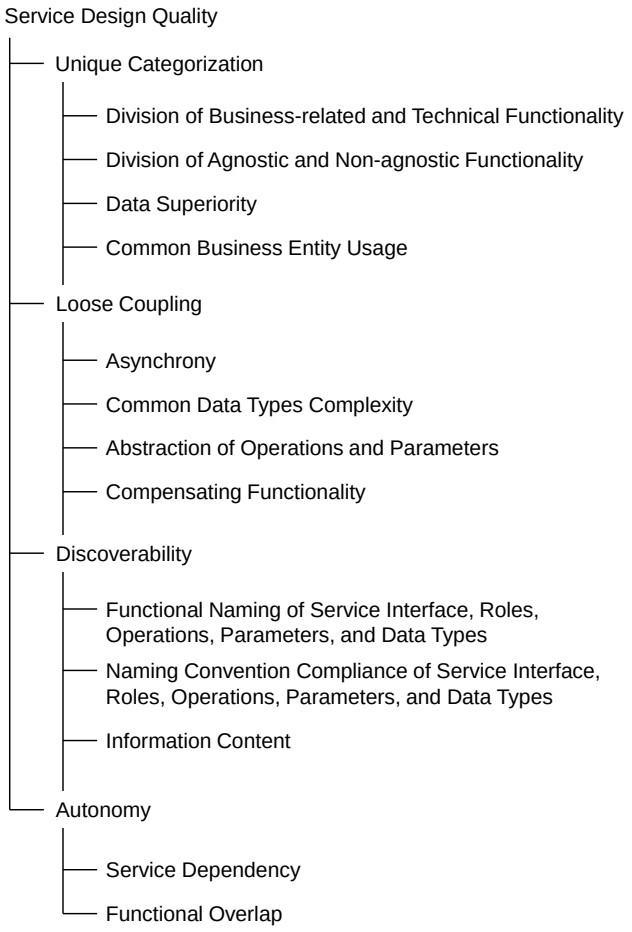


Figure 3. Quality model for web services in service-oriented architecture.

B. Application on Web Service Implementation Artifacts

According to Gebhart et al. [9] in particularly four quality sub-characteristics or criteria can be considered as relevant for the design quality: unique categorization, loose coupling, discoverability, and autonomy. Even though this set of quality characteristics is not expected to be complete it is a good starting point to evaluate the design of a service-oriented architecture and to illustrate the approach.

To apply these quality sub-characteristics on concrete web service implementation artifacts, a mapping of the quality indicators and their metrics is required first. In this section, especially the unique categorization as quality sub-characteristic is considered. This sub-characteristic is comparable to the concept of cohesion in object-oriented systems. It consists of four quality indicators with metrics introduced in [9][30][31]. To illustrate the approach, these metrics are mapped and applied to analyze the service-oriented architecture design.

1) Division of Agnostic and Non-Agnostic Functionality: The background of this metric is that generic functionality should be seperated from specific one so that changes regarding the specific operations do not affect the highly reused ones. It has its origin in the patterns described by Erl [4].

$$DANF(s) = \frac{\left|AF\left(O\left(RI(SI(s))\right)\right)\right|}{\left|O\left(RI(SI(s))\right)\right|} \quad (1)$$

To apply this metric for the scenario, the functions and variables have to be mapped onto elements within XSD, WSDL, and SCA. Table I shows a brief introduction of the element and afterwards a mapping. This mapping specifies where to find this information.

TABLE I. VARIABLES AND FUNCTIONS USED FOR DANF

Element	Description and Mapping
DANF	Division of Agnostic and Non-agnostic Functionality
s	service: the considered service that is provided or required It is represented by a SCA Service or Reference element.
SI(s)	Service Interface: service interface of the service s It is represented by the WSDL document that describes the SCA Service or Reference.
RI(si)	Realized Interfaces: realized interfaces of the service interface si. It is represented by the WSDL PortType that includes provided operations of the service.
O(i)	Operations: operations within the interface i The WSDL Operations within the identified WSDL PortType are expected to be returned.
AF(o)	Agnostic Functionality: operations providing agnostic functionality out of the set of operations o This information has to be determined by an IT expert. It cannot be found within the web service technologies.
| o |	Number of operations o

As result a value of 0 or 1 is desired. These values mean that the service operations provide only agnostic or only non-agnostic functionality. A value between 0 and 1 means that agnostic and non-agnostic functionality has been mixed. In this case, the participants should revise the design. For example the provided operations can be separated into several services.

Based on this mapping information, the metric can be applied for the Manufacturing service that is the SCA Service within the SCA Composite. According to the metric, in a first step the service interface has to be identified. This is the WSDL file Manufacturing.wsdl. Next, the WSDL PortType comprising the provided operations within the WSDL is selected and finally, the operations themselves are returned. Fig. 4 shows the proceeding.

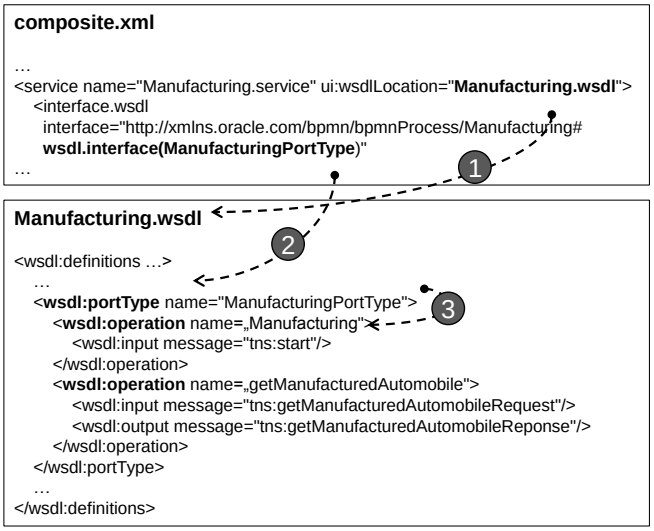


Figure 4. Determination of DANF metric.

After the relevant operations have been identified, the product and quality manager has to decide whether these operations are agnostic or non-agnostic. If he is not capable to answer these questions, he has to ask the developers and estimate the reusability of these operations. In this case, the quality manager comes to the conclusion that the operation "Manufacture" is non-agnostic as it is very specific and cannot be used in other contexts. The operation "getManufacturedAutomobiles" however is agnostic as it provides functionality to request manufactured automobiles, which can be reused in several scenarios. As result the metric returns 0.5, which represents a suboptimal value.

2) Division of Business-Related and Technical Functionality: A metric similar to DANF is DBTF that targets the division of business and technical functionality. It can be mapped in a similar way.

$$DBTF(s) = \frac{\left|BF\left(O\left(RI(SI(s))\right)\right)\right|}{\left|O\left(RI(SI(s))\right)\right|} \quad (2)$$

TABLE II. VARIABLES AND FUNCTIONS USED FOR DANF

Element	Description and Mapping
DBTF	Division of Business-related and Technical Functionality
BF(o)	Business-related Functionality: operations providing business-related functionality out of the set of operations o This information has to be determined by an IT expert. It cannot be found within the web service technologies.

Also in this case, a value of 0 or 1 is desired. These values represent the case that a service provides either only business-related or only technical functionality. In our scenario, all functionality is business-related.

3) Data Superiority: This quality sub-characteristic describes that a service that manages an entity is exclusively responsible for managing it. The metric can be formalized as follows. Most functions have already been described. The others are explained in Table III.

$$DS(s) = 1 - \frac{\left| \begin{matrix} ME\left(O\left(RI(SI(s))\right)\right) \cap \\ ME\left(O\left(RI\left(SI((ALL_S \setminus s))\right)\right)\right) \end{matrix} \right|}{\left| ME\left(O\left(RI(SI(s))\right)\right) \right|} \quad (3)$$

TABLE III. VARIABLES AND FUNCTIONS USED FOR DS

Element	Description and Mapping
DS	Data Superiority
M1 \ M2	Elements of set M1 without elements of set M2 or the element M2
ALL_S	All existing services Represented by all SCA Services
ME(o)	Managed Entities: entities that are managed by operations o This information has to be determined by an IT expert. It cannot be found within the web service technologies.

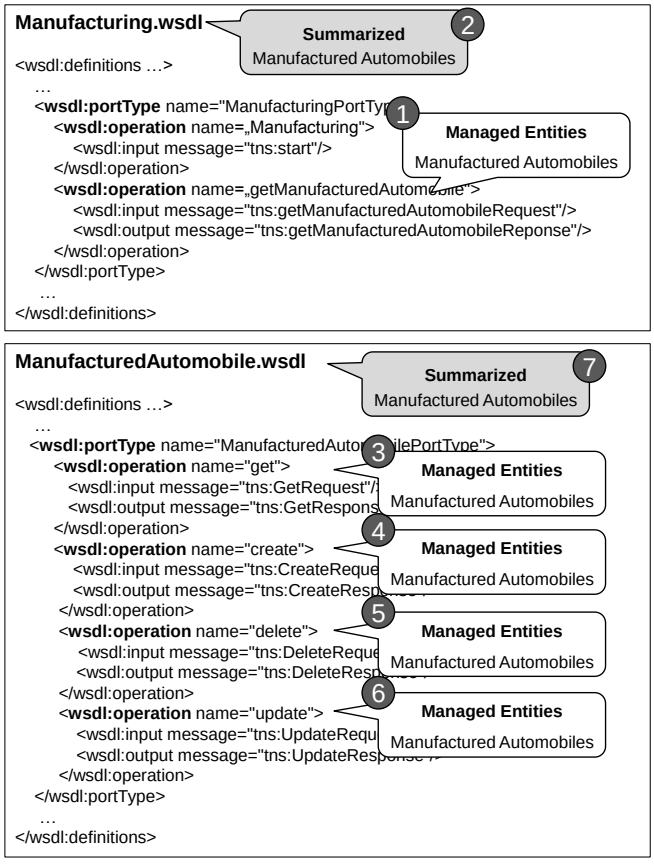


Figure 5. Determination of DS metric.

To illustrate this metric we assume that the ManufacturedAutomobile Reference within the SCA Composite refers to a service described by the ManufacturedAutomobile.wsdl and that no other services are relevant for this metric.

To calculate the metric, the product and quality manager has to consider the provided operations of the Manufacturing service and of all other developed services, i.e., the ManufacturedAutomobile service in this case. Afterwards, the product and quality manager has to decide for each operation whether an entity is managed by this one. Finally, he has to compare the set of managed entities of the services to identify conflicts. Fig. 5 illustrates the proceeding for the Manufacturing service. According to this figure all entities managed by the Manufacturing service are not exclusively managed. The ManufacturedAutomobile service that corresponds to an entity service [1][4] manages manufactured automobiles too. So from a data superiority perspective the Manufacturing service is not ideal and should be revised.

4) Common Entity Usage: Finally, the last quality indicator of the unique categorization quality sub-characteristic can be measured. According to the common entity usage metric, all operations within a service should work on the same entities. This guarantees that entities that do not belong together are managed by different services. In turn, the prior described data superiority ensures that operations that manage the same entities are part of one service.

$$CEU(s) = \frac{\left| OUE\left(\begin{matrix} O\left(RI(SI(s))\right), \\ CMP\left(\begin{matrix} O\left(RI(SI(s))\right), MOUE\left(O\left(RI(SI(s))\right)\right), \\ UE\left(O\left(RI(SI(s))\right)\right) \end{matrix} \right) \end{matrix} \right) \right|}{\left| O\left(RI(SI(s))\right) \right|} \quad (4)$$

TABLE IV. VARIABLES AND FUNCTIONS USED FOR CEU

Element	Description and Mapping
CEU	Common Entity Usage
CMP(o, e1, e2)	Composition: biggest set of entities managed by operations o out of e2 that depend on entitites e1
UE(o)	Used Entities: entities that are used within operations o as input
MOUE(o)	Mostly Often Used Entities: entities that are mostly often used within one operation out of operations o
OUE(o, be)	Operations Using Entities: operations out of operations o that only use entities out of be

This table shows that there is no explicit mapping to web services necessary. All functions that refer to certain elements within a technology have already been mapped by the functions described in Table I and Table III.

Applied on the Manufacturing service, the metric returns the value 1 as all operations that manage entities manage the same. This is also the case for the ManufacturedAutomobile

service. As this entity service provides Create, Read, Update, Delete (CRUD) operations for the same entity, this metric is also ideal for this service. If the ManufacturedAutomobile service would also manage another entity, the CEU metric would return a suboptimal value.

C. Query-Based Static Analysis Approach

The previous section illustrated the mapping of conceptual metrics onto web service artifacts. In this section, the QSA approach is introduced that is afterwards applied to automate the web service evaluation.

As mentioned in the Background section, one central disadvantage of existing architecture evaluation approaches is the usage of an architecture model that is derived from the source code. This architecture model is a representation of a source, however it is limited to the elements defined in a metamodel. This means that either information is lost or that the metamodel and the mapping mechanism has to be enhanced in a way that all necessary information is considered. However, as best practices cover a wide range of information this approach is not practicable. As some best practices refer to technology specifics, the metamodel and the mapping mechanism would escalate. The approach is illustrated in Fig. 6.

