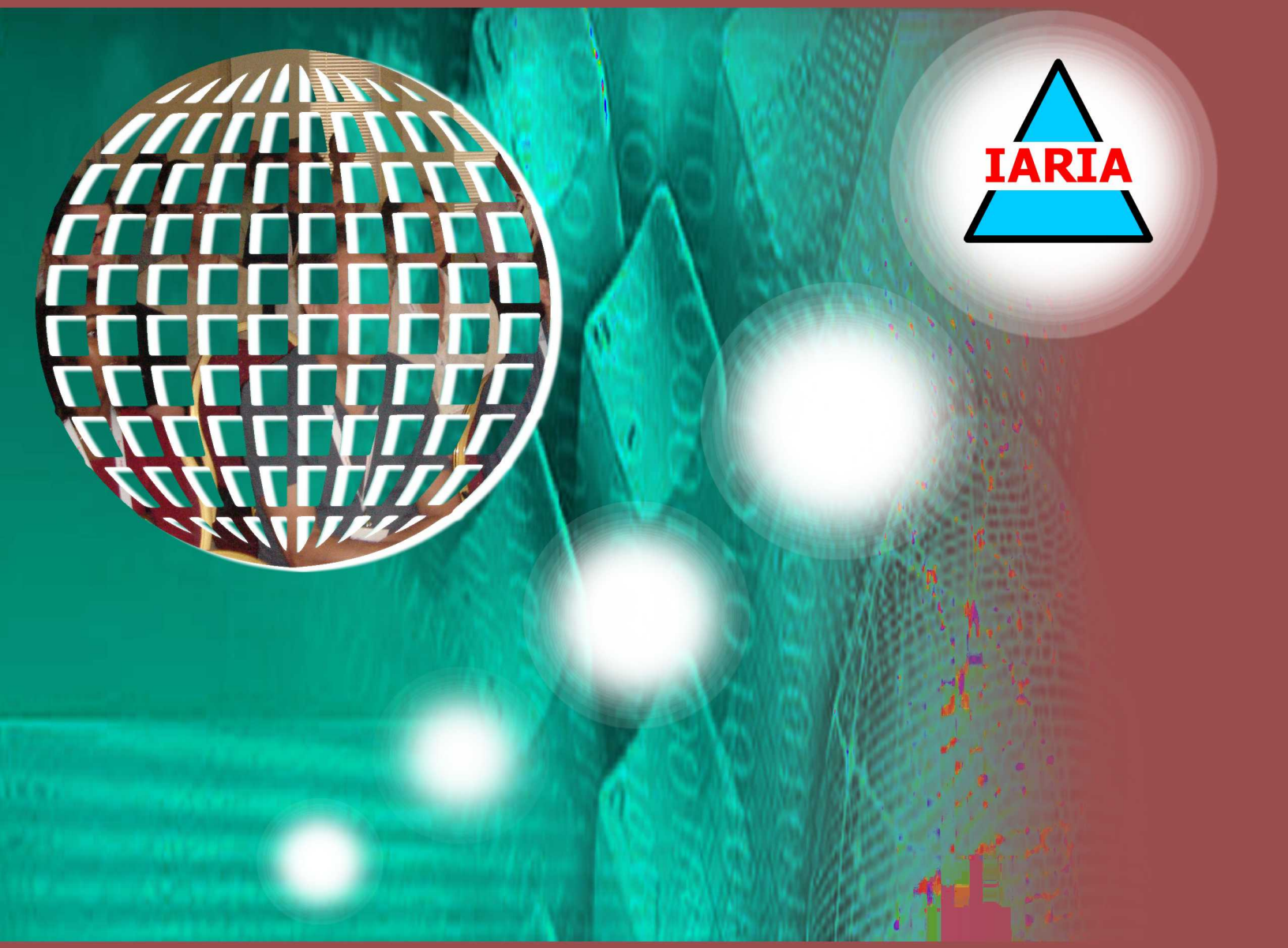
International Journal on Advances in Software



2010 vol. 3 nr. 3&4

The *International Journal on Advances in Software* is published by IARIA.
ISSN: 1942-2628
journals site: http://www.iariajournals.org
contact: petre@iaria.org

Sponsored by IARIA
www.iaria.org

International Journal on Advances in Software
Volume 3, Number 3 & 4, 2010

- Olga Ormandjieva, Concordia University-Montreal, Canada
- Constantin Paleologu, University 'Politehnica' of Bucharest, Romania
- Liviu Panait, Google Inc., USA
- Kenji Saito, Keio University, Japan
- Ashok Sharma, Satyam Computer Services Ltd – Hyderabad, India
- Marcin Solarski, IBM-Software Labs, Germany

Advanced Computing

- Matthieu Geist, Supelec / ArcelorMittal, France
- Jameleddine Hassine, Cisco Systems, Inc., Canada
- Sascha Opletal, Universitat Stuttgart, Germany
- Flavio Oquendo, European University of Brittany - UBS/VALORIA, France
- Meikel Poess, Oracle, USA
- Kurt Rohloff, BBN Technologies, USA
- Said Tazi, LAAS-CNRS, Universite de Toulouse / Universite Toulouse1, France
- Simon Tsang, Telcordia Technologies, Inc. - Piscataway, USA

Geographic Information Systems

- Christophe Claramunt, Naval Academy Research Institute, France
- Dumitru Roman, Semantic Technology Institute Innsbruck, Austria
- Emmanuel Stefanakis, Harokopio University, Greece

Databases and Data

- Peter Baumann, Jacobs University Bremen / Rasdaman GmbH Bremen, Germany
- Qiming Chen, HP Labs – Palo Alto, USA
- Ela Hunt, University of Strathclyde - Glasgow, UK
- Claudia Roncancio INPG / ENSIMAG - Grenoble, France

Intensive Applications

- Fernando Boronat, Integrated Management Coastal Research Institute, Spain
- Chih-Cheng Hung, Southern Polytechnic State University, USA
- Jianhua Ma, Hosei University, Japan
- Milena Radenkovic, University of Nottingham, UK
- DJamel H. Sadok, Universidade Federal de Pernambuco, Brazil
- Marius Slavescu, IBM Toronto Lab, Canada
- Cristian Ungureanu, NEC Labs America - Princeton, USA

Testing and Validation

- Michael Browne, IBM, USA
- Cecilia Metra, DEIS-ARCES-University of Bologna, Italy
- Krzysztof Rogoz, Motorola, USA
- Sergio Soares, Federal University of Pernambuco, Brazil

- Alin Stefanescu, University of Pitesti, Romania
- Massimo Tivoli, Universita degli Studi dell'Aquila, Italy

Simulations

- Robert de Souza, The Logistics Institute - Asia Pacific, Singapore
- Ann Dunkin, Hewlett-Packard, USA
- Tejas R. Gandhi, Virtua Health-Marlton, USA
- Lars Moench, University of Hagen, Germany
- Michael J. North, Argonne National Laboratory, USA
- Michal Pioro, Warsaw University of Technology, Poland and Lund University, Sweden
- Edward Williams, PMC-Dearborn, USA

International Journal on Advances in Software
Volume 3, Numbers 3 & 4, 2010

CONTENTS

From Meta-modeling to Automatic Generation of Multimodal Interfaces for Ambient Computing

José Rouillard
LIFL Laboratory – University of Lille 1
59655 Villeneuve d'Ascq Cedex - France
jose.rouillard@univ-lille1.fr

Jean-Claude Tarby
LIFL Laboratory – University of Lille 1
59655 Villeneuve d'Ascq Cedex – France
jean-claude.tarby@univ-lille1.fr

Xavier Le Pallec
LIFL Laboratory – University of Lille 1
59655 Villeneuve d'Ascq Cedex - France
xavier.le-pallec@univ-lille1.fr

Raphaël Marvie
LIFL Laboratory – University of Lille 1
59655 Villeneuve d'Ascq Cedex - France
raphael.marvie@univ-lille1.fr

***Abstract* — This paper presents our approach to design multichannel and multimodal applications as part of ambient intelligence. Computers are increasingly present in our environments, whether at work (computers, photocopiers), at home (video player, hi-fi, microwave), in our cars, etc. They are more adaptable and context-sensitive (e.g., the car radio that lowers the volume when the mobile phone rings). Unfortunately, while they should provide smart services by combining their skills, they are not yet designed to communicate together. Our results, mainly based on the use of a software bus and a workflow, show that different devices (such as Wiimote, multi-touch screen, telephone, etc.) can be coordinated in order to activate real things (such as lamp, fan, robot, webcam, etc.). A smart digital home case study illustrates how using our approach to design with ease some parts of the ambient system and to redesign them during runtime.**

Keywords* — *Pervasive computing; ubiquitous computing; ambient intelligence; multi-channel interaction; multimodality.

I. INTRODUCTION

Ambient computing is one of the most significant recent advances in Human-Computer Interaction (HCI). Due to the arising of pervasive and ubiquitous computing, the design of HCI has to take into account the context of interactions. The objective is to allow users to interact with a smart system with low constraints through the use of multiple modalities, channels, and devices. In the future, with the availability of new devices and smart objects, ambient computing will allow the definition of services seamlessly interacting with both environment and users.

Our current work takes place in this context of ambient computing. In order to support dynamic unplanned interactions with the user, services have to adapt themselves to their mutating environment – resulting from the user mobility and the variability of her/his context. This requires (a) the availability of distributed devices such as PDA (Personal Digital Assistant), laptops, smartphones, robots, probes, and (b) easing the discovery of these devices.

Currently, development tools that enable us to easily generate and integrate ambient services are lacking. Each piece of software is developed on its own, and then integrated in the system. This introduces additional costs as well as misconfiguration risks. This paper focuses on the design of multi-channel interfaces relying on a workflow engine in order to ease the realization of ambient systems.

This document is an extended version of our previous paper [1]. It is structured as follows. Section two presents related works. Section three explains the background and motivation of this project. Section four gives an overview of our conceptual approach in order to tackle the emerging problems encountered. Section five explains in details our approach from an implementation point of view. Section six describes a case study around the smart digital home thematic and presents the benefits of our approach for the design and generation of multimodal and multichannel interactive systems. Then, a conclusion gives our roadmap for future work.

II. RELATED WORK

Computer frameworks and languages have been proposed specifically to facilitate the development of multimodal interfaces. In the World Wide Web Consortium (W3C) MultiModal Interaction (MMI) framework [2], the interaction manager invokes specific application functions and accesses information in a dynamic processing module. The interaction manager presents the result to the user via one or more output components. Obviously, the interaction manager of this framework is very important because it coordinates data and manages execution flow among various input and output components. It also responds to inputs from the input components, updates the interaction state and the application context, and initiates output to one or more output components. Developers use several approaches to implement interaction managers, including: Traditional programming languages such as C or C++;

Speech Application Language Tags (SALT), which extends HTML by adding a handful of HTML tags to support speech recognition, speech synthesis, audio file replay, and audio capture; XHTML plus Voice (often referred as "X+V"), in which the VoiceXML 2.0 [3] voice dialog control language is partitioned into modules that are embedded into HTML; Formal specification techniques such as state transition diagrams and Harel Statecharts [4].

The OpenInterface project [5] is dedicated to multimodal interaction. In this project, everyday objects can take part in the interaction in ubiquitous computing (including an augmented table for instance) and the user can freely switch from one modality to another according to her/his context: running in the street, at home, in front of a big screen in an airport, etc. This project aims at the design and development of an open source framework for multimodal interaction: the OpenInterface framework.

Those kinds of projects are mainly devoted to the study of multimodal interactions, allowing the usage of more than one device or modality at the same time in order to interact with a main system connected to Internet. Ambient computing increases complexity because related applications are not supposed to manage only devices and modalities, but also channels (cf. Section III.A) in order to allow intelligent and context-aware communications. Our research activity takes place in ambient computing area.

III. BACKGROUND

This background section is divided in three parts: multimodality, user activity, and connection with the ambient environment.

A. Multimodality

Our work tackles the ability of ambient computing to permit context-aware interactions between humans and machines. To do so, we rely on the use of multimodal and multi-channel interfaces in various fields of application such as coaching [6], learning, health care diagnosis, or in-situ marketing. For Frohlich, a channel is defined as an interface that makes a transformation of energy [7]. From a user's point of view, he distinguished voice and movement channels, and from the system's point of view he mentioned audio, visual, and haptic channels.

In the Human–Computer Interaction (HCI) domain, the notion of channel is not used very often and there are very few references to multi-channel research with some exceptions such as the work of [8]: "Often these modalities require specialized channels to allow access modalities such as cameras, microphones, and sensors. A multi-modal multi-channel system faces the challenge of accepting information from any input method and delivering information through the appropriate output methods".

Using a multi-channel approach allows users to interact with several channels choosing the most appropriate one each time in order to exchange with an entity. Such channels could be, for instance, plain paper, e-mail, phone, web site. Using a multimodal approach allows users to employ several modalities in order to interact with a single system. It can be sequential, like first being on the phone then on the web, or synergistic [9], like being on the phone while on the web. This approach implies some synchronization requirements both for the interfaces and knowledge bases used during the interactions.

There are very few tools that support the design and implementation of interfaces having such characteristics [10]. One of our goals is to study and propose infrastructures easing interactions that are both multimodal and multi-channel in an ambient context. In our work, we use the Multi-DMC referential proposed in [11]. It can identify a system based on three criteria: Device (D), Modal (M) and Channel (C). It has two positions (Mono or Multi) for each of the three criteria targeted (DMC). This represents 2^3 (=8) possibilities, which are presented on Figure 1 .

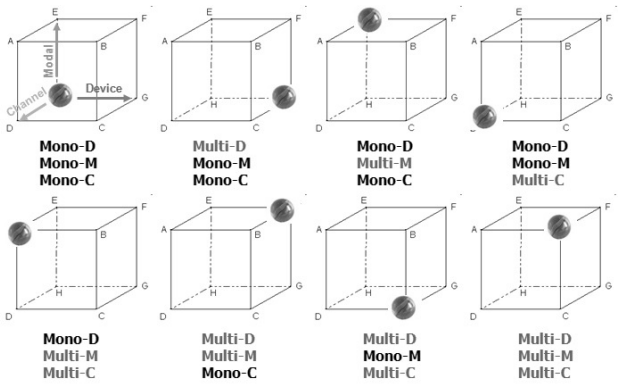


Figure 1. The Multi-DMC referential.

For a given system, one tries to indicate the position of each decisive factor. For example, the system represented on the bottom right of the figure is a multi-device, multimodal, and multi-channel system.

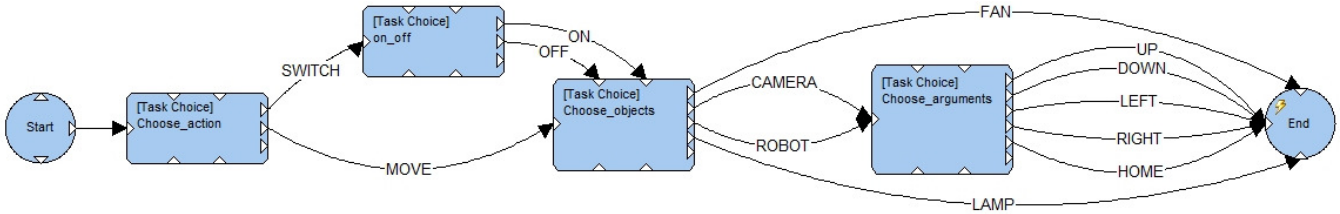


Figure 2. Workflow designed with the Studio Common Knowledge.

B. User activity

In this paper, we are targeting ambient systems, which aim to be user-friendly. Unfortunately, until now such systems are more difficult to conceive and to implement than traditional systems because of the heterogeneity of devices (hardware, software, different locations, etc.). Given its complexity, an ambient system must observe the rules of usability: guidance, low workload, concision, etc. [12]. Therefore, our work is based on concepts identified by HCI domain such as user's activity and logic of use.

The design of interactive systems is based on the notion of tasks and activities, themselves decomposed into sub-tasks/sub-activities whose arrangement is managed by temporal or structural sequences. Among all the approaches used in the design of interactive systems and using these concepts, some are more used such as task models [13][14], Petri nets [15], Statecharts [16], and workflows.

Given all these solutions we have chosen the workflows [17] because they are adapted for non-experts in order to explain their rationale for the use of ambient systems. First, Workflow concepts are as simple as needed to be understood by usual end-users. Second, related modeling languages have been generally designed to be readable by non-(computer)specialists. Finally, they are widespread in information systems and especially in document management systems.

C. Connection with the ambient environment

A major question in pervasive and ubiquitous computing is how to integrate physical objects (screen, chair, coffee machine, etc.) into multimodal applications using technologies such as Radio-frequency identification (RFID), Near field communication (NFC), Barcodes (1D or 2D as QR codes). This will help the users to manipulate freely virtual and real objects with commands like "identify this," "make a copy of that object, here", "move that webcam on the left," etc. We are using the notion of workflow in order to indicate to the user the tasks available at each point of the whole activity flow.

For our work, we are using Common Knowledge [17], which is a cross-platform business rules engine and management system that supports the capture, representation, documentation, maintenance, testing, and deployment of an organization's business rules and application logic. Common Knowledge allows the business logic to be represented in a variety of inter-operable visual formats, including Rete rules, workflows, flowcharts, decision tables, decision trees, decision grids, state maps, and scripts. The engine allows running, testing, and simulating the system behaviors. It can be used through many languages (such as Java, Delphi, VisualBasic, C#, DotNET, etc.) and platforms (Windows, Linux, UNIX).

Figure 2 presents an example of workflow designed graphically using the Studio Common Knowledge tool. It allows following different paths in order to complete a command such as "switch on fan", "move camera down", "switch off lamp", etc.

Figure 3 shows standard and advanced operators used to represent tasks, task choices, split or merge actions, timers, loops, etc. The result is stored using an XML format, in a file with an .aex extension. With our work the resulting system could be used through different modalities of interaction like graphically, vocally, with gesture, RFID, barcodes or a combination of those modalities. Instead of programming applications in an ad hoc fashion, our approach allows to query dynamically the workflow and to propose relevant information to the user while interacting with the system.

The notion of persistence is very important in this context. Indeed, we consider that a global interaction could be the result of many sub-interactions between the system and one or many users. It could also be the result of a sequence of sub-interactions conducted via different kind of channels and modalities.

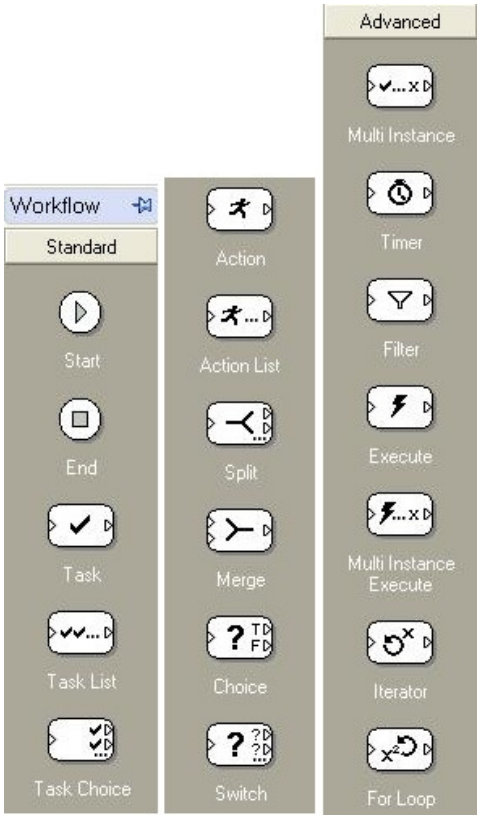


Figure 3. Standard and advanced operators available in Common Knowledge.

The Common Knowledge software supports this persistence feature.

IV. OUR APPROACH

In the context of interaction design based on the DMC referential, we believe, as we explained previously, that meaningful global actions on the system may be the result of a series of sub-actions. These sub-actions can be performed by multiple users cooperating. Several types of devices can be utilized (PC, Smartphone, mobile phone, etc.). Several modalities of interaction, such as direct manipulation (keyboard/mouse), voice, gesture, brain waves, can be employed both in input and output. Finally, multiple communication channels can be exploited such as the telephone or the Internet.

Currently we limit the use to an alternate multimodality (not synergistic). The triggering of a sub-action is based on the FIFO (First In, First Out) principle.

Figure 4 shows our approach based on a software bus. We used for instance the IVY bus [18] and the Web Server Event (WSE) bus (see Section V.B.1) in our experiments, as we will explain later. The "model driven" part mentioned on the figure is used for modeling the activities at a higher level, and mapping resulting models to workflow models, for example. The "engine" part uses an application that queries the generated workflow during the interaction via an Application Programming Interface (API). The "usage" part explains that different kinds of interaction are possible (web client, graphical user interface, vocal user interface, etc.). The "development" part means that the architecture is open in terms of futures applications, technologies and languages. In our approach, the transition from one state to another can be modelized with different tools, such as Petri nets or the usage of workflows for instance. The model driven approach allows working on an abstract level, independently from the chosen technical solution (Petri nets or workflow in our example).

A. Model driven approach

Figure 5 shows that a workflow (middle of the picture) is generated from a high-level model (left of the picture) thanks to a set of model transformation rules. This workflow model is used in order to describe objects and actions that can be applied on those objects using one or more devices in final interfaces (right of the picture).

Our work mainly concerns description of operating and use of multimodal interactions (MMI). The Activity concept is the main notion of our approach. We have experimented a workflow management system (see Section VI) as a support to define the operating logic of MMI and its corresponding execution. However, defining interaction logic may be done at different steps of an application design and so, according to different points of view.

With workflow concepts, we may use complex operators like fork/join, alternatives, variables, composite tasks, to describe some interactions sequences. Using these complex operators corresponds to use software and technical artifacts in order to address functional requirement(s). It may be relevant to define only the interactive requirements without dealing with technical details. The underlying idea is to define a modeling language dedicated to MMI, which contains a minimal set of concepts leaving technical aspects aside in order to easily focus on the interaction concern. With corresponding model transformation rules, the resulting MMI models would be mapped to several technical platforms (other than workflow management system). Thus, operating subtleties underlying the high-level models would be fully described within the generated technical models.

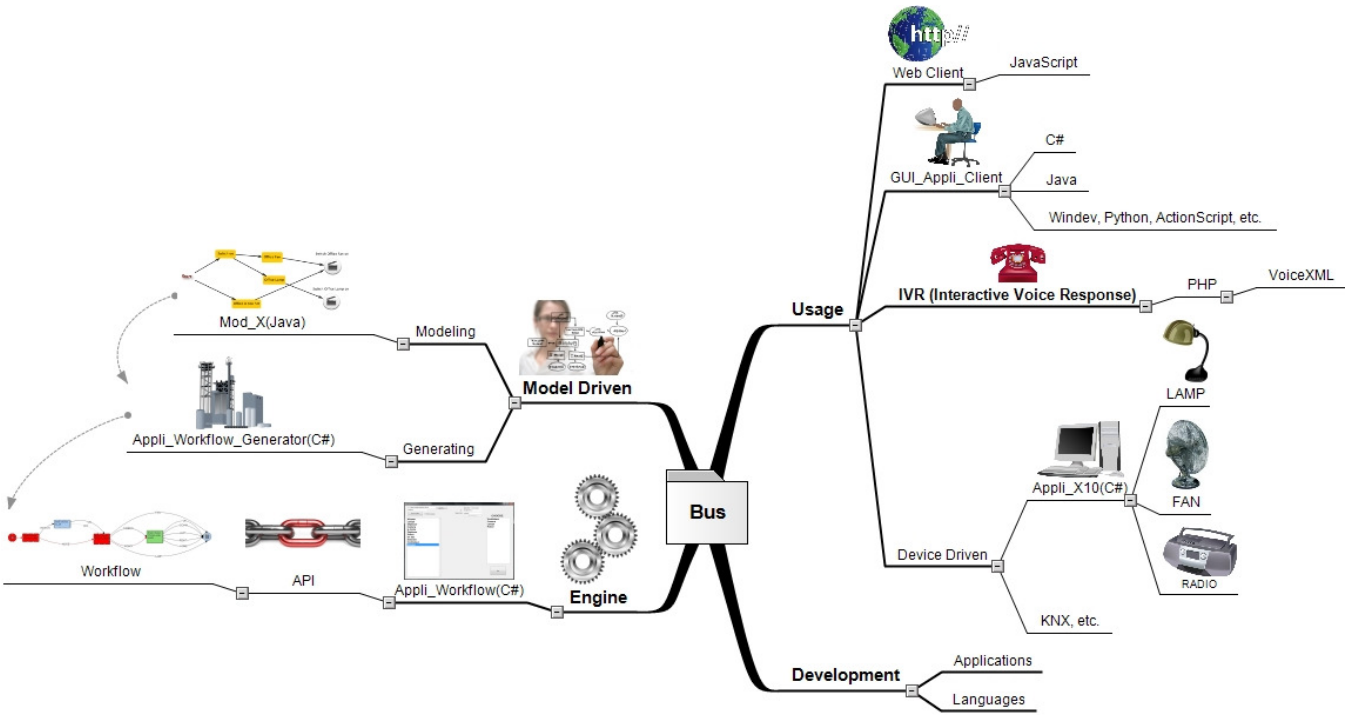


Figure 4. Our approach from meta-modeling to automatic generation of code.

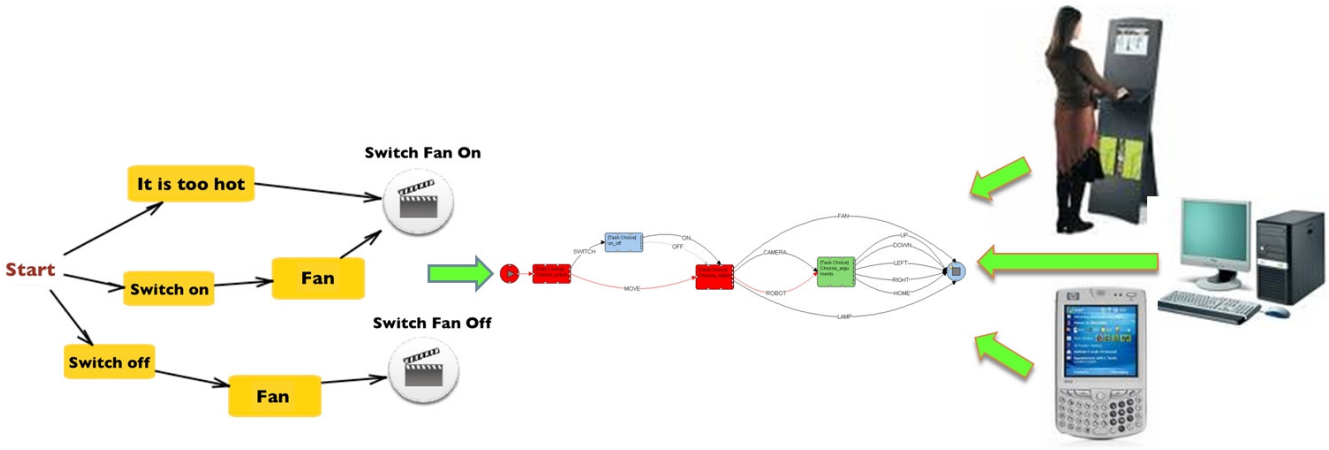


Figure 5. From meta modeling to final interfaces (via a workflow in this case).

This abstraction operation may be repeated in order to propose a simpler modeling language dedicated to end users with some technical skills (like persons who install home automation systems). Finally, we have currently chosen home automation as application domain of our work.

Our approach would have to be tested with other domains like healthcare, e-learning or tourism domains. Indeed, we cannot state that such previous high-level modeling language will still be adapted. In this perspective, we think that domain-oriented modeling languages will be useful in order to better contextualize MMI and to get finer mapping to technical platforms.

For all these reasons, we decided to adopt a Model-Driven Engineering approach, particularly the Object Management Group - Model Driven Architecture (OMG-MDA) declination (abstract towards concrete). We currently focus on an abstract meta-model and a mapping to workflow one. Figure 6 represents what we plan to do and what we have already done (gray rectangle).

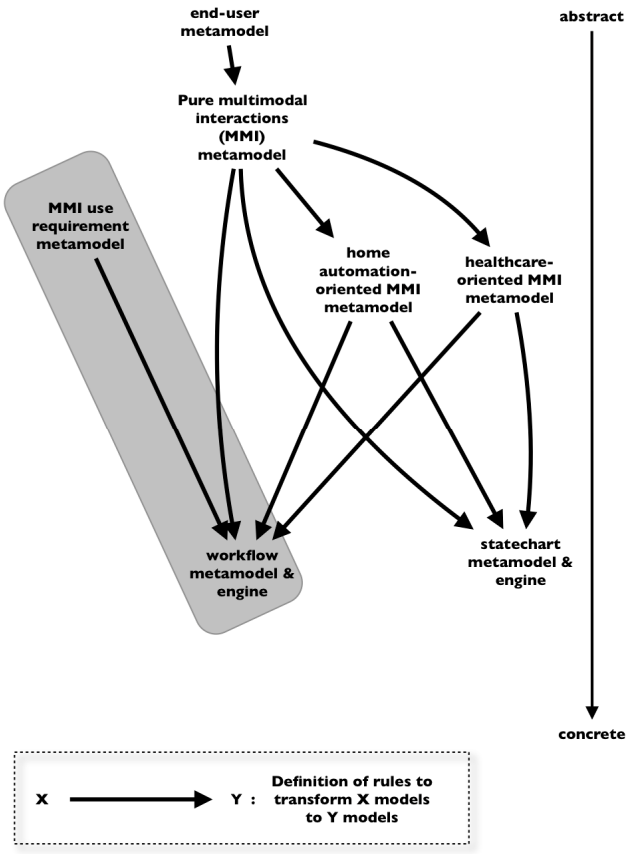


Figure 6. Our Model Driven Engineering (MDE) approach.

B. Conceptual architecture

From a conceptual point of view, our approach is based on the concept of message diffusion between the different actors in our system (an actor can be a user, an application or a device). When an actor wants to do something (for example the user wants to switch on a lamp, or the RFID reader will notify that it has decoded an RFID chip), it sends a message that is then received by all actors. Then the actors have the freedom to perform an action based on this message or not, depending on their needs.

Among the actors, the interpreter of messages has a special significance. It is the 'brain' of the system. Each time it receives a message, it processes it and tries to combine it with previously received messages to produce a higher level of abstraction message. For example, if the RFID reader has sent the message 'FAN chip decoded' and the interpreter has previously received the 'switch on' message, then the interpreter will combine the two for the final message 'switch on the fan'. This message will then in turn be sent to other actors. Among them, the application charged to operate the fan will send the X10 command to switch on the fan.

1) Communication bus

For the actors, several solutions are possible to communicate, such as:

- Pushing information, i.e., send messages to actors, such as broadcasting (sending messages to everyone), multicasting (sending messages only to certain actors), and so on.
- Pulling information. In this case, actors must request information themselves, for example by consulting a database or by consulting an actor responsible for managing the overall ambient system, etc.
- Using a distributed approach such as a multi-agent system.
- Using a centralized approach, such as a communication bus.

We chose to use a communication bus, whose function is to receive the messages and distribute them to all connected actors. This type of solution leaves considerable freedom in the implementation as we shall see later.

2) Device access layer

A communication bus is a relevant component in order to develop applications using interactive devices located in a room among several terminals. Sending a command to/from a remote device or listening/reacting to its events refers to marshalling/unmarshalling mechanisms. Its implementation is time-consuming and decreases code readability.

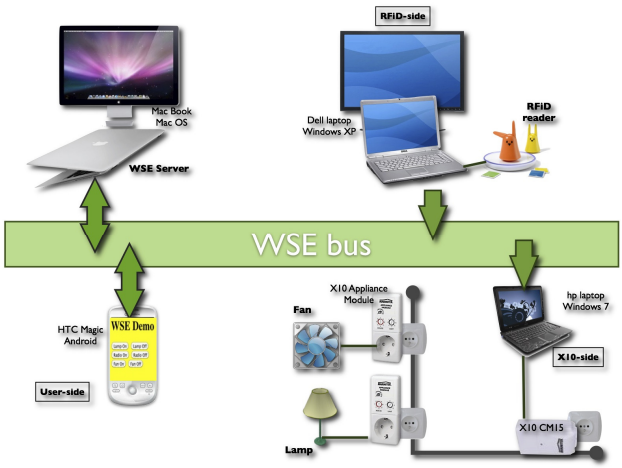


Figure 7. An example of configuration.

Figure 7 illustrates the device access layer based on the following example. A web page is loaded on an Android

mobile phone and can switch on a lamp or a fan through X10. The X10 manager (CM15 module) is connected to a PC (Windows) where a software adapter translates particular messages coming from the communication bus in X10 switch on/off orders. A RFID reader is connected to another PC: when a RFID tag is laid down on the reader, the background of previous web page changes to red, and when the RFID tag is picked up, the background is becomes green. These RFID reactions are possible thanks to a software adapter located to the related PC: for each RFID action, this adapter sends corresponding message on the communication bus.

We call terminals, the android mobile, X10-PC and RFID-PC. Web page and software adapters are called processes. To locate process, we usually mention user-side (Android mobile), X10-side or RFID-side.

To implement the previous example, we may program all processes as following. When user-side sends a "switch fan on" command to X10-side, the related process (i.e., web page) constructs a specific message and sends it through the message bus. The X10-side process receives it, detects it as a X10 order and acts in consequence. When a tag is laid down from the RFID reader, the related adapter reacts by constructing a message and sending it. The user-side process receives the message, detects it as a RFID event and sets the background to red if it is a lay-down event or to green if it is a pick-up one.

Constructing, sending, receiving and detecting messages is a tedious task (long and repetitive) and corresponding code blurs the whole implementation. For this reason it is highly recommended to use an additional software layer that hides messages bus stuff and therefore ease the implementation of MMI application.

V. OUR APPROACH: IMPLEMENTATION

This implementation section is divided in two parts. The first is about model driven engineering, and the second presents the implementation details of our conceptual architecture.

A. Model Driven Engineering

1) Towards a high-level MMI meta-model

As we previously mentioned in Section IV.A, we have adopted a model driven approach (MDA) to get a better separation of concerns (for example, by defining multimodal interactions in dedicated models) and to address the problem of platform heterogeneity.

We use ModX [19] as model framework. ModX is a MOF-tool [20] that we have implemented in 2004. It allows defining abstract and concrete syntaxes; it means meta-models and associated visual representations. ModX-users can create and edit models according to concrete syntaxes. ModX proposes a Javascript API for model transformations.

We have defined a meta-model to describe multimodal interactions requirement. We wanted this meta-model very simple: there is no notion about activity, merge, condition, etc. The main notion of this meta-model is the sentence. A sentence is a sequence of interactions and causes an action/reaction of the ambient system. A term is an interaction that refers to what a user wants to transmit (rather than focusing on the device s/he uses). The meta-model contains 3 concepts (see Figure 8): Start, Term and Action. Start and Action are ways to define the beginning and the end of a sentence. Action is also used to indicate the reaction of the system.

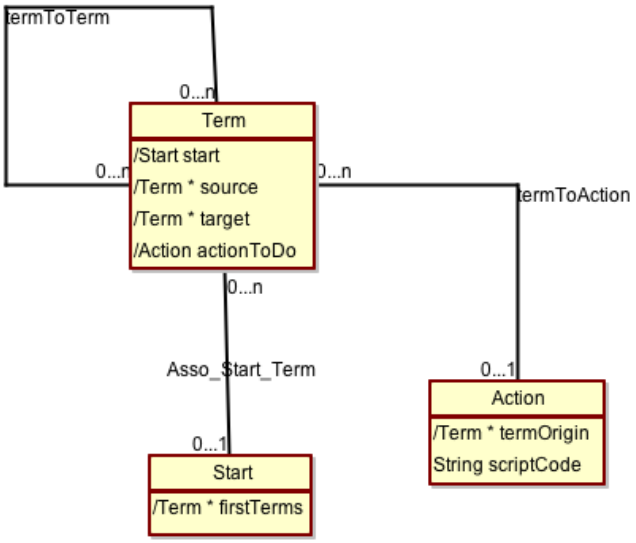


Figure 8. MMI use requirement meta-model.

A Term may be a word or a long expression, and it can be transmitted through different devices. For example, the Term 'Fan' may be indicated through speech recognition, a RFID tag, a QR code, etc. A sentence split into N Terms refers to a sentence with X different interactions.

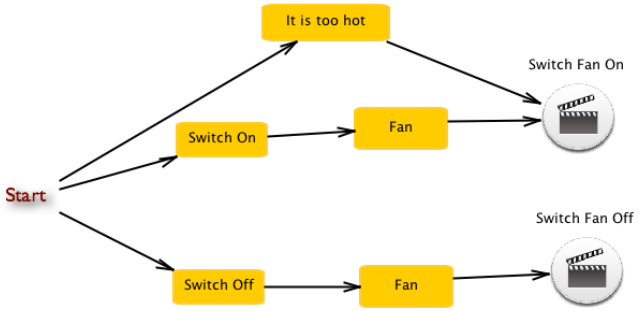


Figure 9. A sample model of multimodal interactions about home automation.

This is illustrated by the model in Figure 9 (Terms and Actions are respectively represented by rounded rectangles and cinema clap in circles).

The sentence "Switch On Fan" (from the Term sequence "Switch On" and "Fan") launches the "switch fan on" action. This sentence refers to two successive interactions and so can use a maximum of two different devices. The same Action is caused by the sentence 'it is too hot', which contains only one Term, so only one interaction that can be performed with one RFID tag or one QR code for the whole expression, for instance. Such a definition also means that "it is too hot" refers to an 'only-one interaction': the previous sentence cannot be constructed by an interaction for "it is too" and another interaction for "hot".

2) Model transformation

To map each MMI use requirement model on Common Knowledge platform, we have defined a set of model transformation rules, implemented as following:

```
1. Create a workflow model

2. Create a starting node

3. Create a taskChoice (STC) connected to the
   previous starting node.

4. For each term (T) connected to the start
     If T is bound to an action (A)
       Create a EndNode (EN)
         EN.caption = A.name
       Associate it with STC,
         association.caption = T.name
     Else
       Create a taskChoice (TC)
         TC.id = T.id
       Associate it with STC
         association.caption = T.name

5. For each term (TA)
     For each its connected term (T)
       If T is bound to an action (A)
         Create a EndNode (EN)
           EN.caption = A.name
         Associate it with TA
           association.caption = T.name
       Else
         Create a taskChoice (TC),
           TC.id = T.id
         Associate it with TA
           association.caption = T.name
```

To summarize these rules, a Term corresponds to a link, i.e., a choice that is done. When a Term is the last of a sequence (and cause an action), an `EndNode` is also created. If the Term points out to other possible choices, a `taskChoice` is created instead.

Object Connections provides a C# API for its workflow engine. It allows creating and editing workflow models. We have implemented a software adapter of this API for our communication bus, called WSE (see below). In this way, the Javascript code (in ModX) corresponding to the previous model transformation, sends WSE messages in order to create elements of workflow model.

B. Implementation of our conceptual architecture

1) Communication bus: WSE

The implementation of a communication bus can be done in several ways, e.g., with the IVY bus as we demonstrated in a previous paper [1] or a multi-agent system [21]. Unfortunately IVY does not work through the web, while using the web is one of our requirements. Therefore we decided to implement our own communication bus, called WSE (Web Server Event).

WSE is the core of our architecture and the central point of traffic. All messages, i.e., user interactions but also actions requested to devices, are carried by WSE (see Figure 11).

WSE is an HTTP-based message bus, like COMET [22]. Such buses are generally dedicated to web pages. Because we focus on interactive devices whose drivers are generally not accessible with JavaScript, we also provide an API in Java and C#. Only a web server supporting PHP scripts, for instance EasyPHP or WAMP (Windows, Apache, MySQL, PHP) is required to install WSE. We choose not to create a standalone WSE server in order to avoid conflict on port 80 with a possible existing web server. Finally we choose to use PHP scripts because of the popularity of this language. Thus, WSE should be installable on most existing / running web servers.

The immediate benefits of this web server-based solution are:

- Multi-OS: if an operating system can access the web, it can use WSE. WSE is therefore compatible with Mac OS, Windows, Android, and Linux.

- Multi-platform: the previous point implies that WSE is running on computers, smartphones, tablets, etc.

- Multi-browser: each operating system has dedicated web browsers. Because we are multi-platform and multi-OS, we are also multi-browser. Thus WSE can be used by Internet Explorer, Firefox, Chrome, Safari, Opera, and so on, as long as they support JavaScript.

- Multi-network: the web access can be done via wired connections, Wi-Fi, 3G. WSE can be used by

all these different modes of connection without restriction. As long as people have access to the web (port 80 is open), they can use WSE. We are therefore not blocked by firewalls. We also tested successfully WSE in our University that offers two different internet accesses, a network dedicated to the staff (teachers, researchers, administration) and a network with a proxy for students.

a) Features of WSE

WSE is multi-languages. Programming a Web application that uses simultaneously a Wiimote [23], a RFID reader and X10 adapters requires handling several programming languages. It can be for instance Java for Wiimote, C# for mir:ror [24] (RFID reader), Javascript for Web application. Currently WSE can be managed with C#, Java, JavaScript, and soon with ActionScript (Flex/Flash) and Python. The only two constraints for languages are to be able to process JSON (JavaScript Object Notation) and support HTTP requests, which can be implemented in any language if necessary.

Installing WSE is very simple. It consists in copying a directory ("Miny/WSE/PutOnWebServer_Root") from the ZIP file available at http://www.lifl.fr/miny, and to place this file in the root of the web server.

WSE provides basically a mechanism for trace. Traces are very interesting for an interactive system, e.g., to do debug, to support the "Undo" command or to analyze user's activities.

Messages routed by WSE are JSON objects. This implies that each message must respect a JSON structure, for instance {"param1":"value1", "param2":"value2", "param3":"value3"}. The advantage is no message format is required. Thus messages like {"action":"open"} or {"whatToDo":"open"} are acceptable. Consequently, each developer can write her/his own message format dedicated to her/his application. For our MINY project, we use the following format: {"action":"…", "actionParams":"…", "object":"…", "objectParams":"…", "location":"…", "locationParams":"…", "fromWhere":"…", "fromWhom":"…"}.

b) Using WSE

To use WSE, simply connect to a session or create one, then send and receive messages. WSE is session-based. All messages within a session are stored in a file, which is named as the session (http://server_url/WSE/traces_files/name). Below is an example in JavaScript and C# for the three steps, connect, send and receive (equivalent code in Java can be found on our web site).

i. Connect to a session

To connect to a session, the user only needs to provide the session name. If the session already exists, WSE connects to it, otherwise the session is automatically created and the connection is established.

JavaScript code:

```
<script LANGUAGE="JavaScript" src="wse.js"/>
…
wse.joinSession("mySession");
```

C# code:

```
using Newtonsoft.Json.Linq;
using Wse;
…
private Wse.Bus myWSEBus;
String serverUrl =
"http://xxx.xxx.xxx.xxx/WSE/traceSession.php";
String sessionName = "mySession";
myWSEBus = new Bus(serverUrl, sessionName);
```

ii. Send a message

To send a message, simply send a JSON object.

JavaScript code:

```
wse.sendMessage
({"action":"switchOn","object":"lamp"});
```

C# code:

```
JObject myMessage = new JObject();
myMessage.Add ({"action":"switchOn", "object":"lamp"});
myWSEBus.SendBusMessage(myMessage);
```

iii. Receive a message

To receive a message it is necessary to declare a listener for messages traveling on the bus. Each time a message is transmitted on the bus, the listener is notified and performs the associated function (Observer pattern). Then the function can extract all the needed information for the application.

JavaScript code:

```
myListener = {};
myListener.newMessageReceive = function
(message)
   { alert("A message has been received: " +
            message);
   };
wse.addListener(myListener);
```

C# code:

```
public class MyListener : IListener
{
   public void NewMessageReceive(string source,
JObject jObject)
   {
      MessageBox.Show("A message has been
received: " + jObject.ToString());
   }
}
…
MyListener myListener = new MyListener();
myWSEBus.AddListener(myListener);
```

2) Device access: Proxy/Stub generator

a) Principles

As explained before, constructing, sending, receiving and detecting WSE messages is a tedious task. For this reason, we have developed a code generator that produces a WSE-based software layer, which handles WSE message operations. With this layer, a programmer uses a remote device as a local device. The following Javascript code shows how to switch on a fan with the devices layer on the example of Section IV.B.2.

```
manager = new Manager("IJAIS2010");

X10 = manager.getX10("328", "Xavier", "Lamp");
// Param 1 : for the office number 328
// Param 2 : around the desk of xavier
// Param 3 : this X10 adapter is dedicated to a
// lamp

X10.switchOn();
```

Here is the code related to RFID events (still in Javascript).

```
rfid = manager.getRFiD("328","all");
// Param 1 : for the office number 328
// Param 2 : for all the office


rfid.layDown = function (stamp) {
     document.body.bgColor = "red";
}
// lay down a RFid tag will set the
// background color of page to red


rfid.pickUp = function (stamp) {
     document.body.bgColor = "green";
}
// pick up a RFid tag will set the
// background color of page to green
```

b) Generator

The code generator produces `userSide.X10` and `userSide.RFiDReader` classes to allow developers to focus on functional/interactive concerns without worrying about remote access. Production of such a class is done from a description of actions (called methods) and events of related devices. The description is JSON formatted and therefore does not imply to use another language.

Here are the two description files corresponding to X10 and et RFID reader devices.

```
{
  name : "X10",
  package : "x10",
  type : "Device",
  constants : {

    object : '"x10"',
    objectParams : null,
    location : null,
    locationParams : null
  },
  methods : {
    switchOn : {},
    switchOff : {}
  }
}
```

```
{
  name : "RFiD",
  package : "rfid",
  type : "Device",
  constants : {
    object : '"RFiDReader"',
    location : null,
    locationParams : null
  },
  events : {
    layDown : {
      stamp : String
    },
    pickUp : {
      stamp : String
    }
  }
}
```

We have defined a generic format inspired from JSON-RPC [25] in order to homogenize the structure of WSE messages that will be exchanged though this devices layer.

This format message protocol is the following:

- action: the expected action (e.g., switchOn) or name of the event (layDown for a RFID reader).
- actionParams: arguments of action or event.
- object: type of device (e.g., X10, RFIDReader).
- objectParams: optional details about the device (for instance X10 has two objectParams: lamp, fan).

Rather than automatic identifiers, we choose to use explicit identifiers, which indicate where the device is.

- location: indicates where the device is (for example: 'Office 328').
- locationParam: details the place in the previous location (e.g., 'Desk of Xavier')

Figure 10 shows a communication between user-side and device-side. Concerning the user-side, the generator produces a proxy class for each description file. For each described method, the proxy (step 2 on Figure 10) contains a corresponding method that consists in creating a WSE message and sending it. If the description defines events, an interface is generated. It contains one method for each event. This interface is associated to the proxy: add/remove listeners methods are added to the proxy while a listener consists in an object implementing the interface. In Javascript, there is no listener interface. The events correspond to methods of the proxy.

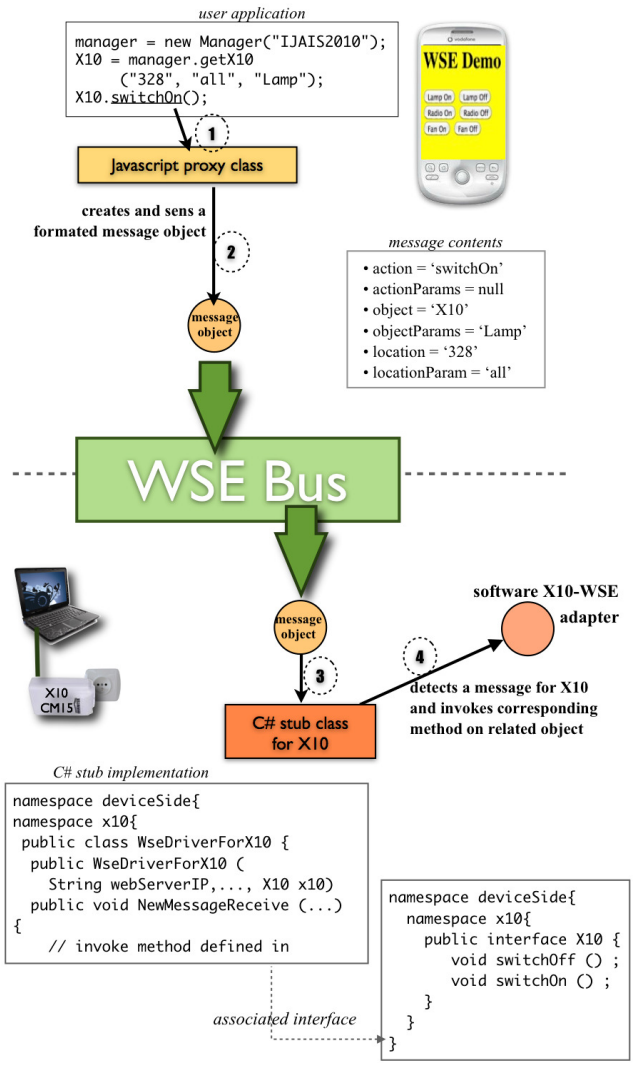


Figure 10. Stub and skeleton on example.

A class `Manager` is also generated and acts as a factory. This class is instantiated with a WSE bus as parameter, and gives access to proxy objects according to location/identification values (step 1 of Figure 10). If a programmer wants to add a new type of device in the device access layer, the code generator can also help her/him by producing code, a stub class, (step 3 of Figure 10) related to WSE stuff. This stub class will have to be connected to another class, a device-WSE adapter (step 4). This one has to interpret a) WSE actions into actions on devices, b) events from device into WSE events. The generation principle is the same as for the user-side but with reversed responsibilities: the skeleton contains a method for each event that device can emit and an associated interface which defines a method for each possible action on the device.

C. Our methodology in a few words

To summarize, here are the major steps to follow to implement our methodology:

1. Identify the devices and the associated actions to use.
2. Define a MMI model to specify the possible interactions that you want to apply through the device actions (cf. Figure 9).
3. Convert this MMI model into a workflow; this step is done automatically in our case.
4. Implement a distributed communications and access to devices. Designers can use the stub/proxy generators (see above), or even can use the already-implemented package we propose for RFID, Androphone, BCI, X10 and IP Camera.
5. Implement the parts that associate interactions to real actions on devices. For instance lay down a specific RFID tag should produce the "It is too hot" interaction.
6. Start the WSE drivers for each device with providing parameters such as IP address, session name, location, etc.
7. Start the workflow engine and the code produced in step 5.

VI. Case Study

This case study section is divided in four parts, which present, respectively, the domain of smart digital home, the architecture of the project, the implementation of this case study and finally, the multimodal aspects of this implementation.

A. Smart digital home

A smart digital home refers to a living space with devices that are connected through wired or wireless networks. The

connected devices may be sensors, actors, consumer electronics, appliances, mobile and PC devices that cooperate transparently for facilitating living and improving usability in the home. Since a variety of devices are present in a smart digital home, convergence and standardization across all the screens of TVs, PCs, appliances and mobile devices, and management of multi-channel interactions is manifestly the key for the success of residential applications.

In our example, several objects are identified in order to be driven remotely: a lamp, a fan, a Rovio robot [26], and a webcam. The possible actions on those objects are the following: move (up, down, left, right, and home) and switch (on/off). As we can see on Figure 2, while the interaction takes place, one of the possible paths of the workflow is followed. Once the final state is reached, a command is sent to the bus.

B. Architecture

For this smart digital home case study, we are using the IVY software bus [18] or our WSE bus, indifferently.

With the IVY bus, a publish/subscribe mechanism is available. Some applications are only subscribers. It means that they need data to prompt information to the user (a synthesized speech for example), to activate appliances (micro-wave oven, washing machine, etc.), or to generate some piece of VoiceXML [3][27] code that will be dynamically generated and used at runtime. Some applications are only sending information to the bus. Others are using the bus to both receive and send data. For instance, the Automatic Speech Recognition (ASR) application usable on a PC needs to receive the different labels corresponding of the speakable words, and oppositely, it sends to the bus the result of the speech recognition engine.

The “Workflow_Engine” application is in charge of the connection with the persistent workflow that we use for this project. It exploits a dedicated API to send the choices of the user to the object connection engine, and to receive the next elements to be presented to the user.

C. Implementation

Our global project was conceived to manage various kinds of devices, sensors, effectors and technologies such as keyboard and mouse, voice over telephone or softphone, QR code, multi-touch screen, Wiimote, Mirror [24] / Reflet [28] NanoZtag RFID, motion webcam, X10 protocol, Rovio robot [26], etc.

Our proposition is based on the architecture illustrated in Figure 11. Three types of elements are present: (1) Interactive components that are detectors and/or effectors, (2) Communication bus for message exchange and (3) Workflow engine. This proposal aims at providing developers the ability to associate to her/his application a multimodal dimension concerning its interactive part. Currently, interactions supported are ruled by only one principle, which is "sentences triggering actions". A sentence consists in a sequence of words that can be triggered by any type of modality (voice, QR code, keyboard/mouse, etc.). To facilitate the writing of such sentences for an application, we use the Task Choice concept [17] in order to factorize words. For example, a sentence may begin by "move" and then be divided into 4 sub-sentences (one for each concerned device). This avoids writing four complete sentences.

An example of path may be the following one: the user activates the button "move" from the Windows application (first sub-action), presents in front of a webcam a QR code identifying the robot (second sub-action) and then pronounces on her/his phone the word "left" (third sub-action). This path is completed and the action "move the robot on the left" is triggered.

Once a model is loaded into the workflow engine, it is executed by the engine that starts with the first task choice. Each time the engine points to a new task choice, the list of possible choices is sent to the bus. This is done by a software agent attached to the workflow engine. Thus, interactive components can subscribe to this type of message, in order, for example, to present the list of choices to the user (as graphical buttons, voice prompt, etc.).

Two other software agents were needed and developed. The first one notifies the workflow engine that a sub-action was performed. This type of agent is attached to an interactive component and translates each relevant interaction into a sub-action that is sent to the bus. The second agent allows to be notified that an action is requested (e.g., switch on fan). Such agent aims to be associated to an interactive component that will translate actions into actual commands on the component, using X10 protocol, for instance.

The three software agents previously mentioned have two roles: to subscribe/transmit on the communication bus and to establish a protocol for discussion between the workflow engine and interactive components. This protocol is based on actions, sub-actions and possible actions. Note that in the model associated to smart digital home, we defined paths so user must first specify the command, then identify the device and finally give a possible parameter for command.

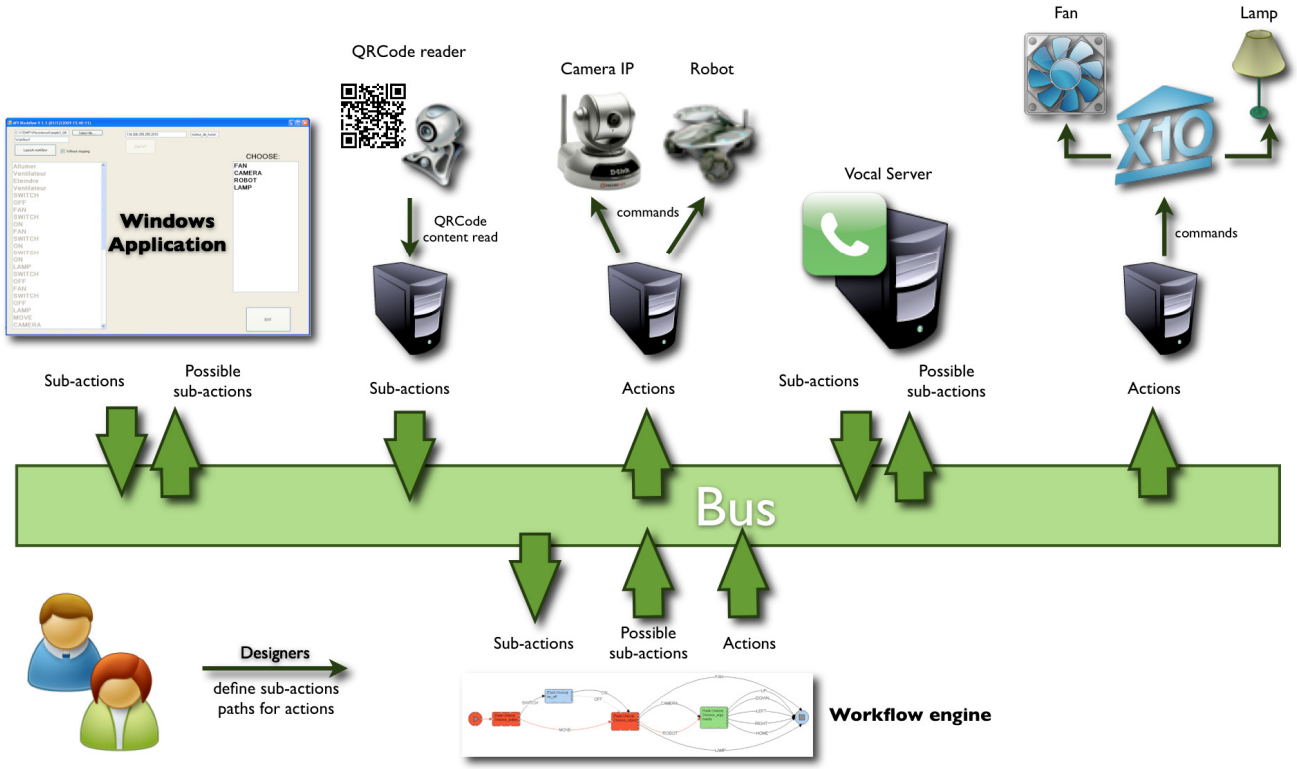


Figure 11. Architecture of our Smart Digital Home project.

The three software agents used the workflow presented in Figure 2 to describe the objects and actions that can be applied on those objects using one or more devices.

D. Multimodality

As previously mentioned, our goal is to provide tools in order to facilitate the design and implementation of multimodal interfaces for ambient computing. Concerning vocal interactions, one big challenge is to provide the designers an easy and robust way to generate code (like VoiceXML [29] for instance) that can integrate grammars related to a particular changing context. Dynamic voice grammars (or entire VoiceXML files) can be generated with our approach, as we can see in Figure 12.

If the designer decides to add a possible new direction, s/he can do it graphically, on the workflow, by adding an arc (called “home” for example), near the up/down/left/right already available. Then with no addition of code, a new possible interaction is available through the workflow. Consequently, one can then pronounce a sentence like “move camera home”, in order to physically make the webcam move.

```
<?xml version="1.0" encoding="UTF-8"?>
<vxml xmlns="http://www.w3.org/2001/vxml"
version="2.0" xml:lang="en-gb">
<form>
<grammar version="1.0" root="GR_VOICE"
mode="voice" tag-format="semantics/1.0">
<rule id="GR_VOICE">
<one-of>
<item>up<tag>out.choice="up";</tag></item>
<item>down<tag>out.choice="down";</tag></item>
<item>left<tag>out.choice="left";</tag></item>
<item>right<tag>out.choice="right";</tag></item>
<item>home<tag>out.choice="home";</tag></item>
</one-of>
</rule>
</grammar>
<field name="choice"><prompt>
Choose among up, down, left, right, home
</prompt>
<filled>
<prompt bargein="false">
The chosen value is: <value expr="choice"/>
</prompt>
</filled>
</field>
</form>
</vxml>
```

Figure 12. Example of VoiceXML code generated by the VoiceXML_Maker agent.

For this case study, we have implemented a multi-device, multimodal, and multi-channel system:

- a Multi-device system because more than one device can be used during the interaction. In our experiments we used many PCs, smartphones and telephones, and a Wii Console.

- a Multi-modal system because more than one modality can be used during the interaction. In our examples we used traditional keyboard/mouse interactions, vocal, gesture and brain computer interaction (BCI). We also used QR codes and RFID tags containing data related to desired actions or objects.

- a Multi-channel system because more than one channel can be use during the interaction. In our smart home case study, it was done across internet and telephone networks.

VII. Conclusion and Future Work

The goal of this paper was to describe how we can facilitate the design of multi-channel and multi-modal interfaces for ambient computing with a model-driven approach. We used a smart digital home case study to explain how to design easily an ambient system using a workflow oriented approach.

Our results show that different devices (such as Wiimote, multi-touch screen, telephone, etc.) can be managed in order to activate real or virtual things. Adding new features (such as appliances, actions, direction, etc.) to an existent system is also very easy and only needs a modification of the workflow.

Our work is orientated toward the production of code generated from model (and meta-model) transformations, and shows that this model-driven approach is encouraging and suitable for the ambient computing domain. With our methodology, a large part of the scripts and applications programs, traditionally coded by developers, can be automatically generated by the ambient system itself.

In the future, this should improve the possibility to detect new objects, persons or possible behaviors dynamically and to respond to them as soon as possible with relevant feature of the ambient system. Thus, it will be challenging to work on the possibility to manage simultaneously different natural languages with a unique model of existing actions.

We will also work on the important point of semantic aspect of the workflow. This will help users for instance when they will not use the commands in the right order. Indeed, a smart system must be able to understand that “move up robot” is the same command as “move robot up”. We are also planning to offer the possibility to dynamically switch from a software bus to another and to manage virtual representation of tangible things (fridge, oven, etc.) in order to allow realistic simulations before real implementation.

VIII. Acknowledgement

The authors would like to thank ObjectConnections, Jaxo Sytem and bcWebCam for providing special tools: Common Knowledge, Cam'A'Bar and bcwebcam.

IX. References

[1] Rouillard, J., Tarby, J.C., Le Pallec, X., and Marvie, R., “Facilitating the Design of Multi-channel Interfaces for Ambient Computing”, The Third International Conferences on Advances in Computer-Human Interactions, ACHI 2010, St. Maarten, Netherlands Antilles, 2010, pp. 95-100.

[2] W3C Multimodal Interaction Activity (MMI), Retrieved January 10, 2011, from http://www.w3.org/2002/mmi/

[3] VoiceXML 2.0., W3C Recommendation (16/03/04), Retrieved January 10, 2011, from http://www.w3.org/TR/voicexml20

[4] Harel, D., “Statecharts: a visual formalism for complex systems”, Science of Computer Programming, Volume 8, Issue 3, pp. 231-274, 1987.

[5] OpenInterface European project. IST Framework 6 STREP funded by the European, Commission (FP6-35182). Retrieved January 10, 2011, from http://www.openinterface.org and http://www.oi-project.org.

[6] Tarby, J.C. and Rouillard, J., “Assistance, advice and guidance with digital coaching”, EAM'06 European Annual Conference on Human Decision-Making and Manual Control Lecture Notes in Computer Science, Springer-Verlag, Berlin, 2006, Valenciennes.

[7] Frohlich, D., “The design space of interfaces, multimedia systems, Interaction and Applications”, 1st Eurographics workshop, Stockholm, Sweden, Springer Verlag, p. 53-69, 1991.

[8] Healey, J., Hosn, R., and Maes, S.H, “Adaptive Content for Device Independent Multi-modal Browser Applications”, Lecture Notes In Computer Science; Vol. 2347, Proceedings of the Second International Conference on Adaptive Hypermedia

and Adaptive Web-Based Systems, pp. 401-405, ISBN: 3-540-43737-1, 2002.

[9] Coutaz, J., Nigay, L., Salber, D., Blandford, A., May, J., and Young, R. M., "Four easy pieces for assessing the usability of multimodal interaction: the CARE properties". In INTERACT, pages 115-120. Chapman & Hall, 1995.

[10] Vanderdonckt, J., Grolaux, D., Van Roy, P., Limbourg, Q., Macq, B., and Michel, B., "A Design Space for Context-Sensitive User Interfaces", Proc. of ISCA 14th Int. Conf. on Intelligent and Adaptive Systems and Software Engineering IASSE'2005 (Toronto, 20-22 July 2005), International Society for Computers and their Applications, Toronto, 2005, pp. 207-214.

[11] Rouillard, J., "Multimodal and Multichannel issues in pervasive and ubiquitous computing", Multimodality in Mobile Computing and Mobile Devices: Methods for Adaptable Usability, Idea Group. Inc, Information Science Reference, ISBN: 978-1-60566-978-6, 409 pages, 2009.

[12] Bastien, Ch. and Scapin, D., "Ergonomic Criteria for the Evaluation of Human-Computer Interfaces", J. M., INRIA Technical report N° 156, 1993.

[13] Bourguin, G., Lewandowski, A., and Tarby J-C., "Defining Task Oriented Components, Task Models and Diagrams for User Interface Design", 6th International Workshop, TAMODIA 2007, Toulouse, France, November 7-9, 2007, Marco Winckler, Hilary Johnson, Philippe A. Palanque (Eds.), Lecture Notes in Computer Science 4849 Springer 2007, ISBN 978-3-540-77221-7, pp. 170-183

[14] Tarby, J.C., "One Goal, Many Tasks, Many Devices: From Abstract User Task Specification to User Interfaces" (Chapter 26). In, Diaper, D. and Stanton, N. The handbook of Task Analysis for Human-Computer Interaction. (pp.531-550). Mahwah, New Jersey: Lawrence Erlbaum Associates, 2004.

[15] Palanque P., Bernhaupt, R., Navarre, D., Ould, M., and Winckler, M., "Supporting Usability Evaluation of Multimodal Man-Machine Interfaces for Space Ground Segment Applications Using Petri net Based Formal Specification". In International Conference on Space Operations (SpaceOps 2006), Rome, Italy, 18/06/06-22/06/06, American Institute of Aeronautics and Astronautics (AIAA), 2006.

[16] Horrocks, I., Constructing the User Interface with Statecharts, Addison-Wesley Professional, 272 pages, 1999.

[17] ObjectConnections, Common Knowledge Studio and engine, provided by ObjectConnections. Retrieved January 10, 2011, from http://www.objectconnections.com

[18] IVY Bus, Retrieved January 10, 2011, from http://www2.tls.cena.fr/products/ivy/

[19] ModX MOF modeling tool, Retrieved January 10, 2011, from http://edutechwiki.unige.ch/en/ModX

[20] MOF OMG Meta-Object Facility, Retrieved January 10, 2011, from http://www.omg.org/mof/

[21] Kubera, Y., Mathieu, P. and Picault, S., "Everything can be Agent!", Proc. of 9th Int. Conf. on Autonomous Agents and Multiagent Systems (AAMAS 2010), van der Hoek, Kaminka, Lespérance, Luck and Sen (eds.), Toronto, Canada, pp.1547-1548, 2010.

[22] COMET, Retrieved January 10, 2011, from http://en.wikipedia.org/wiki/Comet_(programming)

[23] Nintendo, Wii game console and Wiimote controller, Retrieved January 10, 2011, from http://www.nintendo.fr/

[24] Nabaztag, Mir:ror, Nano:ztag, and Ztamp:s, from Violet, Retrieved January 10, 2011, from http://www.violet.net/index_en.html

[25] JSON-RPC: lightweight remote procedure call protocol, Retrieved January 10, 2011, from http://json-rpc.org/

[26] WowWee Group Limited, Rovio robot, Retrieved January 10, 2011, from http://www.omg.org/mof/http://www.wowwee.com/en/support/rovio

[27] VoiceXML 2.1, Recommendation, (19/06/07), Retrieved January 10, 2011, from http://www.w3.org/TR/voicexml21/

[28] Ref:let, An open-source alternative to mir:ror from Violet, under Windows, Retrieved January 10, 2011, from http://code.google.com/p/reflet-mirror/

[29] VoiceXML 3.0, W3C Working Draft (08/08/2008), Retrieved January 10, 2011, from http://www.w3.org/TR/vxml30reqs/

Dynamic Resource Management in Virtualized Environments through Virtual Server Relocation

Gastón Keller and Hanan Lutfiyya
Department of Computer Science
The University of Western Ontario
London, Canada
{*gkeller2,hanan*}*@csd.uwo.ca*

***Abstract*—Virtualization has become an essential technology in the data center. Virtualization improves resource utilization through server consolidation, but it also makes resource management more complex. Golondrina, an autonomic resource management system, was built to use virtual server relocation to handle resource stress situations, that is, situations where the combined resource needs of the virtual servers hosted in a physical machine exceed the resource availability. Experimental evaluation shows that replication offers improvements over migration, and both mechanisms offer improvements over taking no action upon detection of a CPU stress situation. The main contribution of this work is the introduction of virtual server replication as an alternative to migration and the experimental comparison of both mechanisms.**

***Keywords*-virtualization; resource management; migration; replication; autonomic computing**

I. Introduction

A data center is defined as a collection of computing resources shared by multiple applications concurrently in return for payment by the application providers, on a per-usage basis, to the data center provider [2]. To guarantee that an application will always be able to cope with all demand levels the application is statically allocated enough resources so that peak demand is satisfied. The unit of allocation is typically a physical server. This often results in resources being underutilized.

One approach to increasing resource utilization is *server consolidation*, which consists of hosting multiple application servers in one physical server. This approach is possible through *virtualization*. Virtualization refers to an abstract layer between the operating system and the hardware. The layer provides an interface to the actual hardware that allows for the support of a number of *virtual machines*. In a data center a virtual machine would have a server application installed on it. We will use the term *virtual server* to refer to a virtual machine that runs an application server.

Virtualization reduces the unit of resource allocation to fractions of a physical server. This potentially benefits data centers by allowing several applications to make use of the same physical server. If the virtual servers are placed on a physical server based on peak demand, then the physical server can still be highly underutilized. On the other hand, if the virtual servers are placed on a physical server based on the average demand, then this may result in virtual servers competing for the same resources when demand increases. The reason is that demand for an application may increase such that it needs computing resources currently being used by other applications on the same physical server.

The time-varying demand that application servers may experience in a data center [3] suggests that resource allocation should be done dynamically. Dynamic resource management requires monitoring mechanisms and dynamic resource reallocation mechanisms. Golondrina, an autonomic resource management system, was developed with resource utilization sensors for monitoring and *virtual server relocation* mechanisms. Two examples of the latter are *migration* and *replication*.

Migration consists of transferring a running virtual server from one physical server to another. Replication entails the creation of a replica of a virtual server on another physical server. Requests for the virtual server are balanced between the two instances. This should reduce the computing resources needed by a single physical server by distributing requests to two different virtual server instances hosted in two different physical servers. A replica in this work is not an actual copy of the virtual server running at the time, but an instantiation of an image of the virtual server.

The primary focus of this work is to study the use of virtual server relocation to deal with *resource stress situations*, that is, situations where the combined resource needs of the virtual servers hosted in a physical server exceed the resource availability. One reason for this study arises from a challenge in dynamic resource management where it is often difficult to determine the appropriate action in response to a resource stress situation. The goal of our study is to determine effective strategies in the use of different virtual server relocations for effective management of computing resources.

The rest of this paper is organized as follows: Section II provides background on the virtualization software used, Section III describes the resource management system, Section IV presents the experiments, Section V discusses the experimental results, Section VI describes related work, and

Section VII provides a conclusion.

II. Background

This work uses OpenVZ which provides operating system-level virtualization [4], [5]. OpenVZ is essentially a Linux kernel modified to run multiple, isolated *containers* (i.e., virtual user-space environments) on a single physical server or *hardware node*. OpenVZ supports the execution of multiple containers. The containers are isolated program execution environments, which appear as stand-alone servers to users. Each container has its own set of processes including the **init** process, file system, users (including **root**), applications, memory, network interfaces with IP addresses, routing tables, and firewall rules. Information on resource utilization for the hardware node and each container can be retrieved by reading the accounting files in the *proc* filesystem. The host system runs inside a privileged container.

OpenVZ provides container checkpointing, which is the ability to suspend an executing container, save its state to a file and restart it again later. Container live migration is a natural extension of checkpointing. This process consists of two phases. First, the container's file system is copied to the target machine while the container is still running. In the second phase, the container is checkpointed, its state file is transferred to the target machine and a second copy of the container's file system is started. This second copy is incremental, in the sense that it only affects those files that were modified after the first copy. When the second copy finishes, the container is restored from the state file at the target machine. All the network connections are migrated with the container, so the user perceives no downtime, but does perceive a delay in processing.

OpenVZ does not provide container replication, but it can be implemented. The first step is to stop the container that is to be replicated. The second step is to copy the container's file system and configuration file to the target machine. (Once the file system is copied, the original container can be restarted.) The third step is to modify the replica's configuration file with its own information (identifier, IP address, etc). After this last step, the replica can be started. Our current implementation of the replication process avoids stopping the container to replicate by using a stored image of it.

III. Management System

Golondrina was conceived as a multi-resource management system for data centers. This first prototype, however, works with the CPU as its only managed resource. For that reason, we will use the term *CPU stress situation* instead of the more general term *resource stress situation*, as defined in Section I. Similar procedure will be followed with related terms.

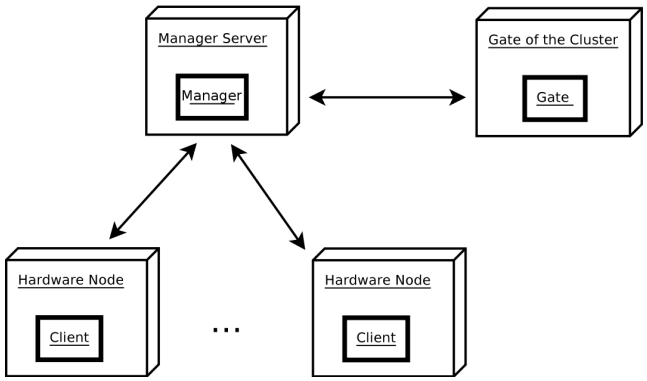


Figure 1. Golondrina's architecture

Golondrina consists of three primary management entities: *Client*, *Manager* and *Gate* (see Figure 1) which are described in this section.

A. Client

Each hardware node has a Client instance that runs in the priviledged container. The Client requires access to the operating system's configuration and accounting files, and the OpenVZ management tools. The Client instance provides the following functionality:

1) **The periodic collection of CPU utilization statistics from the containers and the hardware node.** This is done by reading the hardware node's operating system's accounting files through the *proc* file system. These statistics are sent to the Manager;
2) **Support for migration and replication of containers.** This is done upon a request from the Manager. Migration is provided by one of the OpenVZ Management tools. However, since OpenVZ does not provide a replication feature, the Client has to follow a sequence of steps to trigger a replication. For the container to be replicated requires the following:
   a) An identifier is generated for the replica;
   b) An image of the container is retrieved from a central repository and placed on the target hardware node;
   c) The appropriate configuration files on the hardware node are edited.

B. Manager

Upon receiving the CPU utilization statistics sent by the Client, the Manager stores the statistics. The data model used assumes that the hardware node is an aggregation of containers. The attributes of the hardware node and containers represent CPU utilization metrics. This information is analyzed (III-B1) to execute CPU stress checks (III-B2) and relocation searches (III-B3).

1) Analyzing CPU Utilization Statistics: The CPU utilization statistics sent by the Clients are used to create a *CPU utilization profile* over time. A mathematical model uses the CPU utilization statistics collected at time t_i for predicting the CPU utilization of a container or hardware node at time t_{i+1}.

The mathematical model used is the Auto-regressive Model of Order 1 AR(1) [6], which relies on the last observation in the sequence of observations and two parameters: μ, the mean of the values in the sequence, and θ, which accounts for the variations of the values in the sequence.

Given a sliding window of CPU utilization statistics $W = [u_x, ..., u_t]$ with maximum size w and where $x = max(0, t - w + 1)$. The parameters μ and θ at time t are calculated as follows:

$$\mu_t = \frac{\sum_{i=x}^{t} u_i}{t - x + 1} \quad (1)$$

$$\theta_t = \frac{\sum_{i=x}^{t-1} (u_i - \mu_t) * (u_{i+1} - \mu_t)}{max(1, \sum_{i=x}^{t-1} (u_i - \mu_t)^2)} \quad (2)$$

Having the values u_t, μ_t and θ_t, it is possible to predict the CPU utilization at time $t + 1$:

$$\hat{u}_{t+1} = \mu_t + \theta_t * (u_t - \mu_t) \quad (3)$$

The profiling process uses a historical policy to calculate the container's profiled CPU utilization. This value has to satisfy a given percentile of the container's CPU needs registered in the last window of time $W = [u_x, ..., u_t]$, in addition to the container's current CPU needs. The process sorts in increasing order the collected statistics in W and takes the value corresponding to the 90^{th} percentile. The profiled CPU utilization at time $t+1$ is the maximum of the 90^{th} percentile and the container's predicted CPU utilization plus an additional Δ:

$$\bar{u}_{t+1} = max(90^{th} percentile, \hat{u}_{t+1} + \Delta) \quad (4)$$

2) CPU Stress Detection Mechanism: The Manager executes a CPU stress check on every hardware node that is not currently involved in a relocation. The hardware nodes already involved in relocations (be it as source or target) are likely to change their CPU utilization soon, so they are deemed unstable until all current relocations are completed.

The CPU stress check consists of two steps. First, it verifies whether the predicted CPU utilization of the hardware node exceeds the CPU utilization threshold. Then, if the latter is true, it checks whether *k* out of the previous *n* CPU stress checks also exceeded the threshold, in which case the hardware node is considered to be under a CPU stress situation.

$$(\hat{u}_{t+1} > threshold) \wedge (\sum_{i=1}^{n} (\hat{u}_i > threshold) \geq k) \quad (5)$$

3) Relocation Search: The problem of finding a sequence of relocations is complex. As it is noted in [7], the problem is similar to the NP-hard problem N-dimensional bin packing, but with the additional restriction that the bins are loaded right from the beginning. For this reason, the relocation search uses a greedy strategy to solve the problem.

After the CPU stress check round is completed the hardware nodes are classified as stressed or non-stressed hardware nodes. If both sets are non-empty, then the Manager searches for a sequence of relocations to solve the CPU stress situations. This decision making process consists of determining which containers hosted in stressed hardware nodes will be relocated and which non-stressed hardware nodes will serve as a target for those relocations. The input to the algorithm (shown in Algorithm 1) includes information about stressed hardware nodes (denoted by SH) and non-stressed nodes (denoted by NSH). SH and NSH are assumed to be sorted in descending order of CPU load and ascending order of CPU load respectively. A container is chosen to be relocated (line 3). Currently the implementation starts with the containers using the most CPU cycles. The next step (line 4) is to determine a target hardware node that the container is either replicated or migrated to. The migration or replication is then carried out (line 5).

```
1: for i = 0 to SH.length() do
2:   while SH[i] is stressed do
3:     CT = pickMostHeavilyLoadedCT(SH[i]);
4:     targetHN = pickMostLightlyLoadedHN(NSH);
5:     ExecuteAction(CT,targetHN);
6:   end while
7: end for
```

Algorithm 1: Relocation search

Essentially the policy encapsulated by the algorithm is that for each stressed hardware node the containers using the most CPU cycles should be the first ones to be considered for relocation. Targets of relocation should be as lightly loaded as possible. A check of a potential target hardware node is done to determine if the potential target hardware node is able to accommodate the container to be relocated. For replication, an additional check is done to determine if the target hardware node already has a copy of the container to be replicated.

C. Gate

The Gate component runs in a non-virtualized physical server, which is used as the *gate of the cluster* (i.e., all

service requests come through this physical server). Its responsibility is to update the load balancer's configuration after a replication occurs.

IV. Experiments

The objective of the experiments was to study how the system reacted to CPU stress situations using the container relocation mechanisms.

In order to cause CPU stress situations, load had to be generated for the containers. For this purpose, HTTP requests were sent to web servers running inside the containers. The HTTP requests involved dynamic content so as to increase CPU utilization. With every request, a PHP file was executed to process a two MBytes text file [8], counting the number of words in the file. The execution returned a HTML file with the result of the process. The web servers were Apache [9] instances and the HTTP requests were generated using httperf [10] (running on physical servers in the cluster that were not part of the managed system).

The frequency with which requests were sent determined the *weight* of the generated load. The weight of the load was the percentage of CPU cycles required from one CPU core to handle that load. For example, sending 1 request per second (1 req/sec) resulted in a CPU core being used at 70% capacity.

The metrics used to evaluate the system included lost requests and response time of the web servers. The requests were classified into three categories: lost, failed and successful. Lost requests were those not processed before a client timeout (**client-timo**). Failed requests were those where the server refused a connection (**connrefused**), sent a RESET (**connreset**) or replied with a Server Error 5xx status code (**reply-status-5xx**). A web server's effectiveness was defined as the ratio of the number of successful requests to the total generated requests.

The web servers' response time was measured as the average (**avg**) duration of the established connections (when sending a request, a connection is established between the client and the server, and once the reply is received, the connection is terminated), measured in milliseconds.

The infrastructure on which Golondrina ran consisted of a cluster where one physical server was the *gate of the cluster*, another physical server was a *manager server* and the rest of the physical servers were *OpenVZ hardware nodes*. Each physical server was an Intel Pentium D 3.40GHz (dual-core) with two GBytes of RAM. The containers were built with the default resource allocation provided by OpenVZ.

A. Experiments Design and Configurations

Three different experiments were designed to evaluate Golondrina. Each experiment used the same number of hardware nodes, but the number of containers and the weight of the loads varied.

Each experiment was run three times. For the first run, Golondrina was configured to monitor the CPU utilization and check for CPU stress situations. The relocation mechanisms were disabled. This run provided a baseline, enabling observations on how the environment performed without Golondrina taking corrective actions.

In the second and third runs, Golondrina was configured to use replications and migrations, respectively. The results of these runs were compared with each other and against the baseline.

Golondrina's CPU stress detection mechanism was configured in all cases with a CPU utilization threshold of 0.75. Given that the physical servers possessed two cores, the threshold was equivalent to 150% CPU capacity. (The total CPU capacity of a physical server with x cores is equal to $x{*}100\%$. In this case, the total CPU capacity of the hardware nodes was 200%.) The mechanism was also configured to trigger CPU stress checks every 10 seconds (same frequency with which the monitoring mechanisms were configured).

The HTTP requests sent to the web servers had an associated timeout of 10 seconds and the time span between the start of two different loads during an experiment was 60 seconds.

1) Experiment 1: The managed system consists of two hardware nodes, **bravo02** and **bravo03**, and two containers, **A** and **B**, hosted in **bravo02**. At a given point in time, container **A** receives a load of around 70% (450 requests at a rate of 1 req/sec). After 60 seconds, container **B** receives a load of around 105% (450 requests at a rate of 1.5 req/sec). At this point in time, the hardware node **bravo02** experiences a load of around 175%, which exceeds the CPU utilization threshold of 150%. Thus, **bravo02** is under a CPU stress situation.

In this scenario, no request should be lost since the CPU needs of both containers can be satisfied. However, Golondrina will determine that **bravo02** is under a CPU stress situation and will try to address this situation through a relocation. Golondrina should try first to relocate to **bravo03** the container with the highest load, that is, **B**.

2) Experiment 2: The second experiment is similar to the previous one with the exception that both containers receive a load of around 105% (450 requests at a rate of 1.5 req/sec) each. Consequently, the hardware node **bravo02** becomes CPU stressed with a load of 200% (total CPU capacity).

In this scenario, the web servers running in **A** and **B** will lose requests, due to the lack of spare CPU cycles to allocate to the containers. Golondrina will detect the CPU stress situation experienced by **bravo02** and will respond by triggering a relocation for the container with the highest load (in this case, both containers are candidates). Two replications or one migrations should be enough to terminate the CPU stress situation.

3) Experiment 3: In this experiment, the managed system consists of two hardware nodes, **bravo02** and **bravo03**, and

four containers, **A**, **B**, **C** and **D**, hosted in **bravo02**. One after another, with a 60-second separation in time, the containers receive a load of around 51% (300 requests at a rate of 0.72 req/sec). Thus, **bravo02** experiences a CPU stress situation with a load of 200%.

In this scenario, the web servers hosted in the four containers will lose requests. Golondrina will detect the CPU stress situation and will search for relocations in order to dissipate it. Three replications or two migrations should be enough to terminate the CPU stress situation.

B. Results

This subsection presents the results of the experiments described in Subsection IV-A.

1) Experiment 1: In the first run of the experiment, Golondrina was monitoring the CPU utilization of the hardware node and containers, but no action was taken in response to CPU stress situations. Figure 2 shows the CPU utilization of containers **A** and **B**, and the CPU utilization and predicted CPU utilization (as explained in Subsection III-B1) of the hardware node **bravo02**.

The first time the CPU utilization of **bravo02** went over the 150% threshold was at $t = 11$. Golondrina's CPU stress detection mechanism signaled the problem at $t = 15$. Since no action was taken, the CPU stress situation persisted and was signaled every single time until $t = 39$ (included).

Since there were enough CPU cycles to satisfy the demand of the containers, no request was lost or failed.

The web server **one.com**, hosted in **A**, had an average connection time of 701.9 milliseconds. The web server **two.com**, hosted in **B**, had an average connection time of 904.5 milliseconds.

In the second run of the experiment, Golondrina was to search for feasible replications if a CPU stress situation was detected. Figure 3 shows the CPU utilization of containers **A** and **B**, and the CPU utilization and predicted CPU utilization of the hardware node **bravo02**. Figure 4 shows the CPU utilization of the replicas **A'** and **B'**, and the CPU utilization and predicted CPU utilization of **bravo03**.

The first CPU stress situation in **bravo02** was signaled at $t = 15$. At that time, Golondrina determined that container **B** had to be replicated in **bravo03**. By $t = 16$ the replica **B'** had been created and the load balancer at the *gate of the cluster* was updated. At $t = 17$, **B'** had CPU load, but then it did not process any request for three consecutive periods. As a consequence of container **B'** not receiving any load, the CPU stress situation persisted in **bravo02** and a second CPU stress situation was signaled at $t = 19$. This time container **A** was replicated in **bravo03**. It could be said then that the creation of container **A'** took place due to an improper balancing of the load for the web server **two.com**, hosted in **B** and **B'**.

At $t = 17$ and $t = 22$ it can be seen in Figure 3 and Figure 4 that the curves sloped down. During the periods $(16, 17)$ and $(21, 22)$ the load balancer was being updated, that required it to be restarted. As a consequence, some connections were refused or reset, and hence there was a slight decrease in the reported load.

Table I
EXPERIMENT 1 - PERCENTAGE OF SUCCESSFUL REQUESTS

Web Servers' Effectiveness			
Servers	Run 1	Run 2	Run 3
one.com	100%	99.11%	100%
two.com	100%	98.44%	100%

The web server **one.com**, hosted in **A** and **A'**, had 4 failed requests out of 450 (connrefused 3 connreset 1), which resulted in an effectiveness of 99.11%. The web server **two.com**, hosted in **B** and **B'**, had 7 failed requests out of 450 (connrefused 5 connreset 2), which resulted in an effectiveness of 98.44%.

The web server **one.com** had an average connection time of 749.1 milliseconds. The web server **two.com** had an average connection time of 853.0 milliseconds.

In the third run of the experiment, Golondrina was to look for migrations upon detection of a CPU stress situation. Figure 5 shows the CPU utilization of containers **A** and **B**, and the CPU utilization and predicted CPU utilization of the hardware node **bravo02**. Figure 6 shows the CPU utilization of **B**, and the CPU utilization and predicted CPU utilization of **bravo03**.

A CPU stress situation was signaled at $t = 15$ in **bravo02**. Golondrina determined that container **B** was to be migrated to **bravo03**. At that point, the CPU utilization of both hardware nodes increased, due to the start of the migration process. Since there were spare CPU cycles in both hardware nodes, the containers saw their CPU needs unaffected.

In the period $(26, 27)$, the migration process was completed, but it was not until $t = 28$ that a CPU utilization report from container **B** was sent to the Manager component (running in the *manager server*) by the Client component running in **bravo03**. That report showed a peak of around 140% in CPU utilization, which could be attributed to the hosted web server processing the requests that could not be handled during the suspension period of the migration process.

None of the web servers hosted in **A** and **B** had lost or failed requests. That means that the web servers had an effectiveness of 100%.

The web server **one.com**, hosted in **A**, had an average connection time of 715.2 milliseconds. The web server **two.com**, hosted in **B**, had an average connection time of 991.1 milliseconds.

From the web servers' effectiveness results (Table I), it could be concluded that the replication mechanism is not a convenient tool since requests were lost. However, that loss

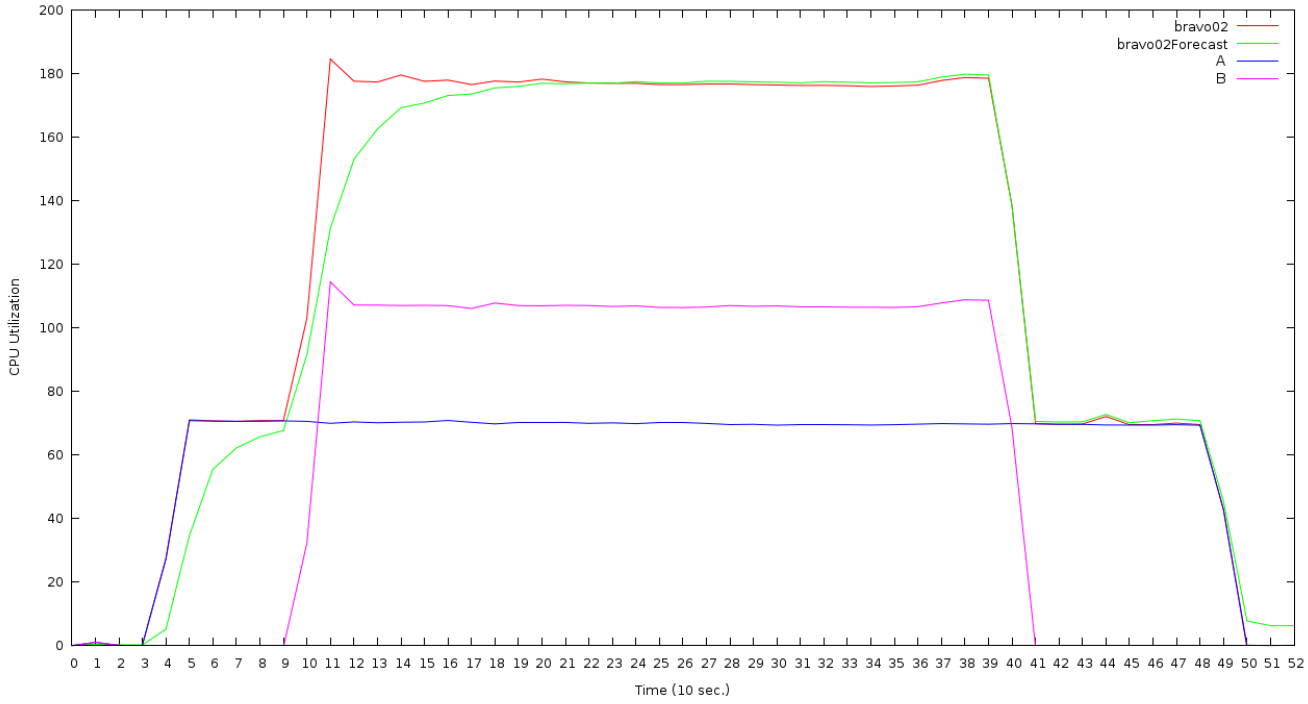


Figure 2. Experiment 1 - No Action

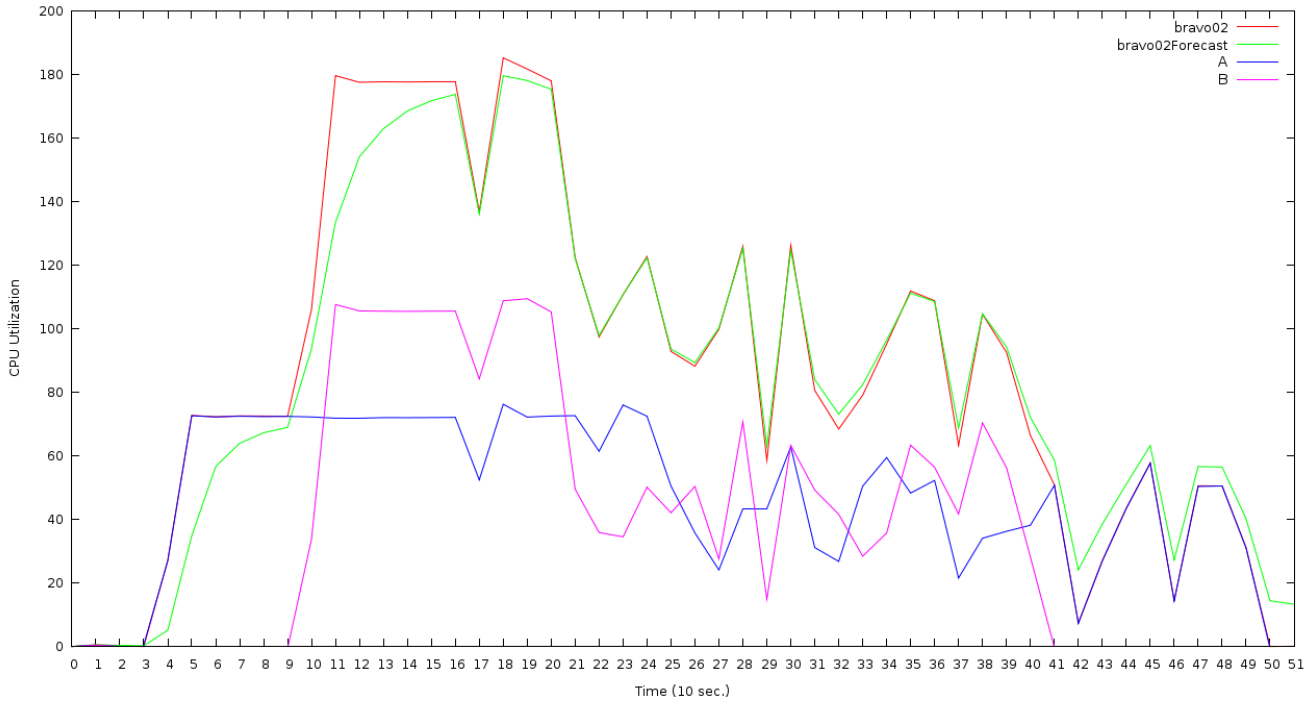


Figure 3. Experiment 1 - Replication, bravo02

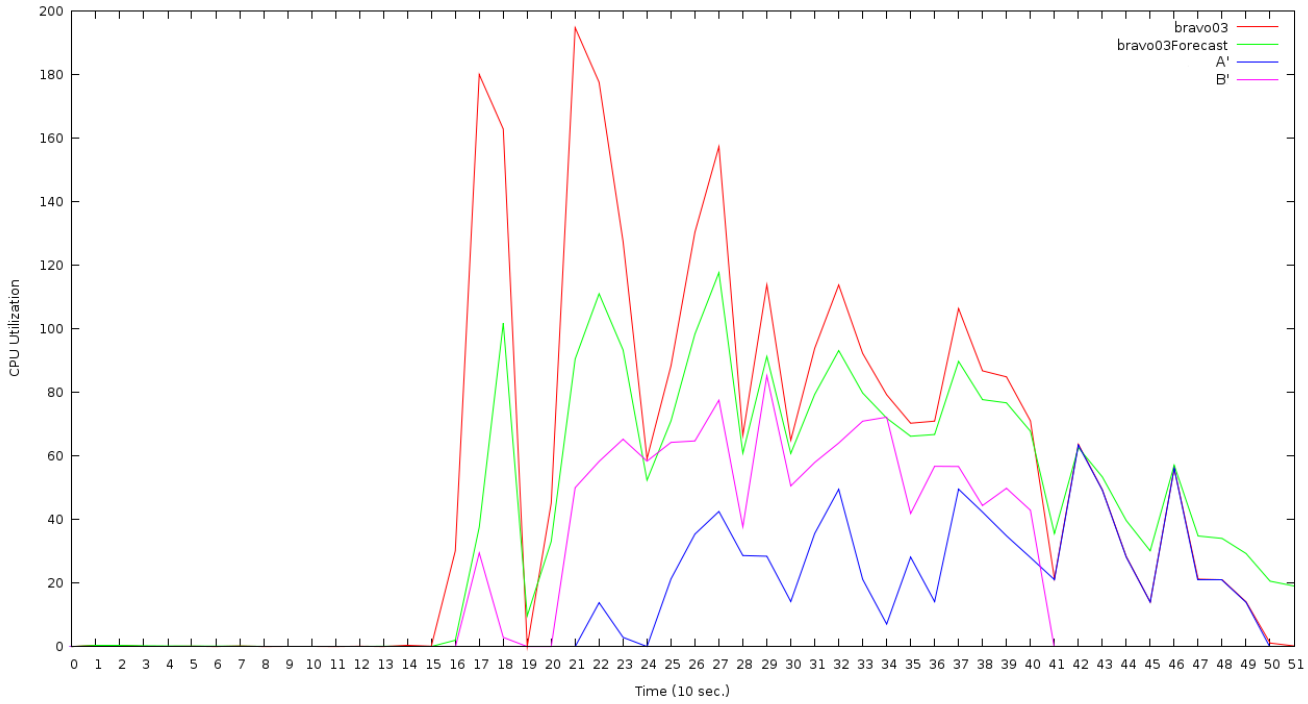


Figure 4. Experiment 1 - Replication, bravo03

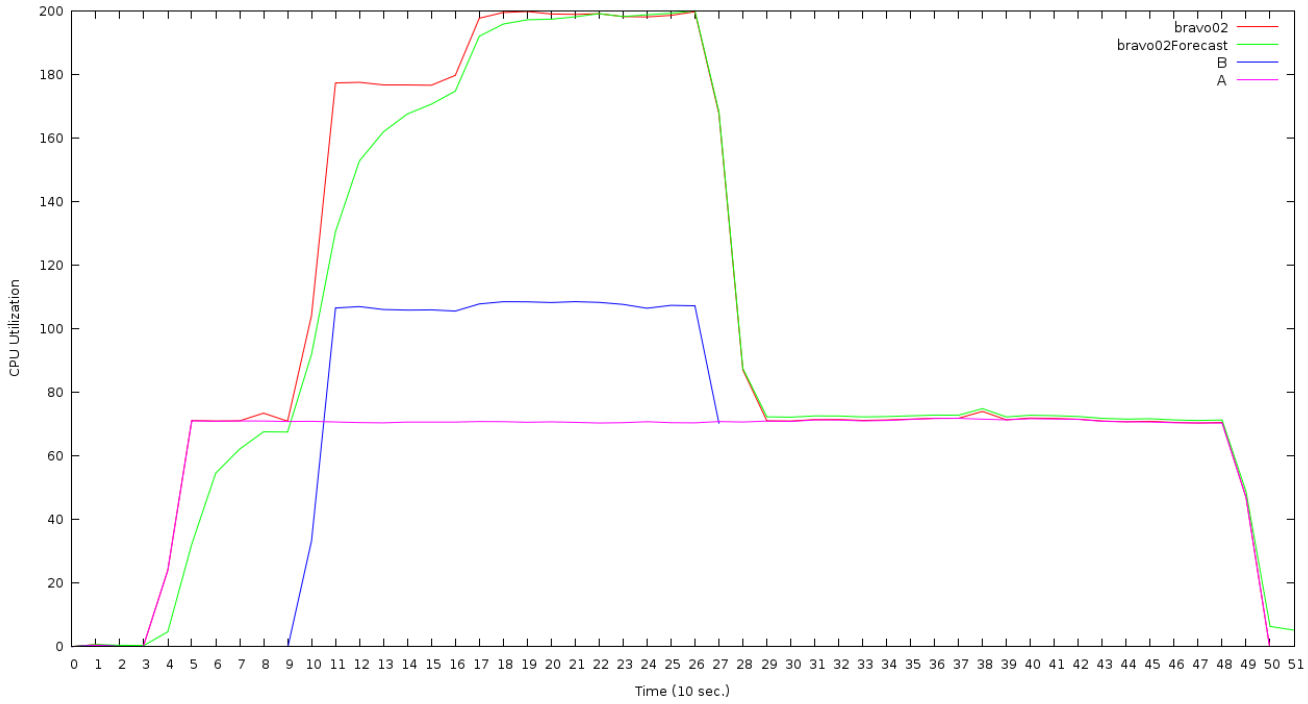


Figure 5. Experiment 1 - Migration, bravo02

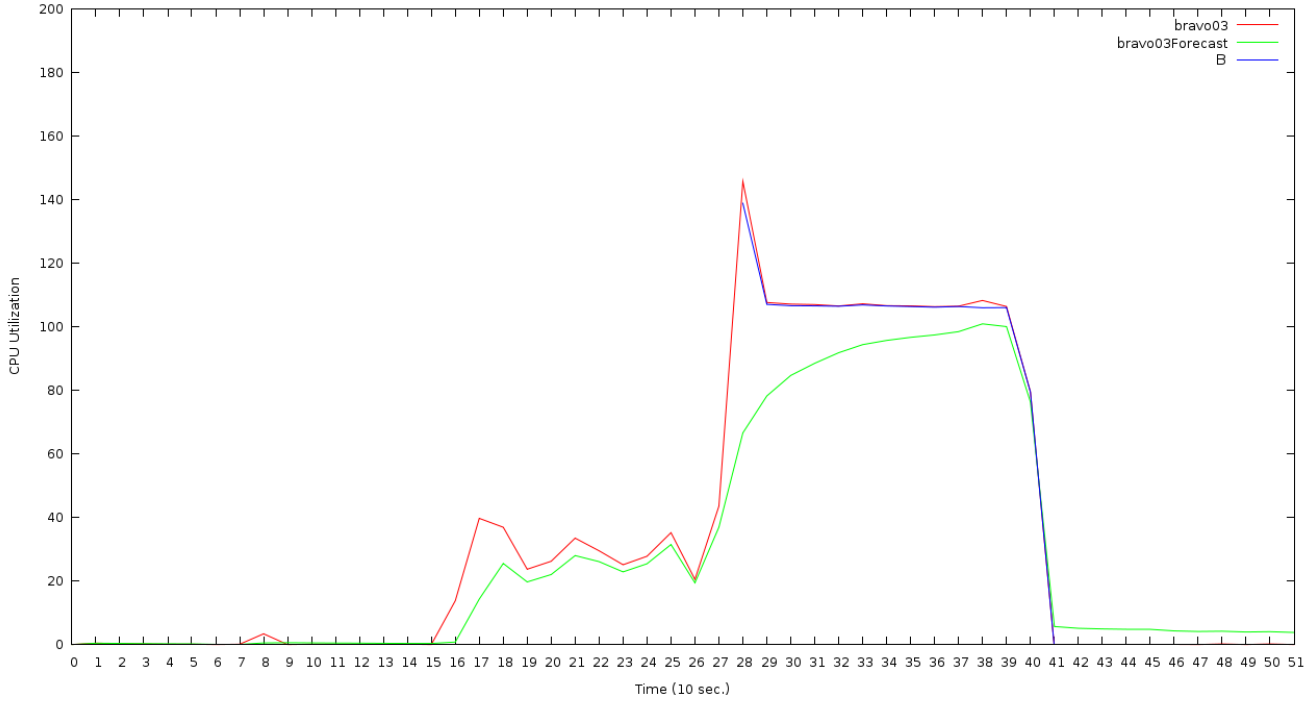


Figure 6. Experiment 1 - Migration, bravo03

Table II
EXPERIMENT 1 - WEB SERVERS' AVERAGE CONNECTION TIME IN MILLISECONDS

Web Servers' Response Time			
Servers	Run 1	Run 2	Run 3
one.com	701.9	749.1	715.2
two.com	904.5	853.0	991.1

can be traced back to the load balancer's need for a restart when updating its configuration after a replication.