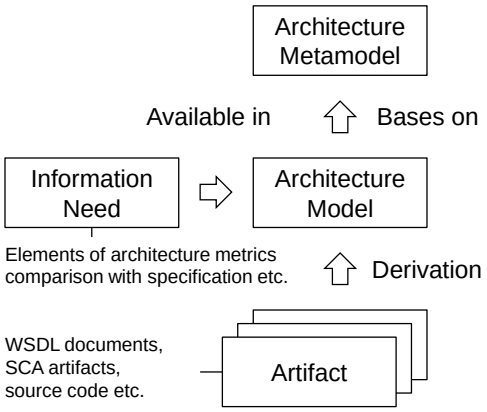


Figure 6. Usual architecture evaluation approaches

The alternative approach as proposed in this article is to query necessary information from artifacts when they are needed. This means that there is no architecture model derived from the source code. Instead we use mechanisms to find information directly in the web service artifacts when they are required. Fig. 7 shows the query-based approach.

Similar to the architecture evaluation approach in Fig. 6, we start with an information need. For example, metrics or their elements, such as the number of available services or the operations of a certain operations, are expected to be determined. Also, the comparison of the architecture to a certain specification might be an information need. Compared to the approach in Fig. 6, the query-based approach does not work on an architecture model that is derived from the artifacts, such as WSDL documents, SCA artifacts, or source code. Instead, a central component, in this case called Analyzer, receives the information need and tries to satisfy it. For that, the Analyzer component has a repository of so-called information providers.

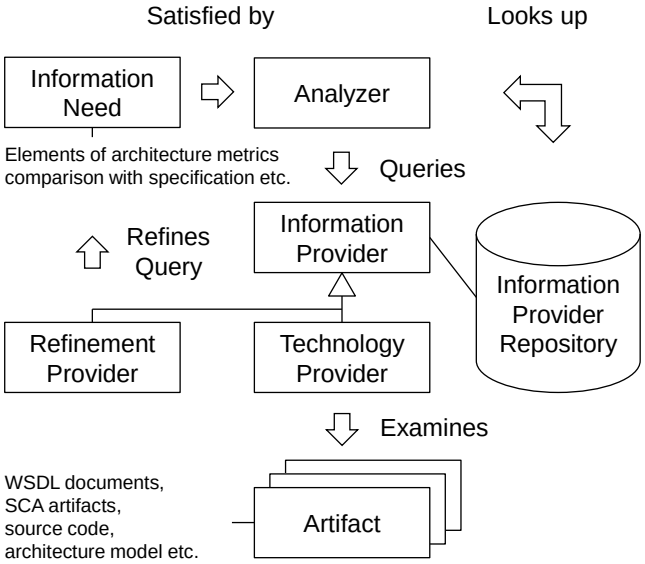


Figure 7. Query-based analysis approach.

An information provider is able to receive a certain query and answer it depending on the expected result. For example, if the information provider is requested to return the number of services within a service-oriented architecture as Integer, the information provider is able to understand this information need and return it in the expected format. We distinguish between Technology Providers and Refinement Providers. Technology Providers are able to answer a query on basis of information contained in certain artifacts, such as WSDL documents, SCA artifacts, source code, models, databases, and so on. For example, if the query is to get all provided services in a service-oriented architecture, a WSDL Technology Provider and a SCA Technology Provider will be called. These providers access WSDL documents and SCA artifacts within the architecture and determine the services in the architecture and return them to the Analyzer component. A Refinement Provider receives a query, refines it into several information needs, and creates the result for the original query depending on the results for these refined information needs. For example, if the query is to get all operations within the service-oriented architecture, a SOA Refinement Provider refines this query into 1) an information need to get all services in the architecture and 2) to get the operations for each of these services. The refined information needs are answered in the same way, i.e., they are satisfied by the Analyzer component that uses information providers. The SOA Refinement Provider uses these results to generate the result for the original query and returns it to the analyzer.

Besides the flexibility and reduced complexity, another advantage of this approach is that for each information provider a different implementation language can be chosen. When deriving a central architecture model from implementation artifacts, often the same transformation language has to be used for all artifacts. In our approach, an information provider that is expected to analyze XML artifacts, such as WSDL documents and SCA artifacts, can

be implemented in another language than an information provider that is expected to work on databases or on hardware information, such as card readers. This has the big advantage that for every purpose the most suitable language can be chosen and that all information that can be requested by any technology can be actually requested and reused in the analysis. Furthermore, languages most people are used to, such as Java and C#, can be applied and no proprietary languages have to be learned. In our case, most of the information providers are directly implemented using Java. This reduces the development time for new information providers and increases the adoption of this approach in real-world projects.

Furthermore, when there is a new artifact that is expected to be considered during the analysis only a new information provider has to be added and the existing logic does not have to be changed. For example, when in the future besides WSDL and SCA also the Business Process Model and Notation (BPMN) 2.0 language is expected to be considered, we only have to add a new technology provider for BPMN that is able to answer queries related to this language. This increases the flexibility and maintainability of this analysis methodology.

D. Query-Based Static Analysis for Evaluation Automation

In this section, the QSA approach is used to automatically evaluate web service artifacts. For that purpose, the mapping knowledge introduced before is implemented as information providers. As illustrated in Fig. 7, an information need can be a metric or elements of a metric. This information need is then sent to the Analyzer component so that it can be satisfied. For that purpose, the Analyzer component uses one or several information providers.

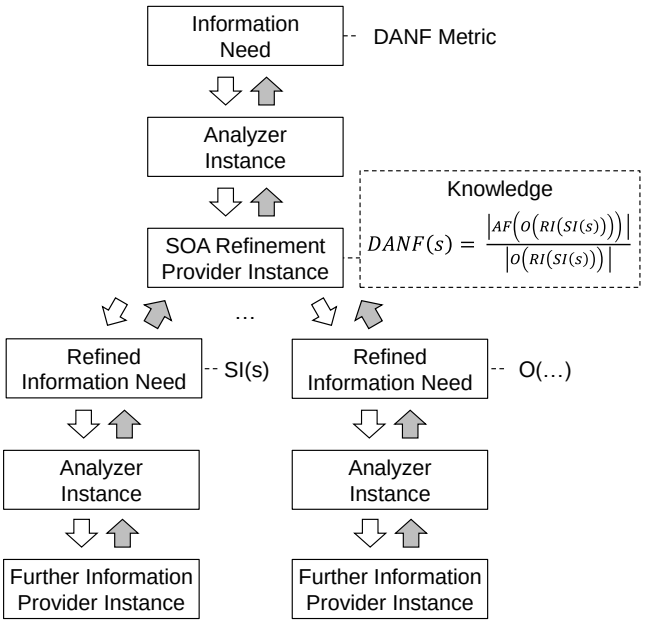


Figure 8. Interaction between Analyzer and SOA refinement provider.

These are able to answer the query. In the previous section, the metrics were introduced and mapped onto web service artifacts. This mapping has shown that 1) the metrics mostly consist of several elements that have to be requested separately and 2) the metrics refer to information kept in WSDL documents and SCA artifacts. Thus, to automate the metrics the following information providers are required:

1) SOA Refinement Provider: The SOA Refinement Provider contains the metrics and describes their refinement. For example, the SOA Refinement Provider knows how to calculate the value*s* for *the* DANF metric. It breaks this metric down into the *metric* elements, requests their result from the Analyzer component, and generates the result for the original query. The interaction between the SOA Refinement Provider and the Analyzer component is shown in Fig. 8.

WSDL Technology Provider: The WSDL Technology Provider examines WSDL artifacts regarding certain information needs. For example, when the information need is to get all services within the architecture, the WSDL Technology Provider checks all WSDL files in the architecture and examines them regarding the service XML element. Aftwards, an element as representer for this service is returned to the Analyzer component. In our case, the WSDL Technology Provider is implemented using Java as the Analyzer component is implemented in Java too. However, in our implementation any other language based on the Java runtime can be chosen. It is also possible to switch from operation calls within the Java Virtual Machine to external web service calls etc. In this case, any other language would be possible too. Fig. 9 illustrates how the WSDL Technology Provider interacts with the Analyzer component.

2) SCA Technology Provider: Similar to the WSDL Technology Provider, the SCA Technology Provider examines SCA artifacts, such as SCA composites. This means that when the Analyzer component needs to satisfy an information need, such as the available services or the service interfaces for a certain service, the SCA Technology Provider examines the SCA artifacts regarding this information. Fig. 9 shows how this information provider works and how it interacts with the Analyzer component.

It is important to mention that several information providers can answer the same type of query. For example, both WSDL documents and SCA artifacts can provide information about available services in the architecture. Also, BPMN processes can provide information about available processes and provided services. So when BPMN is used, also this information has to be considered.

The QSA approach allows to query information independent from how it can be answered. The information provider simply receive the query and try to answer it based on their knowledge. Afterwards, they return the result if possible. The Analyzer component however is responsible to merge the results. So when several available services are returned, both from the SCA Technology Provider and the WSDL Technology Provider, the Analyzer component tries to merge the result so that a holistic view is guaranteed.

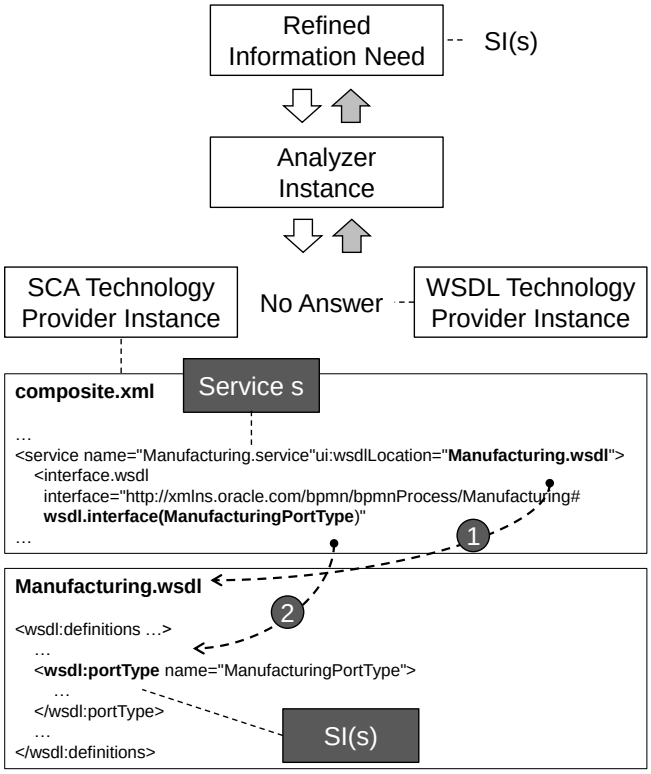


Figure 9. Interaction between Analyzer and technology providers.

E. *Manual Information*

In Section IV.B the mapping of the conceptual metrics onto web service artifacts has been explained. As part of this mapping it was shown that there is some information that cannot be found in the implementation. For example: What about the business-relation of a service operation? Or what about its agnosticity? There is some information that cannot be determined automatically on basis of existing artifacts. To solve this issue, we have created an additional information provider, the Custom Function Results Provider that represents manual information that can be added by experts. For example, when it is not known if a certain operation is business-related, this knowledge can be added by experts and then it will be considered during the analysis process.

F. *Tool Support*

Based on these concepts, we have developed a product that realizes the QSA approach and enables the evaluation of web services regarding the quality model introduced in [9]. The QA82 Analyzer is a generic quality management platform that implements these concepts. Based on the QA82 Analyzer we have developed the QA82 SOA Compliance Center, which implements the SOA quality model introduced before and provides the described information providers. As result, the product and quality manager can automatically perform quality analyses of the developed web services. To demonstrate the behavior of the application we have used it to analyze the introduced scenario. Fig. 10 shows how knowledge can be added by experts.

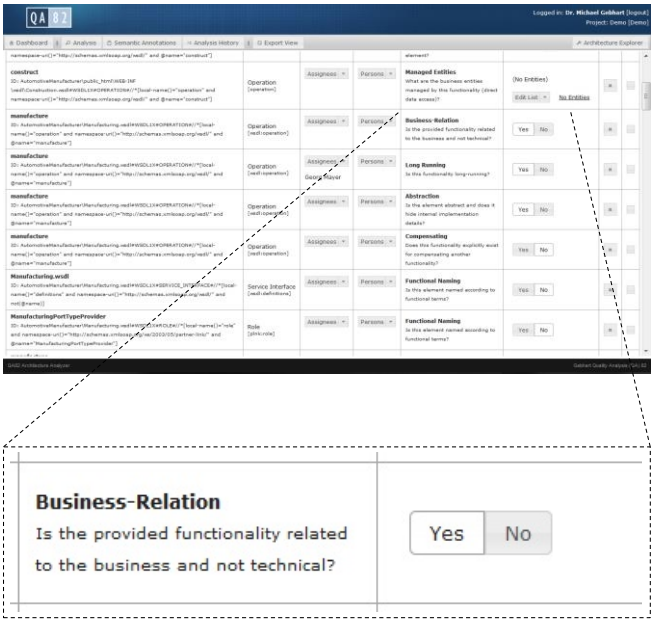


Figure 10. Questions in QA82 Analyzer to add expert knowledge.