The comparison of the web servers' response time does not offer conclusive results (Table II).

In conclusion, when a hardware node experiences a CPU stress situation, but the CPU is not exhausted, no requests are lost. Thus, no action is necessary from the management system. However, migration and replication cause no (serious) performance degradation, so they could be used as preventive actions in case the load was expected to increase.

2) Experiment 2: In the first run of the experiment, Golondrina was monitoring the CPU utilization of the hardware node and containers, but no action was taken in response to CPU stress situations. Figure 7 shows the CPU utilization of containers **A** and **B**, and the CPU utilization and predicted CPU utilization (as explained in Subsection III-B1) of the hardware node **bravo02**.

The first time the CPU utilization of **bravo02** went over the 150% threshold was at $t = 11$. Golondrina's CPU stress detection mechanism signaled the problem at $t = 14$. Since no action was taken, the CPU stress situation persisted and was signaled every single time until $t = 38$ (included).

Starting at $t = 11$ the CPU was equally shared between the two containers, using almost 100% each. However, the number of CPU cycles allocated to each container was not enough for the hosted web servers to process all requests. The web server **one.com**, hosted in **A**, had 101 lost requests out of 450 (client-timo 101), resulting in an effectiveness of 77.55%. The web server **two.com**, hosted in **B**, had 169 lost requests out of 450 (client-timo 169), resulting in an effectiveness of 62.44%.

It can be seen in Figure 7 that during the interval $[36, 38]$ container **B** almost doubled its CPU utilization, taking advantage of container **A** not requesting CPU cycles. This behaviour could be attributed to web server **two.com** processing all the requests that it had not been able to satisfy before due to a lack of CPU cycles.

The web server **one.com** had an average connection time of 2816.3 milliseconds. The web server **two.com** had an average connection time of 3371.6 milliseconds.

In the second run of the experiment, Golondrina was to search for feasible replications if a CPU stress situation was detected. Figure 8 shows the CPU utilization of containers **A** and **B**, and the CPU utilization and predicted CPU utilization of the hardware node **bravo02**. Figure 9 shows the CPU utilization of the replicas **A'** and **B'**, and the CPU utilization and predicted CPU utilization of **bravo03**.

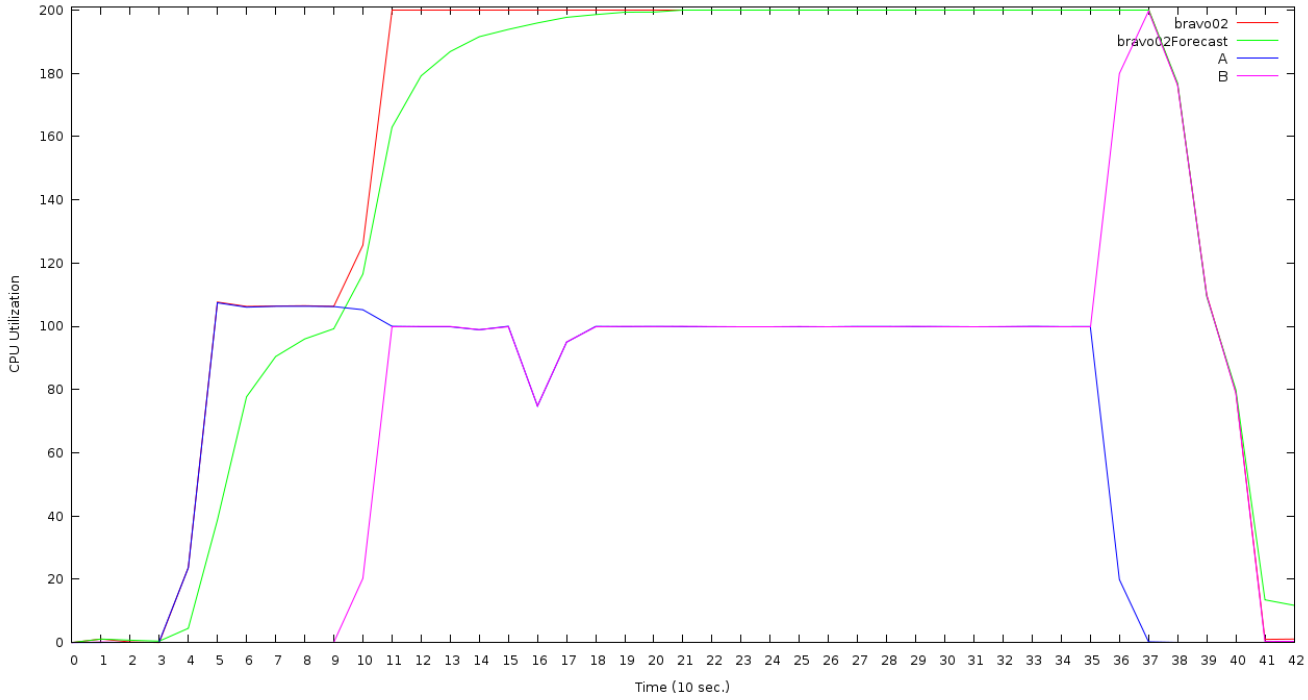


Figure 7. Experiment 2 - No Action

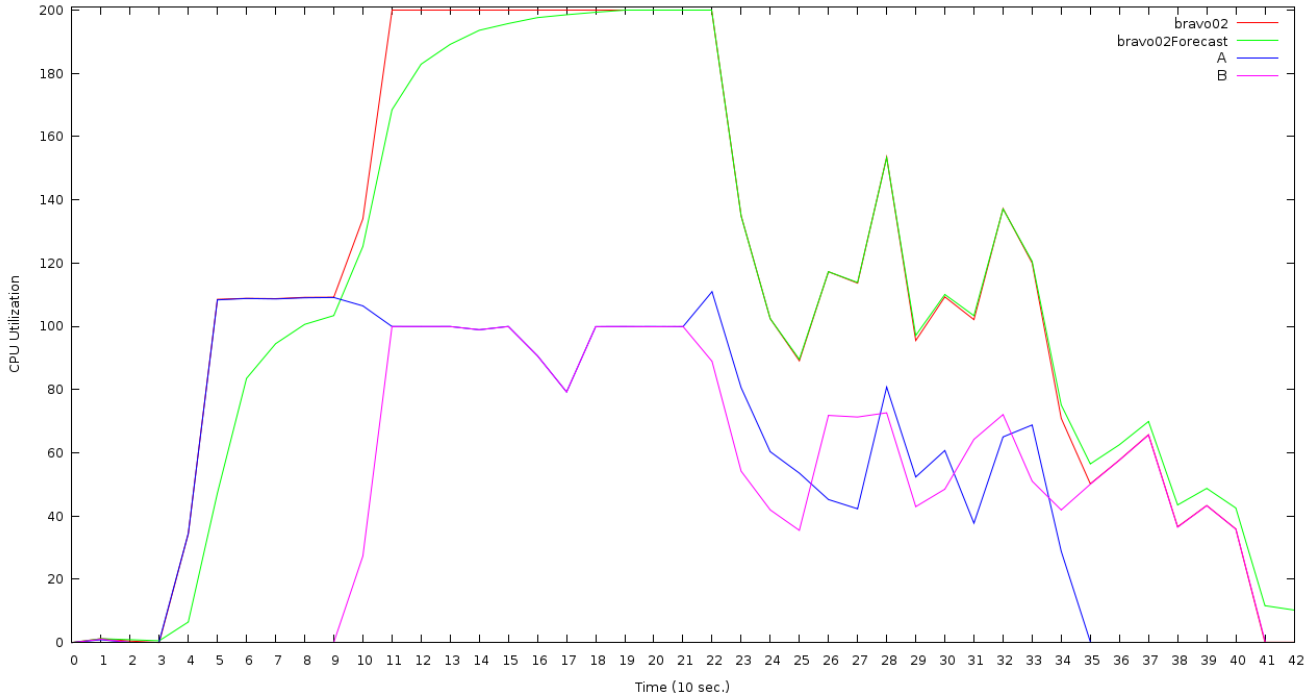


Figure 8. Experiment 2 - Replication, bravo02

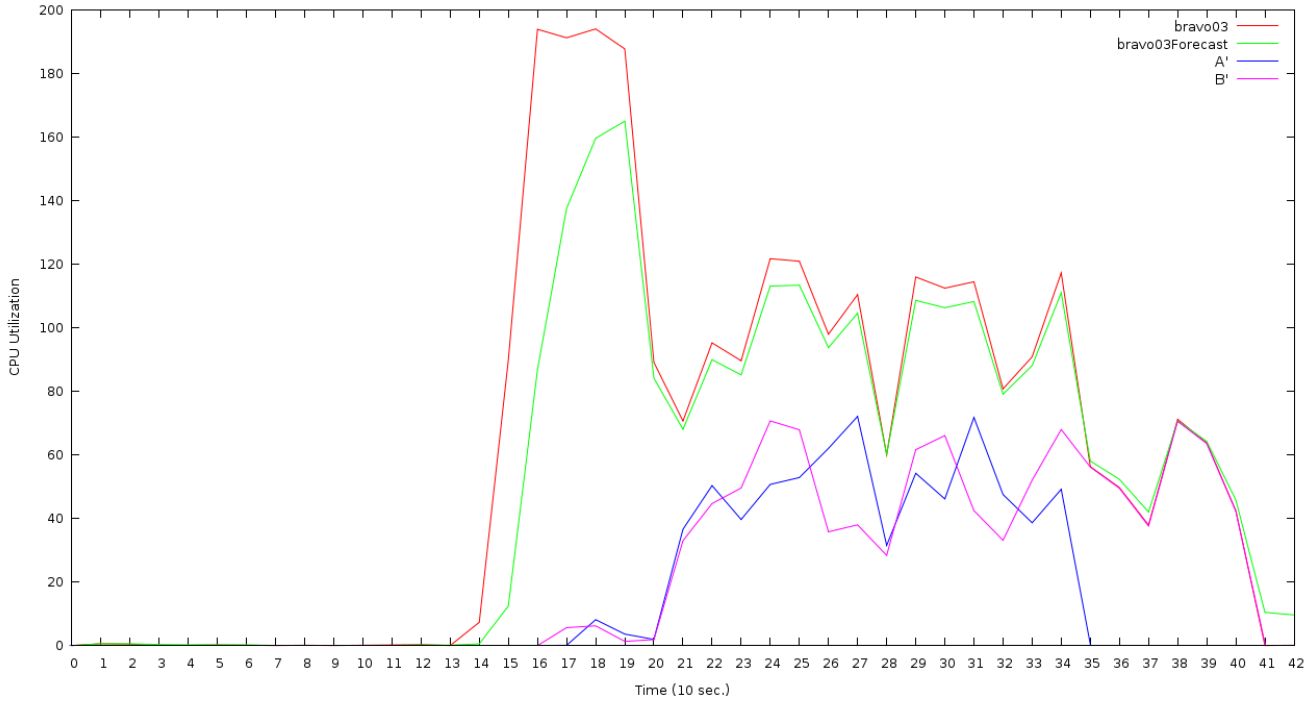


Figure 9. Experiment 2 - Replication, bravo03

The first CPU stress situation in **bravo02** was signaled at $t = 14$. Golondrina determined that both containers had to be replicated in **bravo03**. By $t = 16$ the replicas **B'** and **A'** had been created and the load balancer at the *gate of the cluster* was updated. The load balancer was first updated (and restarted) for **B'** in the period $(15, 16)$ and updated again in the period $(16, 17)$ for **A'**. During those two periods, it can be seen in Figure 8 that the CPU utilization of **A** and **B** decreased, due to connections that were refused or reset.

At $t = 17$, containers **A'** and **B'** had a low CPU utilization and remained with minimal CPU utilization until $t = 20$ (included). During those periods, the requests were handled by **A** and **B**. In consequence, the CPU stress situation continued in **bravo02** and was signaled at $t = 18, 19, 20, 21, 22$. At every one of those five points in time, Golondrina could not find suitable replications since **A** and **B** had already been replicated in **bravo03** and the system does not allow two replicas of the same container to reside in the same hardware node.

The web server **one.com**, hosted in **A** and **A'**, had 21 failed requests (connrefused 4 connreset 17) and 35 lost requests (client-timo 35) out of 450, which resulted in an effectiveness of 87.55%. The web server **two.com**, hosted in **B** and **B'**, had 25 failed requests (connrefused 4 connreset 21) and 29 lost requests (client-timo 29) out of 450, which resulted in an effectiveness of 88%.

The web server **one.com** had an average connection time of 1408.1 milliseconds. The web server **two.com** had an average connection time of 1781.4 milliseconds.

In the third run of the experiment, Golondrina was to look for migrations upon detection of a CPU stress situation. Figure 10 shows the CPU utilization of containers **A** and **B**, and the CPU utilization and predicted CPU utilization of the hardware node **bravo02**. Figure 11 shows the CPU utilization of **A**, and the CPU utilization and predicted CPU utilization of **bravo03**.

A CPU stress situation was signaled at $t = 14$ in **bravo02**. Golondrina determined that container **A** was to be migrated to **bravo03**. The migration process was started, increasing the CPU utilization in **bravo03**. The CPU in **bravo02** was already exhausted, so the migration process competed for the CPU with the containers. **A** and **B** saw a reduction in their CPU allocation in the interval $(14, 20]$ until the migration process ended in the period $(20, 21)$.

In the period $[21, 22]$, container **B** increased its CPU utilization around 180%, and in the period $[24, 25]$ **A** increased its CPU utilization around 195%. The behaviour of both containers could be attributed to the hosted web servers processing the requests that could not be handled during the migration process.

The web server **one.com**, hosted in **A**, had 10 failed requests (reply-status-5xx 10) and 89 lost requests (client-timo 89) out of 450, which resulted in an effectiveness of 78%. The web server **two.com**, hosted in **B**, had 70 lost requests (client-timo 70) out of 450, which resulted in an effectiveness of 84.44%.

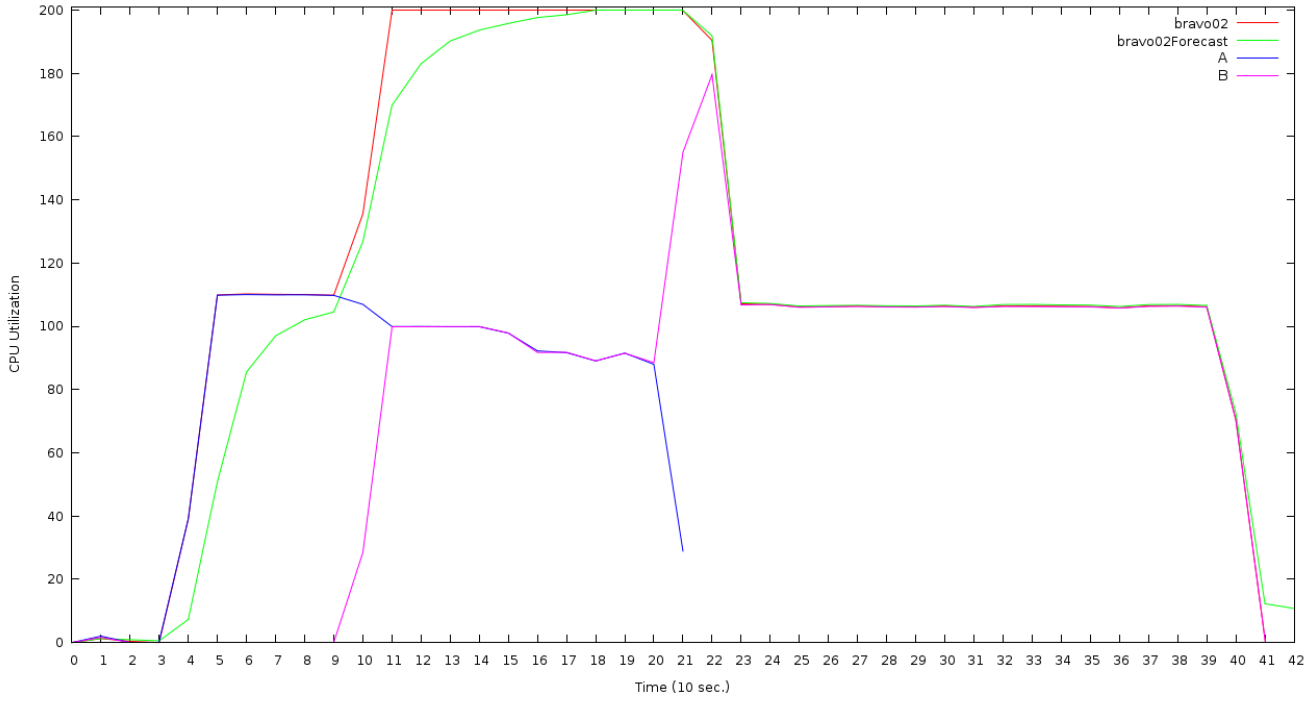


Figure 10. Experiment 2 - Migration, bravo02

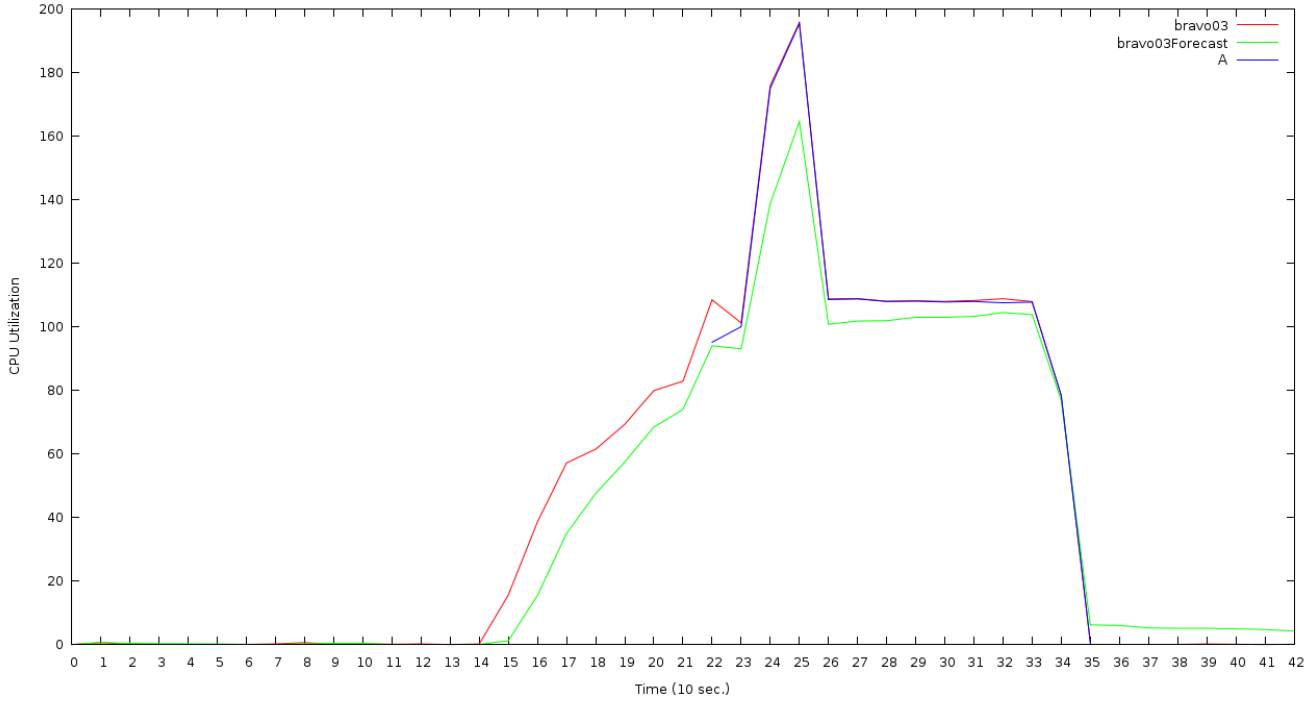


Figure 11. Experiment 2 - Migration, bravo03

Table III
EXPERIMENT 2 - PERCENTAGE OF SUCCESSFUL REQUESTS

Web Servers' Effectiveness			
Servers	Run 1	Run 2	Run 3
one.com	77.55%	87.55%	78%
two.com	62.44%	88%	84.44%

Table IV
EXPERIMENT 2 - WEB SERVERS' AVERAGE CONNECTION TIME IN MILLISECONDS

Web Servers' Response Time			
Servers	Run 1	Run 2	Run 3
one.com	2816.3	1408.1	1723.6
two.com	3371.6	1781.4	1661.6

The web server **one.com** had an average connection time of 1723.6 milliseconds. The web server **two.com** had an average connection time of 1661.6 milliseconds.

From the web servers' effectiveness results (Table III), it can be concluded that migration offers an improvement over taking no action upon detection of a CPU stress situation. However, the migration mechanism does not achieve as much benefit as the replication mechanism does.

The comparison of the web servers' response time (Table IV) shows that both relocation mechanisms help in reducing the average connection time. However, the comparison between the second and third runs does not offer conclusive results.

In conclusion, when a hardware node experiences a CPU stress situation and the CPU is exhausted, some requests will not be satisfied. Both relocation mechanisms offer a convenient solution, since they help to reduce the losses. However, the migration process competes with the containers for CPU cycles in the CPU stressed hardware node, hence diminishing the benefit the migration mechanism could provide.

3) Experiment 3: In the first run of the experiment, Golondrina was monitoring the CPU utilization of the hardware node and containers, but no action was taken in response to CPU stress situations. Figure 12 shows the CPU utilization of containers **A**, **B**, **C** and **D**, and the CPU utilization and predicted CPU utilization (as explained in Subsection III-B1) of the hardware node **bravo02**.

The first time the CPU utilization of **bravo02** went over the 150% threshold was at $t = 17$. Golondrina's CPU stress detection mechanism signaled the problem at $t = 22$. Since no action was taken, the CPU stress situation persisted and was signaled every single time until $t = 51$ (included).

Starting at $t = 23$ the CPU was equally shared between the four containers, using almost 50% each. However, the number of CPU cycles allocated to each container was not enough for the hosted web servers to process all requests. The web server **one.com**, hosted in **A**, had 36 lost requests out of 300 (client-timo 36), resulting in an effectiveness of 88%. The web server **two.com**, hosted in **B**, had 24 lost requests out of 300 (client-timo 24), resulting in an effectiveness of 92%. The web server **three.com**, hosted in **C**, had 39 lost requests out of 300 (client-timo 39), resulting in an effectiveness of 87%. The web server **four.com**, hosted in **D**, had 6 lost requests out of 300 (client-timo 6), resulting in an effectiveness of 98%.

It can be seen in Figure 12 that at $t = 47$, when **A** saw a decrease in its CPU utilization, the remaining containers had a peak in their CPU utilization. This behaviour could be attributed to the hosted web server processing all the requests that could not be satisfied before due to a lack of CPU cycles.

The web server **one.com** had an average connection time of 2916.9 milliseconds. The web server **two.com** had an average connection time of 2462.3 milliseconds. The web server **three.com** had an average connection time of 2537.6 milliseconds. The web server **four.com** had an average connection time of 2268.5 milliseconds.

In the second run of the experiment, Golondrina was to search for feasible replications if a CPU stress situation was detected. Figure 13 shows the CPU utilization of containers **A**, **B**, **C** and **D**, and the CPU utilization and predicted CPU utilization of the hardware node **bravo02**. Figure 14 shows the CPU utilization of the replicas **A'**, **B'**, **C'** and **D'**, and the CPU utilization and predicted CPU utilization of **bravo03**.

The first CPU stress situation in **bravo02** was signaled at $t = 23$. Golondrina determined that containers **A**, **B** and **C** had to be replicated in **bravo03**. By $t = 27$ the replicas **A'**, **B'** and **C'** had been created and the load balancer at the *gate of the cluster* was updated. The load balancer was updated (and restarted) three times (one for each replication) in the period $(27, 28)$. During that period and the following one, it can be seen in Figure 13 that the CPU utilization of **A**, **B**, **C** and **D** decreased, due to connections that were refused or reset.

At $t = 28$, containers **A'**, **B'** and **C'** had a low CPU utilization and remained with minimal CPU utilization until $t = 31$ (included). During those periods, the requests were handled by **A**, **B** and **C**. In consequence, an additional CPU stress situation was signaled in **bravo02** at $t = 30$. Golondrina determined that **D** had to be replicated in **bravo03**, indicating also that the action would not be enough to dissipate the CPU stress situation in **bravo02**.

The web server **one.com**, hosted in **A** and **A'**, had 7 failed requests (connrefused 4 connreset 3) and 2 lost requests (client-timo 2) out of 300, which resulted in an effectiveness of 97%. The web server **two.com**, hosted in **B** and **B'**, had 6 failed requests (connrefused 5 connreset 1) and 5 lost requests (client-timo 5) out of 300, which resulted in an effectiveness of 96.33%. The web server **three.com**, hosted in **C** and **C'**, had 10 failed requests (connrefused

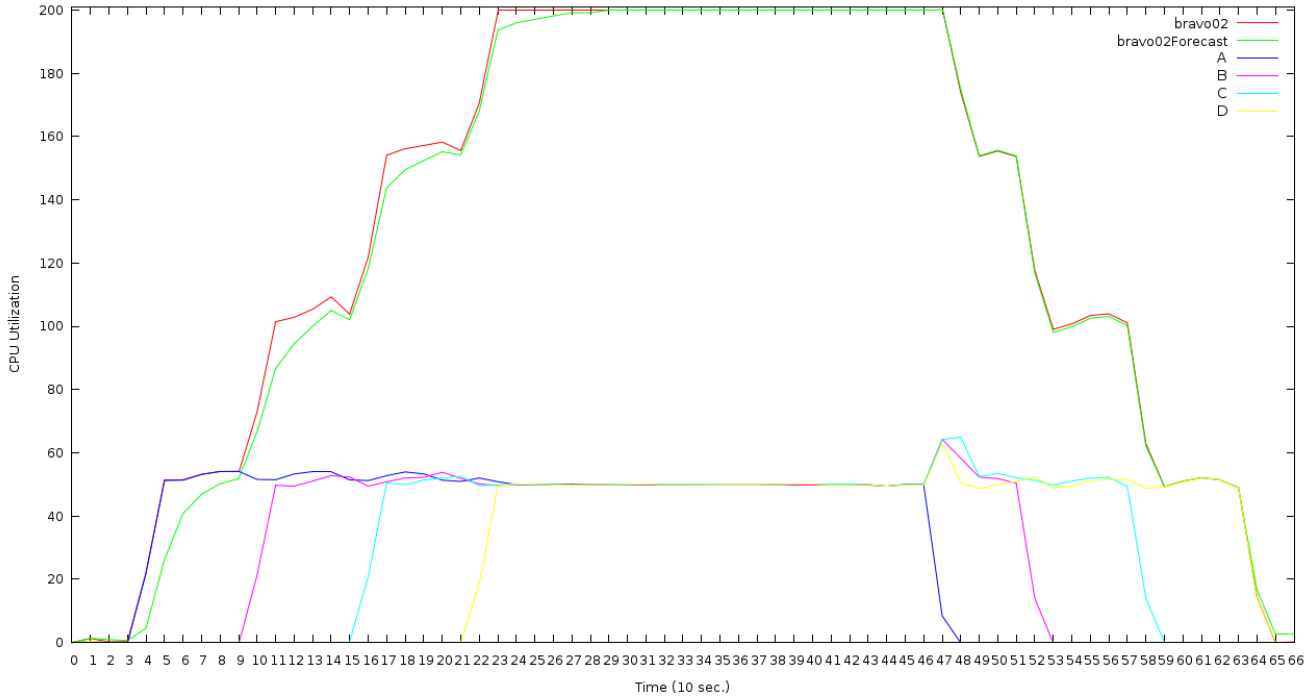


Figure 12. Experiment 3 - No Action

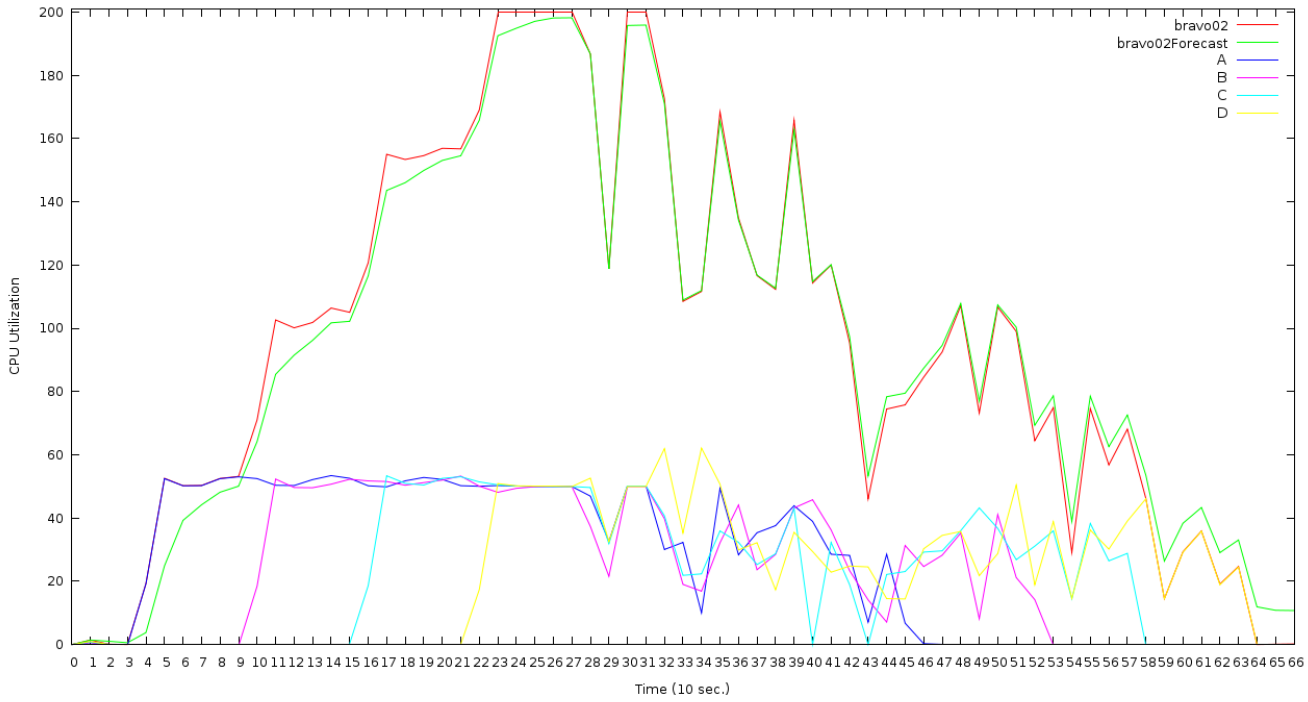


Figure 13. Experiment 3 - Replication, bravo02

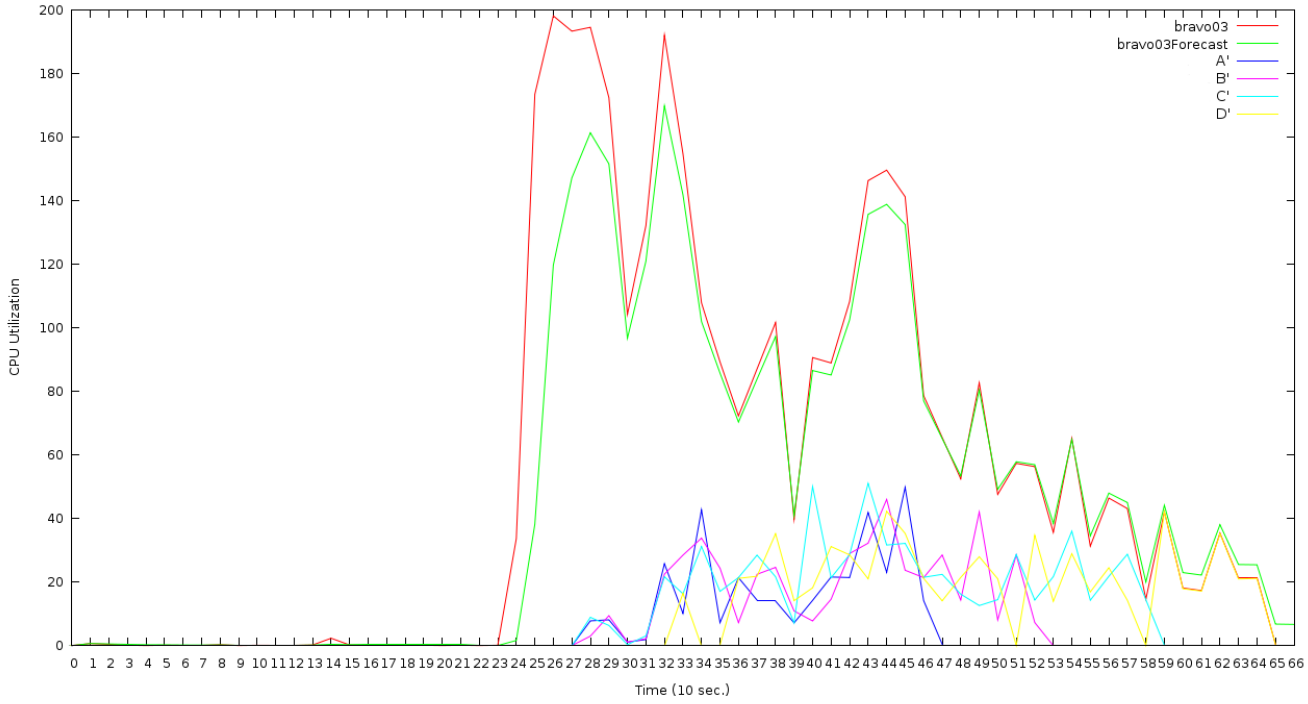


Figure 14. Experiment 3 - Replication, bravo03

5 connreset 5) and 1 lost requests (client-timo 1) out of 300, which resulted in an effectiveness of 96.33%. The web server **four.com**, hosted in **D** and **D'**, had 9 failed requests (connrefused 5 connreset 4) and 2 lost requests (client-timo 2) out of 300, which resulted in an effectiveness of 96.33%.

The web server **one.com** had an average connection time of 1151.3 milliseconds. The web server **two.com** had an average connection time of 1130.8 milliseconds. The web server **three.com** had an average connection time of 1281.0 milliseconds. The web server **four.com** had an average connection time of 1275.9 milliseconds.

In the third run of the experiment, Golondrina was to look for migrations upon detection of a CPU stress situation. Figure 15 shows the CPU utilization of containers **A**, **B**, **C** and **D**, and the CPU utilization and predicted CPU utilization of the hardware node **bravo02**. Figure 16 shows the CPU utilization of **A** and **B**, and the CPU utilization and predicted CPU utilization of **bravo03**.

A CPU stress situation was signaled at $t = 24$ in **bravo02**. Golondrina determined that container **A** was to be migrated to **bravo03**. The migration process was started, increasing the CPU utilization in **bravo03**. The CPU in **bravo02** was already exhausted, so the migration process competed for the CPU with the containers. The four containers saw a reduction in their CPU allocation in the interval $[26, 28]$ until the migration process ended in the period $(28, 29)$.

At $t = 29$, **A** had a peak of around 170% in CPU utilization. Taking advantage of the CPU availability, containers **B**, **C** and **D** increased their CPU utilization during the interval $[29, 33]$, what caused a second CPU stress situation to be signaled at $t = 30$. At that time, Golondrina decided to migrate **B** to **bravo03**. The migration process ended in the period $(35, 36)$ and **B** reached a peak of around 85% in CPU utilization after being migrated.

A final CPU stress situation was signaled at $t = 37$ with the CPU utilization of **bravo02** being 150.48% and becoming 100.5% at the following point in time. Golondrina found no solution to the situation.

The web servers **one.com** and **two.com**, hosted in **A** and **B** respectively, had 18 lost requests out of 300 (client-timo 18), which resulted in an effectiveness of 94%. The web server **three.com**, hosted in **C**, had 14 lost requests out of 300 (client-timo 14), which resulted in an effectiveness of 95.33%. The web server **four.com**, hosted in **D**, had 19 lost requests out of 300 (client-timo 19), which resulted in an effectiveness of 93.66%.

The web server **one.com** had an average connection time of 1051.0 milliseconds. The web server **two.com** had an average connection time of 1412.8 milliseconds. The web server **three.com** had an average connection time of 1343.0 milliseconds. The web server **four.com** had an average connection time of 1248.8 milliseconds.

As in the second experiment, it can be concluded that container migration offered an improvement over taking no action upon detection of a CPU stress situation. Once more, the benefits offered by the replication mechanism exceeded

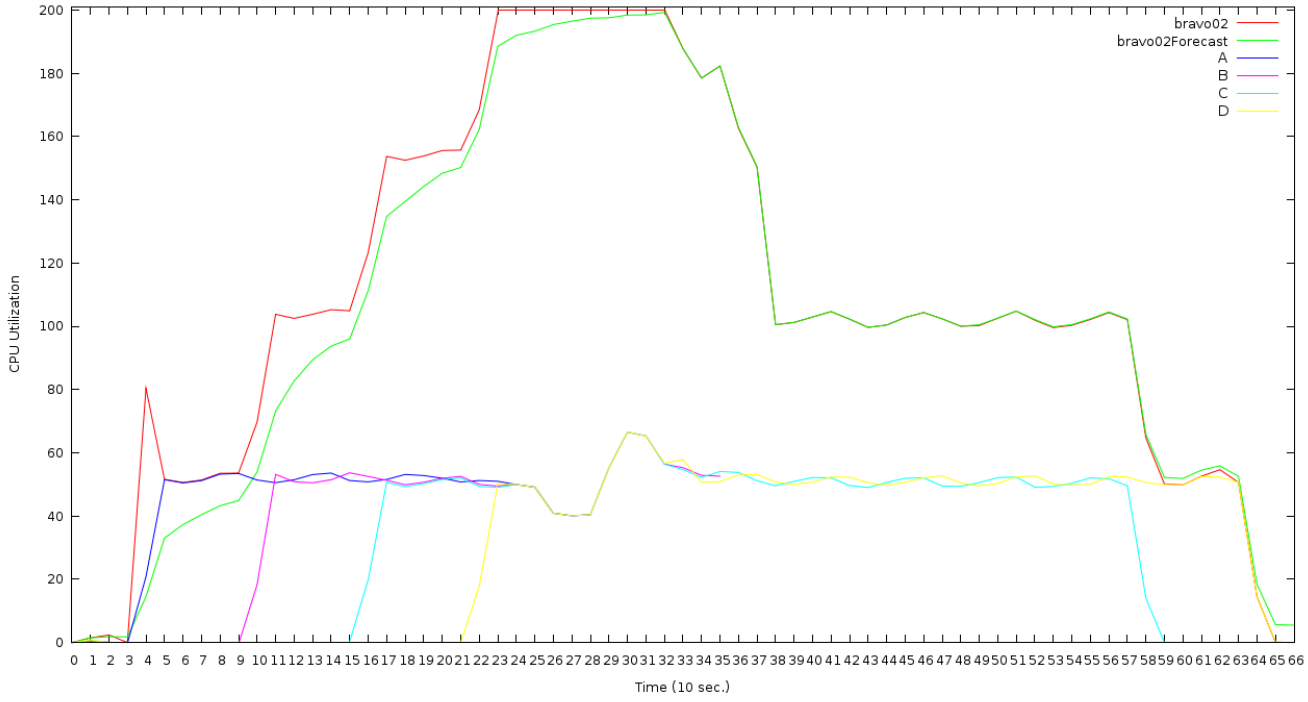


Figure 15. Experiment 3 - Migration, bravo02

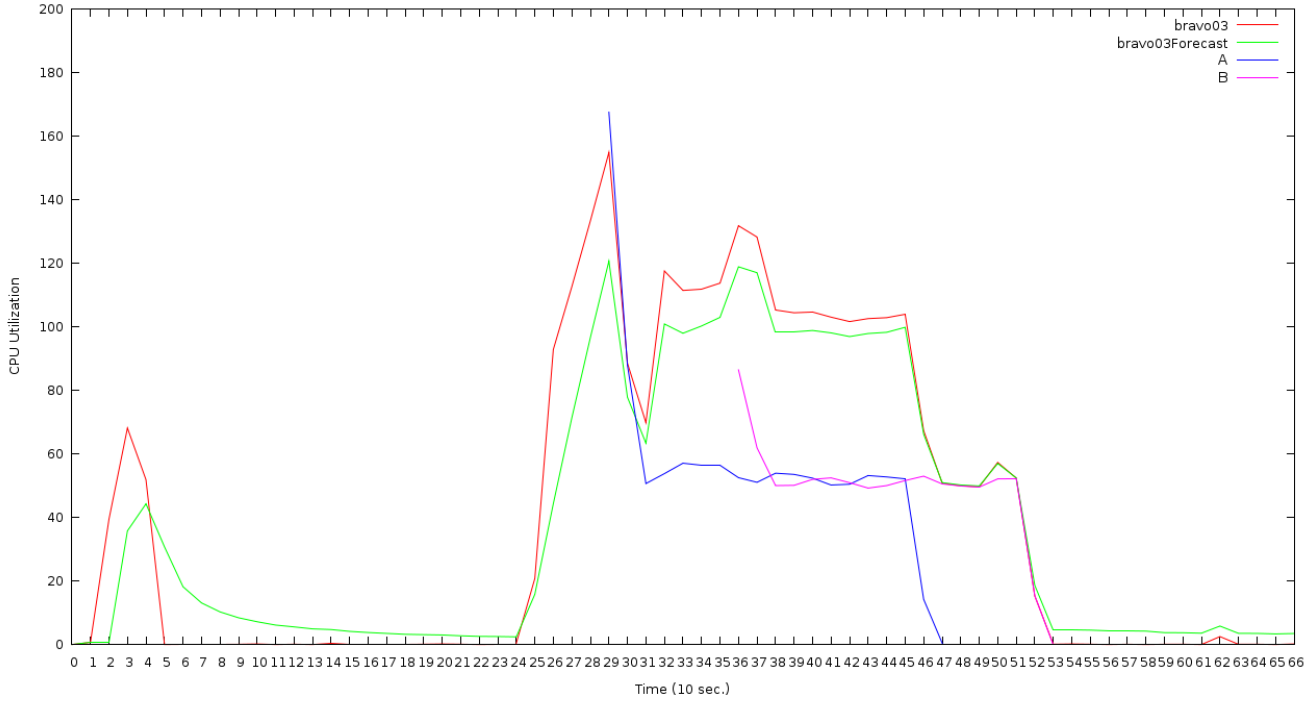


Figure 16. Experiment 3 - Migration, bravo03

Table V
EXPERIMENT 3 - PERCENTAGE OF SUCCESSFUL REQUESTS

Web Servers' Effectiveness			
Servers	Run 1	Run 2	Run 3
one.com	88%	97%	94%
two.com	92%	96.33%	94%
three.com	87%	96.33%	95.33%
four.com	98%	96.33%	93.66%

Table VI
EXPERIMENT 3 - WEB SERVERS' AVERAGE CONNECTION TIME IN MILLISECONDS

Web Servers' Response Time			
Servers	Run 1	Run 2	Run 3
one.com	2916.9	1151.3	1051.0
two.com	2462.3	1130.8	1412.8
three.com	2537.6	1281.0	1343.0
four.com	2268.5	1275.9	1248.8

the benefits obtained through migration.

V. DISCUSSION

The experiments presented in Section IV show that both relocation mechanisms offer an improvement over taking no action upon detection of a CPU stress situation. Even if there are spare CPU cycles to allocate in the stressed hardware node, the mechanisms do not have a negative impact, which supports the use of the relocation mechanisms as preventive actions in case the resource utilization were to continue increasing in the stressed hardware node.

The replication mechanism offers a better improvement over the migration mechanism. One exception is the scenario where enough spare CPU cycles are available at the stressed hardware node for the migration process to use. In that scenario, the hosted containers see no performance degradation during the migration process, and hence the migration mechanism provides the same benefits as the replication mechanism. This suggests the creation of a policy that can be briefly described as follows: *Replication is preferred over migration when the CPU utilization is high. If the CPU utilization is relatively low, then the migration mechanism is preferred.* An algorithm that implements this policy is shown in Algorithm 2.

This policy may result in more replicas being created than there are actually needed, but that can be dealt with by later terminating any of the replicas.

As explained in Section III, this first prototype of Golondrina only manages CPU. Therefore, the prototype is limited in its relocation search feature (Subsection III-B3). A hardware node is classified as non-stressed based only on its CPU utilization. However, the hardware node could be stressed with respect to other resources (such as memory or network

```
1: for i = 0 to SH.length() do
2:   while SH[i] is stressed do
3:     CT = pickMostHeavilyLoadedCT(SH[i]);
4:     targetHN = pickMostLightlyLoadedHN(NSH);
5:     if SH[i].CPULoad > t then
6:       ExecuteReplication(CT, targetHN);
7:     else
8:       ExecuteMigration(CT, targetHN);
9:     end if
10:  end while
11: end for
```

Algorithm 2: Relocation policy

bandwidth). The current version of the relocation search does not address this issue and thus a container that makes use of the stressed resource may be relocated to the hardware node, which intensifies the hardware node's resource stress situation. Future work will address this limitation.

Another interesting outcome of the experiments is the difference between the number of relocations expected in each experiment (as described in Subsection IV-A) and the number of relocations actually triggered by Golondrina to deal with the CPU stress situations.

The first experiment resulted in two replications being triggered when only one should have been enough. The CPU stress situation that resulted in the second replication was caused by an improper balancing of the load for the web server **two.com**, hosted in **B** and **B'**, during the first four periods after the creation of **B'**.

In the second experiment, no unexpected relocation took place. However, multiple CPU stress situations were signaled during the second run of the experiment. The reason why no additional replications were triggered resides on the managed system consisting only of two hardware nodes and Golondrina being restricted by the policy of *not allowing two replicas of the same container to reside in the same hardware node*. Had more hardware nodes been available or that policy not existed, Golondrina would have triggered additional replications. Once more, the CPU stress situations were originated on an improper working of the load balancer.

The third experiment resulted in one additional replication being triggered due to the issue with the load balancer. In addition, a third CPU stress situation was unexpectedly signaled in the third run of the experiment. Be it noted, however, that that third CPU stress situation was more of a *false alarm* given that the CPU utilization of the hardware node was drastically decreasing in the period when the CPU stress situation was signaled.

If the managed system had consisted of more than two hardware nodes, Golondrina would have triggered more relocations during the second and third experiments. Have additional relocations happened, the web servers' effectiveness could have been improved, but with the cost of an overall

lower CPU utilization at cluster-level. This may suggest that Golondrina should give a greater *grace period* to a stressed hardware node where actions have been taken to restore its stability before checking it again for CPU stress situations. This may suggest as well that the CPU stress detection mechanism should take into account CPU utilization trends (*increasing* or *decreasing*) instead of absolute measures of CPU utilization only.

VI. Related Work

Virtualization has become an essential technology in the data center. First, because it enables server consolidation, which may result in lowering costs for building and running data centers. Second, because it creates a very dynamic environment where virtual servers can be resized and relocated on-demand, and new virtual servers can be promptly instantiated. However, this introduces new challenges, which are being addressed by the research community.

Wood et al. studied two approaches to virtual server monitoring: *black-box* and *grey-box* [11]. Black-box monitoring consisted of collecting statistics from the virtual servers without directly contacting them. Grey-box monitoring required running an additional software module inside each virtual server to collect operating system statistics and process application logs. The authors concluded that the grey-box approach enabled the system to make better informed decisions. Still, Golondrina only implements black-box monitoring, since grey-box monitoring could be considered an invasive mechanism that clients might find undesirable.

Zhao and Figueiredo studied the virtual server migration process to be able to predict its performance [12]. After several experiments, they concluded that the migration process' time and performance could be predicted for a number of virtual servers based on measurements from a single virtual server. Kochut and Beaty worked on an analytical model to estimate the improvement in response time achieved through a virtual server migration [13]. Given the current system load and the virtual servers' expected resource demand, the model could help determine whether a migration should be started and which virtual servers to migrate. These studies are important for the development of decision-making support mechanisms that could be used by management entities such as Golondrina's Manager.

Gmach et al. evaluated resource reallocation policies that a node controller could use to periodically do resource reallocation among virtual servers hosted in a physical server [14]. Their study showed that work conserving policies were more effective than non-work conserving policies and that dynamically adjusting workloads' priority resulted in better compliance with Service Level Objectives. These results would be useful if we were to incorporate hardware node-level resource reallocation to Golondrina.

Zhu et al. developed an automated resource management system composed of three controllers that worked at different hierarchical levels and intervals in time [15]. Node controllers would reallocate resources among the workloads hosted in the physical server they were responsible for, pod controllers (a *pod* was a set of nodes) would migrate workloads between physical servers in their pod, and pod set controllers would migrate workloads between pods. The controllers implemented different analytic techniques, such as control theory, bin packing, and trace-based analysis. The integrated work of these controllers offered great results in terms of resource utilization and Quality of Service. Golondrina's Manager component fulfills a similar role to that of the pod controller. However, Golondrina's Manager executes both migrations and replications, whereas the pod controller only executes migrations.

Kumar et al. addressed the lack of coordination between management systems in data centers [16]. They proposed a framework that loosely coupled platform management and virtualization management, and enabled coordinated management actions to be taken. They concluded from their experiments that coordinated management resulted in energy savings, greater stability, and better Quality of Service being provided.

Munasinghe and Anderson worked on developing a data center architecture (hardware and software configuration) that could provide guaranteed Quality of Service to its clients [17]. They developed two mechanisms to do resource scaling: *horizontal scaling* (virtual server replication), and *vertical scaling* (resource allocation adjustments and virtual server migration). Their approach to resource scaling was similar to the one implemented in Golondrina. Their prototype at the moment, however, could not do horizontal scaling.

VII. Conclusion and Future Work

This work is one of the very few (if any at all) that proposes a resource management system for operating system-level virtualized environments. In addition, this is the first study that uses replication as an alternative to migration and compares both mechanisms. Others have proposed to do replication, but have not done it [11]. Others have implemented replication, but not migration [17].

There are many ways in which Golondrina could be extended. One of them is implementing a mechanism to monitor the activity of replicas and stop those that become unnecessary (see Section V). Another extension is to implement a CPU *under-stress* detection mechanism, which would detect and suspend lightly loaded hardware nodes.

Adding memory and network as managed resources is obviously a very attractive extension. A second prototype of Golondrina has already been developed featuring a *memory stress detection mechanism* and a simple heuristic for adjusting memory allocations [18].

Currently, we are studying the interaction between the CPU and memory subsystems and the management strategies that can be implemented to solve different combinations of resource stress situations.

ACKNOWLEDGMENTS

We would like to thank the National Sciences and Engineering Research Council (NSERC) of Canada for their support.

REFERENCES

[1] G. Keller and H. Lutfiyya, "Replication and migration as resource management mechanisms for virtualized environments," in *ICAS '10: Proceedings of the 2010 Sixth International Conference on Autonomic and Autonomous Systems*. Washington, DC, USA: IEEE Computer Society, 2010, pp. 137–143.

[2] A. McCloskey, B. P. Simmons, and H. Lutfiyya, "Policy-based dynamic provisioning in data centers based on slas, business rules and business objectives," in *IEEE/IFIP Network and Operations Management Symposium (NOMS 2008)*, Salvador, Bahia, Brazil, Apr. 2008. [Online]. Available: http://ieeexplore.ieee.org/xpls/abs_all.jsp?arnumber=4575243

[3] P. Padala, K. G. Shin, X. Zhu, M. Uysal, Z. Wang, S. Singhal, A. Merchant, and K. Salem, "Adaptive control of virtualized resources in utility computing environments," in *EuroSys '07: Proceedings of the 2nd ACM SIGOPS/EuroSys European Conference on Computer Systems 2007*. New York, NY, USA: ACM, 2007, pp. 289–302.

[4] (2010, Aug.) Openvz project. [Online]. Available: http://openvz.org/

[5] K. Kolyshkin. (2005) Virtualization in linux. openvz-intro.pdf. Documentation on OpenVZ. [Online]. Available: http://download.openvz.org/doc/OpenVZ-Users-Guide.pdf

[6] G. Box, G. M. Jenkins, and G. Reinsel, *Time Series Analysis: Forecasting And Control*, 3rd ed. Prentice Hall, Feb. 1994.

[7] C. Hyser, B. Mckee, R. Gardner, and B. J. Watson, "Autonomic virtual machine placement in the data center," HP Laboratories, Palo Alto, CA, USA, Tech. Rep. HPL-2007-189, Dec. 2007.

[8] (2010, Aug.) Don quijote by miguel de cervantes saavedra - project gutenberg. [Online]. Available: http://www.gutenberg.org/etext/2000

[9] (2010, Aug.) The apache http server project. [Online]. Available: http://httpd.apache.org/

[10] (2010, Aug.) Httperf site. HP Labs. [Online]. Available: http://www.hpl.hp.com/research/linux/httperf/

[11] T. Wood, P. Shenoy, A. Venkataramani, and M. Yousif, "Black-box and gray-box strategies for virtual machine migration," in *Proceedings of the Fourth Symposium on Networked Systems Design and Implementation (NSDI'07)*, Cambridge, MA, USA, Apr. 2007, pp. 229–242. [Online]. Available: http://www.usenix.org/events/nsdi07/tech/wood.html

[12] M. Zhao and R. J. Figueiredo, "Experimental study of virtual machine migration in support of reservation of cluster resources," in *VTDC '07: Proceedings of the 3rd international workshop on Virtualization technology in distributed computing*. New York, NY, USA: ACM, 2007, pp. 1–8.

[13] A. Kochut and K. Beaty, "On Strategies for Dynamic Resource Management in Virtualized Server Environments," in *MASCOTS '07: Proceedings of the 2007 15th International Symposium on Modeling, Analysis, and Simulation of Computer and Telecommunication Systems*. Washington, DC, USA: IEEE Computer Society, 2007, pp. 193–200.

[14] D. Gmach, J. Rolia, and L. Cherkasova, "Satisfying service level objectices in a self-managing resource pool," in *SASO '09: Proceedings of the 2009 Third IEEE International Conference on Self-Adaptive and Self-Organizing Systems*. Washington, DC, USA: IEEE Computer Society, 2009, pp. 243–253.

[15] X. Zhu, D. Young, B. J. Watson, Z. Wang, J. Rolia, S. Singhal, B. McKee, C. Hyser, D. Gmach, R. Gardner, T. Christian, and L. Cherkasova, "1000 islands: Integrated capacity and workload management for the next generation data center," in *Proceedings of the 2008 International Conference on Autonomic Computing (ICAC'08)*, Chicago, IL, USA, Jun. 2008, pp. 172–181. [Online]. Available: http://ieeexplore.ieee.org/xpl/freeabs_all.jsp?arnumber=4550838

[16] S. Kumar, V. Talwar, V. Kumar, P. Ranganathan, and K. Schwan, "vmanage: loosely coupled platform and virtualization management in data centers," in *ICAC '09: Proceedings of the 6th international conference on Autonomic computing*. New York, NY, USA: ACM, 2009, pp. 127–136.

[17] G. Munasinghe and P. Anderson, "Flexiscale - next generation data centre management," in *UKUUG Spring Conference*, 2008. [Online]. Available: http://homepages.inf.ed.ac.uk/dcspaul/publications/ukuug2008.pdf

[18] A. Pokluda, G. Keller, and H. Lutfiyya, "Managing dynamic memory allocations in a cloud through golondrina," in *Proceedings of the 4th International DMTF Academic Alliance Workshop on Systems and Virtualization Management (SVM'10)*, Oct. 2010, (Accepted to appear).

Coordinated Exploration and Goal-Oriented Path Planning using Multiple UAVs

Christoph Rasche, Claudius Stern, Lisa Kleinjohann, and Bernd Kleinjohann
Faculty of Computer Science, Electrical Engineering and Mathematics
Department of Computer Science
University of Paderborn
Paderborn, Germany
crasche@c-lab.de, claudis@c-lab.de, lisa@c-lab.de, bernd@c-lab.de

***Abstract*—Successful rescue operations after big accidents or natural disasters require a fast and efficient overview of the overall situation. Using aircrafts is an efficient method to achieve such an overview in relatively short time. With recent advances, unmanned aerial vehicles (UAVs) are more and more a viable choice under such circumstances.**

With the number of employed UAVs, the problem of coordination arises as well as proper task allocation among possibly heterogeneous UAVs. Coordination has to be done so that redundant exploration and collision of UAVs with each other are avoided.

This paper presents a hybrid approach for UAV coordination that covers the exploration of unknown terrains as well as goal-oriented coordination and simultaneous task allocation. The approach combines the simplicity of the gradient method with informed A* search and supports prioritized task assignment. It is based on the potential field theory using harmonic functions. Only one single configuration space for representing all relevant information, regarding the terrain and the UAVs is used. The system is suited for highly dynamic environments requiring frequent path recalculations.

***Keywords*-Path Planning, Multiple UAVs, Exploration, Coordination, Potential Field Theory, Harmonic Functions**

I. Introduction

Over the last years, unmanned aerial vehicles (UAVs) received increasing attention in rescue operations. They can be used to explore terrains and are able to relieve humans from dangerous and risky tasks.

Through the use of cameras, it is for instance possible to detect victims or dangerous situations like fire nearby explosives. Such information can be used to coordinate the assistants at the accident site.

Another task, which can be solved is to take measurements in the air, e. g., after a volcanic eruption without risking the life of pilots.

A major task in rescue operations is the exploration of the disaster area. This is important to get an overview of the surroundings and to locate victims. It is necessary to obtain such an overview as fast as possible to start effective rescue operations and–in some cases–to avoid an escalation of the situation, e. g., when explosives are detected near a fire.

Exploratory navigation includes determining all obstacles and goals in a given environment. UAVs travel from their initial positions to one or more different goal positions, while avoiding obstacles and recognizing landmarks and objects. Therefore, the UAVs have to memorize the explored space to plan efficient paths to unknown territory.

In rescue scenarios often places can be identified by the staff where victims are very likely to be found. These areas will be explored first. To take this into account it is possible to mark areas as given goal areas.

The main challenge is to design a system, which covers all the following aspects: autonomous exploration, flying to given targets, and the coordination of multiple UAVs. So, the system to be designed has to manage exploratory navigation as well as goal-oriented planning. Also a task allocation amongst the UAVs is needed.

Another point is that the system has to be efficient. In medical emergencies often every second counts. A system, which needs a long time span to plan paths gives away important time for lifesaving. This means that every calculation, necessary for path planning, should be done without any noticeable delay.

In this paper, an approach combining exploratory navigation, goal-oriented path planning and simultaneous task allocation is presented. It gives a more detailed view of the work presented at the ICAS 2010 [1]. The presented approach is based on artificial potential fields (Section II-D). Path generation then utilizes an informed search algorithm in combination with a gradient method (Section VI). To obtain the desired efficiency, several methods (Section V-C), like a quadtree and an activation window, are used to reduce the calculation costs for path planning. The system ensures a complete exploration of unknown regions and scales well in relation to the number of operating UAVs.

The presented system has two modes of operation, which change automatically during runtime, dependent on the existence or absence of goals: *goal-oriented* and *non-goal-oriented* mode.

Every single UAV has an individual configuration (Section IV-B), amongst others containing its task list. In goal-oriented mode a UAV has one or more prioritized tasks (goals). In non-goal-oriented mode exploration of the complete territory will be done. The approach is cost-efficient because of the combination of the advantages of goal-oriented path planning with the simplicity of gradient methods for

exploration. UAVs without a given goal always move to the nearest unexplored region to optimize exploration. Local as well as global information is used to coordinate multiple UAVs and to generate consistent trajectories. Areas, which probably contain important information can be explored first.

To even support exploration in goal-oriented mode, there is an adjustable tradeoff for the maximum allowed deviation from the shortest path to explore unknown regions (Section V-A). In many cases, gathering new information is also important and a small detour is worth a slight increase of flight time.

The paper is organized as follows. In Section II we show the basics of our approach and some differences to other research. Section III gives an overview of the system we developed. Section IV shows our configuration space, which is used as world model. In Section V we explain how we calculate the potential field using harmonic functions. In Section VI we show the path planning algorithms using the potential field to calculate smooth paths that guarantee obstacle avoidance and always lead to the goal if a path exist. In Section VII we show several results for exploratory navigation including the calculation costs. Finally, Section VIII gives a conclusion and a perspective of future work.

II. Related Work

This section describes related work in the fields of exploration, path planning, and task allocation. Afterwards, the theoretical background for the methods used in our approach is introduced. We start with the basic problem of path planning, which leads to the concept of a configuration space $\mathcal{C}$, and show an overview of the used approaches, based on $\mathcal{C}$.

Today the potential field theory [2] is used in several research areas, e. g., when autonomous robots have to explore terrains [3], [4], [5] or in game theory for path planning of units in strategy games [6]. Several different potential field techniques exist for the different applications.

Exploration of unknown terrain is one of the most important tasks of UAVs. Sawhney et al. [7] presented an exploration system for multiple robots and UAVs in 2D and 3D environments. Their work is focused on an asynchronous exploration strategy in unknown terrains. Asynchronous means that in contrast to synchronous allocation only UAVs, which have stopped are considered for the next exploration of unknown terrain.

Their main goal is to find several paths, which completely explore the terrain, such that the time needed for exploration is reduced as much as possible and the height of the path over any point is constrained to lie beneath an exposure surface. For this purpose they subdivide the terrain using an occupancy grid and plan their paths based on this data structure. In simulations they showed that their approach explores the terrain faster than a synchronous one.

Another part of UAV navigation is goal-oriented path planning. Jung et al. [8] showed a hierarchical path planning approach, including a control algorithm for one single UAV. The UAV had to accomplish the mission objective with limited computational resources. This was to reach a goal destination, while avoiding obstacles.

They start path planning using the A*-algorithm as an informed search algorithm. Afterwards, a path smoothing takes place and the calculated path is followed by an auto pilot.

Like our approach, theirs is based on a multiresolution decomposition of the environment for path planning too, such that a coarser resolution is used far away from the agent, whereas fine resolution is used in the vicinity of the UAV.

For task allocation a scheduling was introduced. To increase the efficiency of their approach a real-time kernel was used. They demonstrated that the UAV with its limited resources managed the mission objective by using a real-time kernel.

Nikolos et al. [9] showed a goal-oriented approach based on evolutionary algorithms, which is capable to navigate through explored and unexplored terrain to a given goal. They distinguish two problems:

- UAV navigation using an offline planner, considering a known 3D environment
- UAV navigation using an online planner, considering a completely unknown 3D environment

In this context offline planning means that a complete path from the initial position of the UAV to the goal position is calculated in advance, whereas in online planning a nearly optimal path will be generated to an intermediate position within the sensors' range. During flight time this is repeated, until the goal position is reached.

They considered the path planning problem within an evolutionary algorithm context, in which B-Spline curves are used to represent the path line. The evolutionary algorithm models the coordinates of the UAV's control points as artificial chromosome genes.

They used the potential field theory combined with a grid-based cell decomposition method as underlying system on which path planning is done. Their results show that it is possible to find feasible paths efficiently by using evolutionary algorithms. This solution could be reached cost-efficiently since only a few iterations were needed to find a feasible solution. The most costly part was to optimize the solution.

All these approaches consider either goal-oriented or non-goal-oriented path planning. In this work we present a combined approach that considers three aspects: exploratory navigation, goal-oriented path planning and simultaneous task allocation. Our approach works in completely unknown environments as well as in (partially) known environments. Paths are always computed offline, which means that our

approach calculate paths to given goals or to unexplored areas a priori and the UAVs follow them. If obstacles appear, which lie in the trajectory of any UAV, a recalculation is triggered.

For example in [7] no task allocation is done and the UAVs are considered to be homogeneous (at least in terms of exploration capability). In contrast, the UAVs in this work are considered to be heterogeneous and, in particular, some tasks can be solved only by a matching UAV. Additionally, in this work exploration of some areas is needed repeatedly as data can become obsolescent over time.

In contrast to most path planning approaches, which focus on the optimization of the path length or where the tests always end after one single exploration of the terrain, we try to lower the time for exploration of a terrain. In our work in some circumstances detours are allowed or even desired (Section V-A). The cases where detours are allowed are in goal-oriented mode. Taking a detour to explore some new terrain leads to an information gain, which can be important. To avoid paths where the detours increase the time needed to reach given goals too much the maximum length of the detours can be defined a priori (Section V-A). Additionally, as information can become obsolescent, explored terrain becomes unexplored again after some time.

A. Path planning basics

Path planning is a wide area of research. A lot of approaches to plan paths are described in the literature. But most approaches are based on a few basic methods. Most commonly used are the following three methods or a combination of them:

1) Roadmap method
2) Cell decomposition
3) Potential field theory

The roadmap method's basic idea is to create a roadmap, which reflects the connectivity of the free space. If such a roadmap exists, path planning is simple. Only the initial and the goal configuration must be connected to the roadmap. A feasible path is found by simply following the path in the roadmap, if both configurations can be connected to it. Otherwise, no feasible path exists. Kazemi et al. [10], e. g., combined the roadmap method with cell decomposition.

We combine cell decomposition with the potential field theory. For this a subdivision of the whole terrain into several subareas is done using a quadtree. Afterwards, each area gets a so-called potential value ϕ. Based on these values path planning can be done. A detailed description of this process is given in the further sections.

All methods are used to solve the basic problem to plan paths, which can be described as follows:

Let $\mathcal{A}$ be a UAV. It moves in a Euclidean space $\mathcal{W}$, represented as R^N (in our case $N = 3$). Additionally, $\mathcal{B}_1, \cdots, \mathcal{B}_r | \mathcal{B}_i \subset \mathcal{W}, (i = 1, \cdots, r)$ are fixed objects distributed in $\mathcal{W}$. Every single $\mathcal{B}_i$ is an obstacle. Assume that the geometry and the position of $\mathcal{A}$ is known. The $\mathcal{B}_i$ can be known but they don't have to be. Further we assume that $\mathcal{A}$ is not restricted in its movement through kinematic constraints. This leads to the following problem:

Given an initial position and direction plus a goal position and direction of $\mathcal{A}$ in $\mathcal{W}$, calculate a path τ consisting of a consecutive sequence of positions and directions of $\mathcal{A}$ under avoidance of contact with any $\mathcal{B}_i$. The path has to start at the initial position and must end at the goal position. Return an error, if such a path does not exist.

All path planning problems have in common that a path from the initial configuration to the goal configuration must be found. This leads to the concept of the configuration space [11].

B. Configuration space

The configuration space $\mathcal{C} \subset R^n$, with n describing the dimension of $\mathcal{C}$ is used to represent the UAV $\mathcal{A}$ within a terrain $\mathcal{W}$. A configuration $q \in \mathcal{C}$ is a tuple $q_1, \cdots, q_n$. The n independent variables can, e. g., represent the position, the role and tasks of a UAV. $\mathcal{A}$ comes with a specific role and tasks, which have to be fulfilled. In our work $\mathcal{A}$ is represented as a point $p \in \mathcal{W}$. The point p is also the position of $\mathcal{A}$ in $\mathcal{W}$. So, UAV $\mathcal{A}$ has a configuration $q_{\mathcal{A}} \in \mathcal{C}$ that consists of all relevant properties of $\mathcal{A}$, e. g., the current position p and its direction. Following the definition of Barraquand et al. [11] one can consider a geometrical application, which maps any configuration q to the point p in the terrain. This map

$$\begin{array}{rcccl} X & : & \mathcal{C} & \rightarrow & \mathcal{W}, \\ & & q & \mapsto & p = X(q) \end{array}$$

is called the forward kinematic map. $\mathcal{C}$ is the union of all possible configurations q.

A Cartesian coordinate system $\mathcal{F}_{\mathcal{W}}$ is embedded in $\mathcal{C}$. In our work a configuration of $\mathcal{A}$ is the specification of the position and direction of $\mathcal{A}$ in consideration of $\mathcal{F}_{\mathcal{W}}$, plus its role, fuel level, maximum and favorite height and its tasks. The position of the UAV in $\mathcal{W}$ is called $p_{\mathcal{A}}$. Due to consistency $\mathcal{C}$ is discretized like the real space using cell decomposition.

1) Occupied space: The workspace of $\mathcal{A}$ contains a finite number of obstacles $\mathcal{B}_i$, $i = 1, \cdots, r$. Obstacles can, e. g., be skyscrapers, mountains or other UAVs, which lie in the trajectory of the UAV. Each obstacle $\mathcal{B}_i$ is in $\mathcal{CB}_i \subset \mathcal{C}$ of those configurations in which the UAV would take a position inside an obstacle $\mathcal{B}_i$. This means:

$$\mathcal{CB}_i = \{q \in \mathcal{C} | X(q) \in \mathcal{B}_i\}, \forall i = 1, \cdots, r.$$

Configurations including these positions are forbidden as they lead to damage of the UAVs.

2) Free space: Due to the existence of obstacles the UAVs are not able to take every position in $\mathcal{C}$. With the definition of $\mathcal{CB}_i$ it is possible to calculate the allowed configurations of the UAVs. This subset of configurations in which the UAVs do not take positions within an obstacle is called free space. This means:

$$\mathcal{C}_{free} = \mathcal{C} \setminus \bigcup_{i=1}^{r} \mathcal{CB}_i.$$

Every configuration $q \in \mathcal{C}_{free}$ is called a free configuration. Using this modeling a collision-free path from a start configuration q_{start} to a goal configuration q_{goal} is a consecutive map $\tau : [0, 1] \rightarrow \mathcal{C}_{free}$ with $\tau_0 = q_{start}$ and $\tau_1 = q_{goal}$.

C. Cell decomposition

The main idea of cell decomposition is the subdivision of $\mathcal{C}$ into disjoint areas, called cells or leaves. An important part is the connectivity graph G. It captures the structure of $\mathcal{C}$. Each subarea is represented through a leaf. A distinction between exact and approximative approaches is made. We use an approximative approach due to cost reductions when dynamic changes take place.

The subdivision is made through the introduction of an internal data structure. This structure has to contain all relevant data, like positions and dimensions of the subareas, positions of obstacles and so on.

One of the most commonly used approaches are grids. They subdivide $\mathcal{W}$ into a number of equally sized subareas. One disadvantage of this approach is that it does not take into account neighboring areas with equal properties, e. g., several neighboring areas, which are occupied space. Neighbors of a current area are all areas, which have one or more border points in common with the current one. For cost reduction it is more efficient to combine such neighboring areas to one big area, so that only one big area has to be taken into account for calculations.

Therefore we use a quadtree for the subdivision of $\mathcal{W}$. The quadtree divides the complete space into four subspaces of equal size. These subspaces are recursively divided into four subspaces. Subdivision stops when the enclosed space of a leaf is of homogeneous type or a pre-defined maximum breakdown is reached. Homogeneous means that the space consists of only one type, e. g., occupied space. If the given breakdown is reached and the space is not homogeneous, an approximation is done: Every leaf which includes occupied space will be marked as occupied space. Otherwise, if the space includes different types of $\mathcal{C}_{free}$, it is marked with that type of space, which constitutes the largest part of the leaf. After subdivision, the complete terrain is represented through the leaves of the quadtree, i. e., the union of all leaves represents $\mathcal{W}$.

One disadvantage of a simple quadtree in contrast to a grid is, that finding neighbor leaves is more costly. It takes logarithmic time instead of constant time when using a grid. This becomes more important as neighbor finding is one of the operations needed most. It is necessary for potential calculations (Section V-B) and for finding paths (Section VI).

To take this into account, we extended the quadtree to a linear quadtree [12]. For transforming a quadtree into a linear quadtree each node of the tree gets a unique code and its level is saved. With this information it is possible to find neighboring nodes with constant time complexity on average.

The approach of cell decomposition is combined with the potential field theory in such a way, that a so-called potential value ϕ is assigned to each leaf of the quadtree.

D. Artificial potential fields for motion planning

The idea of using artificial potential fields for motion planning was introduced in 1985 by Khatib [2]. Using this technique a manipulator moves in a field of artificial forces. The idea is that paths can be obtained through linear superpositions of these fictitious forces or their potentials ϕ, which affect the UAV. The positions to be reached are modeled as attractive poles and obstacles are modeled as repulsive poles for the UAVs. So, a field of forces that affects the complete terrain has to be realized. This can be done using potential values. Therefore, the terrain is divided into sub-terrains, which get potential values. The difference of the potentials of two neighboring areas can be used to model forces. Two types of forces based on two types of potentials exist:

- Attractive forces, which represent goals and pull the UAVs towards them.
- Repulsive forces, which represent obstacles and push the UAVs away from them.

For a proper modeling of potential fields these attractive forces must fulfill two requirements:

1) they have to affect the whole configuration space and
2) they must always lead to the goal.

Figure 1 shows an example for an attractive potential. The goal to be reached is in the middle of the potential field shown in Figure 1 and all surrounding potential values are higher. So, following the descent gradient will always lead to the goal.

Suppose that the first condition is not fulfilled. Then there would exist areas in which no forces act. If the UAV started in such an area, it would never move. Another problem occurs if a UAV flies into such an area. In that case the UAV would usually stop flying. Reaching the goal is not possible then.

The second condition is crucial for reaching the goal, too. If there exist forces, which do not lead to the goal, it would be possible that the UAV selects a trajectory, which makes it impossible to reach the goal. Several possibilities to guarantee forces as demanded exist. One possibility is to

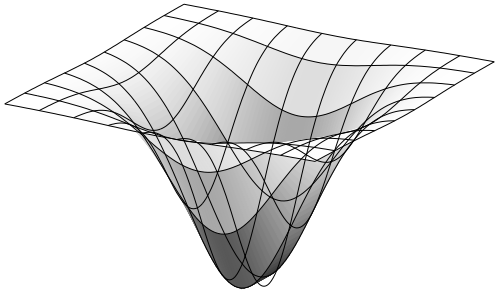


Figure 1. An example for an attractive potential.

adjust the potentials like a cone. The determining value can, e. g., depend on the Euclidean distance to the goal, as it is shown in Figure 1.

Repulsive forces have to fulfill two requirements, too:

1) they have to affect only a specified surrounding area of the obstacle and
2) they must always lead away from the obstacle.

Figure 2 shows an example for a repulsive potential. The obstacle is in the middle of the potential field shown in Figure 2 and all surrounding potential values are less than that of the obstacle. So, following the descent gradient will always lead away from the obstacle.

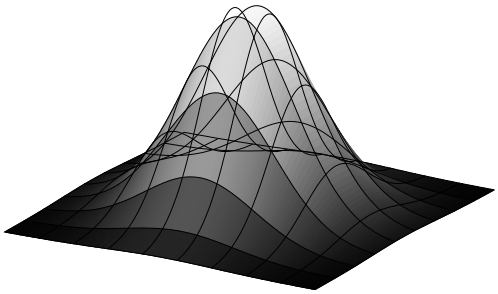


Figure 2. An example for a repulsive potential

The first condition is needed to ensure that obstacles do not affect the complete configuration space. An effect on the complete configuration space can affect the trajectories of the UAVs in such a way that they take unnecessary detours. The second condition ensures that UAVs will not be pulled toward obstacles, which would lead to crashes.

To ensure the two conditions repulsive potentials can be based, e. g., on the Euler's number, as it is shown in Figure 2.

In such a field of forces a path follows the descent gradient within the potential field. It keeps a UAV away from obstacles and pulls it towards the goal.

However, there are some problems when using such potential fields, especially in case of complex and highly dynamic environments. The basic movement in such a field is often done using a gradient method. In this case a UAV is pulled toward the goal. However, obstacles lying around the UAVs trajectory may cause the occurrence of so-called local minima. These minima are one of the most important problems.

A local minimum occurs if a single potential value is less than all surrounding potential values. In that case the driving force vanishes and the UAV gets trapped. Several methods to avoid and to get out of these minima exist. We use harmonic functions [13] for calculating one global artificial potential field. This avoids the generation of local minima.

E. Harmonic Functions

Using harmonic functions for potential value calculation was done first by Connolly and Burns [13] in 1990. One of the advantages of these functions is that one can prove that local minima can be avoided. Additionally, consistent and collision-free paths can be calculated. Harmonic functions satisfy Laplace's equation in n dimensions:

$$\nabla^2\phi = \sum_{i=1}^{n} \frac{\partial^2\phi}{\partial x_i^2} = 0.$$

Namely, ϕ must be two-times differentiable and the second partial derivatives of the potential ϕ must be zero. Additionally, ϕ must be strictly increasing, e. g., dependent on the distance to the goal. The value of ϕ is given on a closed domain Ω in the configuration space $\mathcal{C}$ (Section IV) and satisfies the min-max principle[1] and the uniqueness principle[2] [14], [15]. The min-max principle means that the potential function has its maximum and minimum values at its boundary points (in our case obstacles and goals). So the min-max principle can guarantee that no local minimum can have a potential value less than that of a global minimum and no saddle point can have a greater potential value than an obstacle. The uniqueness principle means that no two areas have the same potential value. This guarantees the absence of flat regions.

Connoly [16] showed that it is possible to generate paths without spurious minima. Combining harmonic functions with Dirichlet's boundary conditions leads to a value-restricted configuration space, which is important for calculations with discrete arithmetic. To respect Dirichlet's

[1]A harmonic function $f(x, y)$ defined on a closed domain Ω takes its minimum and maximum values on the boundary.

[2]If $f(x, y)$ and $g(x, y)$ are harmonic functions defined on Ω such that $f(x, y) = g(x, y)$ for every bound point, then $f(x, y) = g(x, y), \forall x, y$

boundary conditions we bound goal areas with the potential value 0 and occupied areas with the value 1.

Every harmonic function defined on a compact region $\Omega = \partial\Omega \cup \Omega$ satisfies three properties [13]:

1) Every harmonic function is analytic.
2) Select a point q_d to be a goal point with the constraint that $\phi(q_d) = 0$. Set all obstacle boundary points p to some constant $\phi(p) = c$. All harmonic functions satisfy the min-max principle, so ϕ is polar. That means that q_d will be the point at which ϕ attains its minimum value at Ω and all streamlines lead to q_d.
3) If c is set to 1, ϕ will be admissible in our sense. This is a simple normalization.

Additionally, harmonic functions have several valuable properties [13]:

- Completeness up to discretization error
- Fast surface normal computation
- Ability to exhibit different modes of behavior (grazing vs. avoidance)
- Robust control in the presence of unanticipated obstacles and errors
- Lack of spurious local minima
- Linear superpositioning
- Robustness with respect to geometrical uncertainty
- Continuity and smoothness of configuration space trajectories

Completeness in terms of path planning is given if one can guarantee that a path from each initial position to each goal position will be found if such a path exists. It is possible that a coarse discretization disables the finding of a path as shown in Figure 3, where the grey area represents an obstacle. Each leaf, which contains a part of the obstacle is treated as occupied space to ensure collision avoidance. Using a fine discretization as shown in Figure 3(a) makes a path planning from the start to the goal position possible. In Figure 3(b) the coarse discretization makes it impossible to find a route around the obstacle. This discretization errors cannot be compensated by harmonic functions.

Originally, harmonic functions were used for path planning only with a fixed map and one single known goal. But one can show that they also are well suited to dynamic environments with multiple UAVs and goals.

One problem of harmonic functions is their superpositioning. As Connolly mentioned [13], there is no guarantee that in complex or dynamic environments obstacles are avoided when superpositioning is used. The potential values in the neighborhood of an obstacle depend not only on that obstacle's potential, but also on every other obstacle's or possible goals' potentials. If the configuration or the strength changes, the path of a UAV can get infinitely close to the obstacle. The only structure that can be safely modeled as obstacle this way is a point itself, which cannot get affected.

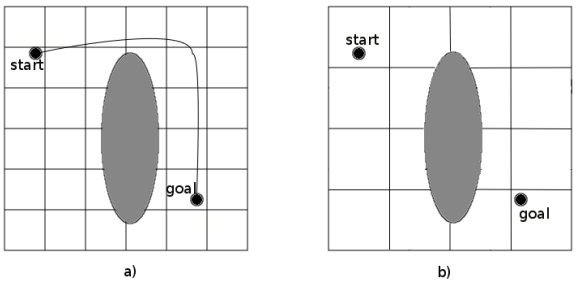


Figure 3. The figure shows two different discretizations of some terrain, including a grey area, which symbolize an obstacle. While in Figure a) a path from the start position to the goal position can be found, using the discretization in Figure b) it is not possible to plan a path from the start to the goal position.

III. Overview of the approach

This section gives a basic overview of our approach. For financial reasons the approach was implemented first as a simulation environment. The detailed description is given in the following sections.

In our work the presented path planning approach is considered to be a centralized one, in terms of using one global configuration space $\mathcal{C}$ to model the world in which the UAVs act. Path planning itself can be done in a centralized manner using a control station or in a decentralized way by the single UAVs.

On the one hand, UAVs have to explore unexplored parts of the terrain, while objects and landmarks must be recognized. Their position and size has to be noticed. Additionally, the UAVs must keep track of regions already visited to plan efficient paths to still unexplored regions of the territory. On the other hand, UAVs have to find collision-free paths from their initial positions to given goal positions. In both cases they act autonomously without any human interaction (except insertion and deletion of goals).

According to [17], it is possible to divide path planning into three general categories:

1) global path planning: A complete path from the position of the UAV to the goal will be computed.
2) local path planning: Several paths which lead towards the goal are planned and compared with each other.
3) reactive path planning: Only the next step is computed dependent on the current situation.

This work presents an approach combining global and reactive path planning for exploratory navigation. Additionally, the global approach is used to find paths to designated goal positions.

To ensure an efficient goal-oriented path planning using multiple UAVs, a coordination of the UAVs is indispensable. The coordination is done using a task allocation system. To ensure this, each UAV has a so-called role (Section IV-B) and schedules the given goals, based on this role.

To explore the terrain and to gain information about it, each UAV has one or two cameras. Different types of cameras, like infrared cameras, are used.

The simulation system currently used to evaluate our approach consists of two types of components, representing a single control station and the UAVs, respectively. Both make use of the same path planning library. The library contains the gradient method as well as the A*-algorithm and a neighbor finding algorithm. Figure 4 shows an overview of the system.

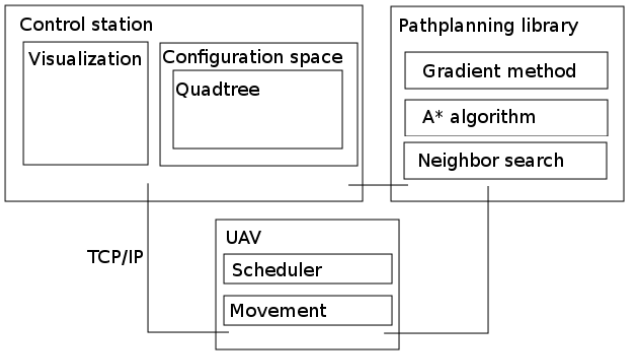


Figure 4. Overview of the designed system.

A. Control station

A control station was realized to represent the configuration space $\mathcal{C}$, which holds the data needed for path planning, including the properties of each UAV and the terrain, like positions of known obstacles, explored and unexplored space, position of the single UAVs, etc.

To distinguish between $\mathcal{C}_{free}$ and $\mathcal{CB}_i$ a subdivision of the terrain is needed. We use a quadtree, which is one of the most commonly used data structures to subdivide 2D terrains. By extending the quadtree to a linear quadtree we are able to find neighbors and to update the structure dynamically in constant time. Dynamic changing of the quadtree is needed to respect changes of the terrain due to exploration and aging of the information about subareas. The quadtree was designed in such a way that the union of the areas represented by all its leaves is the complete terrain.

Based on this subdivision it is possible to use the potential field theory for path planning. Therefore, each leaf gets a potential value, dependent on its distance to goals and obstacles.

The potential values ϕ are used to calculate paths, combining them with a gradient method for exploratory navigation and the A*-algorithm for goal-oriented path planning. For path planning only 2 dimensions are used. With respect to the different camera types of the UAVs, which have different properties, the third dimension, the height of the UAVs over ground is calculated by themselves.

The control station comes with a visualization of the terrain and the UAVs. For this purpose a 3D environment was implemented. It shows the positions of the single UAVs and is able to display their configurations. To display the terrain heightmaps are used. A heightmap is a monochrome picture where the value of each pixel represents the height of a single area of the terrain. The advantage of using heightmaps is that new terrains can be created easily. Additionally, the visualization allows an interaction between the user and the system. It is possible to set up new goals or obstacles and to delete them dynamically during simulation.

B. UAV

The UAVs were implemented as an external program. They consist of the UAV configuration $q_{\mathcal{A}}$ and schedule given goals, based on their role. Additionally, they are able to plan paths, based on a copy of $\mathcal{C}$, they can ask for. Hence, path planning can be decentralized through the UAVs or centralized, as a UAV can ask the control station to plan a path from the UAV position to each goal position.

The UAVs communicate with the control station using TCP/IP. They send their configuration periodically to the control station and can ask for path planning, a copy of the configuration space and additional information.

In our model, a UAV is unable to fly backwards, so it must be able to plan its motion, based on its direction and the goal direction. The movement calculation of a UAV for flying along the calculated path is done by itself.

Additionally, the UAVs have to calculate their height over the terrain. Each UAV has a favorite height with respect to its cameras. The resulting height over ground depends on a UAVs favorite height, the height of the terrain it is currently over and the height of the terrain it reaches next, as it is known so far. Considering also the height of the next terrain leads to a smoother flight when flying over terrains with huge height differences, if for instance steep faces lie in the trajectory of a UAV.

C. Pathplanning library

We developed a library, which can be used by the control station and the UAVs to calculate a path. This ensures that both a centralized path planning by the control station and a decentralized path planning by the single UAVs can be done. Using a library avoids a redundant implementation of the algorithms in the control station and the UAV. Therefore, the library implements the gradient method for exploratory navigation and the A*-algorithm for goal oriented path planning. Additionally, the neighbor finding algorithm is implemented, as it is needed to calculate paths.

IV. Construction of the configuration space

This section gives a more detailed description of the configuration space. We start with the cell decomposition

approach. Afterwards, we present the configuration of a UAV, which is necessary for path planning.

In order to represent the complete scenario, the configuration space has to contain the coordinates and identities of all UAVs, goals, obstacles, known and unknown areas. This section, therefore, describes how the complete area is discretized into leaves using a quadtree. Afterwards, it is presented how additional information regarding the UAVs, like fuel level or individual tasks, are represented. Finally, the section discusses how the recalculation complexity of $\mathcal{C}$ can be decreased to a level at which online calculation is feasible.

A. Quadtree

As mentioned before, the configuration space is divided by a quadtree and each UAV is represented as a point in $\mathcal{C}$. The subdivision is done in several steps. First $\mathcal{C}$ is divided into free space $\mathcal{C}_{free}$ and occupied space $\mathcal{CB}_i$. When representing the UAV as a point in $\mathcal{C}$ the dimensions of the UAVs are not considered. However, to ensure a collision-free path it is necessary to consider their dimensions. This is done using obstacle growing, whereby the occupied space is expanded by the dimensions of the UAVs in all three dimensions.

Finally, the resulting $\mathcal{C}_{free}$ is subdivided into unknown, unexplored, explored and goal space. These space types are represented through different leaf types. This subdivision changes dynamically during runtime.

Note the distinction between unknown and unexplored space. Unknown space represents areas with no information about the terrain height. Unexplored space represents areas without up-to-date information. Hence, during a rescue mission the age of some data is of crucial importance. If the exploration time of a region exceeds a pre-defined threshold, the according region becomes unexplored again, but it can never become unknown again.

The resulting nodes store several properties as shown in Table I.

Property	Description
boundaries	Points to specify the position and the dimensions of the node.
space type	The type of space of the node, e. g., $\mathcal{C}_{free}$ or $\mathcal{CB}_i$.
location code	A unique code, used for neighbor finding with constant time complexity on average.
level	The level of the node (depending on the tree-depth of the node).
potential value	The potential value on which path planning is based.
exploration time	Time point at which the node was explored last time.

Table I
PROPERTIES OF THE QUADTREE NODES

To specify the positions of UAVs and obstacles, a Cartesian coordinate system $\mathcal{FW}$ was embedded into the configuration space $\mathcal{C}$. Hence, each UAV has an associated position and direction within the coordinate system.

The position and dimensions of each node are based on $\mathcal{FW}$, too. Each node is specified through 2 points in space spanning a rectangular region. When extending the quadtree to a linear quadtree a unique code and its level must be saved. In rescue scenarios up-to-date information is necessary. The exploration time is the point in time the node was explored. If the difference between the current time and the exploration time is above a given threshold, the node becomes unexplored again.

B. UAV configuration

Each UAV $\mathcal{A}$ has a configuration $q_{\mathcal{A}}$ within $\mathcal{C}$. It is represented as a 9-tuple $(q_1, \ldots, q_9)$ as shown in Table II.

Tuplenumber	Description
q_1	X - position in space
q_2	Y - position in space
q_3	Z - position in space
q_4	flight direction
q_5	favorite height
q_6	maximum altitude
q_7	role
q_8	task list
q_9	fuel level

Table II
UAV CONFIGURATION ($q_{\mathcal{A}}$)

The position of a UAV in $\mathcal{C}$ is described by three entries and must be unique, because if two UAVs had the same position in $\mathcal{C}$, they would have crashed. For path planning only two entries are used (q_1, q_2). The height of the UAVs (q_3) is neglected as the UAVs can have different favorite heights over ground. Of course the height is important to avoid collisions. It is also important for the cameras of the UAVs. The area the UAVs can explore at one moment depends on the flare angle of their cameras and the height of the UAVs above the terrain.

We distinguish two different height types. The favorite height (q_5) is the height above the underlying terrain a UAV wants to reach. The maximum altitude (q_6) describes the height of the UAV above sea level. In the following simulations (Section VII) sea level is modeled as the zero point of the program. The height of a UAV (q_3) is always the height above sea level.

The maximum altitude is used to partition the terrain into $\mathcal{C}_{free}$ and $\mathcal{CB}_i$. Each part of the terrain, which is higher than the maximum altitude, is $\mathcal{CB}_i$, the remaining terrain is $\mathcal{C}_{free}$.

The configuration of a UAV contains the flight direction (q_4). This direction can also be used to provide a preferred direction in which the UAV will fly first while exploring the terrain. The modeled UAVs can turn on the spot so the direction is not too important for motion planning. Previous tests showed that in this case the use of a preferred direction does not lead to better results concerning the exploration rate

or speed. Hence, there is no favorite direction for the UAVs in the resulting system. Two UAVs cannot take the same position at the same time so the entries q_1, q_2 and q_3 are used to distinguish the single UAVs.

The task list includes all given goals. The types of goals specified so far are shown in Table III. A single task always contains one known goal. It can be UAV-specific or non-UAV-specific, e. g., refueling would be a UAV-specific task, whereas monitoring of areas would be a non-UAV-specific one. Goals can be points, which the UAV has to explore or areas, which have to be monitored with high priority. Each goal is associated with one task. When the fuel level reaches a given threshold, the UAV sets up a new task with a fuel station as goal and highest priority. This task will be executed before all other tasks to avoid damage.

Every UAV also has a role, which is used to realize a scheduling of all given tasks. In this work three roles are defined:

1) *explorer*: An explorer UAV is used mainly for terrain exploration. Goal-oriented path planning will be done only to avoid starvation.
2) *seeker*: A seeker UAV is used mainly to reach given goal points. As long as goals exist goal-oriented path planning will be done and exploration takes place only if no specific goals exist.
3) *surveillant*: A surveillant UAV is used mainly for goal-oriented path planning, too. The difference to the seeker UAV is that it monitors areas rather then flying to given goal points.

C. Decentralized task allocation for multiple UAVs

A role is associated with a specific priority scheme. Each task gets a priority $\mathcal{P} \in \{0, 1, \cdots, 9\}$. The higher $\mathcal{P}$ is the more important the task is. Depending on the role, the task list will be sorted according to the different task priorities. Explorer UAVs first explore the terrain and target specific goals with less priority. Seeker UAVs favor goal points over areas and exploration. Surveillant UAVs favor goal areas over points and exploration. The priority schemes of the seeker and the surveillant roles initialize tasks like goal points or goal areas with high priority to favor these tasks over exploration. Table III shows the implemented scheduling scheme.

Goal	Explorer	Seeker	Surveillant
Monitor areas	2	2	3
Exploration	3	0	0
Fly to goal point	1	3	2
Refueling	9	9	9
Landing	8	8	8

Table III
INITIAL PRIORITIES OF TASKS

The higher the priority the more important the task becomes. If several tasks have equal priority, the processing order of these tasks depends on the position of the corresponding goals. Tasks with goals closer to the UAV are executed first. This ensures fast completion of the tasks and additionally, they are scheduled in such a way that tasks with less distance to each other than to other tasks but equal priority will be processed consecutively.

We distinguish two types of tasks. High priority tasks ($\mathcal{P} > 4$) and low priority tasks ($\mathcal{P} \leq 4$). High priority tasks are tasks, which must be processed as fast as possible as not-processing them will lead to damage of the UAV.

However, this static role assignment does not avoid starvation. That means that if, e. g., only surveillant UAVs are in use and new tasks to monitor goal areas appear faster than the old ones can be finished, flying to goal points would never be executed. To avoid such a behavior the priorities of low priority tasks increase during time. To ensure that these tasks never preempt high priority tasks the increasing stops if $\mathcal{P} = 4$, which is higher than every initial priority of low priority tasks.

Scheduling of tasks, assignment and increasing of the priorities and creation of UAV specific tasks is always done in a decentralized manner by the UAVs.

V. CALCULATING THE POTENTIAL FIELD FOR PATH PLANNING

In this section we describe how the potential values are calculated. This is a two-step procedure. We start calculating a first approximation. Afterwards, we successively enhance the results of the first approximation until we get potential values, which are feasible for path planning. Our approach for potential field calculation has the disadvantage of being quite costly. Hence, to lower the calculation costs in such a way that we can make every calculation without noticeable delay we use several cost reduction techniques, which are described at the end of the section.

We assume that the control station, responsible for coordinating the UAVs, provides a single configuration space $\mathcal{C}$, which is used as a representation of the complete scenario. $\mathcal{C}$ is based on a quadtree (Section IV-A), whereby the leaves carry descriptive information, like a potential value. The quadtree subdivides the terrain in such a way that the union of the leaves represents the complete terrain. Each leaf of the quadtree gets one single potential value. Based on these values, the UAVs decide which leaves they use to find a trajectory to their designated goal position.

Additionally, $\mathcal{C}$ includes all UAVs with their configurations (Section IV-B), every goal and obstacle.

The intention is to maintain only one harmonic function, which describes the entire potential field, concerning multiple goals as well as multiple UAVs, avoiding the negative effects of linear superpositioning of harmonic functions described in [13].

Therefore we model the potential field as a discrete Dirichlet problem for harmonic functions, which always has

a solution and we try to solve it using a relaxation technique. As a relaxation technique to transform the potential field update under Dirichlet boundary conditions the Gauss-Seidel method [18] is used.

To make use of the advantages of harmonic functions, in particular the absence of local minima, the potential values of the configuration space have to fulfill several requirements, e. g., potential values must strictly increase with the distance to the goals, the second partial derivatives have to vanish and so on. To take into account all these restrictions, the calculation process is modeled as an optimization problem to solve the discrete Dirichlet problem.

A. First Approximation

We distinguish two types of potential values ϕ. Bound and unbound values. Bound values are fixed values for goal and obstacle areas, which are set before the calculation starts and they will not be changed during the calculation. These values are the boundaries. Thereby we are able to ensure that obstacles have maximum and goals minimum values as harmonic functions satisfy the min-max principle. We use a potential value range from 0 to 1. Obstacles are bound with 1 and goals with 0.

Every other leaf $u(x, y)$ of the quadtree gets a so-called unbound potential value $\phi(x, y)$, which is greater than 0 and less than 1. These values are calculated for each leaf that is neither obstacle nor goal space. Therefore, we use the following equation. It depends on the distances to the nearest goal area, in sense of Euclidean distance.

$$\phi(x, y) = \xi \cdot \frac{\log\left(\tau\left(x, y\right)\right)}{\log(d)}. \tag{1}$$

Here $\tau(x, y)$ represents the Euclidean distance from the point $(x, y) \in \mathcal{C}$ to the nearest target point. The logarithm of the diagonal d of the complete terrain is used to normalize the values between 0 and 1. $\phi(x, y)$ satisfies Laplace's equation.

For goal-oriented mode it is possible to trade off shortest path calculation and gathering additional information by taking a detour over unexplored regions. To take this into account the ξ-Value was introduced. The value is between 0 and 1 and is set by the user. The lower the value of ξ is, the more attractive unexplored terrain becomes for the path planner. In case of non-goal-oriented path planning (exploratory navigation) ξ is set to 1.

As mentioned, Equation 1 is used to calculate a first approximation for every unbound potential in the environment. It is only a first approximation because not every bound value was respected. This leads to a linear system of equations.

B. Update equation

The first approximation does not consider obstacle areas. But that is necessary to ensure collision avoidance and to guarantee that the potential values are represented through a single harmonic function with its advantages, like reducing the number of local minima. To take this into account the first approximation of the unbound values given by Equation 1 is successively enhanced using a relaxation method. In our work the Gauss-Seidel algorithm is used for relaxation.

Numerical solutions for Laplace's equations can be found through finite differentiation methods. This is possible as the value of a harmonic function at each point is the arithmetic mean of the values of its surrounding points [15]. We use the neighbors of the following four of the existing eight neighbor directions to update the potential value of a leaf $u(x_i, y_j)$:

1) $u(x_{i+1}, y_j)$,
2) $u(x_{i-1}, y_j)$,
3) $u(x_i, y_{j+1})$,
4) $u(x_i, y_{j-1})$.

If we had only equally sized nodes, e. g., when using a grid with equal sized cells, we could calculate the potentials $\phi(x_i, y_j)$ using the following function:

$$\begin{aligned}\phi(x_i, y_j) = \quad & \tfrac{1}{4}[\phi(x_{i+1}, y_j) + \phi(x_{i-1}, y_j) \\ + \quad & \phi(x_i, y_{j+1}) + \phi(x_i, y_{j-1})].\end{aligned}$$

However, when using a quadtree to partition $\mathcal{C}$ as in our approach, the dimensions of the part of the terrain each leaf represents have to be considered. Another point is that through the use of a quadtree a leaf can have several neighbors per direction. In that case we make an approximation and use the mean distance and the mean potential value of the neighbor nodes. We take into account the following distances from the current leaf $u(x_i, y_j)$ to its used neighbors:

$$\begin{aligned}\tau_{right}&: \text{ distance from } u(x_i, y_j) \text{ to } u(x_{i+1}, y_j),\\ \tau_{left}&: \text{ distance from } u(x_i, y_j) \text{ to } u(x_{i-1}, y_j),\\ \tau_{up}&: \text{ distance from } u(x_i, y_j) \text{ to } u(x_i, y_{j+1}),\\ \tau_{down}&: \text{ distance from } u(x_i, y_j) \text{ to } u(x_i, y_{j-1}).\end{aligned}$$

This leads to the following update equation, which is based on [19]:

$$\begin{aligned}\phi(x_i, y_j) = & \frac{\tau_{up}\tau_{down}\left[\tau_{right}\phi(x_{i+1}, y_j) + \tau_{left}\phi(x_{i-1}, y_j)\right]}{\tau_{up}\tau_{down}(\tau_{right} + \tau_{left}) + \tau_{right}\tau_{left}(\tau_{up} + \tau_{down})} \\ & + \frac{\tau_{right}\tau_{left}\left[\tau_{up}\phi(x_i, y_{j+1}) + \tau_{down}\phi(x_i, y_{j-1})\right]}{\tau_{up}\tau_{down}(\tau_{right} + \tau_{left}) + \tau_{right}\tau_{left}(\tau_{up} + \tau_{down})}.\end{aligned} \tag{2}$$

Neighboring leaves are all leaves, which have a border in common. Hence, possibly different leaf sizes are also taken into account. The potential value of the current leaf is represented through $\phi(x_i, y_j)$, the potential of the left neighbor through $\phi(x_{i-1}, y_j)$ and so on.

This modeling leads to a system where the UAVs are driven away from obstacles and led to the goals. Figure 5 shows a sample potential field, calculated using this optimization method. For better visualization the calculated

potential field was mapped to a grid in Figure 5. Here the UAV is in the left upper corner, the goal to be reached is in the lower right corner. The higher parts of the terrain represent occupied space, whereas the lower ones are free space. The UAV will now follow the descent gradient, which always leads it to the neighboring space with the lowest potential value, until the goal is reached.

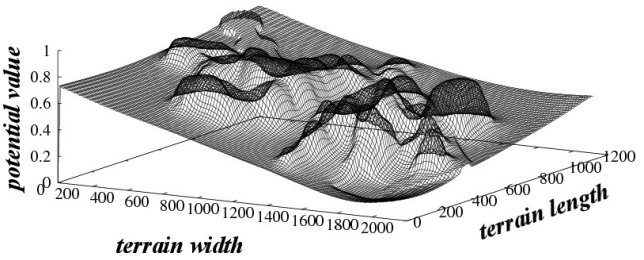


Figure 5. A sample potential field

To reach a given error rate with respect to a perfect harmonic function of $\epsilon \leq 10^{-p}$ when using M leaves for calculation, $\mathcal{O}(pM)$ iterations are needed. Intuitively, choosing a quadtree instead of a grid lowers the calculation costs. However, for large terrains, together with the demand for detailed resolution, this method is still very costly. Several methods reducing these costs are presented in the following section.

C. Cost reduction

The methods presented in the previous sections ensure complete exploration, reaching of given goals and task allocation. These methods are rather complex, especially the recalculation of $\mathcal{C}$. A first approach for cost reduction was an intelligent implementation (Section V-D). When calculating the first approximation a pointer array is created, which holds the leaves to be updated with pointers to their corresponding neighbors. So, iterative loops through this array are used for the update equation, instead of recursively cycling through the quadtree with all nodes. Saving pointers to the neighbors avoids redundant neighbor finding as the quadtree does not change during potential calculations.

However, without any cost reduction techniques the methods still do not well suit embedded systems. Another part for cost reduction is to make recalculations of $\mathcal{C}$ event based. So, recalculation will be done only for the following reasons:

- Insertion of new targets or obstacles
- Deletion of targets or obstacles
- Path-planning request after exploration

After a UAV changed the configuration space through exploration it requests a new path. In that case the central control station recalculates the potential values of the global configuration space. Obviously, even this leads to frequent recalculations of $\mathcal{C}$. In addition, the calculation load depends directly on the number of active UAVs. Therefore, the recalculation step must be performed efficiently.