The QA82 Analyzer or the QA82 SOA Compliance Center creates questions for every information that is not available by any information provider. These questions can be answered by experts and the answer is stored in an internal storage, the Custom Function Result Storage. The Custom Function Result Provider will request this information when it is necessary so that it can be included in the next analysis. In Fig. 10, the expert is asked if the operation "manufacture" is business-related or not. The expert can answer this question by selecting the button "Yes" or the button "No".

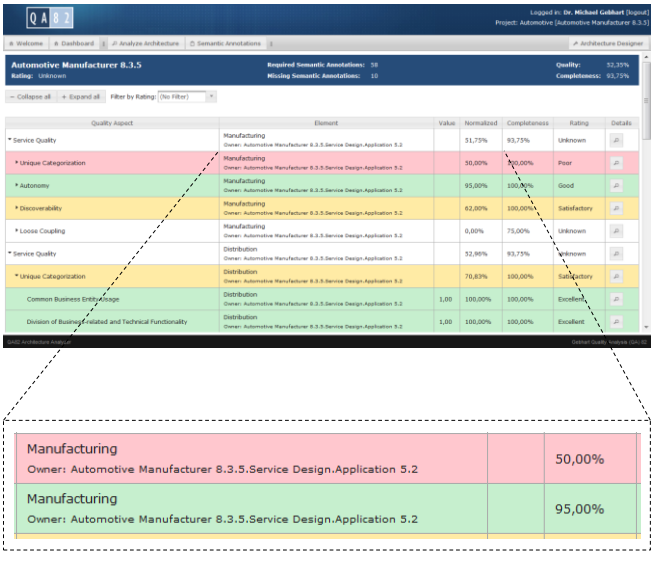


Figure 11. Analysis result in detail.

When an analysis has been performed, it is archived. The user can take a look at the analysis and the entire calculation trace. The analysis represents the prior defined quality model. Fig. 11 shows an analysis result in detail. According to Fig. 11, the Manufacturing service has a design quality of 50%. This means, that the best practices described by the quality model are only partially fulfilled. In our application, the user can select this entry to get more information about how this value has been measured and how it can be improved.

Finally, a quality dashboard shows the recent analysis results and some further information, such as the distribution of certain quality attributes and the number of open questions. Fig. 12 shows the quality of time that is displayed as part of the dashboard.

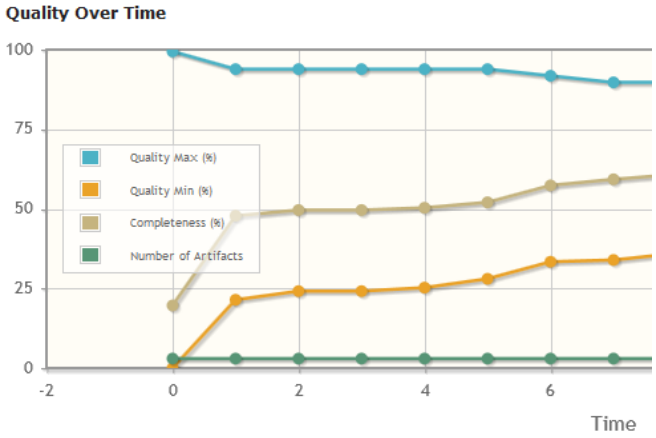


Figure 12. Quality results over time as part of the dashboard.

To sum up, the QA82 Analyzer or the QA82 SOA Compliance Center implement the concepts described in this article and enable their automation. Different views, such as the analysis details view and the dashboard help the user, i.e., the product and quality manager to calculate the quality and to ensure that the developed artifacts comply with wide-spread best practices.

Furthermore, our product is not only for managers or architects. Also developers can now directly take a look at the quality of their artifacts from a design perspective. I.e., they get an impression about how they have considered certain best practices and patterns. If the quality is not optimal, they can interact independently and without being informed by any responsible person. This makes them aware of quality aspects and increases the development speed.

G. Integration into Scenario

Back in our scenario, the quality manager can use the results of the QA82 SOA Compliance Center to inform developers about the design weaknesses. The usage of these metrics in a quality-oriented service design process is illustrated in [32]. Furthermore, as described before, developers can already independently get an insight into the quality of their artifacts.

For example, the result of DANF shows that the two provided service operations "Manufacture" and "getManufacturedAutomobiles" should be separated into two services. In addition, the result of the DS metric shows the conflict between the operations provided by the ManufacturedAutomobile service and the operation "getManufacturedAutomobile" of the Manufacturing service. Summarized, the operation "getManufacturedAutomobile" should be deleted as it provides functionality that is also offered by the ManufacturedAutomobile service. Service consumers using this operation should switch to the ManufacturedAutomobile Service. This and further information are given to all participating persons so that they can improve the artifacts with quality goals in mind.

In addition to the revision hints, the results of the metrics can be used to deliver reports to the management and the customer. For example the product and quality manager can justify cost and investments into quality assurances. Furthermore, the manager can prove the quality of the software by means of objective criteria.

V. Conclusion and Outlook

In this article, an approach was illustrated to measure the design quality of web services in service-oriented architectures regarding wide-spread best practices. For that purpose an existing quality model that refers to SoaML as formalization of a service-oriented architecture design was chosen. By use of another work that describes the mapping between SoaML and web service technologies, this quality model was transferred onto WSDL, XSD, and SCA. By this means the resulting quality model can be directly applied on service-oriented architectures based on web services. For an automation of the web service evaluation the quality-based static analysis approach was introduced. Compared to existing architecture evaluation approaches, the query-based static analysis approach does not derive an abstract architecture model but works directly on developed artifacts. Finally, a software product was shown that implements the approach and enables an automatic quality analysis of developed web service implementation artifacts regarding wide-spread best practices.

To demonstrate the approach a scenario from automotive manufacturing was introduced. In this scenario, a product and quality manager is responsible to ensure the quality of the resulting architecture. Next, the mapped quality model was applied to measure the design quality of services in this scenario. The metrics mapped onto web services enable the product and quality manager to identify weaknesses in the current design and thus give the developers hints about possible improvements. In addition, the results can be used to deliver reports to the management and the customer. Examples for reports are the current quality, the characteristic of certain quality attributes, such as the coupling or autonomy, the number of open questions, and the quality over time. The reports help to prove the high quality and to justify investments in additional quality assurance projects.

Furthermore, developers can perform quality analyses by their own. The metrics reduce the additional effort to interpret the textual descriptions. They directly refer to concrete elements within the used technologies.

As part of our research work, we have created a mapping for all metrics introduced in [9]. We also implemented this quality model as part of the QA82 Analyzer and QA82 SOA Compliance Center [33]. Through this, both product and quality managers and developers can automatically measure their web services regarding the quality model. This further increases the efficiency of the quality assurance process and makes the entire quality topic transparent. All participants are made aware of what quality means and how it can be influenced by developed artifacts. Furthermore, all participants can directly get an insight into the current design quality of developed artifacts.

For the future, we plan to include further quality characteristics both regarding service-oriented architectures and related fields. First, we plan to adapt the approach to analyze services based on the Representational State Transfer (REST) paradigm as it is often applied today. As REST does not prescribe certain interface formalization, we assume that the adaptation will require using more implementation-specific information, such as Java artifacts based on JAX-RS. Second, in collaboration with partners we work on a quality model in the context of business process management (BPM) that enables the determination of quality characteristics regarding the functional quality of modeled business processes based on the Business Process Model and Notation (BPMN) 2.0 [34]. This quality model is expected to be linked with the experiences we gained with the quality model introduced in this article. The results of this BPM quality model will be published as well. Furthermore, it will be supported by our quality analysis product. Finally, we aim to formalize the described metrics in a technology-independent but executable way. With languages, such as OCL [35] or XQuery [36] it is possible to describe queries that refer to a certain technology, such as UML or XML. We will examine the applicability of these languages for our purposes.

REFERENCES

[1] M. Gebhart, “Measuring design quality of service-oriented architectures based on web services,” Eighth International Conference on Software Engineering Advances (ICSEA 2013), Venice, Italy, October 2013, pp. 504-509.

[2] T. Erl, Service-Oriented Architecture – Concepts, Technology, and Design, Pearson Education, 2006. ISBN 0-13-185858-0.

[3] D. Krafzig, K. Banke, and D. Slama, Enterprise SOA – Service-Oriented Architecture Best Practices, 2005. ISBN 0-13-146575-9.

[4] T. Erl, SOA – Principles of Service Design, Prentice Hall, 2008. ISBN 978-0-13-234482-1.

[5] T. Erl, Web Service Contract Design & Versioning for SOA, Prentice Hall, 2008. ISBN 978-0-13-613517-3.

[6] W3C, “Web Services Description Language (WSDL)”, Version 1.1, 2001.

[7] W3C, “XML Schema Part 0: Primer Second Edition”, 2004.

[8] Open SOA (OSOA), “Service component architecture (SCA), sca assembly model V1.00,” http://osoa.org/download/attachments/35/SCA_AssemblyModel_V100.pdf, 2009. [accessed: January 04, 2011]

[9] M. Gebhart and S. Abeck, “Metrics for evaluating service designs based on soaml,” International Journal on Advances in Software, 4(1&2), 2011, pp. 61-75.

[10] W. van den Heuvel, O. Zimmermann, F. Leymann, P. Lago, I. Schieferdecker, U. Zdun, and P. Avgeriou, „Software Service Engineering: Tenets and Challenges,” 2009.

[11] T. Erl, SOA – Design Patterns, Prentice Hall, 2008. ISBN 978-0-13-613516-6.

[12] S. Cohen, “Ontology and Taxonomy of Services in a Service-Oriented Architecture,” Microsoft Architecture Journal, 2007.

[13] N. Josuttis, SOA in Practice, O'Reilly Media, 2007. ISBN 978-0-59-652955-0.

[14] M. Perepletchikov, C. Ryan, K. Frampton, and H. Schmidt, “Formalising service-oriented design,” Journal of Software, Volume 3, February 2008.

[15] M. Perepletchikov, C. Ryan, K. Frampton, and Z. Tari, “Coupling metrics for predicting maintainability in service-Oriented design,” Australian Software Engineering Conference (ASWEC 2007), 2007.

[16] M. Hirzalla, J. Cleland-Huang, and A. Arsanjani, “A metrics suite for evaluating flexibility and complexity in service oriented architecture,” ICSOC 2008, 2008.

[17] S. W. Choi and S. D. Kimi, “A quality model for evaluating reusability of services in soa,” 10th IEEE Conference on E-Commerce Technology and the Fifth Conference on Enterprise Computing, E-Commerce and E-Services, 2008.

[18] OMG, “Service oriented architecture modeling language (SoaML) – specification for the uml profile and metamodel for services (UPMS)”, Version 1.1, 2012.

[19] OMG, “Unified modeling language (UML), superstructure,” Version 2.2, 2009.

[20] S. Johnston, “UML 2.0 profile for software services,” IBM Developer Works, http://www.ibm.com/developerworks/rational/library/05/419_soa/, 2005. [accessed: July 11, 2012]

[21] U. Wahli, L. Ackerman, A. Di Bari, G. Hodgkinson, A. Kesterton, L. Olson, and B. Portier, “Building soa solutions using the rational sdp”, IBM Redbook, 2007.

[22] A. Arsanjani, “Service-oriented modeling and architecture – how to identify, specify, and realize services for your soa,” IBM Developer Works, http://www.ibm.com/developerworks/library/ws-soa-design1, 2004. [accessed: July 11, 2012]

[23] J. Amsden, “Modeling with soaml, the service-oriented architecture modeling language – part 1 – service identification,” IBM Developer Works, http://www.ibm.com/developerworks/rational/library/09/modelingwithsoaml-1/index.html, 2010. [accessed: July 11, 2012]

[24] M. Gebhart, “Service Identification and Specification with SoaML,” in Migrating Legacy Applications: Challenges in Service Oriented Architecture and Cloud Computing Environments, Vol. I, A. D. Ionita, M. Litoiu, and G. Lewis, Eds. 2012. IGI Global. ISBN 978-1-46662488-7.

[25] M. Gebhart and S. Sejdovic, “Quality-oriented design of software services in geographical information systems,” International Journal on Advances in Software, 5(3&4), 2012, pp. 293-307.

[26] M. Gebhart and J. Bouras, “Mapping between service designs based on soaml and web service implementation artifacts,” Seventh International Conference on Software Engineering Advances (ICSEA 2012), Lisbon, Portugal, November 2012, pp. 260-266.

[27] G. C. Murphy, D. Notkin, and K. J. Sullivan, “Software Reflexion Models: Bridging the Gap between Design and Implementation,” IEEE Trans. Softw. Eng., vol. 27, 2001.