The update equation is used mainly to lower the number of local minima. A complete calculation of the configuration space means in our case that no local minima are left. The lowering of the number of local minima is advantageous since trajectories are calculated in such a way that they lead to minima, even if they are only local minima. For the UAVs this results in a detour or in the worst case the UAVs get trapped in a local minimum. Leaving a local minimum is costly as it has to be detected and a safe path out of the minimum has to be calculated.

The costs to calculate the configuration space are directly related to the number of nodes to be updated. Hence, a quadtree was used instead of a grid, which is very common for representing a terrain when using potential fields. To reduce the costs for neighbor finding in the tree–which is necessary for the update equation and the path planning methods–the tree was extended to a linear quadtree. To transform a tree into a linear quadtree a unique code and the level of a node is saved for each leaf. An algorithm, which finds neighbors with constant time complexity on average was implemented. This is a further improved version of [20] with the focus on reducing tree editing costs. With only one configuration space for all UAVs, it is crucial to have a data structure, which can be modified easily–usually this is done several times per second. The modification must be done much more frequently than the potential calculations to respect changes in the terrain, like newly recognized obstacles. In contrast to [20] no additional information about the level difference to the neighbors is stored in the nodes. This significantly increases the editing speed.

Even with this cost reductions, a complete calculation of the configuration space is expensive. Prestes et al. showed [4] that a complete absence of local minima is not necessary for feasible path planning. They used a constant iteration depth to reduce the costs. Based on their results we established break conditions that also allow a dynamic number of leaves. The iterative calculation stops if:

- the potential values remain unchanged,
- the number of local minima is less than 0.5% of the number of updated leaves,
- the iteration depth is greater than or equal to 10% of the number of updated leaves.

The last condition guarantees the termination of the update method. Tests showed that these break conditions lead to an appropriate tradeoff between number of local minima and calculation costs.

Furthermore, leaves with potential values greater than or equal to 0.999 are not considered as local minima. Tests showed that such values usually occur in the neighborhood of newly explored obstacles. Usually such potential values are greater than the potential value of areas, which the UAVs

take. This leads to a significant reduction of the calculation costs.

If a UAV gets trapped in a local minimum, the A*-algorithm is used to leave it. Therefore, the A* searches through the quadtree to find an unexplored area. It takes the first area found and sets it as the next target. Afterwards, a goal-oriented path planning to this area will be done.

For further cost reduction in non-goal-oriented mode, an activation window was established. First all nodes are inactive. After the exploration of an area the newly explored leaves are marked as active. After a given time, leaves can be marked as unexplored again and are no longer active. The first approximation and the following updates will be done only for active leaves.

D. Implementation details

The implementation of the calculation algorithm consists of three main methods. The first method, which is shown as pseudo-code in Listing 1, is used to start the first approximation and to ensure an iterative accomplishment of the update equation until the break conditions are reached. As our first approximation is based on goal distances we have to set up goals even in exploratory navigation. Therefore, we calculate the nearest unexplored or unknown space for each UAV and set this spaces as next targets to be reached. This is done in the method *SetNearestUnexploredAsNextGoal*. While processing the loop for the update equation we check the number of local minima each cycle. Therefor the method *CheckNumberMinima* is called, which checks for each node to be updated if there exists at least one neighbor node with a lower potential value. If one of the break conditions introduced in Section V-C is fulfilled the while loop, which calls the update equation, stops.

```
CalculatePotentialField()
{
  if(exploratory_navigation)
    SetNearestUnexploredAsNextGoal()

  leaf_number = 0
  FirstApproximation(Rootnode)

  max_minima = update_list.length() * 0.005
  number_minima = max_minima + 1
  max_depth = update_list.length() * 0.1
  iteration_depth = 0
  value_change = 1

  while(number_minima > max_minima &&
        iteration_depth <= max_depth &&
        value_change)
  {
    iteration_depth++
    value_change = UpdateEquation()
    number_minima = CheckNumberMinima()
  }

}
```

Listing 1. Pseudo-code of the main function for potential field calculations.

```
FirstApproximation(Treenode)
{
  if(Treenode not leaf)
  {
    for(i = 0; i < 4; i++)
      FirstApproximation(Treenode.child(i))
    return
  }

  if(Treenode.space == Target)
  {
    Treenode.potential = 0
    return
  }

  if(Treenode.space == Obstacle)
  {
    Treenode.potential = 1
    return
  }

  if((Treenode.space == Unexplored ||
      Treenode.space == Unknown)   &&
      exploratory_navigation)
  {
    Treenode.potential = 0
    return
  }

  update_list[leaf_number] = Treenode
  update_list[leaf_number].potential = log(
       target_distance)/log(terrain_diagonal)

  if(Treenode.space == Unexplored ||
      Treenode.space == Unknown)
    update_list[leaf_number].potential *= xi

  SaveNeighborNodes()

  for each(direction)
  {
    distance[direction] = SumDirections()
    distance[direction] /=
      num_neighbors[direction]
  }

  update_list[leaf_number].udr = distance[up]*
       distance[down]*distance[right]
  update_list[leaf_number].udl = distance[up]*
       distance[down]*distance[left]
  update_list[leaf_number].rlu = distance[right]*
       distance[left]*distance[up]
  update_list[leaf_number].rld = distance[right]*
       distance[left]*distance[down]
  update_list[leaf_number].direction =
      update_list[leaf_number].udr
    + update_list[leaf_number].udl
    + update_list[leaf_number].rlu
    + update_list[leaf_number].rld

  leaf_number++
}
```

Listing 2. Pseudo-code of the function to calculate the first approximation

The method used to calculate the first approximation (Listing 2) has several additional requirements. We move recursively through the complete tree but only the leaves are considered. First, we check if a node consists of target space, occupied space or in exploration mode unexplored

space or unknown space. If so, we bound the potential value of the leaf with the corresponding value. If not, we need to calculate an unbound value for the leaf. In that case we use the first approximation equation from Section V-A.

We profit from the fact that there are no quadtree changes during the calculations. We use this property in several ways. As it makes no sense to move through the complete quadtree in each update iteration we first set up a list, where we save pointers to the leaves to be updated later, called *update_list*. As there are no tree changes also the neighbors of the leaves do not change. That makes it possible to calculate them once and save pointers to the neighbor nodes using a method called *SaveNeighborNodes*. It is possible for a node to have multiple neighbors in some directions. In that case we use an approximation for the following update equation as we only save the mean distance to the neighbor nodes in each direction. Finally, we calculate all relevant distances needed for the update equation once and save them, too.

```
UpdateEquation()
{
  value_change = 0
  for(i = 0; i < update_list.length(); i++)
  {
    old_potential = update_list[i].potential

    for each(direction)
    {
      potential[direction] = SumPotentials()
      potential[direction] /=
  update_list[i].num_neighbors[direction]
    }

    new_potential  = update_list[i].udr *
         potential[right]
    new_potential += update_list[i].udl *
         potential[left]
    new_potential += update_list[i].rlu *
         potential[up]
    new_potential += update_list[i].rld *
         potential[down]

    update_list[i].potential = new_potential /
         update_list[i].distances

    value_change+= |old_potential - new_potential|
  }

  return value_change
}
```

Listing 3. Pseudo-code of the function to calculate the update equation

As we made several calculations, needed for the update equation, which is shown in Listing 3, once after calculating the first approximation, the implemented update equation itself is not as complex as it seems to be in Section V-B. We now can use the *update_list* to access the nodes to be updated directly. In case we have several neighbor nodes in one direction we made an approximation in such a way that we use the mean potential value of the neighbors for calculation. After that we just have to multiply the resulting potential values with the relevant distances from Equation 2, sum them, and divide the result through the given complete distance value. Finally, we calculate the difference between the old and the new potential value to check if the potential values change or if a break condition is fulfilled.

VI. Path Planning Approach

In this section we present our approach for path planning. We distinguish between reactive path planning for exploratory navigation and global goal-oriented path planning. Additionally, in case of goal-oriented path planning a coordination method is introduced.

In contrast to the calculation of potential values ϕ, the path planner consider all leaves in all eight directions of the current leaf.

A. Planning paths for single UAVs

Based on the potential field stored in the configuration space $\mathcal{C}$ as described in the previous section, discrete path planning is possible. Here two different approaches are utilized:

1) Gradient based path planning for exploratory navigation (non-goal-oriented)
2) The A*-algorithm for goal-oriented path planning

Since the UAVs are considered to be heterogeneous, e. g., in terms of camera or other equipment, each UAV must be able to reach every point of the terrain. Therefore, it is impossible to assign subareas of the terrain to a single UAV.

In goal-oriented mode sometimes an additional exploration should be done, even if it leads to detours. The length of the detours can be limited. Therefore, two ξ-values are used. One is used during the potential field calculation (Section V-A), the other by the A*-algorithm. Depending on the priority of the goal, the algorithm uses Equations 3 or 4 to calculate the costs (the selection is explained below).

$$f = f_{pre} + \phi(p_{new}) \cdot \xi + \frac{\tau_{goal}}{2d}. \quad (3)$$

$$f = f_{pre} + \tau_{pre} + \tau_{goal}. \quad (4)$$

The costs to fly to the current leaf are f. The current leaf is the leaf, which is currently considered by the path planner. The costs for the start leaf are set to 0. The less f is, the more attractive the leaf becomes for the path planner. Furthermore, f_{pre} describes the costs to reach the predecessor. The potential value of the current node is $\phi(p_{new})$ and for unexplored regions $\xi|0 < \xi \leq 1$ is used to lower the costs even more, dependent on the role of the UAV ($\xi = 1$ for explored regions).

Additionally, the algorithm uses two distances for the calculation: an estimated distance from the current leaf to the goal leaf, τ_{goal}, and the actual distance from the predecessor to the current leaf, τ_{pre}.

Equation 3 is used, if the goals are of low priority ($\mathcal{P} \leq 4$). In this case the costs depend mostly on the potential values and unexplored regions become more attractive. The

complete diagonal of the terrain d is used to normalize the distance values to values between 0 and 1. Dividing by the double diagonal lowers the values even more, which makes the potential values more important for path planning.

Equation 4 is used, if the goals are of high priority ($\mathcal{P} > 4$). In this case it is necessary to reach the goal as soon as possible. The equation achieves this by setting the costs in such a way that the shortest path will be computed.

B. Goal-oriented coordination of multiple UAVs

It is possible that multiple UAVs set the same goal as the next task to process. This behavior should not be completely avoided as a UAV, which sets the goal later than others, may be closer to it, which reduces the time to reach it. After a UAV reached the goal the others are informed that the goal was reached and they remove it from their lists.

However, if multiple UAVs, which are close to each other, select nearly simultaneously the same goal as next goal, this behavior is not efficient. In this case these UAVs would calculate nearly the same paths. Parallel exploration would neither yield considerable information gain, nor would the goal be reached significantly earlier. To avoid this undesired behavior an error handling was implemented. For the handling $dist_{UAV}$ denotes the distance between a subsequent UAV_i and the position of the UAV_1, which did the first calculation for this goal at time T_1. Additionally, $dist_{G_1}$ represents the distance of UAV_1 to the goal at T_1. t_t denotes the minimum time UAV_1 needs to reach the goal. If a UAV calculates a path to the goal position G_1 at time T_1 and another UAV calculates a path to G_1 at time T_i as well, the error handling starts. This handling avoids the calculation of nearly equal paths. It takes into account the points in time T_1 and T_i as well as the distances $dist_{UAV_i}$ and $dist_{G_1}$ in the following way:

$$\left(T_i - T_1 < \frac{t_t}{2}\right) \wedge \left(\frac{dist_{UAV}}{2} < dist_{G_1}\right).$$

If the formula becomes true, a subsequent UAV_i sets the priority for this goal to 0 and moves it to the end of its scheduling list. Hence, even if the UAV, which has set the goal first is not able to reach it, no starvation of that goal would occur.

A more simple way would be to avoid multiple path planning to a single goal. But a goal should be processed as soon as possible and as already mentioned, the first UAV, which makes a path planning, is not necessarily that one, which can reach the goal first. Goals with equal priority are sorted in an ascending order with respect to the UAV's distance to the goals, such that the possible detour to start flying to a goal, which is reached by another UAV first should be not quite as long.

The information about the used terrain can be obtained in two ways. One way is to get the data through third party applications like satellite images. Additionally, the UAVs have on-board cameras for the exploration of unknown areas and to update the information available so far.

VII. Results

For financial reasons the approach was implemented first as a simulation environment and several tests were done to check the costs for calculation and to check if it is able to fulfill all given requirements. The implementation consists of a control station, which models the world and visualizes it in 3D. It was implemented in C++ using OpenGL. Additionally, UAVs were simulated, which connect to the simulation environment using TCP/IP.

In this section we present several tests of the designed system. The focus of our tests was the exploration of terrains. Goal-oriented path planning with parallel exploration was neglected as those tests are presented in [21]. We mainly focus on the time needed to explore given terrains using different numbers of UAVs and the costs for the potential field calculations.

The used terrains were partitioned into $\mathcal{C}_{free}$ and $\mathcal{CB}_i$ in such a way that the UAVs were able to reach the complete free space.

The following tests were done to demonstrate that the system assures a complete exploration of a given terrain. Even by using only a single configuration space the use of multiple UAVs leads to faster exploration rates. Additionally, by using the cost reduction methods introduced in the previous section all calculations can be done online. For testing, a 3D simulation was created, which represents the terrain with the corresponding configuration space and the UAVs. The simulations were run on a desktop PC with a dual core 2.6 GHz CPU.

The tests include the calculation costs for $\mathcal{C}$. The time needed to calculate the configuration space during the different tests is shown in Figure 8(c), Figure 9(c) and Figure 10(c). Each calculation contains the following tasks:

- Assignment of the next terrains to be explored
- Bind goal space and occupied space
- Calculation of the distances to the next goal space
- Calculation of the first approximation
- Determining all neighbors of the relevant leaves
- Update of the potential values
- Check for break conditions

A. Used terrains

The tests were executed on two fictive maps represented as heightmaps (monochrome pictures, see Section III-A) with simulated UAVs. For exploratory navigation the following input parameter are relevant:

- Dimensions of the terrain
- Speed of the UAV
- Maximum altitude
- Size of the heightmap in pixels
- Maximum depth of the quadtree

- Distribution of $\mathcal{C}_{free}$ and $\mathcal{CB}_i$
- Number of UAVs

The first terrain (Figure 6) was represented through a heightmap with a resolution of 256×256 pixels. The underlying quadtree had a maximum resolution of 2×2 pixels per leaf, which led to a tree depth of 7. A terrain of $1000\ m \times 1000\ m \times 160\ m$ was simulated. The UAVs had a maximum altitude of $140\ m$. So, everything above $140\ m$ was treated as occupied space. In our test scenarios $\mathcal{C}$ was partitioned into $\mathcal{C}_{free}$ (72%) and $\mathcal{CB}_i$ (28%). The UAVs flew with a speed of $60\ km/h$ and had a favorite height of $40\ m$ over ground. To explore the terrain, the UAVs used a camera with a flare angle of $90°$. When using such a camera the UAVs were able to explore $5026.55\ m^2$ at one moment when they had reached their favorite height.

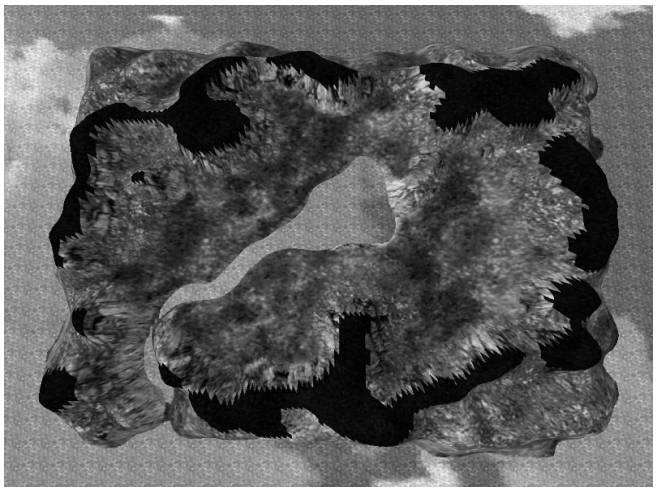


Figure 6. The first test terrain. The black areas are occupied space, the remaining areas are free space.

The second terrain (Figure 7) was represented through a heightmap with the same resolution. The underlying quadtree had a maximum resolution of 2×2 pixels per leaf. A terrain of $1000\ m \times 1000\ m \times 200\ m$ was simulated. The UAVs had a maximum altitude of $120\ m$. So, everything above $120\ m$ was treated as occupied space. In our test scenarios $\mathcal{C}$ was partitioned into $\mathcal{C}_{free}$ (65%) and $\mathcal{CB}_i$ (35%). The UAVs flew with a speed of $60\ km/h$ and had a favorite height of $20\ m$ over ground. To explore the terrain the UAVs used a camera with a flare angle of $90°$. When using such a camera the UAVs were able to explore $1256.64\ m^2$ at one moment when they had reached their favorite height.

The environment is able to consider the time points at which an area was explored. This ensures that areas, which were explored a given time ago become unexplored again to check whether they hold new information. During the following tests this behavior was disabled and areas explored once never became unexplored again.

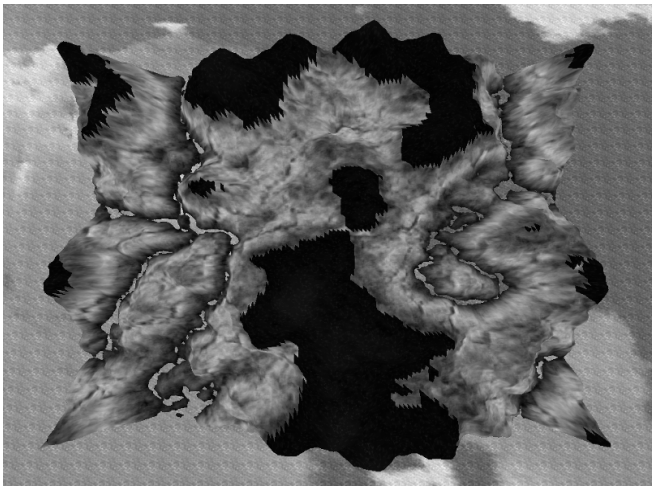


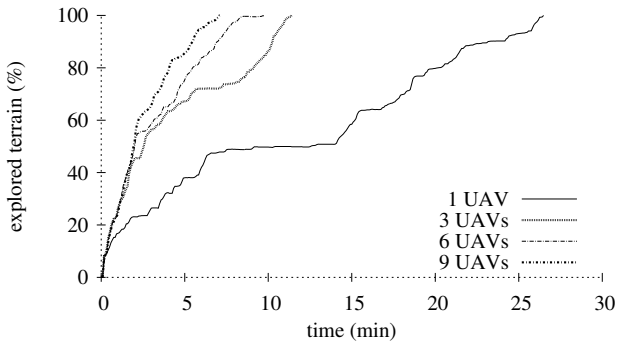
Figure 7. The second test terrain. The black areas are occupied space, the remaining areas are free space.

B. Test results: first heightmap

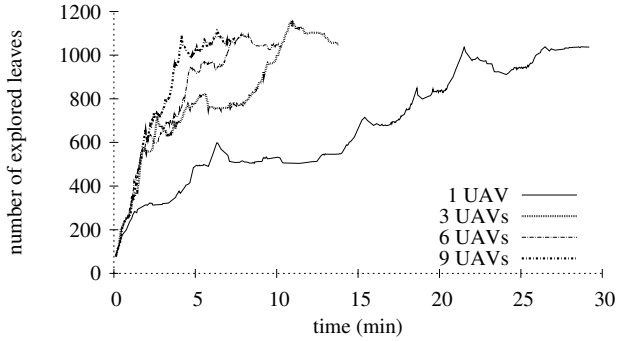
Several tests are presented in this paper. The first test series consists of exploring the terrain. The exploration was done by 1, 3, 6 and 9 UAVs. The partitioning of the terrain was unknown at the beginning. The UAVs had to recognize the occupied space through their cameras. For all tests the exploration was non-goal-oriented. Therefore, no areas had to be explored with high priority and the exploration continued as long as unexplored terrain reachable by the UAVs existed. Mostly, the gradient method was used to reduce the costs for path planning. Additionally, the A*-algorithm was used whenever UAVs got trapped in local minima, which occurred very rarely due to the use of harmonic functions for potential field calculation (Section V-A). Figure 8 depicts the results of the first test series.

Figure 8(a) shows the percentage of explored terrain covered relative to the exploration duration. In every test the complete terrain was explored. By using three UAVs instead of one only half the time for the exploration was needed. Increasing the number of UAVs always leads to a lower time needed for exploration. But tripling the number of UAVs again to nine did not reduce the exploration time by half again, because the more UAVs were used the more often they constrained each other and collisions had to be avoided. This needs time in which the UAVs are not available for exploration. Another point is that the less unexplored terrain is left the more UAVs start to fly to the same terrain to explore it.

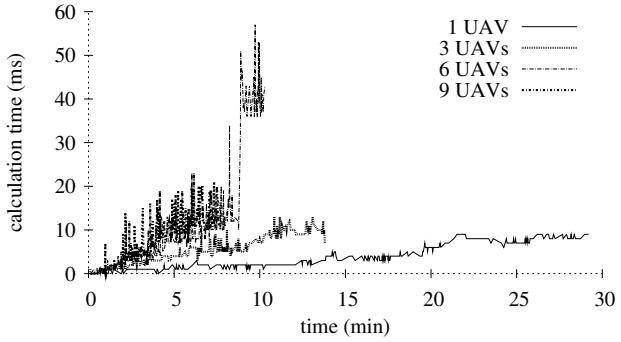
Additionally, the figure shows that until an exploration rate of 70% was reached there are only a few time spans in which the UAVs did not explore large areas of the terrain. This lack-of-exploration behavior occurred after the UAVs explored a large terrain and had missed a few small areas or when the next unexplored area was far away from the UAVs.



(a) Percentage terrain covered



(b) Number of utilized leaves for the calculations of $\mathcal{C}$



(c) Calculation times for single calculations of $\mathcal{C}$

Figure 8. Test result for the first test series

This shows that there is some room for further improvement for faster exploration.

Figure 8(b) shows the number of explored leaves related to the exploration duration. These leaves are those, which have to be updated. It shows that a maximum of about 1200 leaves had to be updated. The number of used leaves increased very fast at the beginning and decreased after a while because the the quadtree's recombining. As shown in the figure the maximum number of leaves remains nearly the same, independent of the number of used UAVs.

Figure 8(c) depicts the costs for the single calculations of the complete configuration space. The costs rise with the exploration rate because of the activation window. The more areas are explored, the more leaves are active and have to be considered for the calculations. But besides this, the costs are relatively constant except for a few spikes. They increase with the number of UAVs, which makes the environment more complex and leads to higher costs. For instance, recalculation is done every time a UAV requests a new path for exploration. So, more UAVs lead to more recalculations, which also increases the complete calculation costs. But the costs are in most cases below 20 ms, which allows online calculations.

A second test series was done for the first heightmap (see Figure 6). Six tests were made to check whether if it makes a difference if the terrain is known or not at the beginning. Additionally, it was checked if it makes a difference if all UAVs start from the same position or from different ones.

For the tests the following six scenarios were created:

1) Test 1: 1 UAV in known terrain.
2) Test 2: 3 UAVs in known terrain and all started from the same position.
3) Test 3: 3 UAVs in known terrain with three different start positions.
4) Test 4: 1 UAV in unknown terrain.
5) Test 5: 3 UAVs in unknown terrain and all started from the same position.
6) Test 6: 3 UAVs in unknown terrain with three different start positions.

Table IV gives an overview of the results. The following values were determined:

- test: The number of the test with the properties described above.
- test duration: The duration it took to explore the terrain.
- iteration depth ∅: The average number of iterations for the calculation of the configuration space.
- number calculations: The number of recalculations of the configuration space.
- costs complete: The complete time needed to make all recalculations of the configuration space.
- costs ∅: The average time a recalculation of the configuration space took.
- diff ∅: The average time between two recalculations of the configuration space.

As expected, the tests with unknown terrain (tests 4 - 6) needed more frequent recalculations of $\mathcal{C}$. A recalculation was done each time new occupied space was detected, what never would happen in known terrains as the occupied space is known at the beginning.

By the use of the break conditions and the introduction of an activation window (Section V-C), a single calculation of the configuration space took relatively little time. Except in the second test case the costs were on average less than 5 ms and this result shows that the calculations can be done before each path planning without causing a noticeable

test	test duration (min.)	iteration depth ∅	number calcula-tions	costs complete (ms)	costs ∅ (ms)	diff ∅ (s)
1	33.75	6.27	92	437	4.75	22.01
2	17.08	7.49	146	861	5.90	7.02
3	19.16	8.14	142	615	4.33	8.10
4	29.33	9.63	302	1337	4.43	5.83
5	13.83	7.99	303	1508	4.98	2.74
6	16.50	8.95	345	1672	4.85	2.87

Table IV
RESULTS OF THE SINGLE TEST CASES FOR THE SECOND TEST SERIES ON THE FIRST TERRAIN.

delay. The higher costs in test case two are the result of several repeatedly computed local minima, which needed many iterations of the update equation to vanish. Because of the higher number of explored nodes at the end of the exploration, the single iterations took a relatively long time.

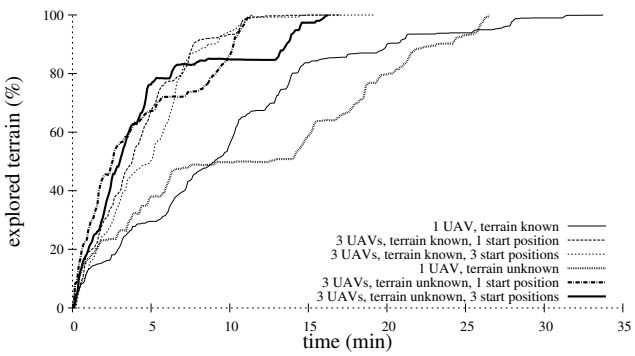
Since a recalculation of the configuration space takes place only before a new path for exploratory navigation is planned, the frequency of recalculating the configuration space was lowered. The first test case needed the fewest number of recalculations. In this test a recalculation was done on average every 22.01 seconds. In the worst case every 2.74 seconds (in test case five) such a recalculation took place.

The results show that the disadvantage of harmonic functions–the high calculation costs–compared to other methods for potential field calculation, which often use linear superpositioning, were reversed.

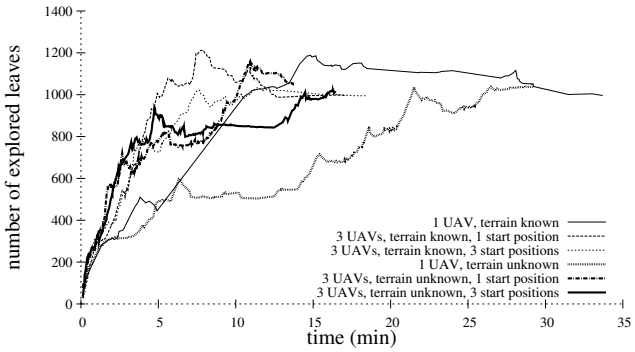
In Figure 9 the test results for the first heightmap are depicted. Figure 9(a) shows the exploration rate compared to the simulation time, which starts at 0 and is measured in real-time. Figure 9(b) shows the number of nodes, which are needed to calculate the configuration space at the single time points. This number has direct influence on the single calculation costs, which are shown in Figure 9(c).

One can see that the exploration was made relatively steady over time, especially during the tests with three UAVs. It made no big difference if all UAVs started from the same position or from three different start positions. The use of three UAVs made an advantage up to an exploration rate of 99% concerning the exploration speed, compared to the use of one single UAV. After a simulation time of, e. g., 7 minutes in the first test case, one single UAV had explored 39.65% of the terrain. When three UAVs made the exploration using the same start parameters they had explored 80.13% in test case two, respectively 81.39% of the terrain in test case three, in the same time span.

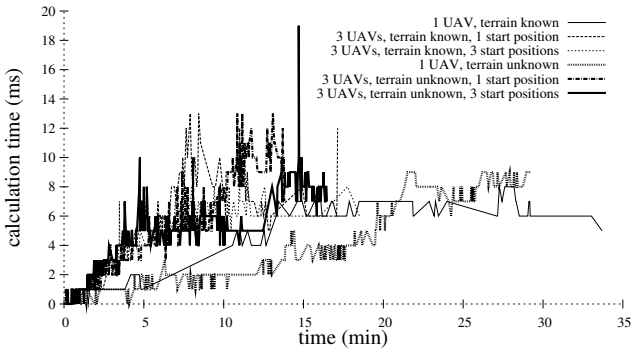
In test cases one and four, where only one UAV was used, it explored the terrain with a steady exploration rate until an exploration level greater than 85% was reached. After that, the exploration rate decreased dependent on time due to the fact that the UAV had to fly over explored terrains to reach the residual unexplored areas.



(a) Percentage terrain covered (first heightmap)



(b) Number of utilized leaves for the calculations of $\mathcal{C}$ (first heightmap)



(c) Calculation times for single calculations of $\mathcal{C}$ (first heightmap)

Figure 9. Test result for the first heightmap

In test case four such a steady exploration rate was not reached permanently. During these tests the UAV had to fly over explored terrains to reach the residual unexplored areas when only a relatively small portion of the terrain was explored. This behavior was observed only for a certain time span. After that, the exploration rate was steady again. Apart from that, a complete exploration of the free space was reached in each test.

Figure 9(c) depicts the costs to calculate the configuration space. One can see again, that the costs rise with the exploration rate of the terrain. This is due to the use of the activation window, which activates more leaves to be

considered for the calculation of $\mathcal{C}$. This behavior can also be seen in Figure 9(b), which depicts the number of utilized leaves for the single calculations. As mentioned before, the behavior that explored terrains can become unexplored again was disabled. Therefore, the number of active nodes only decrease when they are combined.

The calculation costs for $\mathcal{C}$ are, apart from a few spikes, consistent. The only serious anomaly was in the sixth test with 19 ms. This anomaly was caused by several local minima, which needed several iterations to vanish. Except for this spike all other recalculations were between ≤ 1 ms and 10 ms. This shows that no noticeable delay of the exploration occurred, due to the calculations of the configuration space. Combined with the gradient method, which ensures that a route can be started after eight comparisons, this ensures fast path planning.

C. Test results: second heightmap

The six tests from the second test series were repeated on a second terrain (Figure 7) to ensure that they hold for other terrains as well. They are divided in the same six single tests as for the first heightmap.

Table V gives an overview of the results. The values that were determined are the same as for the first heightmap.

test	test duration (min.)	iteration depth ∅	number calculations	costs complete (ms)	costs ∅ (ms)	diff ∅ (s)
1	35.50	7.18	113	334	2.96	18.85
2	19.50	8.12	155	624	4.03	7.55
3	21.16	10.91	160	659	4.12	7.94
4	22.00	12.61	146	646	4.42	9.04
5	7.75	9.40	141	710	5.04	3.30
6	6.66	5.25	150	625	4.17	2.67

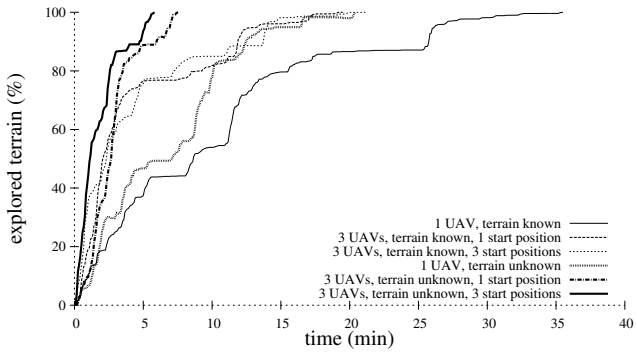
Table V
RESULTS OF THE SINGLE TEST CASES FOR THE SECOND TERRAIN.

The results are mostly the same as those we achieved on the first heightmap. Also, using this heightmap the time for calculating the configuration space does not cause any noticeable delay for path planning. On average the time was below 5 ms except for the fifth test. In the fifth test they were with an average of 5.04 ms not considerably higher.

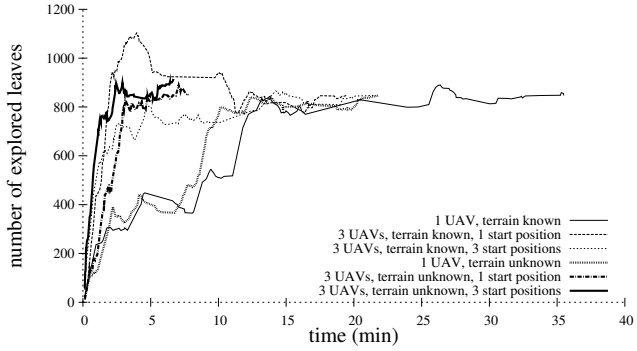
It should be noted, that when using three UAVs instead of one single UAV in unknown terrain, the exploration time was only one-third. This is not due to the fact that the single UAV had to fly much more often over explored terrain than in other tests. But the reason for this is that a high steady exploration rate was reached using three UAVs all the time (see Figure 10(a)).

When using this terrain, the frequency of recalculations of the configuration space was relatively low. It was similar to the first heightmap in such a way that the fewest recalculations were made in the first test case where only one UAV was used. A recalculation was done on average every 18.85 seconds. The most frequent recalculations were done in the sixth test case, where every 2.67 seconds a recalculation took place.

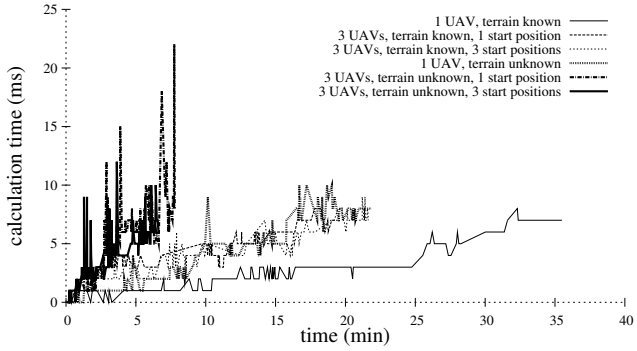
Figure 10 shows the results of the test cases for the second heightmap graphically. Figure 10(a) depicts the exploration over time, Figure 10(b) depicts the number of utilized leaves, which are relevant for the duration to calculate the configuration space. Finally, Figure 10(c) shows the duration of the single calculations of the configuration space.



(a) Percentage terrain covered (second heightmap)



(b) Number of utilized leaves for the calculations of $\mathcal{C}$ (second heightmap)



(c) Calculation times for single calculations of $\mathcal{C}$ (second heightmap)

Figure 10. Results for the second heightmap

Figure 10(a) shows the exploration rates of the terrain for the six test cases dependent on simulation time. It is shown that similar to the first heightmap, when using three UAVs, a steady exploration rate was reached in unknown terrain even

during the complete exploration. In the test cases with known terrain the exploration rate decreased when an exploration level of nearly 80% was reached. The worst rate, which led to the longest time needed to explore the complete free space, was in test one, when using only one UAV in known terrain. Similarly to the tests done on the first heightmap, in every test a complete exploration of the free space was reached.

Figure 10(c) shows the costs to calculate the configuration space for the six test cases. It is shown that the calculation costs were relatively equal, except for the fifth test case with three UAVs in unknown terrain, where several spikes appear. In this test scenario the last calculation of the configuration space was the most costly one (22 ms). But even this is an acceptable value regarding a recalculation every few seconds.

When considering the results one can conclude that the disadvantage of calculating a potential field based on harmonic functions, the high calculation costs, vanished. This was due to the use of convergence criteria (Section V-C), the use of a quadtree (Section IV-A) to reduce the number of utilized leaves instead of using a grid and the introduction of an activation window (Section V-C). Hence, the advantages of harmonic functions can be exploited without suffering from their disadvantage. Always achieving an exploration rate of 100% of the reachable terrain shows that our path planning approach leads to good results even in complex and dynamic terrains.

The frequency of the recalculations of the configuration space shows that the system is a cheap one in respect of the calculation costs. This made it possible to use the designed approach in embedded systems with limited hardware.

Additionally, it was shown that the explored area increased with a steady rate in most cases and the UAVs had to fly relatively rare through explored terrain to reach unexplored terrain. In the cases where this behavior occurred, the exploration rate decreased. So, this is one issue to be solved, to decrease the time needed to explore terrains.

VIII. Conclusion and Future Work

The motivation for this paper was to design an efficient path planning system, which can be used to coordinate multiple UAVs to explore different disaster areas. The requirement was to create an efficient and robust system to coordinate multiple UAVs, including path planning, exploratory navigation and simultaneous task allocation, using only one global configuration space.

A hybrid approach for UAV coordination and efficient exploration of disaster areas was presented. It uses artificial potential fields, combined with an informed search algorithm, and a role system. Additional methods like a quadtree, an activation window, and break conditions were used to find a tradeoff between the number of local minima and computational costs. Until an exploration level of more than 70% was reached, a nearly steady exploration rate was achieved. Three UAVs need only half of the time for exploration in comparison to the time one single UAV needs.

The costs to compute the configuration space were decreased such that an online calculation without any noticeable delay was possible. This is important for highly dynamic environments, where the calculation has to be faster than the changes of the configuration space, in order to calculate efficient paths. In combination with the total exploration of the terrain, this leads to a robust and efficient system.

Future work is to lower the exploration time even more when using multiple UAVs. This can be done by more active coordination methods, e. g., explicit communication between the UAVs. Another focus of future work is to achieve greater autonomy. For this purpose an interaction of the UAVs with each other will be implemented to replace the central control station. This should be done in such a way, that using more UAVs leads to faster exploration of the complete terrain.

Another part of future work is to advance the potential field to n dimensions by including values like exploration times of the leaves and UAV properties into the single potentials. The next step is to extend the potential field to the third dimension of the terrain. Using a 3D potential field will allow the UAVs to move inside of buildings and fly, e. g., below bridges without any additional calculations.

An interesting point of investigation would be to decentralize the field such that each UAV calculates only that part of it, which is relevant for the UAV. In that case properties like fuel level could be included into the potential field to ensure that a UAV flies only to regions of the terrain from which it can fly back with its remaining fuel. Such a modeling of the potential field should reduce the number of methods, which are necessary to control a real UAV.

One step of our research project is to apply the approach to real UAVs. The achieved cost reduction enables the system to be implemented even on embedded systems. Of course, physical properties of real UAVs have to be considered even more. Additionally, sensor errors and the UAV behavior, e. g., in strong crosswind may lead to further need for adaptations when using real UAVs.

When applying the approach to physical UAVs we also need to consider non-holonomic constrains. This will lead to much smoother paths as the UAVs do not have to stop for course changes.

It is planned to extend the skills of the UAVs by introducing an inter-UAV communication, which makes decentralized coordination possible. This leads to the possibility that methods for formation flights can be realized. Several methods for this may be evaluated, e. g., the use of consensus finding [22] in communication graphs or adapting the potential field to hold information for building a formation, e. g., by using bifurcating potentials [23].

ACKNOWLEDGMENT

This work was conducted in the BMBF funded project SOGRO[3].

REFERENCES

[1] C. Rasche, C. Stern, W. Richert, L. Kleinjohann, and B. Kleinjohann, "Combining autonomous exploration, goal-oriented coordination and task allocation in multi-uav scenarios," *ICAS, The Sixth International Conference on Autonomic and Autonomous Systems*, March 2010.

[2] O. Khatib, "Real-time obstacle avoidance for manipulators and mobile robots," *IJRR*, vol. 5, no. 1, pp. 90–98, 1986.

[3] E. Prestes, M. E. Paulo, M. Trevisan, and M. A. P. Idiart, "Exploration technique using potential fields calculated from relaxation methods," *IEEE/RSJ International Conference on Intelligent Robots and Systems*, vol. 4, pp. 2012–2017, 2001.

[4] E. Prestes, P. M. Engel, M. Trevisan, and M. A. P. Idiart, "Exploration method using harmonic functions," *Robotics and Autonomous Systems*, vol. 40, no. 1, pp. 25–42, 2002.

[5] M. Trevisan, M. A. P. Idiart, E. Prestes, and P. M. Engel, "Exploratory navigation based on dynamical boundary value problems," *Journal of Intelligent and Robotic Systems*, vol. 45, no. 2, pp. 101–114, February 2006.

[6] J. Hagelbäck and S. J. Johansson, "Using multi-agent potential fields in real-time strategy games," in *AAMAS '08: Proceedings of the 7th international joint conference on Autonomous agents and multiagent systems*. Richland, SC: International Foundation for Autonomous Agents and Multiagent Systems, 2008, pp. 631–638.

[7] R. Sawhney, K. Madhava, and K. Srinathan, "On fast exploration in 2d and 3d terrains with multiple robots," in *AAMAS*. Richland, SC: International Foundation for Autonomous Agents and Multiagent Systems, 2009, pp. 73–80.

[8] D. Jung, J. Ratti, and P. Tsiotras, "Real-time implementation and validation of a new hierarchical path planning scheme of UAVs via hardware-in-the-loop simulation," *Journal of Intelligent and Robotic Systems*, vol. 54, no. 1-3, pp. 163–181, March 2009.

[9] I. Nikolos, K. Valavanis, N. Tsourveloudis, and A. Kostaras, "Evolutionary algorithm based offline/online path planner for uav navigation," *IEEE SMC*, vol. 33, no. 6, pp. 898–912, Dec. 2003.

[10] M. Kazemi, M. Mehrandezh, and K. Gupta, "An incremental harmonic function-based probabilistic roadmap approach to robot path planning," *Proceedings of the 2005 IEEE International Conference on Robotics and Automation*, April 2005.

[11] J. Barraquand, B. Langlois, and J. C. Latombe, "Numerical potential field techniques for robot path planning," *IEEE Transactions on Systems, Man and Cybernetics*, vol. 22, no. 2, pp. 224–241, Mrz/April 1992.

[3]"Sofortrettung bei Großunfall mit Massenanfall von Verletzten", supported by the Bundesministerium für Bildung und Forschung (BMBF) 13N10164.

[12] K. Aizawa, K. Motomura, S. Kimura, R. Kadowaki, and J. Fan, "Constant time neighbor finding in quadtrees: An experimental result," *3rd International Symposium on Communications, Control and Signal Processing, 2008. ISCCSP 2008.*, pp. 505–510, March 2008.

[13] C. Connolly, J. Burns, and R. Weiss, "Path planning using laplace's equation," *IEEE ICRA*, vol. 3, pp. 2102–2106, Mai 1990.

[14] J. S. Zelek, "A framework for mobile robot concurrent path planning and execution in incomplete and uncertain environments," in *Proceedings of the AIPS-98 Workshop on Integrating Planning, Scheduling and Execution in Dynamic and Uncertain Environments*, 1998.

[15] P. G. Doyle and J. L. Snell, "Random walks and electric networks," 2000. [Online]. Available: http://www.citebase.org/abstract?id=oai:arXiv.org:math/0001057

[16] C. Connolly, "Applications of harmonic functions to robotics," in *IEEE ISIC*, Aug 1992, pp. 498–502.

[17] D. Beck, A. Ferrein, and G. Lakemeyer, "Landmark-based representations for navigating holonomic soccer robots," in *RoboCup 2008: Robot Soccer World Cup XII*, ser. Lecture Notes in Computer Science, vol. 5399. Springer Berlin / Heidelberg, 2009, pp. 25–36.

[18] E. C. Zachmanoglou and D. W. Thoe, Eds., *Introduction to Partial Differential Equations with Applications*. Dover Publications, Inc., 1986.

[19] J. S. Zelek, "A framework for mobile robot concurrent path planning and execution in incomplete and uncertain environments," in *AIPS*, 1998.

[20] K. Aizawa, K. Motomura, S. Kimura, R. Kadowaki, and J. Fan, "Constant time neighbor finding in quadtrees: An experimental result," in *Communications, Control and Signal Processing, 2008. ISCCSP 2008. 3rd International Symposium on*, March 2008, pp. 505–510.

[21] C. Rasche, C. Stern, L. Kleinjohann, and B. Kleinjohann, "Role-based path planning and task allocation with exploration tradeoff for uavs," *ICARCV, 11th International Conference on Control, Automation, Robotics and Vision*, December 2010.

[22] W. Ren, W. R. Beard, and T. W. McLain, "Coordination variables and consensus building in multiple vehicle systems," in *Cooperative Control*, ser. Lecture Notes in Control and Information Sciences, vol. 309. Springer Berlin / Heidelberg, 2004, pp. 439–442. [Online]. Available: http://www.springerlink.com/content/m2gpyjh0mkq4uapc/

[23] D. J. Bennet and C. R. McInnes, "Distributed control of multi-robot systems using bifurcating potential fields," *Robotics and Autonomous Systems*, vol. 58, no. 3, pp. 256 – 264, 2010, towards Autonomous Robotic Systems 2009: Intelligent, Autonomous Robotics in the UK. [Online]. Available: http://www.sciencedirect.com/science/article/B6V16-4XT3HPP-2/2/4338e40c022f12a0582b3655f7a51152

A Framework for Monitoring and Reconfiguration of Components Using Dynamic Transformation

Djamel Belaïd*, Imen Ben Lahmar*, and Hamid Muktar†
*Institut Telecom; Telecom SudParis, CNRS UMR SAMOVAR, Evry, France
Email: {djamel.belaid, imen.ben_lahmar}@it-sudparis.eu
†National University of Sciences and Technology (NUST), Islamabad, Pakistan
Email: hamid.mukhtar@seecs.edu.pk

***Abstract*—Distributed applications can be created using component-based software development. Such applications are defined as an assembly of components requiring services from and providing services to each other. The existing component models provide a description of functional and non-functional requirements of an application. However, this capability is to be determined at the design time of the application. Once deployed, the application cannot be modified to respond to the changing context.**

In order to allow creation of such applications that can be transformed dynamically to respond to changing environments, in this article we propose a framework that allows monitoring and dynamic reconfiguration of different components. These components may be functional components of the user application or other components of the environment on which an application depends. The components of environment may represent the underlying environment (i.e., hardware and network entities) and are presented in our framework in the same way as the application components. A component can monitor other components in order to be aware of their changes. Moreover, the components can also be monitored and reconfigured remotely. If a component is not monitorable or reconfigurable by default, we propose a procedure that transforms it to respond to components requests.

***Keywords*-Framework, component model, monitoring, reconfiguration, UPnP, transformation**

I. Introduction

With the emergence of wireless technologies and ubiquity of hand held wireless devices, application development for the pervasive environments is gaining more and more attention. In such environments, computing is pushed away from the traditional desktop to small hand-held and networked computing devices that are present everywhere we go. As such, Service-Oriented Architecture (SOA) has emerged as a computing paradigm that changed the traditional way of how applications are designed, implemented and consumed in a pervasive computing environment.

One particular approach for developing SOA-based applications is to use component-based application development. Using this approach, an application is defined as a composition of re-usable software components, which implement the business logic of the application domain collectively by providing and requiring services to/from one another. The components required by an application are assembled at the time of application development. Thus, at the time of application deployment, all the components have been defined statically.

However, when considering the broad range of computing devices in pervasive environments (smartphones, PDAs, tablets, laptops, etc.) – with different capabilities and limitations - this approach may not work. Moreover, pervasive environments are highly dynamic due the mobility of users and devices. Thus, an important aspect of pervasive applications is that their realization is very much dependent on their execution context. Due to variability of the environment, modelling the application behaviour needs to satisfy not only the functional requirements in an effective way, but in order to provide better quality of service (QoS) for user satisfaction, it should also consider the current state of the environment in which the application is executing. In addition, the application should also adapt itself according to changing context.

Existing component models like PCOM [1] [18], Fractal [4], OSGi [17] and SCA [5] propose application development using component assembly. In these models, a component offers its capabilities through provided interfaces and consumes functionalities offered by other components through required interfaces. Along with offered and required interfaces, a component may define one or more properties. These properties can also be modified so that a component can be reconfigured dynamically at runtime.

Due to the heterogeneity of the environment, modelling the application behaviour needs to consider the current state of the environment in which the application is executing in addition to its functional requirements. However, most of the existing component models leave such issues to the underlying middleware, which provides a uniform Application Programming Interface (API) or a framework for this purpose [1] [18]. This means that the programmers and the designers have to rely on the functionality of the underlying middleware and such aspects need to be considered during application development life-cycle. All of the reconfiguration aspects, such as determination of reconfigurable properties, have to be decided at development time. Once an application has been developed, and deployed, its composition becomes fixed. A property that was not set to be reconfigurable or monitorable during development remains so and changes need to be made at source code level to make it reconfigurable or monitorable dynamically.

In order to explain these limitations, let's consider an application that provides the functionality of sending large

files from one device to another using a communication link that exists between them. Files are transferred via a WiFi connection and as the size of a typical file is large (e.g., 1 GB each), each file is transferred in several compressed chunks (e.g., 256 KB) to allow quick transfer. On the receiving side, once a device receives a compressed chunk, it decompresses it before merging it with other chunks to reconstruct the whole transferred file.

Due to variability of the WiFi signal strength, the application needs to continuously monitor the network signal to decide the chunk size to be used for sending the file. In case of high signal strength, data can be sent at higher rates and larger chunk size can be used; however, in case of weak signal strength, a smaller chunk size may be applied for a quick transfer. Moreover, to decide the degree of compression, the application needs to monitor the remaining battery powers of the sender and receiver devices. If the battery power of any of the devices is low, it uses lower degree of compression to conserve the battery power required for the compression and/or the decompression. However, if the remaining battery power is sufficiently high for both devices, higher degree of compression can be used for a better throughput over the network. As it can be seen, the decompression degree depends on the used degree of compression. Thus, the use of a lower/higher compression degree for a given file chunk at the sender's side implies that the decompression degree is to be maintained the same on the receiving side. Therefore, whenever the compression ratio changes, the decompression degree should be reconfigured for an efficient transfer of files over the network.

We deduce a few important points from the file sender application. First, the behaviour of the application is dependent on certain properties which are not part of the File Sender application, namely, remaining battery power and network signal strength. Both of them correspond to externally required properties: they do not form the core logic of the application, however, the desired Quality-of-Service (QoS) provided by the application greatly depends on their values. Thus, a mechanism is needed to consider such required properties in the application design without altering the application logic and architecture. Second, the monitoring can be notifications from the provider side or observations from the client side. Third, the reconfiguration of components' properties is dependent on changes of properties of other components. Thus, we require a mechanism that is able to reconfigure dynamically the properties of components whenever there is a need. Finally, since these properties may belong to components found locally or remotely, we need a uniform strategy for accessing local or remote properties to be monitored or reconfigured.

As discussed previously, the above mentioned requirements are not considered by the existing component models. Therefore, in our previous work [2], we have proposed a component model that 1) considers explicitly the required properties of an application in addition to the functional behaviour; 2) allows the monitoring ; and 3) if a property is not monitorable by default, we provide transformation mechanism to render it monitorable.

In this article, we propose the following contributions, some of which extend our previous work [2]. As an extension, first, we present a remote monitoring approach allowing components to monitor their remote required properties. Second, we extend our proposed component model to allow components to reconfigure their local and remote required properties. Third, we present some transformation processes for components to render their properties monitorable or reconfigurable whenever there is a need by a third-party.

The remaining article is structured as follows. In Section II, we first describe our proposed component model and the component assembly, then, in section III, we explain the monitoring and reconfiguration concepts and how components can be transformed to make them monitorable or reconfigurable locally as well as remotely. Section IV describes how components can be declared and how the transformation can be achieved followed by the implementation details in section V. In Section VI we provide an overview of existing related approaches as well as their limitations. Finally, Section VII concludes the article with an overview of our future work.