[28] S. Giesecke, M. Gottschalk, and W. Hasselbring, “The ArchMapper Approach to Architectural Conformance Checks: An Eclipse-based Tool for Style-oriented Architecture to Code Mappings,” 2012, pp. 71–80..

[29] J. A. McCall, P. K. Richards, and G. F. Walters, “Factors in software quality,” 1977.

[30] M. Gebhart, M. Baumgartner, S. Oehlert, M. Blersch, and S. Abeck, "Evaluation of service designs based on soaml," Fifth International Conference on Software Engineering Advances (ICSEA 2010), Nice, France, August 2010, pp. 7-13.

[31] M. Gebhart, S. Sejdovic, and S. Abeck, "Case study for a quality-oriented service design process," Sixth International Conference on Software Engineering Advances (ICSEA 2011), Barcelona, Spain, October 2011, pp. 92-97.

[32] M. Gebhart and S. Abeck, "Quality-oriented design of services," International Journal on Advances in Software, 4(1&2), 2011, pp. 144-157.

[33] Gebhart Quality Analysis (QA) 82, QA82 Architecture Analyzer, http://www.qa82.de. [accessed: July 11, 2012]

[34] OMG, "Business process model and notation (BPMN)", Version 2.0 Beta 1, 2009.

[35] Object Management Group, "Object constraint language", Version 2.0, 2006.

[36] W3C, "XQuery 1.0: an XML query language (second edition)", Version 1.0, 2010.

Multicast Source Mobility Support for Regenerative Satellite Networks

Esua Kinyuy Jaff, Prashant Pillai, and Yim Fun Hu
School of Engineering & Informatics,
University of Bradford, Bradford, United Kingdom
ekjaff@student.bradford.ac.uk, p.pillai@bradford.ac.uk, y.f.hu@bradford.ac.uk

Abstract — **Satellite communications provides an effective solution to the ever increasing demand for mobile and ubiquitous communications especially in areas where terrestrial communication infrastructure is not present. IP multicasting is a bandwidth saving technology, which could become an indispensable means of group communication over satellites since it can utilise the scarce and expensive satellite resources in an efficient way. In Source-Specific Multicast (SSM) the data is sent through a multicast tree from the source to all the receivers. However, if a source is a mobile node moving from one network to another, then special mechanisms are required to make sure this multicast tree does not break. Until now, while many research efforts have been made to provide IP multicast for the mobile nodes, they are mainly focused on terrestrial networks. Unfortunately, the terrestrial mobile multicast schemes are not directly applicable in a satellite environment. This paper proposes a new mechanism to support multicast source mobility in SSM based applications for a mesh multi-beam satellite network with receivers both within the satellite network and in the Internet. In the proposed mechanism, the SSM receivers continue to receive multicast traffic from the mobile source despite the fact that the IP address of the source keeps on changing as it changes its point of attachment from one satellite gateway (GW) to another. The proposed scheme is evaluated and the results compared with the mobile IP home subscription (MIP HS)-based approach. The results show that the proposed scheme outperforms the MIP HS-based approach in terms of signaling cost and packet delivery cost.**

Keywords – Gateway Handover, Mobile Multicast Source, Multi-beam, Regenerative Satellite, Signaling Cost.

I. INTRODUCTION

Traditionally, satellites have been usually treated as a transparent pipe that carries data between a GW and the receivers. Nowadays, the new generation of satellite systems are characterised by support for on-board processing (switching/routing) and multiple spot beams. Regenerative satellites with on-board packet processing can provide full-mesh, single-hop connectivity between two or more satellite terminals/gateways. Multiple spot beams in regenerative satellites further enhance the overall satellite capacity with the help of frequency reuse within different narrow spot beams. These new features enable the satellite to make efficient use of its allocated resources and provide cost effective network services. This paper is an extension of [1] presented at IARIA conference, MOBILITY 2013.

IP multicasting is a technology in which a single copy of IP data is sent to a group of interested recipients. It minimises overheads at the sender and bandwidth use within the network. This explains why IP multicast is considered as an important mechanism for satellite networks, which can have the potential to reach many customers over large geographical areas.

In IP multicasting, there may be many sources sending data to a single multicast group for example: group voice chat. In SSM, a group member of such a multicast group, G may request to receive traffic only from one specific source, S. Unlike in any source multicast (*, G) [2], where a group member might receive unwanted traffic from some sources, in SSM, a group member subscribes to specific multicast channels (S, G) [2] of interest. This implies SSM saves more bandwidth resources than any source multicast. In satellite networks, where bandwidth resources are scarce and expensive, this could be a very significant and compelling factor for SSM. IP mobile multicast over satellites can be used to communicate important service information like the weather conditions, on-going disaster zones and information, route updates, etc., in long haul flights, global maritime vessels and continental trains. Multicasting this information to all the interested parties rather than individually informing them (i.e., unicast) would save a lot of satellite bandwidth resources.

In SSM, a multicast distribution tree is setup with the source at the root and receivers as the end leaf node and the routers forming the intermediate nodes in the tree. The data is then sent from the root with the routers in the network replicating the data only when necessary for delivery until a copy reaches all intended downstream group members. Various issues arise if the receivers or source of the multicast group are mobile and move from one network to another as this may affect this multicast distribution tree. Handover of a mobile multicast receiver from one point of attachment to another has a local and single impact on that particular receiver only. However, the handover of a mobile source may affect the entire multicast group, thereby making it a critical issue.

A mobile multicast source faces two main problems; transparency and reverse path forwarding (RPF). In SSM, a receiver subscribes to a multicast channel (S, G) [2]. During a handover, as the source moves from one network to another, its IP address will change. When the source uses this new IP address, i.e., care-of address (CoA) [3] as source

address to send traffic, the multicast router in the foreign network cannot forward the multicast packets until a receiver explicitly subscribes to this new channel (CoA, G). This is known as the transparency problem.

A multicast source-specific tree is associated to source location, i.e., the source is always at the root of the source-specific tree. The RPF check compares the packet's source address against the interface upon which the packet is received. During handover, the location of the source will change (and consequently its IP address), thus invalidating the source-specific tree due to the RPF check test. Hence, the RPF problem relates to the fact that the mobile source cannot use its home address in the foreign network as the source address to send packets as this will result in a failure of the RPF mechanism and the ingress filtering [4].

This paper is based on the Digital Video Broadcasting Return Channel Satellite (DVB-RCS/RCS2) system, which is an open standard that defines the complete air interface specification for two-way satellite broadband scheme. DVB-RCS/RCS2 is today the only multi-vendor VSAT standard [5]. The return link in DVB-RCS/RCS2 is based on a multiple-frequency time-division multiple-access (MF-TDMA) scheme, where the return channel satellite terminals (RCSTs) are allocated capacity in slots within a certain time and frequency frame. Due to the vendor independence and popularity of the DVB-RCS/RCS2 standard, customers with DVB-RCS/RCS2 compliant equipment have a wide variety of satellite operators and service providers to choose from. This flexibility lowers the equipment and operational costs [6].

The organisation of this paper is as follows. In Section II, the literature review on existing SSM techniques and their applicability to satellite networks are given. Section III presents the new network architecture and the further extended Multicast Mobility Management Unit (M3U) proposed in [1] for source mobility support. Detailed description of the operation and processing proposed mechanism has been provided. New analytical models have been proposed in Section IV, for calculating the signaling cost and packet delivery cost in order to evaluate the performance of the proposed scheme. Section V presents the analysis of the obtained results. Finally, the conclusions are presented in Section VI.

II. PREVIOUS STUDIES ON SSM

A few mobile multicast source support techniques for SSM have been proposed for terrestrial Internet. These are far from being applicable in a satellite scenario. Due to the problems of transparency and RPF, remote subscription [3] –based approaches cannot be applied to mobile multicast sources for SSM. On the other hand, MIP HS-based approach [3] (which relies on mobile IP in terrestrial networks) can support both mobile receivers and sources (including SSM senders) by the use of bi-directional tunnelling through home agent (HA) without facing the problems of transparency and RPF.

Following the MIP HS mechanism, bi-directional tunnelling between the mobile source under target GW and its home GW (serving as HA) [7] could be used to tunnel multicast traffic for delivery onto the source-specific tree. This is used to maintain the mobile source identity. If this MIP HS-based approach is used in mesh satellite networks, the mesh communication concept, i.e., a single hop over the satellite will be lost and there would be some RPF issues when the home GW tries to deliver the traffic onto the source-specific tree over the satellite. Mesh SSM communication, where the receivers and mobile source are all RCSTs of the same interactive satellite network, will no longer be possible since the mobile source has to tunnel traffic from its foreign location to its home GW to be delivered on to the source-specific tree.

The authors in [8] used the shared tree approach proposed Mobility-aware Rendezvous Points (MRPs), which replace the home agents in their role as mobility anchors. It is proposed in this approach that the MRP builds a Multicast Registration Cache (MRC) for mobile multicast sources. This cache is used to map the permanent home address (HoA) of the mobile source with its temporary CoA. Based on the MRC information, a new Multicast Forwarding Table (MFT) format is also proposed, in which each multicast source will be referenced by the two addresses (HoA and CoA) instead of a unique IP address. This solution introduces a new registration method for IP mobile multicast source. The mobile source registers only once with the MRP by sending a Source Registration (SR) message. To send multicast data, the mobile multicast source encapsulates its data packets, and then sends them to the MRP. Before forwarding the encapsulated packets, the MRP checks first whether the multicast packets are coming from a registered and trusted mobile multicast source or not. If so, it decapsulates these packets, and then sends them using the (HoA, G) header to the multicast receivers. When the mobile source moves to a new IP subnet within the MRP service area, the source's MRP is implicitly notified about the CoA change. In case of inter-domain multicasting, if the source moves to a new domain, it has to register again with the local MRP in that domain. The new MRP notifies remote MRPs about the source address change. There is at least one MRP per domain. The MRPs rely on triangular routing and tunnelling to fulfil their role as mobility anchors during intra-domain and inter-domain trees setup. This approach also re-introduces rendezvous points, which are not native to SSM routing. The introduction of new entities/messages for example, the MRP, new registration message (of mobile sources to MRPs whenever they move into a new domain), MRP Peer-to-peer Source Active (SA) [8] and keep-alive messages (required to track the source's MRP attachment point changes) during inter-domain multicasting, coupled with the modification of the standard Multicast Forwarding Table (referenced by the two addresses, HoA and CoA instead of a unique IP address) make this approach very complicated and not suitable for

satellite networks. Also, large number of signaling messages proposed in this mechanism is not good for satellite networks as they consume the scarce and expensive satellite bandwidth.

Authors in [9] and [10] introduced Tree Morphing and Enhanced Tree Morphing (ETM), respectively, which are routing protocol adaptive to SSM source mobility. The concept of the source tree extension or elongation as the source moves from the previous designated multicast router (pDR) to new designated router (nDR) is not applicable in satellite scenario because the delivery tree rooted at the source in one GW cannot be extended to that same source when it moves to a different GW. This makes the fundamental design concept of these extensions not consistent with the nature of satellite networks.

SSM source handover notification approach proposed by authors in [11] suggested adding a new sub-option in the standard IPv6 destination binding option known as SSM source handover notification. During handover, the source after acquiring new IP address will notify receivers to subscribe to the new channel. The problems here are the large amount of signaling traffic over satellite air interface and the fact that some receivers may be unsynchronized to source handovers, leading to severe packet loss.

In [12], the authors proposed multicast source mobility support in proxy mobile IPv6 (PMIPv6) domains for terrestrial networks. Based on the specifications in [13], multicast data arriving from a downstream interface of an Multicast Listener Discovery (MLD) [12] proxy will be forwarded to the upstream interface and to all but the incoming downstream interfaces that have appropriate forwarding states for this group. Thus, multicast streams originating from an mobile node (MN) will arrive at the corresponding Local Mobility Anchor (LMA) [14] and directly at all mobile receivers co-located at the same Mobile Access Gateway (MAG) and MLD Proxy instance. Serving as the designated multicast router or an additional MLD proxy, the LMA forwards the multicast data to the Internet, whenever forwarding states are maintained by multicast routing. If the LMA is acting as another MLD proxy, it will forward the multicast data to its upstream interface, and to downstream interfaces with matching subscriptions, accordingly. One of the drawbacks here is that there are no mechanisms to supress upstream forwarding to LMA even when there are no receivers. This waste of network resources could pose a serious problem in a satellite environment. Triangular routing is also an issue here when a mobile receiver and a source, all having different LMAs are attached to the same MAG. In such a situation, the MAG has to forward traffic upstream to the corresponding LMA of the mobile source, which will tunnel the traffic to the corresponding LMA of the mobile receiver which then tunnels the traffic back to the same MAG for delivery to mobile receiver, causing waste of network resources in the whole domain. The fact that in proxy mobile IPv6 domain, the LMA is the topological anchor point for the addresses assigned to mobile nodes within the domain (i.e., packets with those addresses as destination are routed to the LMA), the role of the LMA and MAG does not fit well into a global interactive multi-beam satellite network with many Transparent/Regenerative Satellite Gateways [15], each having different IP addressing space.