II. Our Approach

In this section, we outline the different concepts involved in our approach. We begin by introducing our component model and component assembly. We then describe the need for component transformation and how they are accompanied by integrating adaptive logic into the application to make it adaptive to the changing context.

A. The Component Model

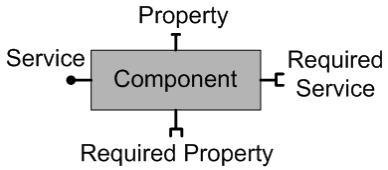


Fig. 1. Component model describing required properties

In an object-oriented paradigm, an object provides its services through a well-known interface, which specifies the functionality offered by the object. In the relatively newer component-oriented paradigm, components may specify, in addition to their provided interfaces, their dependencies on the offered properties of other components using required interfaces. As defined by Szyperski et al. [20] "A software component is a unit of decomposition with contractually specified interfaces and explicit context dependencies only." Thus, a component not only exposes its services but it also specifies its dependencies. Most of the existing component models [1][4][17][5] allow specification of their dependencies for business *services* external to the component. However, they do not allow specification of their dependency for external *properties*. The ability to specify dependency for external properties has two important implications. First, it results

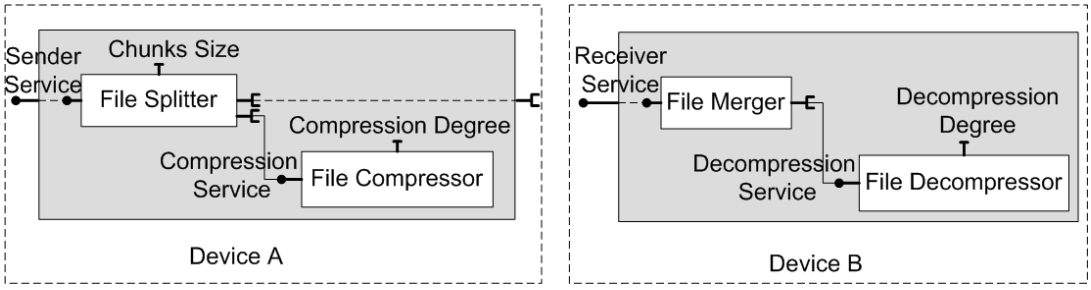


Fig. 2. File Sender Application

in specification at relatively fine granularity thus helping the architects and designers in fine tuning the component's requirements. Second, this fine tuning helps in elaborating the contract between two components because the properties can be enriched with additional attributes that constrain the nature of the contract through appropriate policies. To achieve this objective, in one of our previous works [2], we have proposed a component model that allows expressing this dependency explicitly in terms of required properties, which are provided by other components.

Figure 1 shows main characteristics of a component that provides a service through an interface and requires a service from other components through a reference. The component also exposes a property through which it can be configured. In addition, the component also specifies its dependency on a certain property. This required property, which appears at the bottom of the component, will be satisfied if we can link this component with another component that offers the requested property, thus, solving the dependency.

We use components to represent not only software entities that make up an application, but also to represent hardware and network entities present in the execution environment. For example, a component may represent the screen of a device, a WiFi card, or even user preferences.

B. Component Assembly

Components can be combined together in an assembly of components to build complex applications. For example, figure 2 shows how the File Sender application described in the introductory section can be represented by an assembly of components distributed across two devices: two of them on the sender device (A) and two others on the receiving device (B).

On the sender side (device A) the File Splitter component splits a given file into chunks for an efficient transmission. The appropriate size of the file chunk is determined by the network signal strength. If the signal strength is high, data can be sent at higher rates and larger file chunks (e.g., 1 GB each) will be created. However, if the signal is weak, smaller chunks (e.g., 256 KB) will be created to allow quick transfer. Once a file chunk is created, it is passed to the File Compressor component for compression before sending it to the receiver device. This is done using the service provided by the File Compressor component. The File Compressor component uses an adaptive compression algorithm whose compression ratio depends upon the remaining battery powers of the sending and receiving device. If the remaining battery power of each device is above a certain threshold (e.g., 20 percent), higher degree of compression is used. However, if the remaining battery power of any of the devices is below the specified threshold, lower degree of compression is used by doing quick compression of each chunk thereby conserving the battery power.

On the receiving side (device B), a File Decompressor component is used to decompress the received compressed chunk. The component has a decompression degree property whose value should be the same as the value used for compression by the File Compressor component. Thus, any change of the compression degree must imply the same change of the decompression degree in the File Decompressor component. Once the received chunk is decompressed, a File Merger component combines it with the other decompressed chunks to recreate the transferred file.

C. Component Transformation

Figure 2 shows the File Sender application as defined by the architect. As can be seen, it represents only the functional components of the application and does not show the components external to the application — battery and WiFi — which are required by the functional components for providing the necessary QoS desired by the user. Assume that this application was developed for fixed environments, e.g. a desktop PC connected to a wired network with fixed QoS, in which the application would not need to be adapted.

Our objective is that given such an application, which was not conceived for dynamic environments with changing QoS, we would like to transform it in order to adapt its functionality according to the changing QoS. This transformation can be of two types. If we are interested in knowing the changes in the properties of a component, we need to transform the component to make it monitorable. On the other hand, if we are interested in modifying the properties of a component due to external notifications, we need to make it reconfigurable. Furthermore, a transformation may apply to a component available on the device locally resulting in local transformation or it may apply to some component in remote device, in which case it is known as remote transformation.

In our previous work [2], we proposed a monitoring mechanism to permit an application — based on our component

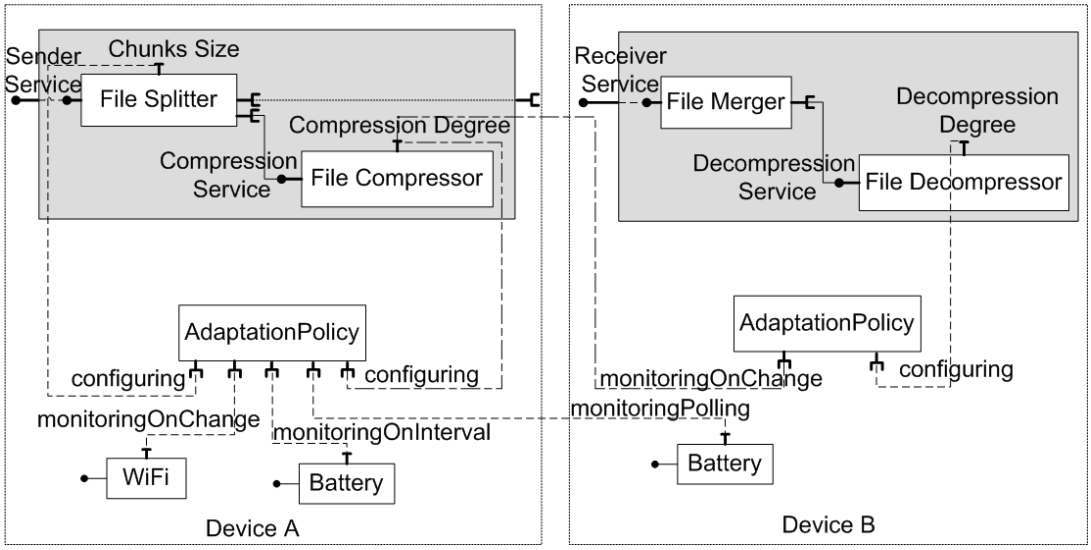


Fig. 3. Monitoring and Reconfiguration of required properties of the File Sender Application

model — to be adapted. However, the approach was limited to monitoring in the local scope. In our present work, we go a step further to address the remote monitoring as well as the local and remote reconfiguration of the components.

Corresponding to our scenario, our application needs to monitor the components of environment: battery and WiFi. Assuming that these components are not monitorable by default, we need to make them monitorable by transformation. Similarly, the functional components of the application need to be reconfigured every time the QoS provided by these non-functional components changes. The File Splitter component needs to be reconfigured for its chunk size property whenever the WiFi signal strength changes. The compression degree of the File Compressor component should also be reconfigured whenever the battery level crosses its threshold. Corresponding to File Compressor component, the File Decompressor component must also be reconfigured with the same degree of decompression as that used for compression.

Since the components are not reconfigurable by default, i.e., when they were defined initially during application assembly, we need to transform them to make them reconfigurable. In the next section, we describe how these transformations can be used along with some adaptation policies to render an application adaptable.

D. Adaptation Logic

Transformation allows a component to be monitorable or reconfigurable. However, only transforming an application does not help in making the application adaptive. For example, in our example application, only by making the WiFi component monitorable and the File Splitter component reconfigurable will not make the application to take adaptation decisions. Instead, we need some *adaptive logics* that will make appropriate adaptation decisions based on certain rules for adaptation. This adaptation logic has to be defined by the architect at design time to make the application adaptable. It is encapsulated in an adaptation policy component which is a functional component and defined following our component model. The adaptation policy components express through their required properties their need to monitor and/or to reconfigure local as well as remote properties offered by components of the adaptable application.

III. Monitoring and Reconfiguration Framework

To make our example application adaptable, we transform its components to make them monitorable or reconfigurable and integrate an adaptation policy component that defines the policy to be used for adaptation. The transformed File Sender application is shown in figure 3. Two Adaptation Policy components have been used to manage the adaptation of the application to respond to adaptation due to the change of the context. The context is defined by the properties of the WiFi and the Battery components and the adaptation is made on the properties of the File Splitter, File Compressor and File Decompressor components. These adaptation policy components express their need to monitor and to reconfigure local and remote properties of other components through their required properties.

On device A, the adaptation policy component expresses through its required properties its need to monitor the *signal strength* property of the WiFi component, the *level* property of the local Battery component and the *level* property of the remote Battery. Depending on changes in the values of these properties, the Adaptation Policy component also expresses its need to reconfigure the *chunk size* property of the File Splitter component and the *compression degree* property of the File Compressor component.

On device B, another adaptation policy component is used to monitor remotely the *compression degree* property changes of the File Compressor and to reconfigure the *decompression degree* property of the File Decompressor component accordingly.

```
public interface GenericProxy {
  Property[]  getProperties();
  Object  getPropertyValue(String propertyName);
  void  setPropertyValue(String propertyName,
                         Object propertyValue);
  Object invoke(String methodName, Object[] params);}
```

Fig. 4. Description of the Generic Proxy interface

If the offered properties of components are not defined as monitorable or reconfigurable resources, we need to transform them to respond to the requests of the adaptation policy components. A transformation is applied dynamically at runtime and is carried out by some predefined components of our framework. For different types of transformation, the framework has defined different components. In the next subsections, we introduce them and we detail the main features of the monitoring and the reconfiguration mechanisms and their transformation processes.

A. Generic Proxy Service

The Generic Proxy Service, provided by our framework, can be applied to any component of an application that we want to introspect before any transformation.

We have defined a general purpose interface GenericProxy that provides four generic methods. These methods are described in figure 4. Each implementation of this interface is associated with a component for which the first method getProperties() returns the list of the properties of the component, the getPropertyValue() returns the value of a property, the setPropertyValue() changes the value of a property and the invoke() method invokes a given method on the associated component and returns the result.

We provide two implementations of the GenericProxy interface (see section V). The first one, LGenericProxy component is for implementation of a local proxy. That is, when associated with a local component it translates its method calls into calls of the associated component. The second one, RGenericProxy component is a remote implementation. That is, when associated with a remote component it forwards the calls of its methods to calls of the associated remote component, over the network.

B. Local Reconfiguration

In parametric adaptation, a component is able to reconfigure the properties of another component. In this context, reconfiguring the required properties of a component is defined as the reconfiguration of the offered properties of another component. For this purpose, we extend our component model [2], in order to allow components to specify their need to reconfigure some of their required properties. For example, in figure 5(a), a component A specifies a required property, offered by the component B, that it needs to reconfigure.

A given property of component can be reconfigured by calling its associated setter method. However, the component that wishes to reconfigure a property of another component does not know a priori the type of this component. To complete the reconfiguration of any component from only the name and type of a property, the reconfigurator component uses an appropriate interface that provides the method **setPropertyValue(propertyName, propertyValue)** to change the value of a property.

However, the component to be reconfigured may not define its properties as reconfigurable resources despite the request. So we need to transform the component to make its properties reconfigurable by offering an appropriate reconfiguration interface. This can be done dynamically by our framework by encapsulating the component with the predefined LGenericProxy component as defined above. The two components are combined together in a single *composite* that offers the services of the original component as well that of the LGenericProxy component. The component can be then reconfigured using the setPropertyValue() method provided by the LGenericProxy component. The framework then replaces the original component with the newly created composite in the application. Figure 5(c) shows the transformation of the component B to render its property reconfigurable by the component A.

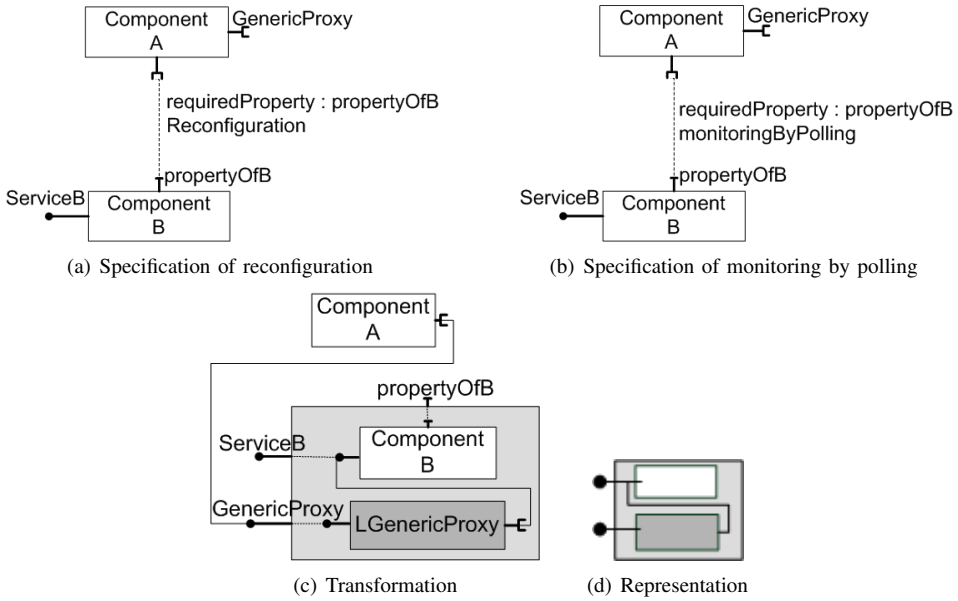
C. Local Monitoring

In [2], we have presented a monitoring approach to allow a component to be aware of required properties changes. Monitoring process consists in informing the interested component about the changes of required properties or notifying it on a regular way or for each variation. We have considered two types of monitoring: monitoring by polling and monitoring by subscription.

Polling is the simpler way of monitoring, as it allows the observer to request the current state of a property whenever there is a need. However, subscription allows an observing component to be notified about changes of monitored properties.

1) Monitoring by Polling: A component may express its need to monitor by polling a required property provided by another component (figure 5(b)). The monitoring by polling of a property can be made by calling its getter method. However, the component that wishes to monitor a property of another component does not know a priori the type of this component. To complete the monitoring of any component from only the name and type of a property, the interested component often uses an appropriate interface that provides the method **getPropertyValue(propertyName)** to request the current state of a property.

However, the component to monitor may not define its offered properties as monitorable by polling resources despite the request. So, we need to transform the component to make its properties to be monitorable by offering an appropriate interface of monitoring. This can be done dynamically by our Framework at runtime in the same way as the transformation process for the reconfiguration since it uses the same LGenericProxy component that provides the needed



(a) Specification of reconfiguration (b) Specification of monitoring by polling

(c) Transformation (d) Representation

Fig. 5. Monitoring by polling and reconfiguration

interface. In figure 5(c), we show the transformation of the component B to render its property monitorable by polling by the component A. Figure 5(d) is a symbol, we used to represent the transformation for monitoring by polling as well as reconfiguration.

2) Monitoring by Subscription: There are two modes of monitoring by subscription: 1) subscription *on change* which specifies that the subscribed component is notified every time the value of the property changes; 2) subscription *on interval* which specifies that the subscribed component is to be notified after a specified time interval. For notification on change, a component must precise the starting time and the duration of notifications. For notification on interval it must specify the notification interval value. It may also precise the starting time and the duration of notifications. The component A must also implement a notification callback through which it will receive appropriate notification messages.

Figure 6(a) shows how component A specifies its need to monitor a required property offered by a component B by subscription on change. Figure 7(a) shows monitoring by subscription on interval. Figures 6(c) and 7(c) shows the symbols we used for denoting subscription on change and subscription on interval, respectively.

For the monitoring with notification mode on interval, as shown in the figure 7(b), each time the MonitoringBySubscription component have to notify the subscriber (the component A), it gets (or monitor by polling) the value of the required property of the component B via the LGenericProxy component.

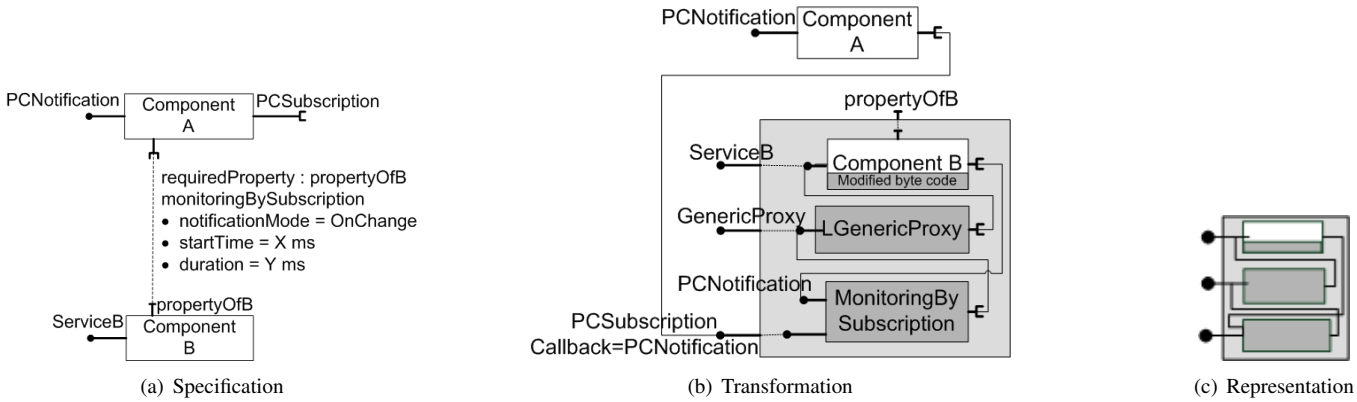
When the notification mode is on change for a required property of B (figure 6(b)), the MonitoringBySubscription component offers a (callback) service of notification PCNotification to the component B so that it can be notified of the changes of a required property and in turn inform all the subscribers of this change. To allow the component B to notify the MonitoringBySubscription for the change of its properties, our framework adds the needed instructions in the byte-code of the component B at runtime.

D. Remote Monitoring and Reconfiguration

To adapt a distributed application in a pervasive environment, some components may be interested in monitoring or reconfiguring properties of other components remotely. For this purpose, components in our framework can specify their need about remote reconfiguration or remote monitoring of some of their required properties.

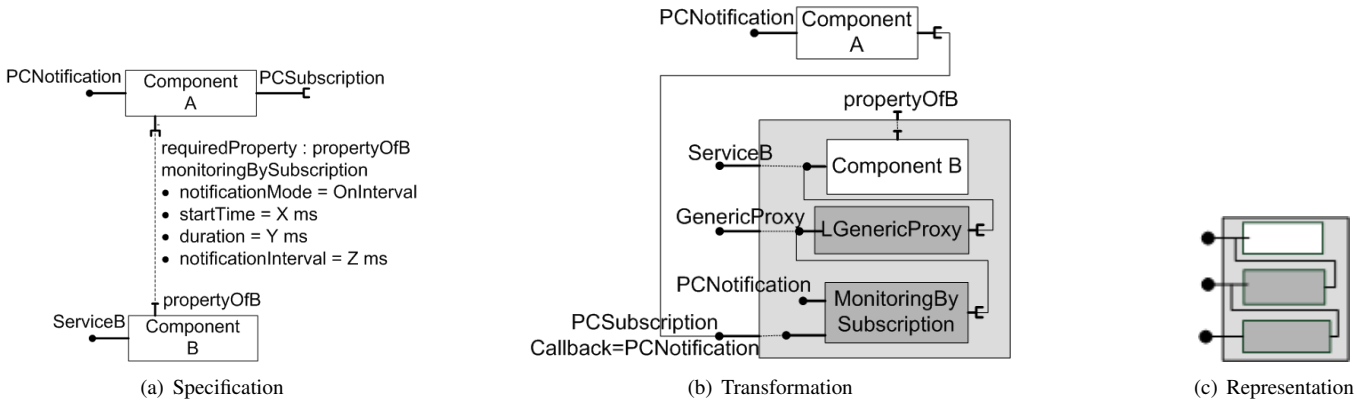
For example, figure 8 shows how component A specifies its need to monitor by subscription a property offered by a remote component B. For the two modes of notification, the component B (the server) must offer a remote subscription service over the network to the component A (the client) and in turn, the component A must subscribe to the remote component B specifying its need. When a change of the property happens, a notification from the component B to A is sent over the network. As for the local case, to provide a remote reconfiguration and monitoring by polling the component B must offer a GenericProxy service and should also be reachable over the network. We note that for the remote purposes there are two transformations: server-side (component B) and client-side (component A).

1) Server-side transformation: The framework first encapsulates the component B in a composite (figure 9(a)) such as defined for the transformations for the monitoring by subscriptions. Then it adds a new component (the RServer component) that integrates the network communication aspects like remote call and event processing. The RServer component has two references: one for the subscription service offered the MonitoringBySubscription component and one for the Gener-



(a) Specification (b) Transformation (c) Representation

Fig. 6. Monitoring by subscription with notification mode on change



(a) Specification (b) Transformation (c) Representation

Fig. 7. Monitoring by subscription with notification mode on interval

icProxy service provided by the LGenericProxy component. The first reference is used to subscribe and the second one is used for the reconfiguration and monitoring by polling on behalf of the client component A. We note that the newly defined composite provides the GenericProxy service and the PCSubscription service in addition to the services of the component B. Thereby it can be used, reconfigured and monitored locally as well as remotely. Figure 9(b) is a symbol denoting the server-side transformation.

2) Client-side transformation: If the component B was on the same device as the component A, we would have used the same composite defined for the monitoring by subscription and connect A to its subscription interface. Since this is not the case, our framework creates the same kind of that composite (figure 10(a)) with the ServerProxy component in place of the component B and for the implementation of the interface GenericProxy we use the RGenericProxy described in section III-A. The ServerProxy component is a byte-code generated component by our framework at runtime (see section V). Its implementation of the service B consists on forwarding the calls to the RGenericProxy component which in turn make the call over the network to the server side. Figure 10(b) is a symbol, we used to represent the transformation in the client-side.

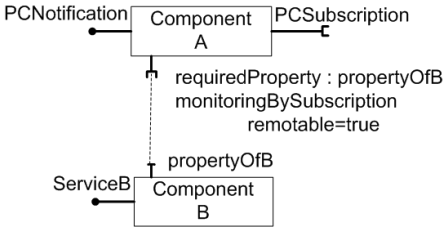


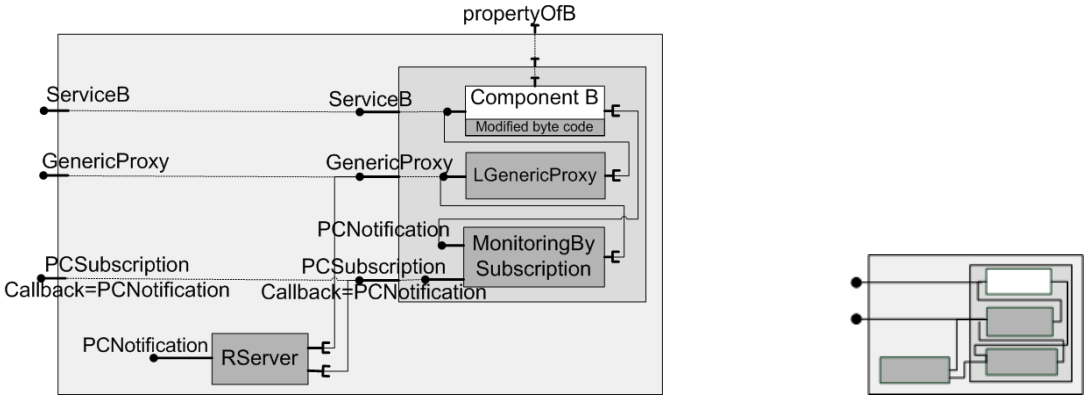
Fig. 8. Specification of remote monitoring

E. Putting it all together

Referring back to figure 3, we assume that signal strength, the compression degree and the local and remote battery levels are not defined as monitorable properties. So we need to transform WiFi, File Compressor and the two Battery components to render their properties monitorable by the Adaptation Policy components.

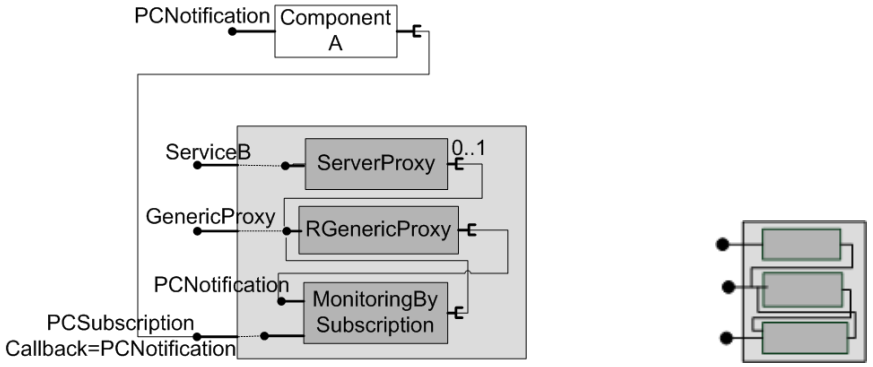
Figure 11 shows the creation of new composites in response of the monitoring request of the Adaptation Policy components. The transformation of the WiFi component corresponds to the creation of a composite as shown in figure 6(c), while the local Battery component is transformed following the figure 7(c).

The Battery component on device B is remotely observed



(a) Transformation (b) Representation

Fig. 9. Remote monitoring and reconfiguration: server side



(a) Transformation (b) Representation

Fig. 10. Remote monitoring and reconfiguration: client side

by the Adaptation Policy component of device A. For this purpose, we require two transformations: a server-side (Battery component) and a client-side (Adaptation Policy component) transformation. The transformation of the Battery component is done following figure 9(b), and a new composite representing the remote Battery component is created in the device A following figure 10(b) to allow the Adaptation Policy component to observe the battery level changes.

Similarly, the File Compressor component (server side) is transformed following the figure 9(b) to allow the Adaptation policy component of device B (client side) to subscribe to the changes of the compression degree property.

For the reconfiguration needs of the Adaptation Policy components, our framework transforms the File splitter and the File Decompressor components following the figure 5(d). The File Compressor component is already transformed into a composite that offers a generic proxy service for reconfiguration.

IV. Architectural Description

The description of an application can be done with the help of an Architecture Description Language (ADL). Instead of inventing a new ADL, we prefer to use one of the existing description languages. In this regard, Service Component Architecture (SCA) [5] provides a rich ADL that details most of the aspects that we are looking for. One of the main features of this component model is that it is independent of particular technology, protocol, and implementation. It consists of a set of services, which are assembled together to create solutions that serve a particular business need. These services correspond to offered services and required ones called references. Along with services and references, a component can also define one or more properties. We use SCA for its extensibility to overcome the missing elements related to required properties, monitoring and reconfiguration aspects.

In our previous work [2], we have extended the standard SCA description by adding the `@requiredProperty` attribute to express explicitly the dependency of components to the offered properties of other components. This attribute specifies the resource whose property will be monitored or reconfigured.

Moreover, we have also specified the `resource` type of each component or property. For a component, the resource type corresponds to the classification of the component in one of the predefined categories, such as software, hardware, and network, etc. Some of these categories have been defined previously as extension of the Composite Capability/Preference Profile (CC/PP) [14] by the authors [16]. For a property,

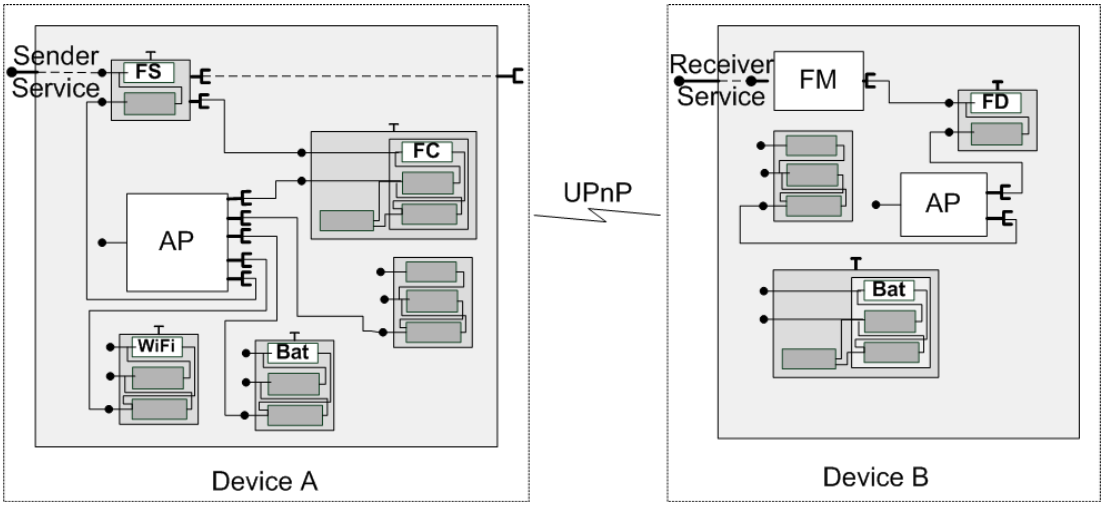


Fig. 11. Transformation of the File Sender Application

```
1<composite name="FileSender">
2   <service name="FileSenderService" promote="FileSplitter/FileSenderService"/>
3    ........
4   <component name="AdaptationPolicy" resource="Software.Component">
5      <service name="PCNotification">
6         <interface.java interface="eu.tsp.iaria-example.PCNotificationInterface"/>
7      </service>
8      <requiredProperty resource="Hardware.Battery" remotable="true" monitoring="ByPolling">
9          <property name="BatteryLevel"/>
10      </requiredProperty>
11     <requiredProperty resource="network.WiFi" monitoring="BySubscription" notificationMode="ON_CHANGE">
12         <property name="SignalStrength"/>
13     </requiredProperty>
14     <requiredProperty resource="Software.FileSplitter" reconfiguration="true">
15         <property name="chunckSize"/>
16     </requiredProperty>
17          ....
18   </component>
19   <component name="FileSplitter" resource="Software.Component">
20     .......
21   </component>
22 </composite>
```

Fig. 12. Description of the File Sender Application using our extended SCA ADL

the *resource* type specifies the component to which this property belongs. The separation between property name and the associated resource type is significant as it allows the transformation of a given component into a monitorable or reconfigurable component for only the specified properties.

Figure 12 shows the description of the File Sender application. As it can be seen (lines 11-13), a `@monitoring` annotation is used to specify the subscription requirement for `SingalStrength` property. Moreover, a `@notificationMode` annotation is used to specify the monitoring by subscription mode. The Adaptation Policy component requires the monitoring of the `SingalStrength` property of the WiFi component, belonging to the network category, by subscription with notification mode *on change* .

In addition to the monitoring feature, the required property allows components to specify their configuration requirements. For this goal, we have extended the `@requiredProperty` attribute to express explicitly the configuration requirements. The extension consists of a new `@reconfiguration` annotation that specifies if the property of resource would be reconfigured or not. For example, the AdaptationPolicy component requires the reconfiguration of the chunckSize property offered by the File Splitter component (lines 14-16).

To handle remote components by monitoring or configuring their offered properties, we used the `@remotable` annotation of SCA specification [5] to specify if the required property is remotable or not. The `@remotable` annotation has boolean value; if it is true, it implies that the requested resource is remotable else it is a local resource. For example, the Battery component (lines 8-10) provided by device B, is remotely

```
1<composite name="FileSender">
2   <service name="FileSenderService" promote="FileSplitter/FileSenderService"/>
3     ........
4   <component name="AdaptationPolicy">
5      <service name="PCNotification">
6         <interface.java interface="eu.tsp.iaria-example.PCNotificationInterface"/>
7      </service>
8      <reference name="PCSubscriptionService" target="WiFiComposite"/>
9      <reference name="PCSubscriptionService" target="BatteryComposite"/>
10     <reference name="PCSubscriptionService" target="UPnPClientComposite"/>
11     <reference name="GenericProxyService" target="FileSplitterComposite"/>
12     <reference name="GenericProxyService" target="FileCompressorComposite"/>
13   </component>
14   <component name=FileSplitterComposite>
15     <service name="FileSenderService">
16         <interface.java interface="eu.tsp.iaria-example.FileSenderInterface"/>
17     </service>
18     <implementation.sca name="FileSplitterComposite"/>
19  </component>
20</composite>
```

Fig. 13. Transformation of the Adaptation Policy component

```
1<composite name=FileSplitterComposite>
2  <service name="FileSenderService" promote="FileSplitter/FileSenderService" />
3  <service name="GenericProxy" promote="LGenericProxy/GenericProxy" />
4  <reference name="FileCompressorService" promote="FileCompressor/FileCompressorService" />
5  <reference name="FileReceiverService" promote="FileSplitter/FileReceiverService"/>
6  <component name="FileSplitter" resource="Software.Component">
7      <service name="FileSenderService">
8         <interface.java interface="eu.tsp.iaria-example.FileSplitterInterface"/>
9      </service>
10     <implementation class="eu.tsp.iaria-example.impl.FileSplitterImpl"/>
11     <reference name="FileCompressorService" target="FileCompressorComposite/FileCompressorService" />
12     <reference name="FileReceiverService" target="FileMerger/FileReceiverService" />
13  </component>
14  <component name="LGenericProxy" resource="Software.Component">
15     <service name="GenericProxy">
16         <interface.java interface="eu.tsp.iaria-example.GenericProxy"/>
17     </service>
18     <implementation class="eu.tsp.iaria-example.impl.LGenericProxy"/>
19     <reference name="FileSplitterService" target="FileSplitter/FileSenderService"/>
20  </component>
21</composite>
```

Fig. 14. Transformation of the File Splitter component

monitorable by polling by the Adaptation Policy component.

Figure 13 shows the transformation of our extended SCA description of the File Sender application into standard SCA description. As it can be seen, the required properties of the Adaptation Policy component are transformed into references to the created composites (lines 8-12).

In figure 14, we show the transformation of the File Splitter component to a new composite to render its property reconfigurable by the Adaptation Policy component. The created composite exposes in addition to the **FileSender** service of the File Splitter component, a GenericProxy service that is provided by a Local Generic proxy component allowing the reconfiguration of the chunckSize property.

V. IMPLEMENTATION

In order to validate our approach, we have implemented a prototype of our approach in Java. The prototype implements our framework as services that offers the various transformation mechanisms to the applications.

For dynamic transformation we required code level manipulation for which we used the open source software JAVA programming ASSISTant (Javassist) library [13]. Javassist is a class library for editing byte codes in Java; it enables Java programs to define a new class at runtime and to modify a class file when the Java Virtual Machine (JVM) loads it. The implementation of the **LGenericProxy** component implements the GenericProxy interface (figure 4). It is based on the Java reflection API. As defined in the java API specification [12],

the java API java.lang.reflect provides classes and interfaces for obtaining reflective information about classes and objects. Reflection allows programmatic access to information about the fields, methods and constructors of loaded classes, and the use reflected fields, methods, and constructors to operate on their underlying counterparts on objects.

For remote communication, we have used the Universal Plug and Play (UPnP) [21] technology. A UPnP network consists of UPnP devices that act as servers to UPnP control points, the clients. The control point can search for devices and invoke actions on them. The **RServer** component, as shown in figure 9 (a), integrates the network communication aspects like remote method call and event processing on behalf of the server component it represents and is implemented as a UPnP device. For that purpose, it uses the services (interfaces) of the server component, generates a UPnP device and services (actions and stateVariables) descriptions and starts the device to be detected (by the UPnP control points) over the network. When it receives from control points a call as a UPnP action, it translates it as a call to the appropriate method of the LGenericProxy component which in turn calls the associated method of the server. When it is notified, by the **MonitoringBySubscription** component, for a change of one of the server properties it modifies the related state variable and then the device sends an event over the network with the new value of the property so the control points that has subscribed to that change receives this notification.

The **ServerProxy** component, in figure 10 (a), is used in place of a server when this last is on another device. Thanks to the javasist API, it is Java generated implementation of the services (interfaces) of the server. The calls to its methods are forwarded to the appropriate method of the component, implementing the GenericProxy interface, it references.

The **RGenericProxy** in figure 10 (a), is implemented as an UPnP control point. When it starts, it searches for a UPnP device with the same type of RServer component and subscribe to the UPnP events related to the change of the variables state of this server component. Each time it receives a state variable change event it notifies the MonitoringBySubscription component that in turn notifies the interested subscribers (e.g. ServerProxy component). The RGenericProxy also implements the GenericProxy interface. Each call to a method of this interface is transformed as an UPnP action call to the device server (i.e. RServer component).

Finally, to allow a component to notify the MonitoringBySubscription for the change of its properties, the open-source Javassist is used, at runtime, to inject the notification code in the property setter of the byte-code of the component implementation (class).

VI. Related Work

The monitoring and reconfiguration issues have been extensively studied in different contexts, notably in the area of software components that has become an accepted standard for building complex applications. In this section, we detail some of the existing related approaches as well as their limitations. But let's first give an overview of some component models that become an accepted standard for building complex applications.

The OSGi component model [17] enables components to hide their implementations from other components while communicating through services that are shared between components. It allows specification of properties when registering components in the service registry. This ensures searching and binding of components having specific properties. An advantage is that these properties cover a wide range of characteristics such as context, quality of service, and other non-functional aspects. However, the major drawback is that the specification of properties is done at the code level; so properties are tied to the functional code.

Another relevant component model is SCA [5], which provides a programming model for building applications and systems based on a Service Oriented Architecture. More recently, there has been SCA extension for event processing [6]. It is based on the publish/subscribe model, which allows components to produce and consume events through channels. These events may be sent to notify a consumer about changes occurring on the producer's side, without the consumers having knowledge of the producer's functionality. Currently, the SCA specifications of existing runtimes do not provide any transformation mechanism to render a property nor monitorable neither reconfigurable.

The PCOM component model [1][18] allows specification of distributed applications made up of components. Components reside within a component container that contains listeners allowing application programmers to be notified whenever a parameter or a communication changes or new components are discovered. Each component explicitly specifies its dependencies in a contract, which defines the functionality offered by the component, i.e., its offer, and its requirements with respect to local resources and other components. To model the required properties, the syntactical description can be enriched with properties of typed name-value pairs for offers and typed name-value-comparator triples for requirements. Using this description, the system can automatically determine whether an offer can satisfy a requirement.

In a recent work [10], the PCOM container was extended by two services; assembler and application manager, to separate the task of calculating the configuration from the application execution. The application manager is responsible for managing the life cycle of application and also the selection of the configuration algorithm. However, the assembler is responsible for calculating a valid configuration. Clearly, using a fully distributed assembler, for instance, requires the availability of an instance of this assembler on each device.

One of the limiting factors of PCOM resource description is that resources are not standardized: there is no formal way of resource description and the different types of resources can not be distinguished from one other. Another problem is that the non-functional aspects of an application are also treated in the programming API. This means the component developer has to take care of the non-functional aspects (monitoring,

binding, and unbinding of the components) at the code level. Moreover, PCOM does not support the remote monitoring of components' parameters, since each container is reponsible of the monitoring of the hosted components.

In [3], Beugnard et al propose a process that makes functional aspects of components independent from observational ones. This separation of concerns allows the advantage of changing observations without modifying the core part of components. They have also defined a set of predefined components dedicated to observation that can be attached to any functional component. Both the functional and observation components are defined declaratively and then using a kind of weaver they can be integrated to result in context-aware components.

While their approach is quite general, we can identify two limitations compared to our proposed approach. First, instead of defining a few dedicated observation components, we propose that any of the components in our system can be transformed into observation component by adding to it the capabilities of observation. Second, since this transformation is done dynamically, we can selectively specify the properties of a component that we want to make observable.

As a monitoring and a reconfiguration middleware, we cite MADAM (Mobility and ADaption enAbling Middleware) [8], which is a middleware for runtime adaptation with: context management, adaptation management and configuration management. Their objective is to adapt the application at runtime in response to context changes. For this purpose, the middleware provides a Context Manager to monitor context when this latter changes. It is responsible for context reasoning, such as aggregation, derivation, and prediction in order to provide the Adaptation Manager component with relevant context information when context changes occur. The Configurator middleware component is responsible for reconfiguring an application by deleting or replacing component instances, instantiating components, transferring states, etc.

However, MADAM middleware does not support the remote monitoring and the remote reconfiguration and it is limited to the local scope. Moreover, it does not support the transformation of components to render them monitorable or reconfigurable resources.

In the same context, we cite the AwareWare middleware that tends to facilitate the development of applications to be more adaptive in such a heterogeneous environment [22] [23]. The AwareWare middleware consists of Awareness measurement tools that are used to measure and to collect awareness data. The awareness manager organizes these tools and provides system independent query and notification interfaces for adaptive applications. They consider two basic methods, pull and push, for querying distributed awareness information. An adaptation decision module and adaptation policy language are used to reconfigure applications. The adaptation policy defines rules to determine how the application changes its behaviours, by changing the applications component inter-connections and tuning parameters.

However, the reconfiguration is limited to the hosted components and it does not cover the remote components. Moreover, AwareWare middleware does not support the transformation of components to monitorable of reconfigurable resources. However, our framework is able to transform the components into monitorable or reconfigurable resources to reply to the components' request.

In another recent related work [7], a reconfiguration middleware handles the reconfiguration of applications in heteregenous environment. Towards this objective, Corradi et al propose to partition the middleware logic into two reconfiguration layers that basically feature an application logic layer and a non-functional layer on top of a very minimal kernel layer. Each reconfiguration layer features a monitoring engine whose aim is to keep track of current status of elements of the (monitored) layer above. A reconfiguration enactment engine concretely executes the reconfiguration actions determined by policies.

Compared to our approach, the major drawback of the reconfiguration middleware that does not support the remote monitoring and the remote reconfiguration of applications. Further, the proposed middleware does not consider the transformation of components to monitorable or reconfigurable resources despite the components' request.

The approaches in [9] and [11] focus on the adaptation of system behaviour at runtime. They consists of reusable infrastructure corresponding to probes, and resource discovery components to support respectively, monitoring of properties changes, and quering for new resources. An adaptation engine is used to carry out the necessary reconfiguration by using some adaptation operators and adaptation strategies.

However, the reconfiguration of components does not cover the remote components and it is limited to the local scope. Moreover, they do not support the transformation of components to render them monitorable or reconfigurable by other components. However, in our approach, we propose to model explicitly these features through required properties of components, and even these latter are not defined by default as monitorable or reconfigurable resources, our framework is able to transform them to reply to components request.

In [15], Melisson et al propose pervasive binding to provide support for service discovery in SCA-based applications. This binding is called UPnP binding since it is based on UPnP protocol [21]. Towards this objective, they propose to integrate UPnP into FRASCATI platform (a SOA platform for the SCA standard)[19] to support the remote call of a service that is advertised via the UPnP protocol. The FRASCATI platform was modified to supply spontaneous communications between SCA components using UPnP bindings.
However, in our approach, the SCA composite is transformed by adding UPnP components allowing its remote monitoring and its remote reconfiguration using the existing open-source SCA runtimes (e.g. Newton, Tuscani, etc.). Moreover, their approach is limited on the remote call and it does not take into account neither the remote monitoring nor the remote reconfiguration. However, in our present work, we rely on UPnP protocol to manipulate remote components by monitoring their

changes and reconfiguring their properties in addition to the remote call.

Project	Monitoring		Reconfiguration	
	Local	Remote	Local	Remote
Madam Middleware [8]	√		√	
Rainbow Middleware [9]	√		√	
AwareWare Middleware [22] [23]	√	√	√	
Reconfiguration Middleware [7]	√		√	
PCOM Component Model [1][10]	√		√	√
SLCA Component Model [11]		√	√	
Our Framework	√	√	√	√

TABLE I
EXAMPLES OF MONITORING AND RECONFIGURATION APPROACHES

In table I, we summarize the most important features related to some of the cited middlewares and component models. We compare them regarding the monitoring level, i.e., if it is executed locally or remotely, and the similarly the reconfiguration level. As it can be seen, most of the given approaches focus on the local monitoring and/or local reconfiguration mechanisms. There are limited approaches that consider remote monitoring and/or remote reconfiguration. And even if they exist, their presented mechanisms are not defined in appropriate way. Moreover, in our knowledge, none of the cited middleware supports the transformation of components to render them monitorable or reconfigurable resources.

VII. CONCLUSION AND FUTURE WORK

In this article, we proposed an approach for monitoring and dynamic reconfiguration of component-based applications. The flexibility offered by our approach is that any software, hardware, or network component that one wants to monitor or reconfigure, but that does not offer these capabilities inherently, can be transformed to offer these functionalities — given that they are representable by a software component. These aspects are treated independently of the functional code and, hence, do not make the situation more complex for the designers and developers.

For this purpose, we proposed a generic component model that allows unified specification of hardware, software, and network components. In our component model, a component specifies its provided and required services, and its properties, but in addition, it also specifies the required properties that are to be monitored or reconfigured. If the required properties are not monitorable or reconfigurable by default, transformation processes are done dynamically by our framework to reply to the component request. Some Java implementation details of the prototype were provided in the article.

We are integrating our prototype into an SCA runtime. This will allow us to test the feasibility of our approach in real-world scenarios.

VIII. ACKNOWLEDGMENTS

This work is partially supported by French ANR through Seamless (SEamless and Adaptive Services over MultipLe AccEsS NetworkS) project number 07TCOM018.

REFERENCES

[1] Christian Becker, Marcus Handte, Gregor Schiele, and Kurt Rothermel. PCOM - A Component System for Pervasive Computing. In *Proceedings of the Second IEEE International Conference on Pervasive Computing and Communications*, PERCOM '04, pages 67–76, Orlando, Florida, USA, 2004.

[2] Imen Ben Lahmar, Hamid Mukhtar, and Djamel Belaïd. Monitoring of non-functional requirements using dynamic transformation of components. In *Proceedings of the 6th International Conference on Networking and Services*, ICNS'10, pages 61–66, Cancun, Mexico, 2010.

[3] Antoine Beugnard, Sophie Chabridon, Denis Conan, Chantal Taconet, Fabien Dagnat, and Eveline Kaboré. Towards context-aware components. In *Proceedings of the first international workshop on Context-aware software technology and applications*, CASTA '09, pages 1–4, Amsterdam, Netherlands, 2009.

[4] Eric Bruneton, Thierry Coupaye, Matthieu Leclercq, Vivien Quéma, and Jean-Bernard Stefani. The fractal component model and its support in java: Experiences with auto-adaptive and reconfigurable systems. *Software Practice and Experience (SP&E)*, 36:1257–1284, 2006.

[5] Open SOA Collaboration. SCA Assembly Model Specification V1.00. http://www.osoa.org/, 2007.

[6] Open SOA Collaboration. SCA Assembly Extensions for Event Processing and Pub/Sub V1.00. http://www.osoa.org/, 2009.

[7] Antonio Corradi, Enrico Lodolo, Stefano Monti, and Samuele Pasini. Dynamic reconfiguration of middleware for ubiquitous computing. In *Proceedings of the 3rd international workshop on Adaptive and dependable mobile ubiquitous systems*, ADAMUS'09, pages 7–12, London, United Kingdom, 2009.

[8] Jacqueline Floch, Svein Hallsteinsen, Erlend Stav, Frank Eliassen, Ketil Lund, and Eli Gjorven. Using architecture models for runtime adaptability. *IEEE Software*, 23:62–70, March 2006.

[9] David Garlan, Shang-Wen Cheng, An-Cheng Huang, Bradley Schmerl, and Peter Steenkiste. Rainbow: Architecture-based self-adaptation with reusable infrastructure. *IEEE Computer*, 37:46–54, October 2004.

[10] Marcus Handte, Klaus Herrmann, Gregor Schiele, and Christian Becker. Supporting pluggable configuration algorithms in pcom. In *Proceedings of the Fifth IEEE International Conference on Pervasive Computing and Communications Workshops*, PERCOMW'07, pages 472–476, NY, USA, 2007.

[11] Vincent Hourdin, Jean-Yves Tigli, Stéphane Lavirotte, Gaëtan Rey, and Michel Riveill. SLCA, composite services for ubiquitous computing. In *Proceedings of the International Conference on Mobile Technology, Applications, and Systems*, Mobility'08, pages 11:1–11:8, Yilan, Taiwan, 2008.

[12] Java 2 Platform API Specification. http://download-llnw.oracle.com/javase/1.4.2/docs/api/java/lang/reflect/package-summary.html.

[13] JAVA programming Assistant. http://www.csg.is.titech.ac.jp/ chiba/javassist/.

[14] Cédric Kiss. Composite capability/preference profiles (cc/pp): Structure and vocabularies 2.0. w3c working draft 30 april 2007. http://www.w3.org/TR/2007/WD-CCPP-struct-vocab2-20070430/, 2007.

[15] Rémi Mélisson, Daniel Romero, Romain Rouvoy, and Lionel Seinturier. Supporting Pervasive and Social Communications with FraSCAti. In *3rd DisCoTec Workshop on Context-aware Adaptation Mechanisms for Pervasive and Ubiquitous Services*, CAMPUS'10, Amsterdam, Netherlands, 2010.

[16] Hamid Mukhtar, Djamel Belaïd, and Guy Bernard. A policy-based approach for resource specification in small devices. In *Proceedings of the second International Conference on Mobile Ubiquitous Computing, Systems, Services and Technologies*, pages 239–244, Valencia, Spain, 2008.

[17] OSGI. Open services gateway initiative. http://www.osgi.org, 1999.

[18] Stephan Schuhmann, Klaus Herrmann, and Kurt Rothermel. A framework for adapting the distribution of automatic application configuration. In *Proceedings of the 5th international conference on Pervasive services*, ICPS'08, pages 163–172, Sorrento, Italy, 2008.

[19] Lionel Seinturier, Philippe Merle, Damien Fournier, Nicolas Dolet, Valerio Schiavoni, and Jean-Bernard Stefani. Reconfigurable SCA applications with the frascati platform. In *Proceedings of IEEE International Conference on Services Computing*, SCC'09, pages 268–275, Bangalore, India, 2009.

[20] Clemens Szyperski. *Component Software: Beyond Object-Oriented Programming*. Addison-Wesley/ACM Press, Boston, MA, USA, 2nd edition, 2002.

[21] UPnP Forum. UPnP Device Architecture 1.1. http://www.upnp.org, 2008.

[22] Qinag Wang and Liang Cheng. Awareware: an adaptation middleware for heterogeneous environments. In *IEEE International Conference on Communications*, Paris, France, 2004.

[23] Qinag Wang and Liang Cheng. A flexible awareness measurement and management architecture for adaptive applications. In *IEEE Global Telecommunications Conference*, GLOBECOM'04, Dallas, Texas, USA, 2004.