The authors in [1] proposed a solution consistent with the DVB-RCS/RCS2 satellite network specifications that supports SSM source mobility within the satellite network. The idea of reserving IP addresses for the mobile Return Channel Satellite Terminals (mRCSTs) in all foreign networks (i.e., under all potential target gateways) is not efficient in utilisation of the allocated IP address space. This will lead to scalability issues especially with increasing number of satellite terminals requiring IP addresses, as the IP address space is limited. This paper is an extension of our previous proposal in [1] with the following modifications:

- No IP addresses are reserved for mobile sources (mRCSTs) in foreign networks.
- The satellite is a regenerative one with on-board processing (OBP) at layer 2 of the protocol stack, capable of replicating multicast traffic on-board the satellite. This further saves the satellite bandwidth resources as only one copy of the multicast traffic will be sent to the satellite air interface for all beams with interested receivers instead of one for each beam as was proposed in [1].
- The functioning, location and the type of messages issued by the Multicast Mobility Management Unit (M3U) proposed here are quite different from those proposed in [1]. These changes further help in providing an effective support for source mobility.

III. Proposed Multicast Source Mobility Mechanism for SSM in Regenerative Satellite Network

The satellite terminals like the regenerative satellite gateways (RSGW), RCSTs and mobile RCSTs are assumed to be IP nodes with layer 3 capability. In this satellite network, the routing function is organised as a 'decentralized router'. In a client/server [16] like architecture, part of the routing functions are located in the RCSTs/RSGWs/mRCSTs (clients) and the other part of them within the Network Control Centre (NCC), i.e., routing server. Each time a client needs to route an IP packet, it asks the server for the information required to route this packet. The routing information sent by the server (NCC) is then saved in the client.

Each time an IP packet comes into the satellite system, the ingress RCST/RSGW determines where to send the packet, the final target being to get the destination RCST's MAC address. The ingress RCST/RSGW look within its routing table to find if the route on the satellite path exist. If it does not exist, it issues an ARP towards the NCC, through the connection control protocol (C2P) [15] connection request message [16]. Since the connection and switching

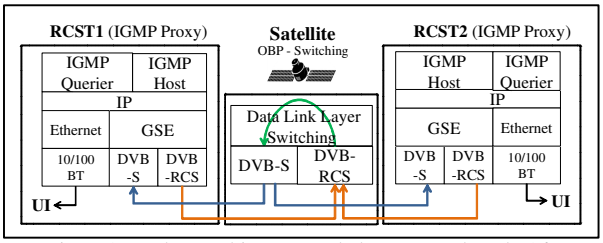


Figure 1. Mesh IP multicast control plane protocol stack [16]

on-board the satellite is defined here at layer 2 of the protocol stack, the knowledge of MAC addresses of the RCSTs is mandatory to establish a connection. This means that the NCC provides the mechanisms required to associate the IP address and MAC address of a RCST/RSGW [16].

In the control plane, Internet Group Management Protocol (IGMP) messages are exchanged between the NCC and the IGMP Proxy contained in the RCSTs. Also, the IGMP messages are exchanged between User Terminals and IGMP Querier included in the RCSTs as shown in Figure 1. IGMPv2 general Query, Specific Group Query, Report and Leave messages are exchanged over the satellite air interface between the NCC and the RCSTs/RSGWs/mRCSTs.

As illustrated in Figure 1, all satellite entities transmit using DVB-RCS and receive using DVB by Satellite (DVB-S). The two existing satellite transmission standards, DVB-RCS and DVB-S are combined by the OBP into a single regenerative multi-spot satellite system allowing a full cross-connectivity between the different up link and down link beams.

A. Network Architecture

Figure 2 shows the network architecture, where a mobile multicast source is located at its home network in beam 1 and the receivers are in beams 1, 2, 3 and 6. GW_A1, GW_A2, GW_A3, GW_A4, GW_A5 and GW_A6 serves beams 1, 2, 3, 4, 5 and 6, respectively. The multicast receivers in the terrestrial network as shown in Figure 2 are served through GW_A1. The mobile source sends out just one copy of multicast traffic and the OBP replicates the traffic, one for each of the four beams that has interested receivers. GW handover (GWH), which involves higher layers (i.e., network layer), will take place at the overlapping areas between beams. IP multicast source mobility support is therefore implemented at GWH.

B. Source Mobility Support with Multicast Mobility Management Unit (M3U) at Gateway Handover.

In order to develop an effective solution to support source mobility, the following general assumptions have been made:

- The regenerative satellite has OBP – switching at layer 2 to provide on-board connectivity between different beams.
- The NCC will act as the IGMP querier for the satellite network in addition to its normal functionalities.
- The NCC enables the establishment of point-to-multipoint connection between mobile source (mRCST) and all listening RCSTs/RSGWs.
- All RCSTs function as IGMP Proxy, i.e., IGMP Router and Querier on its user interface (interface towards the internal LAN) and an IGMP Host on the satellite interface.
- All RCSTs, mRCSTs and RSGWs are mobility-aware nodes and can process mobility instructions.

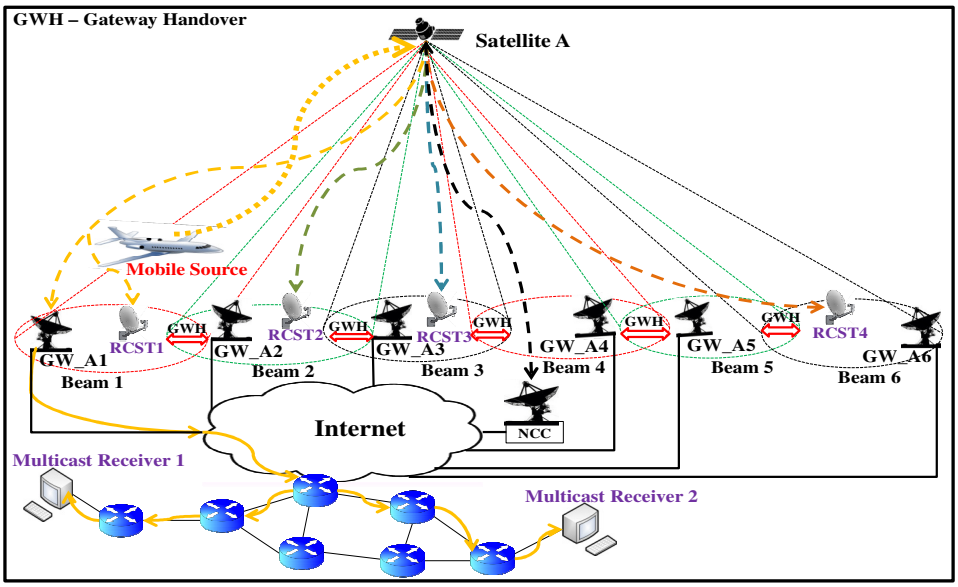


Figure 2. Mobile Source at Home Network (GW_A1)

TABLE I. PROPOSED NEW MESSAGES

Message Name	Type	Source	Destination	Content	Purpose
Service Interface Update Message (SIUM)	Multicast	NCC	All SSM RCSTs/GWs Receivers + mobile source	IP addresses of mobile source in both old and target GWs. Instructions to update source list (add mobile source new IP address) in service interface of specified channel.	To avoid each listening RCST/GW from sending IGMP Join Report on to the satellite air interface after receiving channel re-subscription from terrestrial SSM receivers.
Source Handover Message (SHM)	Internal Signaling	M3U	NCCu	A Request to establish point-to-multipoint link btw source & all listening RCSTs/GWs (from previous tree)	To establish new delivery tree to all listening RCSTs/GWs without them sending any IGMP join report to new channel (CoA, G). To reduce tree establishment time.
Channel Update Message (CUM)	Multicast	Mobile source	All SSM Receivers	IP addresses of mobile source in both old and target GWs. Instructions to receivers to update channel subscription to new mobile source IP address	For all SSM end receivers to update their channel subscription from (S, G) to CoA, G) For Internet receivers to start building the new delivery tree to the target GW.

A new Multicast Mobility Management Unit (M3U) responsible for control plane signaling to provide mobility support for multicast sources is proposed. This new M3U entity located at the NCC is equipped with the following:

- A database of all mRCSTs, each identified by its physical (MAC) and IP addresses
- A 'Message Chamber' which can issue the new proposed signaling messages.

Three new types of messages shown on Table I have been proposed in this paper. It is proposed that any mRCST should be able to issue Channel Update Message (CUM) after receiving Service Interface Update Message (SIUM) from the NCC during GWH. Details of these messages are given in Table I.

When the NCC receives the synchronization (SYNC) burst from the mobile source (mRCST) containing the handover request, it will retrieve the target beam identity from its database and determine whether the beam belongs to a different GW. Once the NCC establishes that the target beam belongs to a different GW, a GWH is initiated. The NCC will then update its service information (SI) tables which include Terminal Burst Time Plan (TBTP), Super-frame Composition Table (SCT), Frame Composition Table (FCT) and Time-slot Composition Table (TCT). The NCC will send an SNMP Set-Request message that includes the updated SI tables and the routing update information (RUI) of the mRCST to the target GW to ensure that the target GW gets ready for connection with the mRCST. Upon reception of the SNMP Set-Request message, the target GW will allocate bandwidth resources and IP address to the mRCST according to the new burst time plan sent by the NCC. The SNMP Get-Response message is then sent by target GW to the NCC.

Once the NCC receives the SNMP Get-Response message from target GW, the M3U immediately issues the Source Handover Message (SHM) to the NCC unit (NCCu), requesting the point-to-multipoint link between the source and all the listening RCSTs/GWs (from previous tree). SHM is internal signaling within the NCC (i.e., between M3U and NCCu). Upon reception of SHM, the NCCu will make the resources available and then instructs the OBP to establish the required connections. This is immediately followed by the M3U issuing the SIUM to all RCSTs/GWS involved in this particular channel, including the mobile source. The SIUM contains both the mobile source old and new IP addresses in the old and new GWs, respectively. The SIUM also contains instructions for all listening RCSTs/GWs to update source list (add mobile source new IP address) in the service interface for requesting IP multicast reception [17]. This will create a new channel that contains the mobile source new IP address (CoA) under the target GW. This action ensures that subsequently, when the RCSTs/GWs receive IGMP join Report from downstream receivers for this new channel, no IGMP report will be sent to the satellite air interface since the channel already exist in the RCST/GW multicast routing table. The creation of this new channel by the SIUM is possible in satellite networks because the NCC knows:

- The mac and IP address of all active RCSTs/GWs,
- The newly acquired IP address of the mobile source,
- All RCSTs/GWs that are members of the channel involving the mobile source.

Therefore, the NCC can enable the establishment of a point-to-multipoint connection between the mobile source and all the listening RCSTs/GWs directly. This reduces the amount of traffic on the satellite air interface, thus saving scarce and expensive satellite bandwidth resources. The PID of the channel may remain the same. Upon reception of SIUM, the mobile source immediately issues CUM, i.e., CUM is triggered by reception of SIUM. The CUM is sent just like any multicast user traffic by the mobile source through source-specific tree to all SSM receivers.

After 1 round trip delay of issuing SIUM (for mobile source to receive SIUM and issue CUM), the NCC issues SNMP Set-Request message, which includes the mRCST (mobile source) identity and the SI tables to the source GW. The source GW then acknowledges the NCC by sending a SNMP Get-Response message. Once the SNMP Get-Response message is received from source GW, a GWH command is issued to the mRCST from NCC in a Mobility

Control Descriptor carried in a Terminal Information Message Unicast (TIMu) using the old beam. The source GW now updates its route mapping table and released resources used by the mRCST. Once the mRCST receives the handover command, it synchronizes with the NCC and the target GW, retunes itself to the target beam.

Figure 3 shows the proposed signaling sequence to support multicast source mobility for SSM at GWH. This signaling sequence contains the proposed new messages integrated into the standard GWH signaling sequence as described in the DVB-RCS specification in [7]. The NCC acting as satellite IGMP querier keeps control of the multicast groups and also builds the SSM tree based on the on-board connectivity between different beams. Periodically, the NCC sends out the Multicast Map Table (MMT) [16] to all multicast receivers. The MMT which contains the list of IP multicast addresses each associated with a specific Program Identifier (PID) enables listening RCSTs/GWs to receive multicast traffic from groups which they are members of. When the NCC receives an IGMP join report for SSM, the M3U checks the source-list to see if some sources are mRCSTs. If some sources are identified as mRCSTs, the M3U will a keep record of them in its database.