DSCTP Congestion Control Algorithm Based on Dynamic Policies

Jie Chang, Wen'an Zhou, Junde Song, Feng Yu, Zhiqi Lin
Beijing University of Posts and Telecommunications, China
cjbupt@gmail.com, zhouwa@bupt.edu.cn, jdsong@bupt.edu.cn, yfbupt@gmail.com, linzhiqi07@gmail.com

***Abstract*—This paper introduces DSCTP (Dynamic Stream Control Transmission Protocol), a sender-side, transport-layer protocol that modifies the standard Stream Control Transmission Protocol (SCTP) protocol. Although SCTP provides support for multi-homing, the basic reason for such a provision was to improve reliability of associations, simultaneous transfer of new data to multiple paths is currently not allowed in SCTP. DSCTP adopts SCTP's multi-homing feature to distribute data across multiple end-to-end paths in a multi-homed SCTP association. DSCTP aims at exploiting congestion control algorithm of Transmission Control Protocol (TCP) and SCTP. Through the use of dynamic policy management framework, DSCTP switches the transmission onto the alternate path using DSCTP's flexible path management capabilities. We can gain significant throughput improvement if simultaneously transfer new data across multiple paths to the receiver. In this article, these techniques include transmission start, flow control, network monitoring, generation of policies, routing switching, and congestion recovery. Extensive simulations under different scenarios highlight the superiority of the proposed solution with respect to TCP and the standard SCTP implementation.**

Keywords-DSCTP; dynamic policy management framework; SCTP; congestion control

I. INTRODUCTION

Computer networks have experienced an explosive growth over the past few years and with that growth have come severe congestion problems [17]. In recent years, the portfolio has been surging as the network used widely. Especially in recent years, the development of IP telephone (VoIP) and Internet Protocol Television (IPTV), transferring voice, video and multimedia information in the Internet becomes inevitable. The core problem is the transmission of multimedia data services and real-time communications, and how to provide a certain quality of services for these services. From the protocol of the transport layer, the traditional transport protocols, TCP provides both reliable data transfer and strict order-of-transmission delivery of data. Some applications need reliable transmit without sequence maintenance, while others would be satisfied with partial ordering of the data. In both of these cases, the head-of-line blocking offered by TCP causes unnecessary delay [10]. User Datagram Protocol (UDP) lacks reliable guarantee mechanism for the transmission, and because it has no congestion control mechanism, so the unfair competition for bandwidth can causes network congestion even collapse [1]. In recent years, SCTP protocol was proposed by IETF, called as a modified TCP protocol, which has both advantages of TCP and UDP. Congestion control is one of the basic functions of SCTP. For some applications, it may be likely that adequate resources will be allocated to SCTP traffic to ensure prompt delivery of time-critical data -- thus, it would appear to be unlikely, during normal operations, that transmissions encounter severe congestion conditions. However, SCTP must operate under adverse operational conditions, which can develop upon partial network failures or unexpected traffic surges. In such situations, SCTP must follow correct congestion control steps to recover from congestion quickly in order to get data delivered as soon as possible. In the absence of network congestion, these preventive congestion control algorithms should show no impact on the protocol performance [11]. Due to the bottlenecks of the current network equipment handling capability, service needs a lot of data transfer currently, so it often results in the status of network congestion. Although conditions can be alleviated by improving hardware, due to the limit of the development of hardware manufacturing technology and economic costs, frequent replacement of hardware is usually unrealistic and not a long-term plan. Therefore, only from the perspective of improving network congestion control to improve the network condition is feasible.

Congestion control is a method used for monitoring the process of regulating the total amount of data entering the network so as to keep traffic levels at an acceptable value. This is done in order to avoid the telecommunication network reaching what is termed: congestive collapse. Congestion control mostly applies to packet-switching network. A wide variety of approaches have been proposed, however the "objective is to maintain the number of packets within the network below the level at which performance falls off dramatically" [16].

Congestion control is usually focused on the design of transport layer protocol, like TCP. However, whether TCP or SCTP are not have desired performance. It is necessary to provide a new transmission protocol to meet the needs of real-time data for streaming media. The remaining sections of paper are aimed to apply dynamic policies into SCTP protocol. With a comprehensive analysis of the characteristics of the traditional congestion control mechanism and to fully exploit the own characteristics of SCTP, and based on the idea of dynamic policy management, a modified SCTP protocol – DSCTP was proposed. DSCTP is not a new protocol, only to increase the dynamic SCTP congestion controls mechanisms, so that it can always be adjusted in data transmission according to the network environment. Section II overviews several ideas and mechanisms used by SCTP; some are compared with TCP and UDP to highlight similarities and differences. Section III

formulates the characteristics and the defect of SCTP congestion control mechanism. Section IV formulates a refined dynamic policy management framework based on Ponder model. In Section V, the framework is applied into SCTP and describing dynamic congestion control algorithm of DSCTP. In Section VI, simulation results show DSCTP dynamic congestion control protocol can not only reduce the risk of overall network congestion occurs, but also improve the overall efficiency of the network.

II. OVERVIEW OF SCTP

In this section, we provide an overview of the protocols that we are using in this paper: SCTP [14]. SCTP is defined in RFC2960 [14] with changes and additions included in the Specification Errata [15].

A SCTP versus TCP and UDP

Today most applications use either TCP [10] or UDP [1]. Applications that need a reliable in-order delivery of the bytes sent by its peer uses TCP, whereas ones that can tolerate a certain degree of loss prefer UDP, primarily because UDP provides speedier delivery of packets. Most applications prefer TCP over UDP and applications use TCP including file transfer applications, electronic mail and the worldwide web. UDP is used by streaming audio/video applications for which timely delivery is more important than reliability. SCTP was recently adopted by IETF, and is a reliable transport protocol that operates on top of a connectionless packet based network such as IP. It was originally designed to be a general purpose transport protocol for message oriented applications, as is needed for the transportation of signaling data. SCTP is a transport layer protocol and its services are at the same layer as TCP and UDP. Instead of the three phase connection setup for TCP and best effort to delivery for UDP, the initialization of an association is completed after the exchange of four messages. Another important difference between SCTP and TCP is the support for multi-homed nodes in SCTP, i.e. nodes which can be reached using more than one IP addresses [13]. So if the routing is configured in such a way that these IP addresses are accessible through different paths, multi-homing gives SCTP a certain network level fault tolerance.

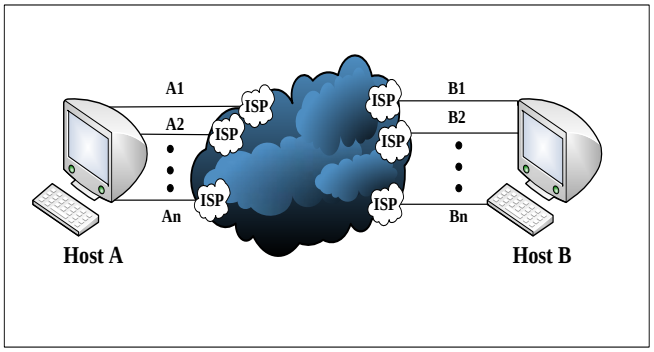


Figure 1. Example Multi-homed Topology

Unlike TCP and UDP, SCTP supports multi-homing at the transport layer to provide redundancy at the path level, thus increasing association survivability in the case of a network path failure [8]. An SCTP endpoint may bind to multiple IP addresses during association initialization. As the Figure 1 shows, we can contrast SCTP with TCP to further explain SCTP's multi-homing feature. There are n^2 distinct TCP connections are possible between Hosts A and B: (A_1, B_1) , (A_1, B_2) , (A_1, B_3) ,..., (A_1, B_n) , (A_2, B_1),..., (A_2, B_n),..., (A_n, B_1),..., (A_n, B_n) .SCTP congestion control algorithms are based on RFC 2581 [12], and include Selective Acknowledgement (SACK)-based mechanisms for better performance. Similar to TCP, SCTP uses three control variables: a receiver's advertised window (RWND), a sender's congestion window (CWND), and a sender's slow start threshold (SSTHRESH).

At association startup, a primary path is defined for each SCTP endpoint, and is used for normal sending of SCTP packets. The idea of SCTP load balancing is to use only primary path to transfer. Because the congestion control mechanism of SCTP is similar to TCP, apply congestion control mechanism to load sharing, there would be unnecessary fast retransmit problems.

A single SCTP association (session), is able to use alternatively anyone of the available IP-addresses without disrupting an ongoing session. However, this feature is currently used by SCTP only as a backup mechanism that helps recovering from link failures. SCTP maintains the status of each remote IP address by sending Heartbeat messages and it is thus able to detect a specific link failure and switch to another IP address. Another novel feature is that SCTP decouples reliable delivery from message ordering by introducing the idea of streams. The stream is an abstraction that allows applications to preserve in order delivery within a stream but unordered delivery across streams. This feature avoids HOL blocking at the receiver in case multiple independent data streams exist in the same SCTP session. Congestion control was defined similar to TCP, primarily for achieving TCP friendliness [14].

B SCTP Reseach Activities

SCTP is standardized in the IETF first in the Signaling Transport Work Group (SIGTRAN WG) and since 2001 it has found a new home in the transport Area Work Group (TSV WG). The Protocol Engineering Laboratory (PEL) directed by Professor Paul Amer at the University of Delaware is dedicated to the research, development, and improvement of new and existing computer network protocols. PEL researchers are investigating innovative transport protocol alternatives to TCP and UDP (such as SCTP) emphasizing these alternatives within army networks to provide efficient communications under mobile, ad-hoc network conditions [9]. The ongoing development of alternative transport protocols (e.g., SCTP) which provide several benefits over traditional transport protocols such as

TCP and UDP, especially in supporting army and/or multimedia applications. Current focus is on transport layer multi-homing and multi-streaming.

III. SCTP CONGESTION CONTROL

A Compare of Network congestion and congestion control

Before SCTP congestion control mechanisms are discussed, we must first clear and definite two concepts: network congestion and congestion control.

When an excessive number of packets reach a certain part of the communication network, there is no time for this part of network to deal with all the data, resulting in decreasing the network performance, and even lead to a suspend of the network communication services, which produce the network congestion. Congestion control has no exact definition, its purpose is to use some means of avoiding the network congestion situation, while in case of congestion the state can resume as soon as possible.

The reason may be due to limited storage space as node cannot cache all the receiving packets and the packet loss or due to data units arrive at the number far exceeds of node's processing capacity and cause delay, and the fundamental reason is that network data traffic flow and node processing speed.

B Congestion Control in TCP

TCP and UDP are the most common IP network transport layer protocol, TCP has its own congestion control mechanism, while UDP has not, although some scholars in recent years have been studying how to add UDP congestion control mechanism, but this is not in the discussion areas.

In general, TCP congestion control mechanism can be divided into open-loop control and closed-loop control of two kinds. Open-loop control focus on prevention, hopes to avoid congestion by perfect design, closed-loop control is solution-focused, trying to relieve and control after the congestion occurs. Therefore, TCP congestion control algorithm designed mainly from two aspects, the basic TCP congestion control algorithms will include: slow start, congestion avoidance, fast retransmit and fast recovery [12].

1) Slow start and congestion avoidance: TCP congestion control requires two main parameters, Congestion Window (CWND) and Slow-Start Threshold (SSTHRESH). Sender sends window SWND = min (CWND, RWND), RWND is the receiver window and it usually has little impact on the send window. Therefore CWND directly determine the size of the send window, and the size of send window directly determine the size of the send packet. CWND = 1 when data sent initially, when each sent data packet is successfully confirmed, CWND will be increased by 1. If all the data packets in the send window have been identified, CWND will be double. The growth process of the congestion window is called slow start; in fact, in the slow start phase, CWND in the ideal case is exponential growth. However, CWND cannot be unlimited growth, we must set a threshold SSTHRESH for it, if CWND> = SSTHRESH, CWND can be increased by 1 only when all the data packets in the send window have been confirmed. This time is called the congestion avoidance phase, CWND increases linearly.

TCP will set a retransmission timeout RTO when a data packet is sent, when the round-trip time RTT exceeds RTO's setting time, it means that this packet is lost, and has to retransmit this packet, which is called network congestion. At this point, the network come into congestion recovery phase, TCP congestion control algorithm shall do the following:

Set CWND = min (4*MTU, max (2*MTU, 4380byte)) [14] [15];

SSTHRESH = RWND;

Set SSTHRESH = max (CWND/2, 4*MTU);

Re-enter slow start phase

2) Fast retransmit and fast recovery: As waiting for the RTO will make the network transmission efficiency decreased, fast retransmit mechanism would like to find a way to search for an alternative way of congestion discovery. Whenever an endpoint receives a SACK that indicates that some TSNs are missing, it should wait for two further miss indications (via subsequent SACKs for a total of three missing reports) on the same TSNs before taking action with regard to Fast Retransmit. And then enter the congestion recovery phase, during this period the system shall do the following:

Set SSTHRESH = max (CWND/2, 4*MTU);

Set CWND = max (CWND/2, 4*MTU) +3;

Enter the congestion avoidance phase directly;

This is the basic algorithm of fast recovery. During the period of congestion recovery phase, using fast recovery to replace slow start is known as TCP congestion control algorithm.

C SCTP Congestion Control Mechanism

SCTP protocol is called a modified TCP protocol, which has the characteristics of both TCP and UDP. SCTP is connection-oriented in nature, but the SCTP association is a broader concept than the TCP connection. The term "stream" is used in SCTP to refer to a sequence of user messages that are to be delivered to the upper-layer protocol in order with respect to other messages within the same stream. This is in contrast to its usage in TCP, where it refers to a sequence of bytes. TCP guarantees in-sequence delivery of data to its upper-layer protocol within a single TCP session. This means that when TCP notices a gap in the received sequence number, it waits until the gap is filled before delivering the data that was received with sequence numbers higher than that of the missing data. On the other hand, SCTP can deliver data to its upper-layer protocol even if there is a gap in TSN if the Stream Sequence Numbers are in sequence for a particular stream (i.e., the missing DATA

chunks are for a different stream) or if unordered delivery is indicated. Although this does not affect CWND, it might affect RWND calculation. The biggest difference between SCTP and TCP, however, is multi-homing.

Because the multi-homing nature, the unique nature of SCTP, its congestion control mechanism is different from TCP. Once the congestion happened on the link, the transmitter will choose another IP address with the same host to continue the transmission. As SCTP provide the connection oriented service, and the connection for each IP address has already established, there is no need to setup a new connection. In this way the overhead of setting up and releasing the connections can be minimized to a large extent. In traditional way of STCP, the slow start and the congestion avoidance phases are still needed by the link, however, because of using SACK in SCTP to confirm, the transmitter only need to retransmit packets which have not be confirmed when the link enter the fast recovery phase after the congestion happened. SACK changes the acknowledge mechanism of TCP, TCP only confirm packets that all ready received, while SACK would send the acknowledgment which contain the disordered information to the receiver, by doing this the transmitter will minimize blindness of the retransmission.

D The Defect of the Traditional Congestion Control Mechanism

The traditional method of congestion control emphasis on closed-loop control, but open-loop control has not much achievement. When dealing with congestion control, both of TCP and SCTP take the way of "congestion discovery"—a kind of mechanism for congestion control. Although SCTP reduces the blindness of retransmission by using the SACK confirmation method, it can still not get rid of hysteresis due to passive wait for the congestion. The method of congestion control is too simple, the initial start is so slow and some other problems are all have defects. Besides, TCP and current SCTP use only one destination address at any given time to transmit new data. SCTP restricts sending new data, which can act as probes for information (such as available bandwidth, loss rate, and RTT) to only one primary destination. Consequently, an SCTP sender has minimal information about other paths to a receiver [7].

In order to improve these deficiencies, this paper presents a SCTP-based congestion control algorithm. The protocol which uses of this new congestion control algorithm is called dynamic SCTP (DSCTP). DSCTP fully exploits SCTP feature, also it is flexible, proactive and accurate to carry on the congestion control. More importantly, it can adjust its congestion control policies according to network conditions before the congestion happen. DSCTP is not a new protocol, it only improves SCTP congestion control mechanism, and the basic idea comes from the dynamic policy management. DSCTP uses multiple destinations simultaneously, CWND growth in DSCTP demands tracking the latest TSN received in order per destination, information not coded directly in a SACK. On the other hand, a DSCTP sender maintains more accurate information about all paths, since new data are being sent to all destinations concurrently. This information allows a DSCTP sender to better decide where to retransmit.

IV. DYNAMIC POLICY MANAGEMENT

Policy is a set of rules for the management and control of network resources. The essence of policy based network control is to view network as a finite state machine, the node within the network is assigned a specific state according policy rule [2].

Traditionally static policy is generated by the PMT [3] through the administrator, while the dynamic policies are generated based on the parameters which are collected by the network, feedback mechanisms and policy algorithms. Networks are controlled by dynamic policies, which fluctuate accompanied by the changing environmental message, feedback from contributing factors, so that dynamic policies are self-adjust and can reflect the real network environment.

Policy repository should be maintained from instant to instant; policies are added, deleted or modified upon the alteration of network environment. Real-time network management can be achieved automatically without intervention of human network operators. This is a big virtue of dynamic policy management process. Dynamic policy management makes full use of network resources, upgrade efficiency and guarantee regular service provision.

Dynamic policy management framework is based on IETF policy management framework [4] and PONDER policy framework. The framework clearly describes how the dynamic policy management network works. In Fig. 1, the following module is shown:

PMT (Policy Management Tool): It provides a visual management interface, the policy administrator through PMT to add, modify and delete policies.

PB (Policy Base): It stores policy.

PEP (Policy Enforcement Point): Execution of specified actions, for example: to request policy, to update policy, to delete policy.

DPS (Dynamic Policy Scheduler): Centralized management and scheduling of policy.

DPIP (Dynamic Policy Information Point): It collects environmental information continually and uses this information to update policy.

DFQB (Forecast Query Builder): It predicts environmental information and assists DPIP to create new dynamic policies.

DPC (Dynamic Policy Cache): The policy pre-distributed by PDP is stored in the policy cache.

DPS (Dynamic Policy Self-management): It detects conflicts between policies.

PEP (Policy Enforcement Point): It executes of specified actions.

PDP (Policy Decision Point): It is composed of trigger detection and handling, rule location and applicability analysis, network and resource-specific rule validation and device adaptation functions.

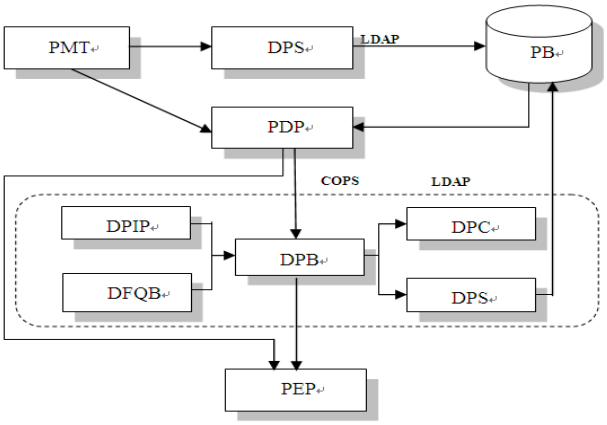


Figure 2. Dynamic Policy Management Framework [5]

PMT dynamically stores policy objects to PB through DPS and maintains PB. When it is necessary, PMT passes policy objects to PDP. PDP can obtain the corresponding policy object from PB according to the policy request submitted by PEP. DPIP continuously collects environmental information, with the assistance of DFQB, in order to adapt to the ever-changing network environment. Then the collected information is passed to PDP to produce new dynamic policies. DPC is used to store policies which are issued in advance by PDP to the DPC. Finally DPS detects policy conflicts, then, it will place policies in PB if no conflict is detected. And then DPB will pass the corresponding dynamic policy rules to PEP. DPS takes a centralized management and scheduling upon policies, taking advantage of the idea of scheduling algorithm. The conventional policy storage is putting policies directly into PB. When request for policies arrives, PDP will search for the suitable policies by doing comparison one policy by one policy. It is such a difficult procedure when it comes to large-scale PB, consuming a long time to search out one policy. On contrary, DPS can centralized manage and schedule policies so that it can decrease the conflicts and increase the efficiency [6].

V. DYNAMIC ALGORITHM DESCRIPTION

DSCTP uses dynamic congestion control algorithm which describes the congestion control mechanism in detail in respect of transmission start, flow control, network monitoring, generation of policies, routing switching, and congestion recovery. First, we appoint sender as A, receiver as B, A and B jointly appoint a number of IP address $\{A_1, A_2, ..., A_n\}$ $\{B_1, B_2, ..., B_n\}$. If the service X is intended to send data from A to B, we assume that host A_1 is selected as the sending host.

A Transport Start Phase

Compared with the traditional SCTP, the major improvements of DSCTP are listed in the following aspects：

- When A_1 is intend to send data, firstly, select a host B_i randomly.
- If A_1 had never sent data to B_i, A_1 sends a request REQ to PDP, and REQ includes the current service type X as well as the environmental parameters $PARAM_1, ..., PARAM_n$ collected from the DPIP; If A_1 has ever sent data to B_i, A_1 sends a request REQ to PDP, and REQ includes the previous value of CWND and SSTHRESH in addition to the current service type X and the environmental parameters collected from the DPIP.
- PDP receives REQ, and if A_1 is the first time send data to B_i, PDP sends a query request to PB to query the matching policies. If PDP finds the appropriate policies in PB, then PB returns PDP a policy packet which includes the value of CWND and SSTRESH currently should be set. If A_1 is not the first time to send data to B_i, firstly, PDP queries the policies in the DPC, and if the types of service are the same, then DPC returns the policies to the PDP; else, PDP queries the policies in the DPB, if the match is found, then DPB returns the policies to PDP; otherwise, PDP queries the policies in the PB, if find the appropriate policies, then PB returns it to PDP. The final policies PDP obtains should be assertion whether it is suitable for A_1 send data to B_i.
- If PDP can obtain the return policies, it will further modify the policies according to its own special rules, and return the final result to A_1; if PDP cannot obtain the return the policies, then PDP will return A_1 a policies in accordance with the default rule.
- If A_1 is the first time to send data to B_i, A_1 will set its own CWND and SSTRESH according to the policies received. If A_1 is not the first time to send data to B_i, and return the assertion of policies is TRUE, that means A_1 can send data to the current B_i, and send data according to the value of CWND and SSTRESH before; otherwise, A_1 selects other

IP address to resend to PDP and repeats the process before.

B Congestion Avoidance Phase

In the control of sending quantity of packets, DSCTP also uses TCP and SCTP mechanism. In other words, use CWND and SSTHRESH control of numbers of packets. CWND and SSTHRESH in a similar way with slow start and congestion control. The only difference is CWND not start with 1. Whenever CWND is less than SSTHRESH, an SCTP endpoint must use slow-start algorithm to increase CWND only if the current congestion window is being fully utilized. When CWND is less than or equal to SSTHRESH, Only when all of packets are confirmed can CWND be increased; otherwise, CWND must not be increased.

C Network Monitoring Phase

Detection of network conditions is a core operation of dynamic adaptive network. A has to constantly monitor the link status of each B_i. It should need to take appropriate actions or continue to complete sending segment of packet or continue to send packet after switching IP address when certain link of B_i is found of congestion. Detailed process is described as follows:

- The request of determining the status of connection should be sent from A to PDP, which includes of its CWND and SSTHRESH.
- The value of M should be calculated according to certain rules of CWND, SSTHRESH, environmental parameters of DPIP and forecast parameters of DFQB. M is a number from 0 to 10, which indicates the level of the current status of connection. The greater the value of M indicates that the worse the status of connection should be, and then the less appropriate the data should be sent.
- PDP should get the value of M, then according to the current type of service and other special rules which determine the level of the current link status. In addition, PDP should write the new policies into DPB. The new policies should include the current environment parameters, such as the value of CWND and SSTHRESH and the level of current links. The meaning of the generation of policies should be that the level of link environment is LEVEL in the current environment when all the environmental parameters do not exceed the current value, CWND and SSTHRESH also do not exceed the current value.
- Write the current generation of policies into DPC and replace the original policies. The latest policies should be saved in DPC forever. If the next is the same type of service, check whether the matching policies of DPC first, because the policies of DPC is closest to the current network status.
- PDP should return a result to A_1 according to the calculated M and its own special rules whether the current link for sending data.

D Dynamic Policy Generation Phase

We explain the dynamic generation of policies. PDP can obtain the corresponding policy object from PB according to the policy request submitted by PEP. Then collect environmental information by DPIP and dynamically determine threshold M.

1) Add Global Load Map: In order to elevate efficiency in the bi-driven algorithm [18], an entity called Global Map (GM) is added to the receiver, indicating the each host's instantaneous load value of the system, such as server load, server busy factor, number of active connections, server hardware and so on. The load value is calculated in accord with the pre-defined rules, taking into weighting factors such as the number of pending requests, request types, host processing capacities etc. The algorithm should be executed as follows. New requests should be assigned to light-load nodes according to pre-defined rules. Meanwhile, all host load values are polled; when one idle host is detected, an uncompleted task in high load value host will be assigned to idle host.

2) Dynamically Determine Threshold M: Dynamic scheduling algorithm use dynamic policy management framework to determinate the threshold M dynamically. DPIP collects performance parameters of every host. When new request comes, the load balancer makes a request to PDP and then should contain operations described as follows:

- The value of each parameter will be mapped to a value between 0 and 10, then according to the importance of each parameter of the system calculate the weighted average value, this is the value of the current system performance:
  $$v' = \frac{v - \min_A}{\max_A - \min_A} * 10$$
  where $\min_A$ is the minimum value of parameter, $\max_A$ is the maximum value, v is the current value and v' is the standard value.
  $$F = (f_1, f_2, \ldots f_n)$$
  where f_i is the weighted average value and $f_i = \mu * v'$. μ is the weight value expressing the importance of parameter. F is the standardization of vector.
- According to F, it would be easy to find w in PB, which is pre-defined.
- According to the vector of λ in DFQB, M value would be calculated:
  $$m_{(t,i)} = \sum_{\alpha \in A} \varphi(\alpha) * \omega_{(t,\alpha)}$$

$m_{(t,i)}$ represents the performance value of host i at the time t, $\varphi(\alpha)$ represents mapping function of attribute α. $\omega_{(t,\alpha)}$ is the weight value of attribute α at the time t.

$$\lambda_{(t,i+1)} = \lambda_{(t,i)} \left| m_{(t,i)} - m_{(t-1,i)} \right| \Big/ \sum_{i=1}^{n} \left| m_{(t,i)} - m_{(t-1,i)} \right| \lambda_{(t,i)}$$

$\lambda_{(t,i)}$ represents the parameter that effects the system of host i at the time t.

$$\mathrm{M}_t = \mathrm{M}_{t-1} + \sum_{i=1}^{n} \lambda_{(t,i)} \left(m_{(t,i)} - m_{(t-1,i)} \right)$$

where M_t is the threshold of the system. As it can be different in the effect to the whole system when the performance of each host changes, we have to get a average value as the threshold for this system.

E Routing Switch Phase

When A receives policies to change routing, the link appears in a congestion situation, then the routing must be switched.

On the one hand the transmitter should continue to send data to the original address, but on the other hand the process above described should be carried out again when it is in the sending process of a data segment, Once it selected an appropriate IP address B_k, then it should immediately stop sending data to B_i and resend all packets to confirm of the current send window, it also need to send a notification along the original link. The passed node should clear the cache of packets from A_1 after it received notice. Then it will send the remaining data of this data segment to B_k. From A to B_i, CWND and SSTHRESH should not be changed and should be retained.

F Congestion Recovery Phase

The dynamic policy generation is based on the forecast and it is characterized by bias. When the congestion occurs, the congestion recovery mechanism of DSCTP is relatively simple and it only waits for enough time to recovery, then send the packet according to the algorithm of DSCTP. It will not affect network communications even when the waiting time is long enough because there are several routings to choose for SCTP, so the sending endpoint can wait for a long time before sending packet when there has congestion. In addition, after congestion, all nodes on the congested link will clear part of the buffer after receiving instructions. Therefore it can be assumed that it makes the link A to link Bi return to normal after the waiting time. Then the data can be sent by the algorithm of DSCTP. Before congestion occurs DPB should write the policies into PB (It should be the same as the tactics stored in DPC) and set the network status into poor. As this policy was originally wrong, so this policy should be removed from the DPB and DPC. This is the process of dynamic error correction.

G Algorithm Analysis

Compared with the traditional SCTP, the major improvements of DSCTP are listed in the following aspects：

- There are more than one transport addresses in the active state that can be used as a destination address to reach that endpoint, so they can share the network load. The traditional SCTP usually uses the same destination address until being instructed by the upper layer to do, otherwise SCTP may change to an alternate destination in the event an address is marked inactive [11]. However, DSCTP can send data to multiple IP address simultaneously, and no matter data is sent to any one IP address, all of which can be correct confirmation, thus this can greatly enhance the flexibility and the overall processing capabilities of the network.
- According to multiple network parameters, DSCTP can implement the congestion control. The traditional congestion control in SCTP only concerned with some congestion control parameters, such as Congestion Window (CWND), Slow Start Threshold (SSTHRESH), Round-trip Time (RTT) and so on. The monitor of congestion appears very rough. However, DSCTP not only concern the above parameters, but also implement congestion control by environmental parameters. In fact, many environmental parameters can be an indicator to some degree: such as the processor payload of the receiver, the size of the cache, the total number of streams on the current transmission link, the total number of router on the link and the CPU used in the router and so on. All of these parameters may indicate a certain level of congestion.
- DSCTP monitor the network status， and also feedback information to predict the status of link. To avoid network congestion, CWND should be incremented by 1*MTU per RTT if the sender has CWND or more bytes of data outstanding for the corresponding transport address. Or passive monitoring the congestion status and then rapid recovery from congestion state through SACK. To predict the possibility of congestion in DSCTP. As potential congestion, if they predict potential congestion, they should take appropriate action to ensure congestion avoidance immediately.
- Like TCP and SCTP, a DSCTP endpoint uses the following three control variables to regulate its transmission rate, such as RWND, CWND and SSTHRESH. SCTP and DSCTP also require one additional control variable, partial_bytes_acked, which is used during congestion avoidance phase to facilitate CWND adjustment. Beginning data

transmission into a network with unknown conditions or after a sufficiently long idle period requires SCTP to probe the network to determine the available capacity. Through the phases of the slow start and the congestion avoidance, to make the network best transport efficiency, but some urgent high-speed data services have to experience this slow process. DSCTP maintains these modes of the slow start and congestion avoidance, but the value of the initial congestion window is not equal to 1 and after recovery from congestion, according to the state of the sending endpoint and the current network environment, the values of CWND and SSTHRESH are not fixed and by all of these environment parameters to define.

- All network activities in DSCTP are in the guidance of dynamic policies. Based on the service characteristics, the requirements of the network and so on, the network administrators create enough permanent policies in advance, which are widely used in the vast majority situations of the network. And also through the feedback mechanism and prediction mechanism to create temporary policies, which are effective for a given period of time because they collect the current network environment parameters. These policies can either be replaced by new dynamic policies, or be deleted by themselves at the end of the life period.

H Deficiency of DSCTP

- DSCTP cannot avoid congestion. In fact, any kind of congestion control mechanism could not completely avoid any congestion. It can only try to reduce congestion. At the same time, it can ensure implementations of fast-retransmit and fast-recovery as fast as possible. The essence of DSCTP is to see the receiving endpoint as a cluster system, by adding dynamic scheduling algorithms to make the network load more balancing. By this way it can increase the processing capacity of the receiving endpoint, it can also reduce the load of the single host. The ultimate goal is to avoid congestion phases. However, when the network load is big enough, DSCTP cannot avoid this situation.
- DSCTP cannot pay much attention to congestion recovery. As the above mentioned, DSCTP mainly focused on how to avoid congestion, but to the question of how to recovery from congestion, it only obtains the values of CWND and SSTHRESH, it does not provide other effective solutions. This is largely based on DSCTP is multi-homed, for DSCTP, it can has multiple links at the same time, if one of the link has not send data for a long time, it would not have too much influence on the network. So if the link has some problem, it should be wait enough time for the network recovery, not pay much attention on how to fast-recovery from congestion.
- All endpoints in DSCTP must reserve some resources to tackle additional communication information all the time, such as environmental parameters report and delivery, policy request and policy delivery and so on. This extra cost will definitely reduce the efficiency of the transport network, especially when the network load is huge, it will make the network environment more terrible. However, resource reservation algorithm in DSCTP is different from the traditional resource reservation algorithm. It does not need all the resources for transferring data in network, just keep small part of the resources to handle control events. If this method can greatly reduce the probability of congestion is also worth it because congestion in the transport network will lead to low-level efficiency.

VI. SIMULATION TEST

In order to verify the dynamic algorithm for the congestion control is more efficient than the traditional congestion control algorithm, not only can greatly reduce congestion occurred, but also can improve the efficiency of the network transmission. The system testing must be performed using VS2008 simulation for two kinds of algorithms of congestion control by comparing two mechanisms of sending the same quantity of packets, from congestion numbers (CN) and experimental time (ET). EV is used to simulate values of network conditions.

A Experimental condition:

To simplify the protocol we should make some hypothesis:

- Only concerned with receiving buffer occupancy, hops from source to destination and round-trip time three environment variables, and ignored other environment variables.
- The service of sending data in the network belongs to the same type.
- Using one-to-many communication mode, which is a source point correspond to multiple destination points. The sending endpoint send data to any one of the receiving endpoint and get the confirmation are all represented the success of sending data.
- In the real network, it often occurs that too large data traffic can cause congestion in any subnet. Adding a variable EV in this experiment to simulate. EV is a value from 0 to 1, when EV is close to 1, the network load is in a high situation, but when EV is close to 0, the load of network is in a low situation. In short, we can see EV as the ratio of all of data in the network and carrying capacity of network, so it can reflect the current network load conditions.

Because there are none of perfect network protocols based on dynamic congestion control algorithm, considering

of fairness, not to choose the existing SCTP as a experimental subject and construct two experimental objects based on TCP protocol.

Object 1: SCTP simulation:

- One source endpoint can establish TCP connections with 10 destination endpoints;
- Each connection has its own parameters of congestion control;
- From initial status, the source endpoint only chooses one destination endpoint and sends data to it;
- The source endpoint always remains sending data to the same destination endpoint;
- The sending endpoint uses a slow start and congestion avoidance to regulate the congestion window;
- Using SACK to confirm;
- Once the source endpoint has congestion with the current destination endpoint. The sender has to choose another destination endpoint to send packet, and send congestion window at a fixed value.

Object 2: DSCTP simulation:

- One source endpoint can establish connections with 10 destination endpoints;
- Each connection has its own parameters of congestion control;
- The sending endpoint can sends data to any host at any time;
- The sending endpoint can process data and environmental information at the same time;
- The sending endpoint based on environmental information to predict the status of the network;
- According to the predicted value the sending endpoint choose to continue to send data to the current host, or choose to send data to other hosts;
- If you need a switch, the sending endpoint retransmit all data in the send window to a new destination endpoint, and notify the original host to clear receiver's buffers and use of the size of the original send window to set a new send window.

In fact, the essence of simulation is to contrast two experimental subjects. CN and ET correspond with the changes of EV.

B Test Results:

Assume that transmit 100000 packets, each size of packet is 1500 byte (the maximum size of Ethernet), the sum size is 150 MB. Using of two kinds of congestion control mechanisms. From the below we can see that the value of CN and ET (s). As shown in the below, we can divide EV into three parts: first EV is from 0 to 40, and then EV is from 40 to 90, the last EV is from 90 to 100. The value of EV is from 0 to 100, when the value of EV is equal to 0, network is in an idle status; when the value of EV is equal to 100, network is in a fully congestion status; we only test the value of EV from 10 to 95 because the value of EV is lower than 10, the network basically has no data traffic. But if the value of EV is higher than 95, the network is in a state of paralysis. The value of ET is the time interval from the first packet transmitted to the last packet, which expresses the efficiency of the transport network. The smaller the ET, the higher transmit efficiency is. The value of CN is the times of congestion which expresses the ability of controlling network congestion. The smaller the CN, the higher congestion capability is. SCCA expresses static congestion control algorithm and DCCA expresses dynamic congestion control algorithm.

Table I and Table II list the required EV of 10, 20, 30, 40, 50, 60, 70, 80, 90, 93, 95 respectively, under SCCA and DCCA. What Table I contains is CN. From Table I we can see that CN of SCCA is from 0.2 to 24.2, while CN of DCCA is from 0 to 4.2. Based on Table I, draw a two-dimensional graph of Fig. 3. What Table II contains is ET. From Table II we can see that ET of SCCA is from 41.1 to 2505.4, while CN of DCCA is from 50.7 to 2641.7. Based on Table II, draw a two-dimensional graph of Fig. 4.

TABLE I. COMPARE CN OF DCCA WITH SCCA

EV	10	20	30	40	50	60	70	80	90	93	95
SCCA	0.2	0.5	1.1	1.8	2.8	3.9	5.7	8.5	14.2	18.6	24.2
DCCA	0	0.1	0.2	0.3	0.4	0.5	0.8	1.2	2.2	3.7	4.2

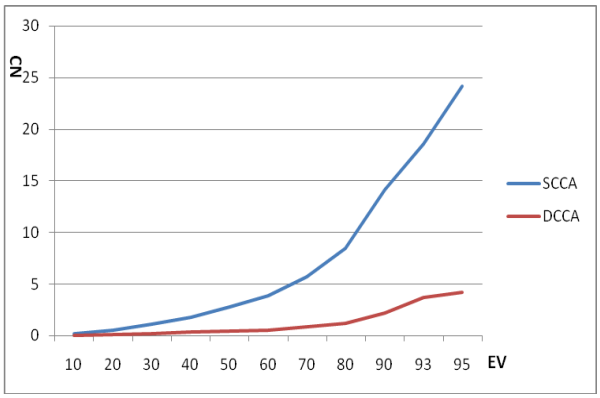


Figure 3. Compare CN of DCCA with SCCA

From Table I and Fig. 3 we can see that the value of CN in the SCCA and the value of CN in the DCCA are nearly equal to 0 when the value of EV is lower than 20. When the value of EV is higher than 95, the increase of CN will increase drastically eventually converge towards endless. From this we can see that DCCA is always better than SCCA.

Table I and Table II list the required EV of 10, 20, 30, 40, 50, 60, 70, 80, 90, 93, 95 respectively, under SCCA and DCCA. What the table contains is CN. From Table I we can see that CN of SCCA is from 0.2 to 24.2, while CN of DCCA is from 0 to 4.2. Based on Table I, draw a two-dimensional graph of Fig. 3.

TABLE II. COMPARE ET OF DCCA WITH SCCA

EV	10	20	30	40	50	60	70	80	90	93	95
SCCA	41.1	72.6	117.8	193.5	286.1	404.1	569.8	834.2	1457.8	1590.4	2505.4
DCCA	50.7	86.8	137.2	187.7	254.2	301.3	475.9	723	1384.9	1865.2	2641.7

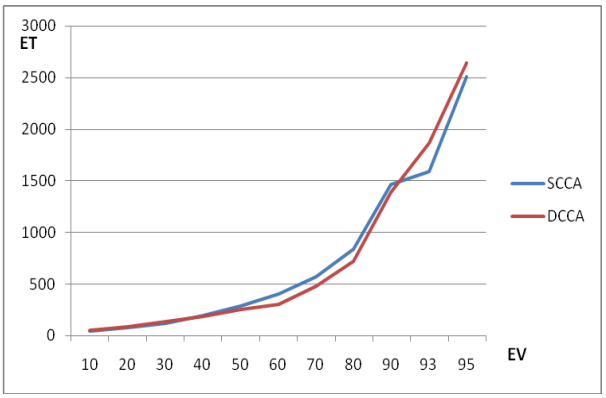


Figure 4. Compare ET of DCCA with SCCA

From Table II and Fig. 4 we can see that the values of ET in the DCCA and in the SCCA have two cross points, one is EV=40 and the other is EV less than 90. This is because DCCA use the dynamic congestion control mechanisms to reduce the probability of congestion occurs and improve the overall efficiency of transport network.

At the point of EV is equal to 40, two curves of ET gradually begin to coincide. This is because in a network with high load situation, no way can be taken to avoid network congestion occurs. The network was in a state of paralysis.

We can see that ET of DCCA is lower than ET of DCCA when the value of EV from 40 to 90. Because DCCA uses the dynamic congestion control mechanisms to reduce congestion occurs and also improves the network transmission.

When the value of EV is higher than 90, the increase of ET will increase drastically eventually converge towards endless. Because there is no way to avoid network congestion occurs when they are in a heavy-load conditions. From this we can see that DCCA is always better than SCCA.

C Result Analyzing

From the above results can draw the following conclusions, according to Table I, Table II, Fig. 3 and Fig. 4:

In the good network conditions, transport efficiency of dynamic congestion control policy is a little below the traditional congestion control policy. Considering the network was in an unoccupied state, so that the performance of internet would not cause too much influence. Because there is hardly any task in the transmit link in the light-load conditions, there is no significant difference between DCCA and SCCA.

In the normal-load conditions, using dynamic congestion control algorithm could reduce the probability of congestion occurs and improve the network transmission. In the heavy-load conditions, using dynamic congestion control algorithm and traditional congestion control algorithm could not change the network congestion situation.

Thus, the dynamic congestion control algorithm does help to reduce the network congestion occurs, and also help to enhance overall efficiency of the transport network.

VII. SUMMARY AND CONCLUSION

The main idea of this paper is applying the dynamic policy management framework to SCTP. Congestion control is a method used for monitoring the process of regulating the total amount of data entering the network so as to keep traffic levels at an acceptable value. The traditional SCTP initial start is so slow and SCTP use only one destination address at any given time to transmit new data. Therefore, the new algorithm design uses dynamic congestion control algorithm which describes the congestion control mechanism in detail in respect of transmission start, flow control, network monitoring, generation of policies, routing switching, and congestion recovery. Detection of network conditions is a core operation of dynamic adaptive network. The actions such as dynamically determining the value of the threshold and dynamically transferring tasks are practically based on the principle of prediction and feedback. However, merely using prediction and feedback is not enough to complete dynamic scheduling. What is more important is to integrate the dynamic policy management framework. Developed on the basis of conventional static policy management framework, this dynamic framework has the characteristics of self-government, which allows the whole algorithm to keep working without the involvement of the administrator. In this way, the policy scheduling can guarantee that it will keep up with the rhythm of change on the system, thereby, significantly enhancing the transferring efficiency.

ACKNOWLEDGMENT

The project of the research on hierarchy and protocol of dynamic control and management for packet service oriented Heterogeneous / Converged Network is supported by Ministry of Technology and Science of People's Republic of China under Grant No.2007AA01Z204 and Sino-Swedish collaboration research program under Grant No. 2008DFA11950.

REFERENCES

[1] Postel J., "User Datagram Protocol". RFC 768, IETF, Aug. 1980.

[2] http://www.rfc-editor.org/rfc/rfc3060.txt. [accessed: January 20, 2011]

[3] http://tools.ietf.org/html/draft-ietf-policy-arch-00. [accessed: January 20, 2011]

[4] http://www.rfc-editor.org/rfc/rfc3318.txt. [accessed: January 20, 2011]

[5] Dulay N., Lupu E., Sloman M., and Damianou N., "A Policy Deployment Model for the Ponder Language", An extended version

of paper in Proc. IEEE/IFIP International Symposium on Integrated Network Management(IM' 2001), Seattle, May 2001, IEEE Press.

[6] Liu X. J., Liu Y. H., Wei D., and Liu H. Y., "Dynamic Policy Based Network Management Scheme in Mobile Environment", 2008 International Symposium on Computer Science and Computational Technology.

[7] Iyengar J. R., Amer P. D. and Stewart R., "Concurrent multipath transfer using SCTP multihoming over independent end-to-end paths", Proc. IEEE/ACM Transactions on Networking, Vol. 14(5), pp. 951-964, Oct 2006.

[8] Fracchia R., Casetti C., Chiasserini C. F. and Meo M., "WiSE: Best-Path Selection in Wireless Multihoming Environments" Proc. IEEE Transactions on Mobile Computing, Vol. 6(10), pp. 1130-1141, Oct. 2007.

[9] http://pel.cis.udel.edu/. [accessed: January 20, 2011]

[10] Postel J., "Transmission Control Protocol". RFC 793, IETF, Sept. 1981.

[11] http://www.rfc-editor.org/rfc/rfc4960.txt. [accessed: January 20, 2011]

[12] Allman M., Paxson V., and Stevens W., "TCP Congestion Control". RFC 2581, IETF, Apr. 1999.

[13] Humaira K., Brad P. and Alan W., "SCTP versus TCP for MPI", Proc. Thirteenth International Symposium on Temporal Representation and Reasoning, TIME 2006.

[14] Stewart R., Xie Q., Morneault K., Sharp C., Schwarzbauer H., Taylor T., Rytina I., Kalla M., Zhang L., and Paxson V., "Stream Control Transmission Protocol," RFC 2960, Oct. 2000.

[15] Stewart R., Arias-Rodriguez I., Poon K., Caro A., and Tuexen M.,"Stream Control Transmission Protocol specification errata and issues,"draft-ietf-tsvwg-sctpimpguide-16.txt, Apr. 2006.

[16] http://en.wikiversity.org/wiki/What_is_Congestion_control%3F. [accessed: January 20, 2011]

[17] Jacobson V.,"Congestion Avoidance and Control". Proc. Computer Communication Review, Vol. 25(1), pp. 157-173, Jan. 1995.

[18] Eager D. L., Lazowska E. D. , and Zahorjan J., "A Comparison of Receiver-Initiated and Sender-Initiated Adaptive Load Sharing", Performance Evaluation,Elsevier, Amsterdam, Holland, Vol. 6. pp. 53-68,1986.

Bioinformatics: From Disparate Web Services to Semantics and Interoperability

Mikael Åsberg, Lena Strömbäck
Department of Computer and Information Science
Linköpings Universitet
Linköping, Sweden
Email: m.asberg.watch@gmail.com, lena.stromback@liu.se

***Abstract*—In the field of bioinformatics, there exists a large number of web service providers and many competing standards regarding how data should be represented and interfaced. However, these web services are often hard to use for a non-programmer and it can be especially hard to understand how different services can be used together to create scientific workflows. In this paper we have performed a literature study to identify problems involved in developing interoperable web services for the bioinformatics community and steps taken by other projects to address them. We have also conducted a case study by developing our own bioinformatic web service to further investigate these problems. Based on our case study we have identified a number of design issues important to consider when designing web services. The paper is concluded by discussing current approaches aimed at making web services easier to use and by presenting our own proposal of an easy-to-use solution for integrating information from web services.**

***Keywords*-bioinformatics; XML; web services; interoperability; semantics**

I. Introduction

In our previous work [1] we studied the interoperability of web services within the bioinformatics community. This article extends the work by providing a more comprehensive literature review, more details on our work, and a first description of our current work in the field.

In the field of bioinformatics, there has been an explosion of data over the past years. For example, in the Molecular Biology Database Collection: 2008 update [2], 1078 databases are listed, 110 more than in the previous update [3], which itself also contained 110 more databases than the update before that. These databases are being maintained and hosted by a large number of autonomous service providers. Many of them are only concerned with a single database and its tools. Regarding how data should be represented and formatted and how its corresponding tools and algorithms should be interfaced, there exists a large number of standards [4] [5]. As an example, P. Lord et al. say in [6] that ”there are at least 20 different formats for representing DNA sequences, most of which have no formal specification”. Many standards and specifications evolved in an ad-hoc fashion and there is no wide-spread agreement on when a particular standard should be used.

Remotely accessing the resources maintained by the service providers can often be done in more than one way. Many service providers have constructed www-based interfaces to their resources, meant to be used by humans. The user does not have to use any specialized software or use a specific platform to access the resource, a common web browser is sufficient. As an alternative to browser-based interfaces, many service providers in the bioinformatics community offer programmatic access by using web services [7]. A web service is any service available over the Internet using XML as its messaging system, and it is independent of platform and programming language. The web services we specifically mean here are those falling under the category of RPC (remote procedure call) web services, being parts of Internet API:s. Such web services allow for programmatic and batch access. An example web service method from the bioinformatics community is *get_genes_by_enzyme*, which is part of the web service API that allows access to KEGG (Kyoto Encyclopedia of Genes and Genomes) [8]. This particular method takes an organism and an enzyme ID in string form and returns its corresponding genes. Another example is EBI Soaplab Web Services for EMBOSS programs [9], which is a large set of web services providing web access to many EMBOSS programs.

Given the nature of biology, a bioinformatician often has to use multiple service providers to conduct his or her research. By combining several resources and processes, local or remote, bioinformaticians can create scientific workflows [10]. There exists a number of different tools for creating scientific workflows, for example Taverna [11] [12] [13] and VisTrails [14] [15]. These tools allow the user to create a data pipeline, a workflow, using a number of different resources. Taverna caters specifically to bioinformaticians while VisTrails was originally targeted at visualization. An alternative to flexible workflow systems like Taverna and VisTrails are specialized bioinformatic grids [16] that tie remote and local resources together in a client to provide a unified view. Such grids can often be successful on a small scale, but development costs and network restrictions prevented them from becoming the de-facto standard for integration of bioinformatic services [16].

However, the task of creating bioinformatics workflows can be very difficult because of the fact that there are so many service providers and there is little consensus about data formatting and interfacing. Thus, bioinformaticians face a massive interoperability problem. A browser-based form is a convenient way to work if one just wants to do a few

stand-alone look-ups in a given database, but when one needs to perform many look-ups and queries on several disparate resources it becomes unfeasible. This is true not just because of the fact that these web-based interfaces may be poor at supporting programmatic and batch access, but also of the intricate data reformatting that may be involved in using the output from one resource as input to another. Web services, by themselves, do not solve the interoperability problem. They are simply a way to access remote resources programmatically. A problem facing users is to determine how different sets of web services can be used together to create the desired workflows.

In this paper we have performed a literature study to give an overview of other projects that have been introduced to enhance semantic interoperability and discoverability for bioinformatic web services. Section II discusses the problem with designing interoperable web services and Section III discusses semantic frameworks that have been introduced in bioinformatics community to alleviate the problem of lacking interoperability. In Section IV we discuss scientific workflow systems that are becoming a more and more popular way of interacting with multiple bioinformatic resource providers to form workflows. This way of working has many benefits, but introduces new problems that need to be considered. Section V discusses our case study, where we developed our own web service to get a hands-on experience of the problems involved with web service design. The case study was also aimed at designing a web service capable of performing data integration on the fly. Section VI summarizes the problems and design issues we encountered. In Section VII we end the paper by discussing other approaches to interoperability and also present our continued work, BioSpider.

II. Designing interoperable web services

Our main issue is how to design web services for the bioinformatics community that are interoperable with other services. We focus on technologies used in this community.

SOAP-based web services are sometimes called XML-based web services. Everything that is passed between the users of a web service and the web service itself is in XML form. This means that input parameters and output data for a given web service method are also serialized in XML. In order for clients to use a particular web service, they need a description, the *WSDL*, to learn about, which methods are available, what parameters those methods require and what kind of data is returned.

As an example, let us create a web service of the Java method (showing its signature only) *String getXItem(String id)*. It accepts a single string and returns a single string value. In Figure 1 we see what the created data types for our example method looks like in the WSDL description.