Upon reception of CUM by SSM receivers in the Internet, a new SSM delivery tree construction to the target GW is triggered as shown in Figure 4 (compared to that in Figure 2). Figure 4 shows the mobile source now in beam 2 after a successful GWH. If the Target GW was not a member of the old multicast channel, it will issue an IGMP join report to NCC as soon as it gets the updated channel subscription request (PIM-SSM Join) from receivers in the Internet. The target GW now becomes part of the mesh receivers within the satellite network as it assumes the responsibility of serving receivers in the Internet. But if the target GW was already a member, a multicast reception state will simply be created against the interface upon which the PIM-SSM join was received. It should be noted here that CUM is delivered through serving GW to the SSM receivers in the Internet before the resources used by the mobile source in the serving GW are released and also before retuning and switching by the mobile source to the target GW begins. This is so, because it is only through the serving GW (old SSM delivery tree) that CUM can reach all the SSM receivers in the Internet.

When the SSM receivers in the LAN behind the listening RCSTs receive the CUM, they will update their channel subscription by issuing unsolicited IGMP join report towards the RCST. Upon reception of the IGMP join report, the RCST (IGMP Proxy) will check its multicast routing table to see whether that channel already exist. On checking, the RCST will discover the existence of the channel in its multicast routing table thanks to the action of SIUM as described above. Therefore, this will prevent the RCST from issuing IGMP join Report onto the satellite air interface, thus saving satellite bandwidth resources.

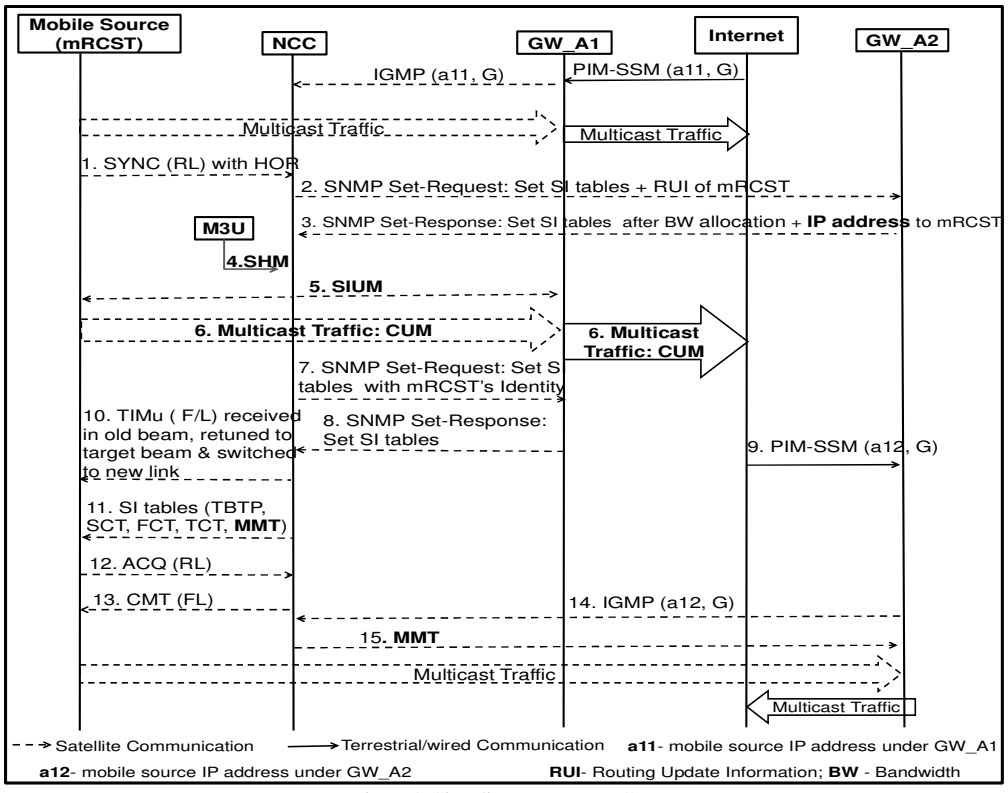


Figure 3. Signaling sequence at GWH

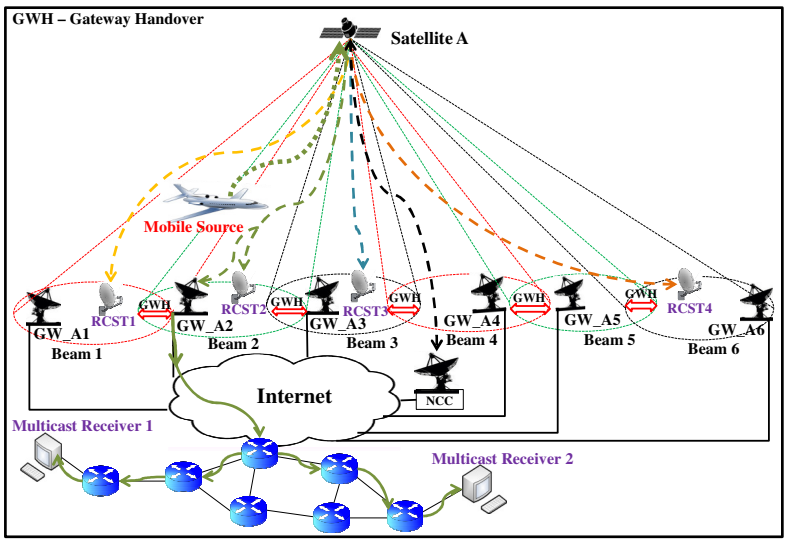


Figure 4. Mobile source at foreign network (GW_A2)

C. *M3U operation and processing*

Figure 5 shows the processing flowchart of the control plane information (signaling traffic) through the M3U. For correct signaling to take place, M3U must be able to identify the following:

- An IGMP packet (i.e., an unsolicited IGMP join report) in order to add the requesting RCST/GW on the delivery tree.
- Mobile multicast source or receiver (mRCST) and differentiate between the two.
- GWH request and target GW.
- Target GW signaling (SNMP) to get the mRCST newly allocated IP address.

1) *IGMP Packet Identification*

When the NCC receives any signaling traffic, the M3U checks the IP destination address and the protocol number on the IP packet to determine whether it is an IGMP packet. If the IP destination address is equal to 224.0.0.1 (for IGMPv1&2) or 224.0.0.22 (for IGMPv3) and the protocol number is equal to 2, then the IP packet is an IGMP packet and is then it is sent to Stage 2 in Figure 5, otherwise, it is sent to Stage 4.

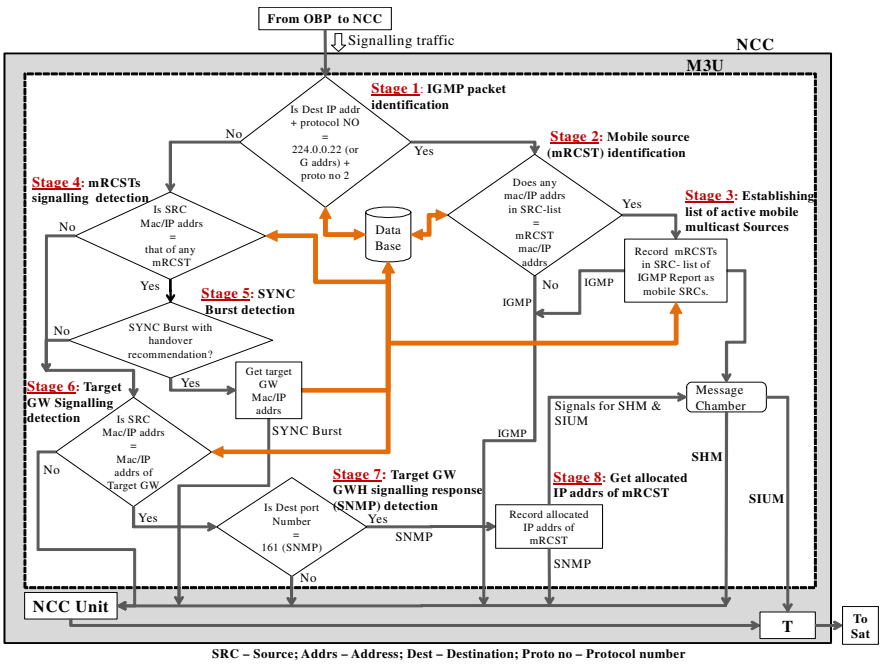


Figure 5. M3U source mobility support processing for SSM during GWH

2) mRCST Identification

In Stage 2 of Figure 5, the task is to determine whether the source-list in the received IGMP packet contains any mobile source (mRCST). The M3U checks the IP addresses contained in the source-list against the list of mRCSTs in the database to find out whether the requesting RCST/GW is requesting to receive multicast traffic from a mobile source (mRCST) or not. If source-list contains any mRCSTs, then those mRCSTs are mobile multicast sources. The mRCSTs contained in source-list of received IGMPv3 join report are then recorded in Stage 3 as mobile sources based on the analysis in Stage 2 given above. Finally, the IGMP packet is then forwarded to the NCC (querier).

3) mRCST Signaling Detection

At Stage 4, the main task is to separate signaling traffic coming from any mRCST from those of fixed RCST. To do this, the M3U has to check the source mac/IP address of the signaling traffic received against the database to establish whether it is coming from a mRCST or not. All signaling traffic coming from any mRCST is sent to Stage 5 for close examination to find out whether they are synchronisation (SYNC) burst containing handover recommendation while the rest is sent to Stage 6. Once it is confirmed that it is a SYNC burst in Stage 5, with handover recommendation, then the target GW identity is known and its MAC/IP address recorded. Following this process, a table of mRCST versus target GW (identified by their MAC/IP addresses) can be established for all mRCSTs in the whole interactive satellite network. This now prepares the M3U to expect GWH signaling response from the target GW.

4) Target GW Response Detection and the mRCST allocated IP address recording

Now, knowing the identity of the target GW (from the handover recommendation), signaling traffic from the target GW can be tracked within the NCC to find out whether it is the response to the GWH request initiated by the NCC. This is very important because earlier knowledge of the allocated IP address to the mRCST by the target GW contained in this GWH response is very crucial here for further signaling.

Therefore, Stage 6 examines the source MAC/IP address of all signaling traffic to see whether it is that of the target GW. If it does, then the packet is sent to Stage 7, if not, then to NCCu. In Stage 7, the destination port number of the packet is checked to find out whether it is equal to that of SNMP (i.e., 161), the signaling protocol used in GWH as specified in [7]. If this is so, then, the packet is sent to Stage 8, where the allocated IP address to the mRCST in the target beam is extracted and recorded. Once the M3U is aware of the mRCST's IP address in the target beam, it immediately issues the SHM to the NCCu, requesting for a point-to-multipoint connection establishment as explained above. It is therefore imperative that the M3U gets the mRCST's IP address in the target beam as soon as possible in order to minimise the multicast handover latency during GWH. If the destination port is not equal to 161, then, the packet is simply sent to the NCCu for normal signaling. The issuing of SHM is immediately followed by that of SIUM to all mesh SSM receivers including the mobile source as explained above.

The uniqueness about this proposal are: the new re-subscription mechanism of the satellite receivers and gateways to the new multicast channel (CoA, G) after every GW handover without the issuing of IGMP join report over the satellite air interface, the absence of encapsulation (tunnelling) and triangular routing paths throughout the system and its compliance with DVB-RCS/S2 specifications. If all the listening RCSTs/GWs were to individually issue IGMP join reports to the satellite air interface for re-subscription after GWH, the total number would be enormous and will put a lot of strain on the satellite bandwidth resources. The proposed solution will significantly save satellite bandwidth resources.

IV. ANALYTICAL MODEL

Under this section, analytical models for GWH signaling cost and packet delivery cost (when mobile source is away from home) are developed to evaluate the proposed mobile multicast source GWH procedure for SSM. These are then compared with MIP HS–based approach (see Section II), which appears to require only minimal changes to support multicast source mobility for SSM in a satellite environment. The other schemes for multicast source mobility SSM which are mainly defined for terrestrial networks will require major changes to be applicable in satellite networks. This explains why the performance evaluation of the proposed M3U scheme is compared only with that of the MIP HS-based approach.

TABLE II. MESSAGE SIZE AND NUMBER OF HOPS

Notation	Description	Value
M_{SYNC}	SYNC message size	12 bytes
M_{SNMP}	SNMP Request/Response + SI tables message sizes + RUI + allocated BW and IP address	636 bytes
M_{TIM}	Terminal Information Message size	35 bytes
M_{SI_t}	SI tables (TBTP, SCT, FCT, TCT, MMT) message size	152 bytes
M_{ACQ}	Acquisition Burst message size	12 bytes
M_{CMT}	Correction Message Table size	30 bytes
M_{MMT}	Multicast Map Table message size	30 bytes
M_{SIUM}	Service interface update message size	50 bytes
M_{SHM}	Source handover message size	30 bytes
M_{CUM}	Channel update message size	50 bytes
M_{PIM_SM}	PIM-SM message size	64 bytes
M_{IGMP}	IGMP message size	64 bytes
M_{DHCP}	DHCPDISCOVERY/DHCPOFFER/ DHCPRQUEST/DHCPACK message size	300 bytes
M_{MIP_Reg}	MIPv4 Registration Request message size	74 bytes
M_{MIP_Rep}	MIPv4 Registration Reply message size	48 bytes
M_{IPv4}	Size of IPv4 packet header	20 bytes
M_{DATA}	Multicast data size	120 bytes
h_{2ST}	Number of hops between any 2 satellite terminals	1
h_{GW-INT}	Number of hops between satellite GW and Internet nodes through internet	10

In this analysis, signaling cost (C_{sign}) and the packet delivery cost (C_{PD}) before and after GWH are evaluated. Signaling cost is defined as the accumulative signaling overhead for supporting mobile multicast source GWH in a multi-beam satellite network and is calculated as the product of the size of mobility (handover) signaling messages and their hop distances [18]. Packet delivery cost (C_{PD}) on the other hand is the accumulative traffic overhead incurred in delivering a packet along a routing path. CPD is calculated by multiplying the data packet size by the hop distance. Here, only the packet delivery cost within the satellite network (satellite receivers) will be considered before and after the GWH for both our scheme and the MIP HS–based approach. Table II shows the messages sizes and number of hops used for the analysis. These parameters are referenced from [1][19][20]. The hop distance between any two satellite terminals under different GWs is assumed to be 1. This is because each GW has a different IP address space, hence a different IP network.

A. Modelling the Proposed M3U scheme

Figure 6 shows the signaling messages (extracted from Figure 3) involved in the proposed M3U scheme for multicast source mobility support. It is assumed here that the target GW was not yet a member of the multicast channel served by the mobile source, so an IGMP join report is issued by the target GW to NCC after receiving an updated channel subscription request (PIM-SSM Join) from receivers in the Internet (see Figures 2 and 4). It is also assumed in Figure 6 that SNMP–Request and SNMP-Response messages carrying the SI tables, RUI and allocated bandwidth resources + IP address have the same packet length or size. From Figure 6, the signaling cost per GWH for the proposed M3U scheme $C_{sign}^{(M3U)}$ is given by:

$$C_{sign}^{(M3U)} = \begin{matrix} C_{SYNC} + 4C_{SNMP} + C_{SIUM} + C_{CUM} + C_{PIM-SM} \\ + C_{TIM} + C_{SI-t} + C_{ACQ} + C_{CMT} + C_{IGMP} + C_{MMT} \end{matrix} \quad . (1)$$

Where each of the terms in (1) represents the cost of each signaling message shown in Figure 6.

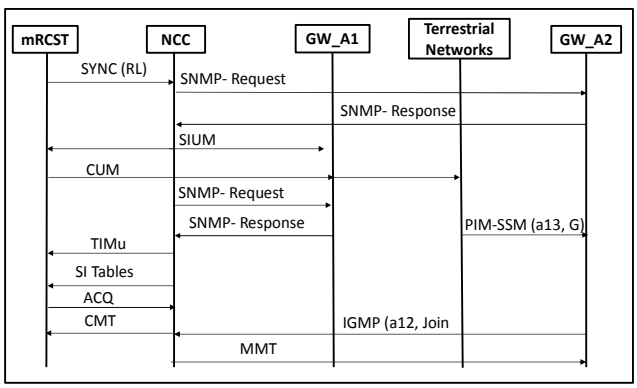


Figure 6. Signaling messages when using M3U

Substituting the cost value (message size × hop distance) for each term in (1) and re-arranging implies $C_{sign}^{(M3U)}$ is given by:

$$C_{sign}^{(M3U)} = \begin{matrix} \alpha h_{2ST}(M_{SYNC} + 4M_{SNMP} + M_{SIUM} + M_{CUM} + M_{TIM} + M_{SI\_t} + M_{ACQ} \\ + M_{CMT} + M_{IGMP} + M_{MMT}) + \beta h_{GW\_INT}(M_{CUM} + M_{PIM\_SM}) \end{matrix} \quad . (2)$$

Where α and β are weighting factors for wireless (satellite) and wired links, respectively. They are used to emphasize the link stability [18][21].

The packet delivery cost $C_{PD}^{(M3U)}$ for each multicast packet to any receiver within the satellite network (i.e., mesh communication) is given by:

$$C_{PD}^{(M3U)} = \alpha M_{DATA} h_{2ST} \, . \quad (3)$$

The packet delivery cost before and after GWH under this scheme will remain the same. This is because no extra hop will be traversed by the packet after GWH.

The packet delivery cost per multicast session $C_{PDS}^{(M3U)}$ can be determined using the average session transmission rate λ_S, from the mobile source and the average session length in packets E_S [18][22] . This is calculated as the product of λ_S, E_S and $C_{PD}^{(M3U)}$ (i.e., the packet delivery cost for one multicast packet). This implies packet delivery cost per multicast session $C_{PDS}^{(M3U)}$ is given by [18] [22]:

$$C_{PDS}^{(M3U)} = \lambda_S \; E_S \; C_{PD}^{(M3U)} \, . \quad (4)$$

B. Modelling of MIP HS-based approach

The MIP HS or bi-directional tunnelling approach relies on mobile IP architectural entities, i.e., HA and mobile node (mRCST). When the mobile source moves away from its home network at GW_A1 to a foreign network at GWA_2, it has to register its care-of address [23] to its HA at home network.

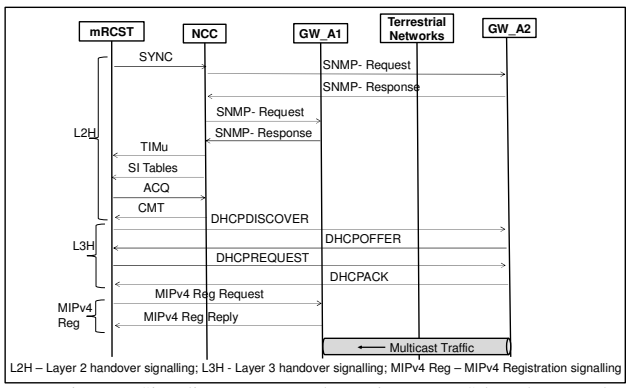


Figure 7. Signaling messages when using MIP HS-based approach

Details of MIP HS-based approach can be found in [3]. Figure 7 shows the signaling messages involved during GWH using MIP HS-based approach. The details of the content of Figure 7 can be found in [7][19][23][24] .

Similarly as in (1) and (2) above, the signaling cost per GWH for MIP HS-based approach, $C_{sign}^{(HS)}$ is given by:

$$C_{sign}^{(HS)} = \begin{matrix} C_{SYNC} + 4C_{SNMP} + C_{TIM} + C_{SI-t} + C_{ACQ} + C_{CMT} \\ + 4C_{DHCP} + C_{MIP-\mathrm{Re}\,q} + C_{MIP-\mathrm{Re}\,p} + 2C_T \end{matrix} . \quad (5)$$

Where C_T is the cost of tunnelling each IPv4 packet header.

$$C_{sign}^{(HS)} = \begin{matrix} \alpha h_{2ST}(M_{SYNC} + 4M_{SNMP} + M_{TIM} + M_{SI\_t} + M_{ACQ} \\ + M_{CMT} + 4M_{DHCP} + M_{MIP\text{-}\mathrm{Re}q} + M_{MIP\text{-}\mathrm{Re}p} + 2M_{IPv4}) \end{matrix} . \quad (6)$$

The packet delivery cost $C_{PD\_brfore}^{(HS)}$ for each multicast packet to any receiver within the satellite network (i.e., mesh communication) before GWH in MIP HS-based approach is given by:

$$C_{PD\_before}^{(HS)} = \alpha M_{DATA} h_{2ST} . \quad (7)$$

After GWH, the packet delivery routing path changes as the mobile source has to first tunnel the multicast data to its HA at home network for delivery into the source-specific tree. This implies the multicast data will under a double hop communication from the mobile source to reach the listening RCSs/RSGWs. Hence, the packet delivery cost after GWH $C_{PD\_after}^{(HS)}$ is given by:

$$C_{PD\_after}^{(HS)} = \alpha h_{2ST} (2M_{Data} + M_{IPv4}) . \quad (8)$$

Similarly as in (4), the packet delivery cost per multicast session after GWH for MIP HS-based approach is given by:

$$C_{PD\_after}^{(HS)} = \lambda_S \; E_S \; C_{PD\_after}^{(HS)} . \quad (9)$$

V. NUMERICAL ANALYSIS AND RESULTS

Assuming here that the satellite beams (coverage area) are circular and of identical dimensions, the border crossing rate of the mobile source (mRCST) or in other words, the frequency at which GWH is taking place f_{GWH} is given by [18][22]:

$$f_{GWH} = \frac{2V}{\pi R} . \quad (10)$$

Where V is the average velocity of the mobile source and R is the radius of the circular satellite beam.

The total signaling cost C_{T_Sign} to support the multicast source mobility is therefore given by the product of the signaling cost per GWH and the frequency of GWH. So, from (2) and (10), the total signaling cost for the proposed M3U scheme C_{T_Sign} is given by:

$$C_{T\_Sign}^{(M3U)} = \frac{2V}{\pi R} C_{sign}^{(M3U)} . \quad (11)$$

Similarly, from (6) and (10), the total signaling cost for MIP HS-based approach is given by:

$$C_{T\_Sign}^{(HS)} = \frac{2V}{\pi R} C_{sign}^{(HS)} . \quad (12)$$

For numerical evaluations, the parameters in Table II and the following are used in the analytical models presented in Section IV: E_S = 10, α = 2, β = 1 [18][21].

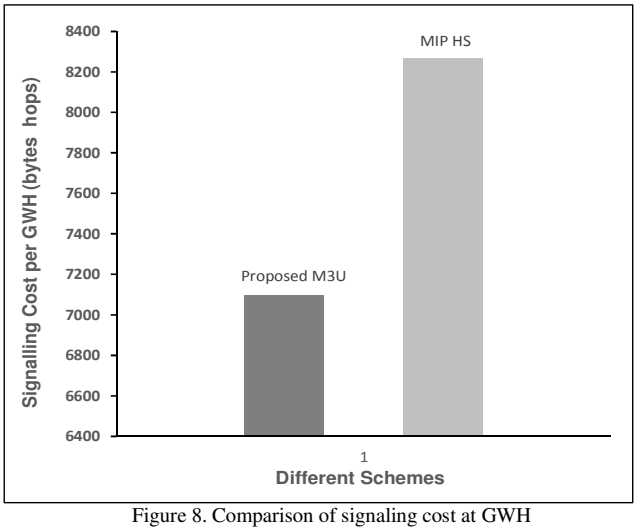


Figure 8. Comparison of signaling cost at GWH

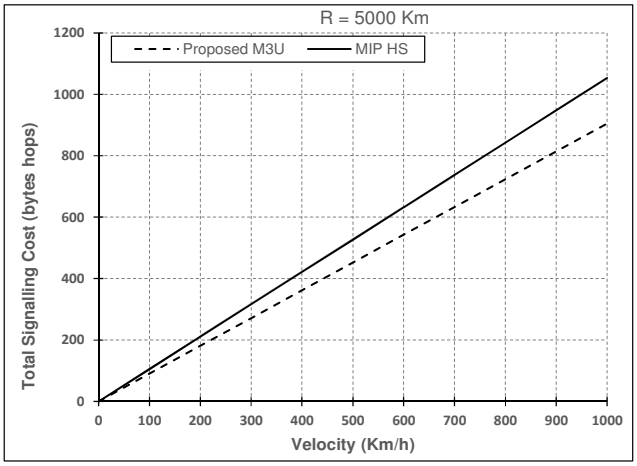


Figure 9. Variation of total signaling cost with velocity

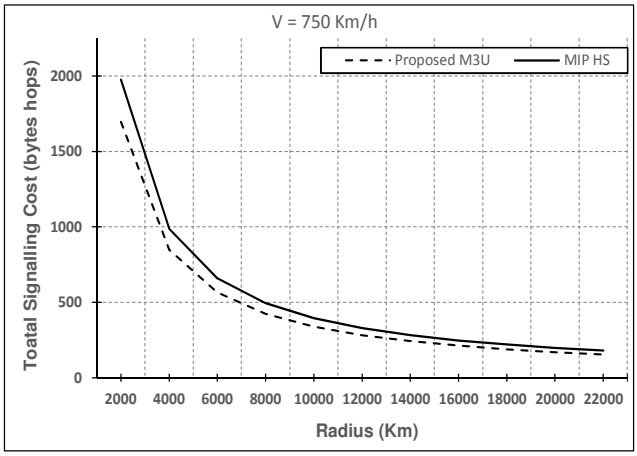


Figure 10. Variation of total signaling cost with radius

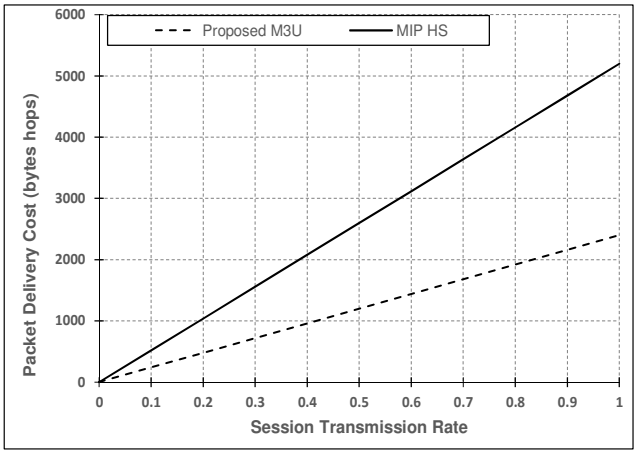


Figure11. Comparison of packet delivery cost

A. *Signaling Cost*

Figure 8 shows the signaling cost at GWH for the proposed M3U scheme as compared with the MIP HS-based approach. These results are obtained by substituting the numerical values of the parameters in (2) and (6), respectively. From Figure 8, it can be seen that signaling cost of the MIP HS-based approach is much higher than that in the proposed M3U scheme. The extra signaling cost for location update at the HA is one of the major reasons for the higher GWH signaling cost in MIP HS-based approach. By making use of (11) and (12), the total signaling cost during GWHs for the proposed M3U and MIP HS-based schemes, respectively, are investigated in Figures 9 and 10. In Figure 9, the radius of the satellite beam is set at 5000 Km and the total signaling cost is measured as the velocity of the mRCST (mobile source) is varied from 0 to 1000Km/h. Figure 9 reveals that the total signaling cost increases as the velocity of the mobile source increases. This is expected, since the higher the velocity, the more the frequency of GWH (border crossing) and hence, the higher total signaling cost. It can also be deduced from Figure 9 that the total signaling cost for MIP HS-based approach is generally higher than that for the proposed M3U scheme. These results show that in a similar multi-beam satellite network providing mobility support, satellite terminals on slow moving platforms like the maritime vessels will incur less signaling cost (overhead) than those on fast moving platforms like long haul flights (aircrafts).

On the other hand, Figure 10 shows how the total signaling cost changes with varying satellite beam radius at a fixed mobile source velocity of 750Km/h. As shown in Figure 10, the total signaling cost reduces as the radius of the satellite beam increases. This is true because the larger the satellite beams (radius), the fewer the number of GWHs required by the mobile source travelling at a constant velocity. But the smaller the satellite beam, the more the number of GWHs required for any satellite terminal travelling at a constant speed.

More GWHs implies more signaling cost and vice versa. Although the recent trend in satellite beam size is moving towards narrow beams instead of big beams, the main reasons are the power requirements of the RCST and the frequency reuse (to increase capacity). Figure 10 also shows that the proposed M3U scheme outperforms the MIP HS-based approach in total signaling cost against radius of satellite beam.

The results in Figure 10 could be particularly important to designers of global multi-beam satellite networks that support mobility, as the sizes of the GW beams will have an effect on the overall handover overhead.

B. *Packet Delivery Cost*

The packet delivery cost for both schemes after GWH obtained by making use of (4) and (9), are investigated in Figure 11. The display in Figure 11 shows that packet delivery cost increases as the session transmission rate increases. Also, Figure 11 shows that for any particular session transmission rate, the packet delivery cost for MIP HS-based approach is much higher than that for the proposed M3U scheme. This is consistent with the fact that in the proposed M3U scheme, there is mesh communication with a single hop over the satellite even when the mobile source is away from home and also, there is no encapsulation (tunnelling) of multicast packet at all in any stage. But in MIP HS-based approach, packet delivery has to undergo a double hop transmission over satellite (i.e., through HA), thus incurring higher packet delivery cost. Also, the higher multicast packet delivery cost in MIP HS-based approach when the mobile source is away from home is due to the fact that tunnelling is employed to route packets between the mobile source and the HA. The extra IP packet header here increases the packet delivery cost.

From all the results presented in Figures 8, 9, 10 and 11, the proposed M3U scheme outperforms the MIP HS-based approach.

VI. CONCLUSION AND FUTURE WORK

Support for IP mobile multicast over bandwidth constrained environments like satellites is very important, as it efficiently makes use of the available bandwidth resources and thus provide cost effective network services. Due to transparency and reverse path forwarding problems, the handover of a mobile multicast source in SSM from one IP network to another will result to the breakage of the multicast delivery tree. While some solutions to support multicast source mobility in SSM have been proposed for the internet, it was seen that these are not very suitable in a satellite network.

This paper proposes a suitable solution for multicast source mobility for SSM in a multi-beam satellite network. It presents the network architecture and proposes a new Multicast Mobility Management Unit (M3U) located at the NCC. Also, three new control messages have been proposed to provide IP mobility support to the mobile multicast source during GWH. The functioning of the M3U and the new control messages provide an elegant and effective solution for the mobile multicast source transparency and RPF problems in SSM.

Performance evaluation for the proposed M3U scheme and the MIP HS-based approach was carried out using signaling cost during GWH handover and packet delivery cost after GWH. Provided other factors remain constant, the results obtained show the following:

- The total GWH signaling cost is directly proportional to the speed of the mobile source, i.e., the higher the speed, the higher the total GWH signaling cost and vice versa.
- The total GWH signaling cost is inversely proportional to the radius of the satellite (gateway) beam, i.e., the total GWH signaling cost reduces as the radius of the satellite beam increases and vice versa.
- The packet delivery cost is directly proportional to the session transmission rate. This means that the packet delivery cost increases as the session transmission rate increases and reduces as the session transmission rate reduces.

In all scenarios investigated, the results obtained show that the proposed M3U scheme outperformed the MIP HS-based approach in terms of total GWH signaling cost and packet delivery cost when the mobile source is away from home network.

For future work, ways of integrating the proposed M3U scheme into PMIPv6-based IP mobility over satellite will be examined. This could potentially lead to faster and better handover performance compared to the individual M3U or PMIPv6 scheme.

REFERENCES

[1] E. K. Jaff, P. Pillai, and Y. F. Hu, "Source mobility support for source specific multicast in satellite setworks," in *MOBILITY 2013 : The Third International Conference on Mobile Services, Resources, and Users*, Lisbon, Portugal, 2013, pp. 69 - 74.

[2] H. Holbrook, B. Cain, and B. Haberman, "Using internet group management protocol version 3 (IGMPv3) and multicast listener discovery protocol version 2 (MLDv2) for source-specific multicast," IETF RFC 4604, August 2006.

[3] I. Romdhani, M. Kellil, L. Hong-Yon, A. Bouabdallah, and H. Bettahar, "IP mobile multicast: challenges and solutions," *Communications Surveys & Tutorials, IEEE,* vol. 6, pp. 18-41, First Quarter 2004.

[4] P. Ferguson and D. Senie, "Network ingress filtering: defeating denial of service attacks which employ IP source address spoofing," IETF RFC 2827, May 2000.

[5] "DVB fact sheet- August 2012: return channel satellite," DVB Project Fact Sheet, August 2012.

[6] "SatNet DVB-RCS vs. proprietary VSAT systems," Advantech Satellite Networks DVB-RCS, April 2006.

[7] "Digital video broadcasting (DVB); interaction channel for satellite distribution systems; guidelines for the use of EN 301 790 in mobile scenarios," ETSI TR 102 768, April 2009.

[8] I. Romdhani, H. Bettahar, and A. Bouabdallah, "Transparent handover for mobile multicast sources," in *Networking, International Conference on Systems and International Conference on Mobile Communications and Learning Technologies, 2006. ICN/ICONS/MCL 2006. International Conference on*, 2006, pp. 145-145.

[9] T. C. Schmidt and M. Wählisch, "Extending SSM to MIPv6—problems, solutions and improvements," in *selected papers from TERENA Networking Conference, Computational Methods in Science and Technology*, Poznań, 2005, pp. 147-152.

[10] T. C. Schmidt, M. Wählisch, and M. Wodarz, "Fast adaptive routing supporting mobile senders in source specific multicast," *Telecommunication Systems, Springer,* vol. 43, pp. 95 – 108, February 2010.

[11] C. S. Jelger and T. Noel, "Supporting mobile SSM sources for IPv6," in *Global Telecommunications Conference, 2002. GLOBECOM '02. IEEE*, November 2002, pp. 1693-1697.

[12] T. C. Schmidt, S. Gao, H. Zhang, and M.Waehlisch, "Mobile multicast sender support in proxy mobile IPv6 (PMIPv6) domains," IETF, draft-ietf-multimob-pmipv6-source-03, February 2013.

[13] B. Fenner, H. He, B. Haberman, and H. Sandick, "Internet group management protocol (IGMP)/multicast listener discovery (MLD)-based multicast forwarding ("IGMP/MLD proxying")," IETF RFC 4605, August 2006.

[14] S. Gundavelli, K. Leung, V. Devarapalli, K. Chowdhury, and B. Patil, "Proxy mobile IPv6," IETF RFC 5213, August 2008.

[15] "Satellite earth stations and systems (SES); broadband satellite multimedia (BSM); connection control protocol (C2P) for DVB-RCS; background information," ETSI TR 102 603, January 2009.

[16] "Satellite earth stations and systems (SES); broadband satellite multimedia (BSM); regenerative satellite mesh - B (RSM-B); DVB-S/DVB-RCS family for regenerative satellites; Part 2: satellite link control layer," ETSI TS 102 429-2, October 2006.

[17] B. Cain, S. Deering, I. Kouvelas, B. Fenner, and A. Thyagarajan, "Internet group ganagement protocol, version 3," IETF RFC 3376, October 2002.

[18] L. Jong-Hyouk, T. Ernst, and C. Tai-Myoung, "Cost analysis of IP mobility management protocols for consumer mobile devices," *Consumer Electronics, IEEE Transactions on,* vol. 56, pp. 1010-1017, 2010.

[19] "Digital video broadcasting (DVB); transport of MPEG-2 TS based DVB services over IP based networks," ETSI TS 102 034 V1.3.1, October 2007.

[20] X. Jiang and U. Narayanan, "Performance analysis of mobility support in IPv4/IPv6 mixed wireless networks," *Vehicular Technology, IEEE Transactions on,* vol. 59, pp. 962-973, 2010.

[21] J. H. Lee, T. Ernst, D. J. Deng, and H. C. Chao, "Improved PMIPv6 handover procedure for consumer multicast traffic," *Communications, IET,* vol. 5, pp. 2149-2156, 2011.

[22] L. Jong-Hyouk, J. M. Bonnin, Y. Ilsun, and C. Tai-Myoung, "Comparative handover performance analysis of IPv6 mobility management protocols," *Industrial Electronics, IEEE Transactions on,* vol. 60, pp. 1077-1088, 2013.

[23] C. Perkins, "IP mobility support for IPv4," IETF RFC 3344, Aug. 2002.

[24] "Digital video broadcasting (DVB); guidelines for DVB IP phase 1 handbook," ETSI TR 102 542, November 2006.

www.iariajournals.org

International Journal On Advances in Intelligent Systems

ICAS, ACHI, ICCGI, UBICOMM, ADVCOMP, CENTRIC, GEOProcessing, SEMAPRO, BIOSYSCOM, BIOINFO, BIOTECHNO, FUTURE COMPUTING, SERVICE COMPUTATION, COGNITIVE, ADAPTIVE, CONTENT, PATTERNS, CLOUD COMPUTING, COMPUTATION TOOLS, ENERGY, COLLA, IMMM, INTELLI, SMART, DATA ANALYTICS

issn: 1942-2679

International Journal On Advances in Internet Technology

ICDS, ICIW, CTRQ, UBICOMM, ICSNC, AFIN, INTERNET, AP2PS, EMERGING, MOBILITY, WEB

issn: 1942-2652

International Journal On Advances in Life Sciences

eTELEMED, eKNOW, eL&mL, BIODIV, BIOENVIRONMENT, BIOGREEN, BIOSYSCOM, BIOINFO, BIOTECHNO, SOTICS, GLOBAL HEALTH

issn: 1942-2660

International Journal On Advances in Networks and Services

ICN, ICNS, ICIW, ICWMC, SENSORCOMM, MESH, CENTRIC, MMEDIA, SERVICE COMPUTATION, VEHICULAR, INNOV

issn: 1942-2644

International Journal On Advances in Security

ICQNM, SECURWARE, MESH, DEPEND, INTERNET, CYBERLAWS

issn: 1942-2636

International Journal On Advances in Software

ICSEA, ICCGI, ADVCOMP, GEOProcessing, DBKDA, INTENSIVE, VALID, SIMUL, FUTURE COMPUTING, SERVICE COMPUTATION, COGNITIVE, ADAPTIVE, CONTENT, PATTERNS, CLOUD COMPUTING, COMPUTATION TOOLS, IMMM, MOBILITY, VEHICULAR, DATA ANALYTICS

issn: 1942-2628

International Journal On Advances in Systems and Measurements

ICQNM, ICONS, ICIMP, SENSORCOMM, CENICS, VALID, SIMUL, INFOCOMP

issn: 1942-261x

International Journal On Advances in Telecommunications

AICT, ICDT, ICWMC, ICSNC, CTRQ, SPACOMM, MMEDIA, COCORA, PESARO, INNOV

issn: 1942-2601