The WSDL description shows that for our example method, two complex schema types have been declared, one for the input parameter and one for the result. However, these types are simply wrappers around the XML Schema primitive type *xs:string*. Herein lies the interoperability problem, because these types have no semantics attached to them. We do not know the semantic meaning of the input and output data, only their syntactic meaning!

```
<xs:complexType name="getXItem">
   <xs:sequence>
      <xs:element minOccurs="0"
                  name="arg0" type="xs:string"/>
   </xs:sequence>
</xs:complexType>
<xs:complexType name="getXItemResponse">
   <xs:sequence>
      <xs:element minOccurs="0"
                  name="return" type="xs:string"/>
   </xs:sequence>
</xs:complexType>
```

Fig. 1. Type Declarations in WSDL-file

When programming in a strongly-typed language like Java or C# and one needs to have semantics for syntactically simple types (like strings or integers), the common practice is to introduce new classes or hierarchies of classes. By doing that, we can enforce semantic rules at compile time and prevent, for example, scenarios where users are passing strings that represent journal abstracts to methods expecting database identifiers in string form. However, this will not work in a web service context, because, in order to achieve interoperability between different programming languages and platforms, the types used in the host language must have corresponding XML Schema types like integers or strings.

The WSDL description for a web service can be seen as the grammar of that particular web service. Thus, all providers have their own grammars, albeit specified using the same XML language constructs. In order to achieve true semantic interoperability we need a common grammar, a common stock of reference. Within the bioinformatics community, a number of different semantic frameworks have been proposed to provide a common grammar, for use as a common stock of reference regarding data types, naming, operations etcetera.

Another thing to consider is that not all parameters for a given web service method may hold the same merit. Some parameters constitute the fundamental data needed by the service. These are the important parameters. But many methods also accept additional parameters, whose purpose is simply to tweak the behavior of the service. When looking for semantically suitable web services, a bioinformatician is not interested in these kinds of parameters as they would only serve to complicate the task at this point [6].

A problem with the web services themselves can also be that a given method is semantically simple and has a limited and well-defined purpose. This programming paradigm is mostly considered a good thing, but if the building blocks of a workflow become too fine-grained it will be difficult to create more advanced workflows without requiring a lot of ”glue” in the form of data formatting or even programming. However, it may be the case that possible interoperability between two web services is easier to establish if their parameters are fundamentally simple and well-defined. This leads to the main point of this article: how to design web services with clear

semantics that can be easily used in bioinformatic workflows.

In the following section we will give a presentation of some of the more notable semantic frameworks that have been introduced to provide a single grammar to describe a host of autonomous service providers.

III. Semantic Frameworks in Bioinformatics

Several projects have attempted to address the above problems and here we give a brief overview of the three most important in the field of bioinformatics today: BioMoby [17], ^{my}Grid [18], and the more recent BioCatalogue [19] [20]. These projects tackle not only the problem of helping users to determine how services from different providers can work together to form workflows, but also assist in the discovery of relevant services.

A. BioMoby

The BioMoby project was initiated in 2001 and has a main branch, MOBY-Services (MOBY-S) and a sub-branch known as SSWAP (Simple Semantic Web Architecture and Protocol). However, according to the BioMoby web page [21] these two branches are to be merged in the future.

In the **MOBY Services** branch three ontologies provide semantic discoverability and interoperability along with strictly enforced naming rules, and we present them briefly below:

- **Service ontology** - this ontology contains operational classifications used to label web service methods. There is a root type called *Service* and a tree of sub-types, such as *alignment* or *rendering*. The purpose of this ontology is to assist in the discovery of web service methods that are useful or interesting for a given task or problem.
- **Namespace ontology** - this ontology can be seen as a flat list of different namespaces that can be used to semantically describe data that is consumed and/or produced by a web service method. It also enforces a method for how to name identifiers. A problem in the bioinformatics field is that identifiers have not been named in a consistent and reliable way. The namespace ontology derives information from Cross-Reference Abbreviations List from the Gene Ontology consortium [22] and defines distinct naming rules that are used dependably and reliably. For example, *Antirrhium majus* (Snapdragon) gene names live in the *DragonDB_gene* namespace.
- **Object ontology** - this ontology is similar to the service ontology in the sense that it can be viewed as a tree. The root node is called *Object* and between all nodes and its children there exists an *IS-A* relationship, e.g., an *AlignedSequence* is an *Object* and an *AlignedDNASequence* is an *AlignedSequence*. Two nodes, in different parts of the tree, can also have a *HAS-A* relationship (either one-to-one or one-to-many), e.g., the type *Annotation* is a direct child of *Object* and it has two *Integers*, but is itself not an *Integer*. An instance of an object that is returned from a web service method is serialized into XML and contained in an envelope (that is also XML). Envelopes can contain cross-references that are supplied by the data provider and they describe related pieces of information. Since cross-references are valid objects by themselves they may be used directly, without additional formatting or computation to discover services from other providers that operate on them [23]. In Figure 2 we can see a small fraction of this ontology.

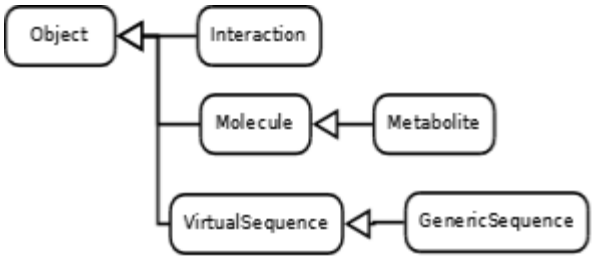


Fig. 2. Fraction of BioMoby object ontology

The BioMoby team runs Moby Central, which is a web service registry where all service definitions can be found, expressed in the ontologies described above. When a new actor wants to participate in the BioMoby project they register their objects, namespaces, and service classifications in the registry. This implies something that is very important and fundamental to BioMoby: the ontologies described above are *end-user extendable*.

SSWAP stands for Simple Semantic Web Architecture Protocol and was previously called Semantic-MOBY or simply S-MOBY. The approach taken by this sub-branch differs from the one taken by the MOBY-Services project. Instead of maintaining three user-extendable ontologies centrally, it defines a minimal messaging structure and relies on the wealth of the available third party ontologies, such as OBO [24], to define meaning, syntax and interoperability between services. In [17] the authors say that *"SSWAP has shown exciting early success in achieving interoperability between a small number of participating providers. It remains to be seen, however, if the complexity of reasoning over an open-world system, and/or the potential dilution of compatibility between resources due to an increasing number of ontological possibilities, will interfere with the desired goal of straight-forward, maximum interoperability between bioinformatics Web resources"*.

B. ^{my}Grid

The ^{my}**Grid** project is a part of the UK government's e-Science program and it seeks to enable bioinformaticians to discover and chain together disparate bioinformatics services to create workflows, *in silico* experiments. For describing a bioinformatics domain and the properties of its services there exists a domain ontology that is stored centrally and maintained and generated by an expert. Semantic descriptions of services are made using a lightweight RDF data model using terms from the domain ontology. These descriptions are extendible by users. The project makes a distinction between domain services and shim services. Domain services are the centerpieces, those that perform scientific functions. The archi-

tecture should not automatically select those for the scientist, because there could be alternatives that only the scientist doing the experiment should select among. Shim services, on the other hand, does not have a scientific function per se, but are only used as glue to connect two domain services that are not directly compatible. The architecture should automatically insert available shim services where they are needed without the scientist having to intervene. The Taverna software can interact with ^{my}Grid-based sources, and also with BioMoby-ones through a plugin [25].

C. BioCatalogue

The BioCatalogue project [19] [20] was launched in June 2009 and is a joint venture between EMBL-EBI and the ^{my}Grid project. It aims to assist web resource users within the Life Sciences community to find relevant services for their research and assist in determining how different services can operate together. Another goal is to act as a registry for suppliers of services, which will allow any given supplier to increase the size of its user base.

The project tries to tackle the problem of services becoming stale, disappearing, or changing by monitoring both their actual availabilty and their supposed function. To put it another way, the four main issues listed on their homepage (http://www.biocatalogue.org/) effectively point out the objectives of the project:

- *"Web Services are hard to find."*
- *"Web Services are poorly described"*
- *"My Web Services are not visible"*
- *"Web Services are volatile"*

By finding solutions to the problems listed above, the BioCatalogue project aims to become the central hub for web services in the Life Sciences community. It doesn't aim to become a supplier of scientific web services themselves, only to bring the vast variety of already existing ones under one umbrella to the benefit of the community.

The BioCatalogue project has noted that finding an interesting web service is only part of the problem. It's not enough just to have a central registry of services where suppliers can register their service. Often the documentation for a service is mediocre or outdated and comes with few or no examples, making the service hard to use. Some services also require certain operations to happen in sequence to be able to produce a meaningful result, and this fact is not always clear from the get-go.

To minimize this problem of understanding a given web service, the BioCatalogue project employs rich annotations for all registered services. The annotations for a given service are not derived from a single source (e.g., the supplier of the service). Instead it's comprised of information from several parties: the supplier of the service, a domain expert curator employed full-time who has sub-curators to assist him or her, the user base itself, and usage patterns that are automatically collected. Together, these entities evolve the annotations for a given service over time to make it better and more consistent. The role of the curator is to oversee the process to ensure that guidelines are followed in order to avoid annotations to be given in an inconsistent manner. Annotations for services can be divided into four main categories:

- Functional - these annotations describe the purpose of the service, what kind of operations it can perform and what data it will operate on and produce. This category is further divided into sub-categories, pin-pointing the task of the service more explicitly, e.g., alignment or text mining. Example input data and other usage scenarios often accompany these annotations in order to help users.
- Operational - these annotations detail any particular considerations that must be taken into account in order to successfully use the service. As noted above, some services require operations to happen in sequence (i.e., the individual methods of the service cannot be seen as separate islands) in order to produce a sensible result. Such things are annotated in this category.
- Profile - here automatically collected data and other comments by users regarding the service are maintained.
- Provenance - contains information about the supplier of the service and an audit trail detailing any changes to service over time are kept here.

BioCatalogue can be accessed through a web portal [20], but an API is also provided to allow programmatic access. A user using the web portal can look for services by searching for scientific function, data types, provider, country etcetera. If the user is unsure about the type of some data he or she has, BioCatalogue can analyze an excerpt of it to determine its type. Regarding data, input and output data are tagged with ontological terms from the ^{my}Grid project and if the users know that information it's straightforward to find services that fit perfectly semantically. All services from the ^{my}Grid project have been imported into BioCatalogue and work is currently underway to merge with other big repositories like BioMoby.

IV. Scientific Workflow Systems

The main aim of this paper is to study interoperability of web services, i.e., how web services can be designed to be easily used together to solve an information integration task. From a technical point of view there are several ways to combine web services into more complex tasks, however, one approach in common use within the bioinformatics community is scientific workflows. Scientific workflow and workflow-based systems [11] [12] [13] [14] [15] [26] [27] [28] have emerged as an alternative to ad-hoc approaches for documenting computational experiments and designing complex processes. They provide a simple programming model whereby a sequence of tasks (or modules) is composed by connecting the outputs of one task to the inputs of another. Workflows can thus be viewed as graphs, where nodes represent modules and edges capture the flow of data between the processes.

The actual features and representation of a scientific workflow differ between the systems, due to varied needs from application areas and users. There is ongoing work to create one common model for provenance, e.g., the Open Provenance Model [29] and a mediation approach [30]. However, currently

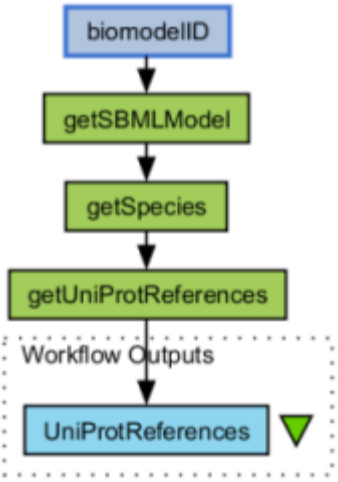


Fig. 3. Sample workflow illustrating iteration in Taverna

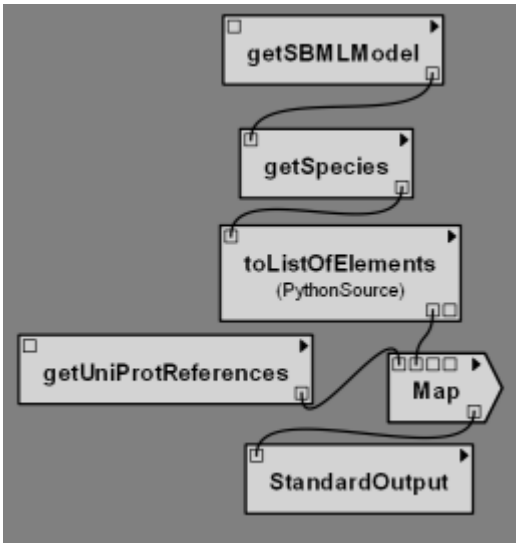


Fig. 4. Sample workflow illustrating iteration in VisTrails

systems tend to work in their own internal format albeit it is becoming common to provide conversion to other formats. In practice this means that a web service can be more or less easy to use within the scientific workflow framework dependent on its design and the available features for the chosen tool.

In this work we have chosen to work with two different workflow systems, VisTrails [14] [15] and Taverna [11] [12] [13]. VisTrails is a workflow system that supports exploratory computation tasks. It has a graphical user interface that is used for the composition and execution of workflows. Data and workflow provenance is uniformly captured to ensure reproducibility of results by others. Workflows can be composed by program libraries (Python) or by external web services. VisTrails has been used in the fields of biology and earth science. Taverna is designed specifically for bioinformatics applications. As VisTrails, Taverna has a graphical user interface for creating and executing workflows. Workflows are composed by making use of external services such as web services, BioMart, BioMoby and SoapLab services.

There are several differences between VisTrails and Taverna in terms of how they represent provenance and what functionality they offer. However, for this work the most important difference is how they represent iteration, which is a common task in bioinformatics. Taverna offers a straightforward solution. Figure 3 shows an example of a Taverna workflow. Whenever a module returns a list of results the next module is iteratively applied on all results in the list. In this case *getSpecies* returns a list of all species in the model and *getUniProtReferences* is applied on every species in the list. VisTrails does not offer this feature, instead they offer a number of control flow modules. This includes control flow modules such as conditions and a map module for iteration. By using the map module we can apply the next module on all results in a list. The resulting VisTrails workflow corresponding to the Taverna version is shown in Figure 4.

V. Case Study

To further explore the design issues encountered when designing bioinformatic web services, we have performed a case study where we implemented our own service. The main objectives of the service were that it should be easy to use, it should help with data integration, and the semantics should be possible to model in frameworks such as MOBY-Services. Regarding data integration, we wanted to investigate ways of following links between different data sources using a single service. Right now, our service links three databases together: the (curated) BioModels Database [31], the UniProt Knowledgebase (UniprotKB) [32], and the RCSB Protein Data Bank (PDB) [33]. The service is not intrinsically bound to just these three databases forever, but could be expanded to work on others.

A. Design of the web services

The web service is written in Java [34] using the Eclipse Web Tools Platform [35], but it is platform neutral in the sense that it can be used from any WSDL-enabled language and platform without any additional dependencies. Using the SBML library [36], the service loads SBML models from the BioModels database, and the loading of a model can be seen as an entry point to the web service. From this model the user can extract UniProt references that can be found in the annotations for some species in the BioModels data. The UniProt references are used, by the server, to obtain the corresponding UniProtKB XML-files and, from those PDB-references can sometimes be extracted. A PDB file is returned to the caller who can use it for visualization. The service can be seen as having several stateful subsets: when you ask questions about BioModels or its species you need a BioModel or species ID, but when you obtain a UniProt reference then that becomes the key you use and likewise the PDB reference

```
getSBMLModel(biomodelFileName) : BioModel ID
getSpecies(modelID) : Species IDs
getNthSpecies(modelID, n) : Species ID
getNumberOfSpecies(modelID) : Number of Species
getSpeciesSBMLID(speciesID) : Species SBML ID
getSpeciesSBMLName(speciesID) : Species SBML name
getUniProtReferences(speciesID) : UniProt references
getPDBReferences(uniprotID) : PDB references
getPDB(pdbReference) : PDB file
```

Fig. 5. Web Service Method Listing

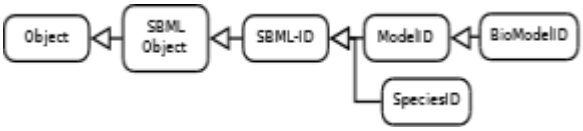


Fig. 6. Type ontology for our web service

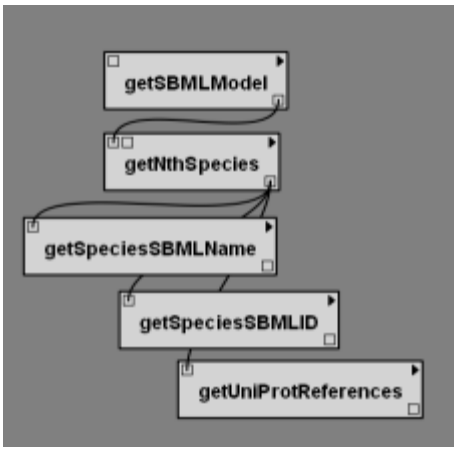


Fig. 7. Fetching name, id, and references for a given species

becomes the identifying token that is used for the PDB part of the service. In figure 5 we see a listing of the core methods.

Figure 3, from the previus section, is an example of a Taverna workflow utilizing the web service. It's a straightforward workflow that loads a BioModel. From the BioModel a list of ID:s to its species is obtained and for each species any UniProt references are determined. The method *getUniProtReferences* that is called only takes a single species ID so Taverna automatically iterates over our list of species ID:s and calls the method once for each. The resulting output is a list of lists that shows how species are annotated with UniProt references. In Figure 4, the previous section also showed the same workflow modeled using the VisTrails system. The VisTrails variant of the workflow is a bit more complicated than its Taverna counterpart. This is due to the fact that iteration is not automatic in VisTrails, but modeled in the workflow itself, using special control flow modules like map.

BioModel-files and UniProtKB-files are never passed verbatim between the client and the server, instead ID:s are used to identify a particular file. The only large data object that is passed is the output of the *getPDB*-method that represents the PDB itself. The server stores the latest release of the curated biomodels in a database and it will fetch UniProtKB-files and PDB-files when they are needed from their respective resource providers. Any downloaded files are cached for performance reasons.

All input and output parameters to the methods that make up the web service are simple types like strings or integers. Since all types used can be modeled using XML Schema primitives we are not tied to a particular platform. It also means that our interface is fit for modeling in MOBY-Services. A question that arises when modeling an interface in MOBY-Services is if one should use namespaces or types for new items. We have chosen to define new types. Figure 6 shows how the types can be realized for our service. For UniProt and PDB references there are already existing namespaces defined that can be used. See also the descriptions of input and output parameters in Figure 5.

Determining the proper level of granularity of your interfaces is hard. If it's too fine-grained, you could end up with an explosion in the number of methods that might require glue to perform complex tasks. If you design a service where the methods are complex and operate on complex data structures the service will tend to be tied to a very limited set of tasks and it might require the user to perform data-reformatting tasks, especially if he or she wants to use other services as part of his or her workflow. In our implementation the goal was to design a service that ties together several different databases by following links in the data while at the same time having a simple interface to allow for interoperability and to severely limit the need for the user to reformat data. If the primary goal of the service was to obtain PDB data then a much more compact interface could have been made, where the methods *getSpecies*, *getUniProtReferences*, and *getPDBReferences* could have been replaced with a method that given a BioModel ID returns a list of PDB:s. Such a method would be very easy to use for that particular task, but it could not be used in some other context.

B. Using the web services

The design of the web services is important for its ease of use and the possibility to combine the services into extended functionality. In this section we will exemplify and discuss two of the major design choices that we considered in our case study. These were to design a service that ties together several different databases by following links in the data and providing a simple interface to allow for interoperability and to severely limit the need for the user to reformat data. This is demonstrated by the two examles in Figure 4 and 7.

Here we use the services to find references to information in the UniProt database, but by combining them in another way the user can select other information about each species.

If the primary goal of the service was to obtain PDB data then a much more compact interface could have been made, where the methods *getSpecies*, *getUniProtReferences*, and *getPDBReferences* could have been replaced with a method that given a BioModel ID returns a list of PDB:s. Such a

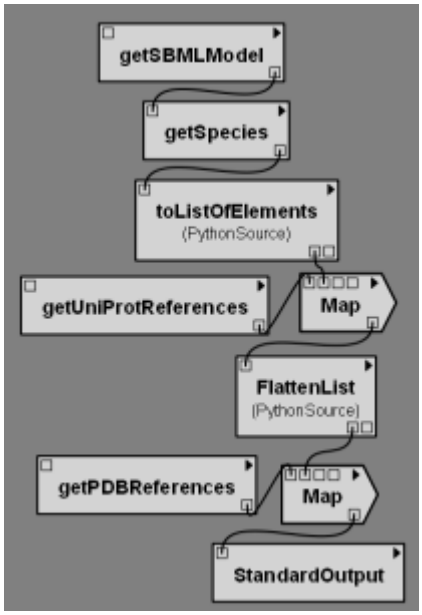


Fig. 8. Find PDB references in VisTrails

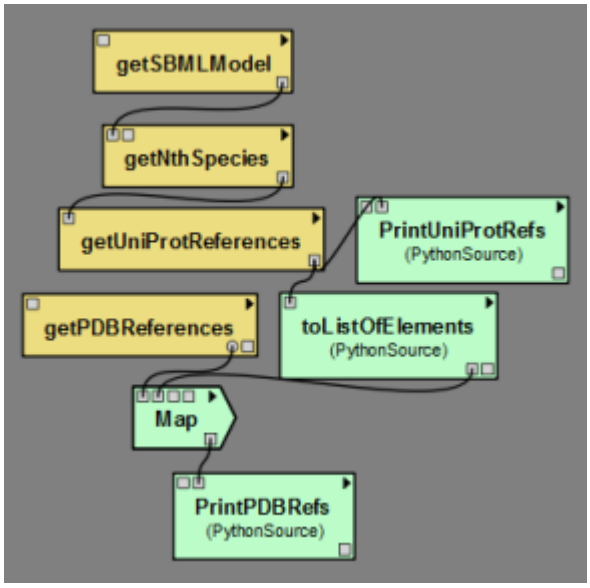


Fig. 9. Find PDB references using parameter exploration and iteration

method would be very easy to use for that particular task, but it would not have allowed the variation demonstrated by the previous example.

As shown by pervious examples, in bioinformatics, it is common to have lists of data as the result of some operation that one might want to pass to another operation. Even though both Taverna and VisTrails have facilities for creating iterative control flows it still remains a complex task. As an example, we can study the workflow where the user loads a BioModel, obtains an ID for each of its species and for each species determines any UniProtreferences. So far, this is the same example as before, but what if we want to follow the UniProt links to find any PDB references? The more straightforward solution is shown in Figure 8.

Here we reformat the list of UniProt references and find a set of PDBreferences for each of them. The drawback with this solution is that the connection between each species and the resulting PDBreference is lost. Maintining them requires building complex datastructures during the iteration. In Figure 9 we show an alternative solution in VisTrails. For iteration, the workflow utilizes a combination of VisTrails features for parameter exploration and control flow modules like maps.

This implementation prints the desired results of the workflow. In Figure 10, we see the output of the workflow from VisTrails when we are using BioModel BIOMD0000000003. Three short Python scripts are used for printing results and for data type conversion. A main objective when constructing this workflow was to preserve information regarding links, i.e., we want to know exactly how species are annotated with UniProt references and how UniProt references links with PDB references. This goal has been fulfilled. A careful study of the output reveals that only the species with index one has UniProt references and gives *findPDBReferences* something to work on. Since map returns a list of lists we can tell how UniProt references are connected to PDB references. In this case they both point to the same PDB file.

VI. Design issues

In this section we summarize our experiments identifying the important design issues to consider when designing web services for bioinformaticians. The goal is to allow bioinformaticians to create complex workflows using multiple service providers that are easily located and those workflows should require a minimum amount of glue in the form locally-performed programming. We have divided these issues into the following categories (in no particular order): semantics, chainability, granularity, data representation, and data passing.

A. Semantics

When creating workflows, bioinformaticians need to be able to automatically discover services that perform some scientific function or find services that either produce or consume data in some given format. The key to enabling this automatic discovery is semantics and here is where frameworks such as MOBY-Services or ^{my}Grid come into the picture. Registering your service in frameworks such as MOBY-Services or ^{my}Grid will allow users to automatically find it when it fits at the semantic level and not just at the syntactic level.

```
Species 0 UniProt refs: []
Species 0 PDB refs: []
Species 1 UniProt refs: ['P24033', 'P35567']
Species 1 PDB refs: [['1P2A'], ['1P2A']]
Species 2 UniProt refs: []
Species 2 PDB refs: []
```

Fig. 10. Find PDB references workflow in VisTrails - output

B. Chainability

Chainability is closely related to semantics, because we need semantics to be able to attain it. Chainability means the ability to automatically list consumers for a given type of data, consumers that fit semantically. Sometimes a service cannot work on your data directly but may be able to work on some subset of the data or requires re-formatting of the data. This requires so called shim services that perform extraction or formatting on some existing data to make it usable by another service, but have no scientific domain function by themselves. Shim services are important to discover automatically. One step towards realizing semantically correct chainability has been made through frameworks such as BioMoby and ^{my}Grid where the semantics of the function of a web service and of its input and output data is specified. However, the framework must also be clever enough to suggest shim services automatically. Say a user has some piece of data in a given format and wants to perform some scientific function on it. The data might need to be reformatted before the methods performing a scientific function can operate on it. The reformatting is done by shim services, and those should be discovered automatically by the framework even though they themselves do not perform the scientific function the user was searching for per se. In [37], D. Koop et al. presents a method of suggesting suitable services by using predictions.

C. Granularity

Should we make stateless or stateful web services? How fine-grained should they be? Very fine-grained methods could be seen as following the programming paradigm of divide-and-conquer. A large problem is broken down to a large number of very small, restricted, and well-defined mini-problems. In a procedural context one could then image writing a procedure (a function, a method) for each mini-problem and this would allow for modularity, ease of testing, re-usability, and ease of documentation. It might not always be the best way when catering to non-programmers because it becomes a complex task of putting all the pieces together in a workflow if there are lots of them. However, if the building blocks in the toolbox are all fundamentally simple, this might lead to a lot of glue in workflows in the form of shim services or programming. On the other hand, if the function of a service is fundamentally simple, it will be easier to describe its semantics. This allows for interoperability. A complex web service where too much functionality has been shoe-horned into a few methods will be very hard to use outside the particular purpose it was designed for. This kind of web service would be hard to use as a tool in a toolbox.

Another issue to take into account is whether the service should operate on lists or single items. Operating on lists can be tempting for performance reasons due to overhead associated with web service calls. However, the output from such methods will become more complex and it may be difficult or impossible to tell how results are connected to input data. The capabilities of workflow tools regarding support of iterative control flows would influence the design of the web service.

D. Data representation

One very important issue when designing a web service operating on complex data is the data format, such as the SBML model. The choice of representation is important when passing complex objects as arguments between web services, but also to enable an understanding of the semantics of the services. In our case we have chosen to use available XML standards for bioinformatics [4] [5], such as SBML and UniPROT. This is a benefit, as it makes the functionality of the service transparent to anyone familiar with the standard, e.g., in our case the naming and the functionality of the SBML services have a direct relationship to the entities defined by SBML.

Another aspect is that data representation for web services is closely related to data formats available for export on the web. Therefore it is natural to reuse the work already invested in this area instead of inventing new representations. Our case study shows that using available data formats works well. In addition we avoid unnecessary conversions of data by using formats where data is already available.

E. Data passing

When using web services we want to avoid passing large amounts of data back and forth when we do not need to. If the data is not generated by the client and is available in the public domain and it is the responsibility of the web service to manipulate or study this data in order to compute some result that is to be returned to the caller, then that data should stay on the web service server as much as possible. The clients should only see the results they are interested in. In our example implementation we found that using IDs worked well as links to pieces of data.

VII. Other Approaches

In the previous sections we have discussed interoperability issues regarding bioinformatic web services, we followed with a look at some semantic frameworks that have been introduced to tackle those issues. We also presented a detailed case study where we identified and discussed design issues related to the construction of web services.

In this section we will present three other approaches that aim to make web services easier to use for bioinformaticians. First we present TogoWS [38] [39], which is a web service and data-integration proxy for a number of service providers. We follow that with a presentation of SeaHawk [40], that serves as a front-end to the BioMoby framework presented earlier. The section is concluded with a presentation of our future work, the BioSpider, which in short can be described as plugin-based framework for modelling disparate, but yet connected bioinformatic web resources.

A. TogoWS

The TogoWS project [38] [39] sprung out from ideas seeded and problems recognised at the BioHackathons in 2008 and 2009. In these BioHackathons it was concluded that interoperability was a major problem in bioinformatics, not only because of data representation but also because technical decisions made by some certain service providers, forcing clients to use a particular programming language or environment. It was also noted that there are projects aimed at tackling this problem, such as BioMoby, but that many providers have not made their services compatible with these frameworks due to the hefty investment in server-side work required, making these proposed solutions not completed.

Due to the amount of data reformatting required to be able to use data from one service provider with another provider, a large number of third-party, client side libraries have been developed. There are libraries for different programming languages aimed at different tasks and these include the set of libraries (BioPerl, BioPython, BioRuby, BioJava) provided by the Open Bioinformatics Foundation [41]. However, the team behind TogoWS decided that their service should itself provide the data reformatting features offered by these libraries. This would relieve the user of the burden of having to install these libraries and write code to interface with them.

The team behind the TogoWS project decided to build a web service frontend for different suppliers, acting as a proxy between them and the users. Several major service providers were included under this umbrella. Operations were divided into two main categories: data retrieval and analysis. It was decided that using a REST-based API was best for data retrieval because then a uniform URI-scheme for the participating databases could be devised. For analysis operations, it was decided that SOAP was the best option because here results returned and parameters can be very complex and running time can be substantial, making REST a less than ideal choice. In the first phase, a unified SOAP-based interface was developed for several service providers located in Japan, and certain technical limitations found in some services were worked-around.

By developing a front-end that is itself a web service too, the clients can continue to use whichever tools they like to call these services and instead of waiting for service providers to agree on interoperability issues, the TogoWS project makes it happen for them.

B. SeaHawk

SeaHawk [40] is a front-end for BioMoby (more specifically, MOBY Services), written in Java, and developed at the University of Calgary (Canada). The people behind the project reviewed numerous other front-ends available for MOBY-S, categorised them, and evaluated what their respective strengths and weakness were, in order to build, as they see it, a better client. A major problem they identified with other front-ends is how they deal with actual data. MOBY-S employs several ontologies to deal with service specification, naming, data type hierarchies and relationships etcetera and all communication payload is wrapped in XML structures called Moby Envelopes. Many of the other front-ends either required the user to have extensive knowledge of the layout of those ontologies or were very limited in their expressiveness (in order to reduce complexity).

These findings inspired the idea of developing a front-end that was data-centric, the bioinformatician using the tool should focus on his or her actual data and not worry about implementation details. Since ontological terms do have to be specified in order to operate within MOBY-S, the SeaHawk client generates the required Moby Envelopes under the hood for the user. Data can be anything from service output, formatted HTML, rich-text files, text files in certain biologicial formats, or subsets and through its interface SeaHawk offers many ways to access the data.

While making it easier for novice programmers to work with the tool and focus on the data, the focus is also on using the system in a workflow manner and SeaHawk was recently enhanced to be able to generate Taverna workflows from its operations [42].

C. BioSpider

Our ongoing work to address the above problems is a tool that goes under the name of BioSpider. It's inspired by one of the core research ideas that we investigated in this paper: having a web service perform on-the-fly data integration on a few databases we knew had references to each other in one way or another.

In BioSpider, we do not focus on a particular technology like web services, but the emphasis is on the data itself. As we have discussed in this article, there are several autonomous providers of data (along with tools corresponding to that data) in the bioinformatics community. Even though there is an abundant number of data and service providers in the bioinformatics community, the actual data in databases themselves are not self-contained but full of references to other databases. These references represent important information for bioinformaticians performing their research, but the sheer intricacy and massiveness makes it very diffult for them to get a bird's eye view of how different data sets and entries are connected.

This is where BioSpider comes in. BioSpider is at its core a framework, a model for creating a single graph, representing data from disparate sources that have references to each other in some way. This graph is in effect the result of data integration operations performed between two connected sets of data.

The framework also comes with a rule-set for displaying this graph in a graphical user interface. It's through this user interface the user sees a unified view of the data, and he or she can follow links in the data to more sources, or apply actions on different data items (nodes) in the graph. These actions include visiting web pages, invoking web services or even running third-party tools. Figure 11 shows an example of the what the graph can look like for the user.

Fig. 11. Graph of connected data sources as visualized by BioSpider

In technical terms, BioSpider is written in Java. It consists of a set of classes that make up the core framework where connections between data items and actions that are available are specified. We have separated the actual framework from the data to allow the rule-set used by the framework to be extendible by users. One extension is the introduction of support for a completely new database, i.e., completely new functionality, or it can be an extension of the functionality of an already known data source. Another possible extension is additional actions that can be performed on data items or new references to other sources that can be exploited.

Our goal with the framework is to avoid hard-coded connections to any given data set and just provide a set of rules for describing data sets, actions that can be performed on items in the data and connections to other sources. The framework should be expressive to be able to capture all kinds of different data and actions but still be easy to extend.

We forsee two kinds of users. The first user is the normal user, using the framework as an ordinary desktop application to explore data in a unified way and perform actions on the data. The second user is a power user who will extend the rule set, making the tool even more powerful. These extensions should be easy to feed back to the community, or to the authors of the framework. Recently, we evaulated the extensibility of the framework and the results were encouraging.

In the future, we envision being able to in one way or another incorporate or utilize work done by other projects, like the BioCatalogue project, to further extend and improve our framework for the benefit of the user.

VIII. Conclusion

In this article we performed a literature and case study to examine the situation for bioinformaticians with a need for creating complex workflows using multiple service providers. They face severe interoperability problems because it can be very difficult to discover appropriate services and determine how they can be used in conjunction. When designing web services for the bioinformatics community we have identified several issues that need to be addressed to achieve a high discoverability and interoperability. The individual web service methods should be fundamentally simple so the semantics is easy to describe and the method should operate on data that is semantically well defined. This will allow the service to be registered in frameworks such as BioMoby and ^{my}Grid. Registration of services will assist users in discovering the service and decide how it can be used with other services.

References

[1] M. Åsberg and L. Strömbäck, "Interoperable and easy-to-use web services for the bioinformatics community - a case study," in *The Second International Conference on Advances in Databases, Knowledge, and Data Applications DBKDA 2010*, 2010.

[2] M. Y. Galperin, "The molecular biology database collection: 2008 update," *Nucleic Acids Research*, vol. 36, 2008.

[3] ——, "The molecular biology database collection: 2007 update," *Nucleic Acids Research*, vol. 35, 2007.

[4] L. Strömbäck, D. Hall, and P. Lambrix, "A review of standards for data exchange within systems biology," *Proteomics*, vol. 7, no. 6, pp. 857–867, 2007.

[5] L. Strömbäck, V. Jakoniene, H. Tan, and P. Lambrix, "Representing, storing and accessing molecular interaction data: a review of models and tools," *Briefings in Bioinformatics*, vol. 7, no. 4, pp. 331–338, 2006.

[6] P. W. Lord, S. Bechhofer, M. D. Wilkinson, G. Schiltz, D. Gessler, D. Hull, C. A. Goble, and L. Stein, "Applying semantic web services to bioinformatics: Experiences gained, lessons learnt," *International Semantic Web Conference*, pp. 350–364, 2004.

[7] E. Germani, *Web Services Essentials*. O'Reilly, 2002.

[8] M. Kanehisa, S. Goto, S. Kawashima, Y. Okuno, and M. Hattori, "The KEGG resource for deciphering the genome," *Nucleic Acids Research*, vol. 32, 2004.

[9] M. Senger, P. Rice, and T. Oinn, "Soaplab - a unified sesame door to analysis tools," in *UK e-Science- All Hands Meeting 2003*, 2003.

[10] B. Ludäscher, I. Altintas, C. Berkley, D. Higgins, E. Jaeger, M. Jones, E. A. Lee, J. Tao, and Y. Zhao, "Scientific workflow management and the Kepler system," *Concurrency and Computation: Practice & Experience*, vol. 18, no. 10, pp. 1039–1065, 2006.

[11] The Taverna Team, "Taverna - open source and domain independent workflow management system," Accessed january 16th 2011. [Online]. Available: http://www.taverna.org.uk/

[12] D. Hull, K. Wolstencroft, R. Stevens, C. Goble, M. Pocock, P. Li, and T. Oinn, "Taverna: a tool for building and running workflows of services." *Nucleic Acids Research*, 2006.

[13] T. Oinn, M. Greenwood, M. Addis, N. Alpdemir, J. Ferris, K. Glover, C. Goble, A. Goderis, D. Hull, D. Marvin, P. Li, P. Lord, M. Pocock, M. Senger, R. Stevens, A. Wipat, and C. Wroe, "Taverna: lessons in creating a workflow environment for the life sciences," *Concurrency and Computation: Practice and Experience*, 2006.

[14] The VisTrails Team, "VisTrailsWiki," Accessed january 16th 2011. [Online]. Available: http://vistrails.org/

[15] L. Bavoil, S. P. Callahan, P. J. Crossno, J. Freire, C. E. Scheidegger, C. T. Silva, and H. T. Vo, "Vistrails: Enabling interactive multiple-view visualizations," *In Proceedings of IEEE Visualization*, 2005.

[16] P. B. T. Nerrincx and J. A. M. Leunissen, "Evolution of web services in bioinformatics," *Briefings in Bioinformatics*, vol. 6, no. 2, pp. 178–188, 2005.

[17] The BioMoby Consortium, "Interoperability with Moby 1.0-it's better than sharing your tootbrush!" *Briefings in Bioinformatics*, vol. 9, no. 3, pp. 220–231, 2009.

[18] K. Wolstencroft, P. Alper, D. Hull, C. Wroe, P. Lord, R. Stevens, and C. Goble, "The myGrid ontology: bioinformatics service discovery," *International Journal of Bioinformatics Resesearch and Applications*, vol. 3, no. 3, pp. 303–325, 2007.

[19] J. Bhagat, F. Tanoh, E. Nzuobontane, T. Laurent, J. Orlowski, M. Roos, K. Wolstencroft, S. Aleksejevs, R. Stevens, S. Pettifer, R. Lopez, and C. A. Goble, "BioCatalogue: a universal catalogue of web services for the life sciences," *Nucleic Acids Research*, 2010.

[20] The BioCatalogue Project, "BioCatalogue.org - Home," Accessed january 16th 2011. [Online]. Available: http://www.biocatalogue.org

[21] The BioMoby Consortium, "BioMoby Semantic MOBY," Accessed january 16th 2011. [Online]. Available: http://biomoby.open-bio.org/index.php/semantic-moby/

[22] The Gene Ontology Consortium, "GO database abbreviations," Accessed january 16th 2011. [Online]. Available: http://geneontology.org/cgi-bin/xrefs.cgi

[23] M. Wilkinson, D. Gessler, A. Farmer, and S. L, "The BioMOBY project explores open-source, simple, extensible, protocols for enabling biological database interoperability," *Proceeding of the Virtual Conference on Genomic and Bioinformatics*, vol. 3, pp. 16–26, 2003.

[24] Object Management Group, "Life sciences analysis engine specification," Accessed january 16th 2011. [Online]. Available: http://www.omg.org/technology/documents/formal/lsae.htm

[25] E. Kawas, M. Senger, and M. D. Wilkinson, "BioMoby extensions to the taverna workflow management and enactment software," *BMC Bioinformatics*, 2006.

[26] Information Sciences Institute, "Pegasus:home," Accessed january 16th 2011. [Online]. Available: http://pegasus.isi.edu

[27] The Kepler Project, "The Kepler project - Kepler," Accessed january 16th 2011. [Online]. Available: http://kepler-project.org

[28] The Swift Project, "Swift," Accessed january 16th 2011. [Online]. Available: http://www.ci.uchicago.edu/swift

[29] L. Moreau, J. Freire, J. Futrelle, R. E. McGrath, J. Myers, and P. Paulson, "The open provenance model," 2008. [Online]. Available: http://eprints.ecs.soton.ac.uk/14979/1/opm.pdf

[30] T. Ellkvist, D. Koop, J. Freire, C. Silva, and L. Strömbäck, "Using mediation to achieve provenance interoperability," in *IEEE Workshop on Scientific Workflows*, 2009.

[31] EMBL-EBI, "BioModels database," Accessed january 16th 2011. [Online]. Available: http://www.ebi.ac.uk/biomodels-main/

[32] UniProt Consortium, "UniProtKB," Accessed january 16th 2011. [Online]. Available: http://www.uniprot.org/help/uniprotkb

[33] RCSB, "RCSB protein data bank," Accessed january 16th 2011. [Online]. Available: http://www.rcsb.org/pdb/home/home.do

[34] Oracle Corporation, "Oracle technology network for java developers," Accessed january 16th 2011. [Online]. Available: http://www.oracle.com/technetwork/java/index.html/

[35] The Eclipse Foundation, "Web tools platform (WTP) project," Accessed january 16th 2011. [Online]. Available: http://www.eclipse.org/webtools/

[36] B. J. Bornstein, S. M. Keating, A. Jouraku, and H. M., "Libsbml: An api library for sbml." *Bioinformatics*, 2008.

[37] D. Koop, C. E. Scheidegger, S. P. Callahan, J. Friere, and C. T. Silva, "Viscomplete: Automating suggestions for visualization pipelines," *IEEE Transactions on Visualization and Computer Graphics*, vol. 14, no. 6, pp. 1691–1698, 2008.

[38] T. Katayama, M. Nakao, and T. Takagi, "TogoWS: integrated SOAP and REST APIs for interoperable bioinformatic web services," *Nucleic Acids Research*, 2010.

[39] Database Center for Life Science, "TogoWS," Accessed january 16th 2011. [Online]. Available: http://togows.dbcls.jp/

[40] P. M. Gordon and C. W. Sensen, "Seahawk: moving beyond HTML in web-based bioinformatics analysis," *BMC Bioinformatics*, 2007.

[41] The Open Bioinformatics Foundation, "Open bioinformatics foundation," Accessed january 16th 2011. [Online]. Available: http://www.open-bio.org/

[42] P. M. Gordon, K. Barker, and C. W. Sensen, "Helping biologists effectively build workflows, without programming," in *Proceedings of the 7th International Conference on Data Integration in the Life Sciences - DILS 2010*, 2010.

Implementing Row Version Verification for Persistence Middleware using SQL Access Patterns

Fritz Laux
Fakultät Informatik
Reutlingen University
D-72762 Reutlingen, Germany
fritz.laux@reutlingen-university.de

Martti Laiho
Dpt. of Business Information Technology
Haaga-Helia University of Applied Sciences
FI-00520 Helsinki, Finland
martti.laiho@haaga-helia.fi

Tim Lessner
School of Computing
University of the West of Scotland
Paisley PA1 2BE, UK
tim.lessner@uws.ac.uk

***Abstract*—Modern web-based applications are often built as multi-tier architecture using persistence middleware. Middleware technology providers recommend the use of Optimistic Concurrency Control (OCC) mechanism to avoid the risk of blocked resources. However, most vendors of relational database management systems implement only locking schemes for concurrency control. As a consequence a kind of OCC has to be implemented at client or middleware side. The aim of this paper is to recommend Row Version Verification (RVV) as a mean to realize an OCC at the middleware level. To help the developers with the implementation of RVV we propose to use SQL access patterns. For performance reasons the middleware uses buffers (cache) of its own to avoid network traffic and to reduce disk I/O. This caching, however, complicates the use of RVV because the data in the middleware cache may be stale (outdated). We investigate various data access technologies, including the Java Persistence API and Microsoft's LINQ technologies in combination with commercial database systems for their ability to use the RVV programming discipline. The use of persistence middleware that tries to relieve the programmer from the low level transaction programming turns out to even complicate the situation in some cases.**

The contribution of this paper are patterns and guidelines for an implementation of OCC at the middleware layer using RVV. Our approach prevents from inconsistencies, reduces locking to a minimum, considers a couple of mainstream technologies, and copes with the effects of concurrency protocols, data access technologies, and caching mechanisms.

***Keywords*-persistence middleware, caching, data access pattern, row version verification.**

I. Introduction

Databases provide reliable data storage services, but especially for business critical applications the use of these services requires that applications will obey the concurrency control protocol of the underlying Database Management System (DBMS).

In this paper we look at the application development from the Online Transaction Processing (OLTP) point of view, and especially on the modern mainstream commercial DBMS used by industry , namely DB2, Oracle, and SQL Server, with ISO SQL standard as the common denominator. The ideas described here are extensions of the work first presented in [1]. A cornerstone of data management is the proper transaction processing, and a generally accepted requirement for reliable flat SQL transactions is the ACID transaction model defined by Haerder and Reuter [2]. The acronym ACID stems from the initials of the four well known transaction properties: Atomicity, Consistency, Isolation, and Durability. However, the original definition for Isolation "Events within a transaction must be hidden from other transactions running concurrently" cannot be fulfilled by the mainstream commercial DBMS systems. They only provide a concurrency control (CC) mechanism that, according to the isolation level settings, prevents a transaction itself from seeing changes made by concurrent transactions, and protects the transaction's updates against overwrites from other transactions during its execution time. The implemented isolation levels follow roughly the definitions in the ISO SQL standard, but with semantics of the used CC mechanism.

A typical CC mechanism used to ensure the isolation of concurrent SQL transactions in mainstream DBMS, for example DB2 and SQL Server, is some variant of multi-granular Locking Scheme Concurrency Control (which we call shortly LSCC, whereas in literature the more general term pessimistic concurrency control is often used). LSCC systems use exclusive locks to protect writes until the end of a transaction, and shared locks protect read operations against concurrent modifications of the data. The term multi-granular refers to a mechanism of applying locks, for example, at row level or table level, and compatibility issues of these locking levels are solved using special intent locks. The isolation level declared for the transaction fine-tunes the duration of the shared locks. Locking may block concurrent transactions, and may lead to deadlocks as a kind of concurrency conflict, in which the victim transaction is chosen by the DBMS according to internal rules among the competing transactions. Another typical CC mechanism is some variant of Multi-Versioning Concurrency Control (MVCC), which is used for example by Oracle and a specially configured SQL Server database. In addition, Oracle also uses locking at table level and provides the possibility of programmatic row level locking by the SELECT .. FOR UPDATE variant

of the SQL select statement.

The MVCC mechanism always allows reading of committed data items without blocking. In case of concurrency competition between transactions, the first writer transaction wins and conflicting updates or write operations are prevented by raising serialization exceptions.

The third available concurrency control mechanism is server-side optimistic concurrency control (OCC), as presented by Kung and Robinson [3], in which transactions only read contents from the database while all changes are first written in the private workspace of the transaction, and finally at transaction commit phase - after successful validation - the changes will be synchronized into the database as an atomic action. In case of concurrency competition between transactions the first transaction to COMMIT is the winner and others will get a serialization exception, however, after requesting the COMMIT only. Currently server-side OCC has not yet been implemented in any commercial mainstream DBMS systems. The only implementation of which we know is the Pyrrho DBMS [4] and VoltDB [5], [6]. Both products focus on transaction processing in the Cloud and apply OCC to overcome the drawbacks of locking that significantly reduces the concurrency, hence, the response time and scalability crucial for Cloud storage services. Please note, however, that scalability is a general requirement and not limited to Cloud storage services. One reason, why OCC might be more appealing for the Cloud, is the basic assumption of OCC that conflicts are rare and especially applications that are built on top of the cloud are not primarily OLTP applications with a high concurrency on the same record.

With the advent of multi-tier web applications, or more precisely, decentralized and loosely coupled transactional systems, client-side OCC has gained new attention. Providers of enterprise architecture frameworks (like Java Enterprise Edition (JEE)) and persistence middleware (like object relational mappers, e.g., Hibernate) propose to use optimistic concurrency control to avoid the risk of blocked resources and orphan locks. Developers face now the situation that they have to implement a kind of optimistic concurrency control over the existing concurrency control provided by the DBMS. But shifting the burden to the middleware or application is a tricky task [1]. First, there is a need to distinguish between business/user transaction and SQL transaction. Second, we have to deal with transactional legacy applications that bypass the middleware.

Definition: (SQL transaction) A *SQL transaction* is defined as finite sequence of SQL statements that obey the ACID properties.

This definition is equivalent to the general transaction definition from Härder and Reuter [2], but restricted to SQL databases. SQL transactions are supported by relational DBMS.

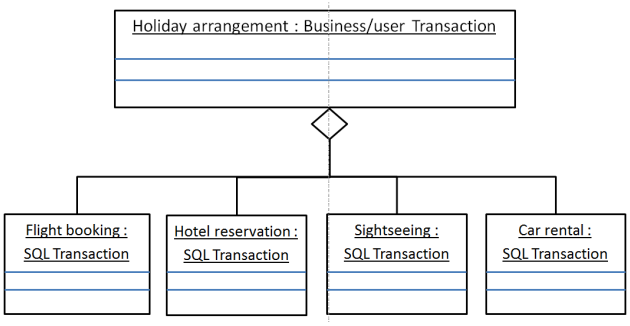


Figure 1. Example UML diagram for a business/user transaction

Definition: (Business/user transaction) A *business/user transaction* is an unit of work that holds the ACID properties. The unit of work is defined by the business (application) as an aggregate of applications, each one executing at least one SQL transaction.

Ideally business/user transactions should be supported by a transaction coordinator, that ensures the overall ACID outcome. But the result is not necessarily based on Herbrand semantic. Depending on the business rules there might be more than one successful outcome (e.g. an alternative result) or compensation transactions could be involved. As example, think of a complex business transaction like booking a holiday arrangement that involves multiple, non-integrated applications, e.g. flight booking, hotel reservation, car rental contract, or siteseeing tour (see Figure 1). If flight and hotel bookings succeed and car rental fails, but the siteseeing tour is possible, this could be considered as an alternative successful business transaction. However, a car rental without flight or hotel booking should be forbidden. Each of the applications that is part of a business/user transaction comes with its own SQL transactions that might be already committed before another application happens to fail, hence, a compensation for already committed SQL transactions is needed. In this paper the only aggregate used for RVV is a sequence of SQL transactions.

In general, the design of a multi-tier architecture requires to split a business/user transaction into a sequence of SQL transactions for several reasons:

- The business transaction needs to access more than one DBMS
- Access to a database is executed under different sessions because of session pooling
- The business transaction uses legacy systems not designed to support distributed transactions.

This leads to a situation where the DBMS is unaware of the complete business/user transaction because different sessions and different transactions form the actual transaction. On the one hand, the splitting avoids automatic locking for a possibly unpredictable time, on the other hand, it requires additional mechanisms to ensure the global consistency of

the business/user transaction.

But, if concurrent business transactions are splitted into multiple SQL transactions these may interfere without the possibility for any help by the DBMS. For instance a lost update could arise due to interleaved execution. Therefore, the applications, the middleware, and the DBMS need to co-operate somehow.

The implementation of a concurrency control mechanism in every application is inefficient and depends on the skills of the programmer. Therefore it is preferable to implement the concurrency control protocol in the middleware for the benefit of all applications using this middleware. The implementation of a concurrency control protocol providing full serializability is often to restrictive and derogates performance. A good compromise is the Row Version Verification (RVV) discipline that avoids the lost update problem from the business transaction's view.

The contribution of this paper is how to apply the data access patterns of [11] to different middleware technologies. The novel parts are techniques how to reliably implement RVV discipline for complex business transactions as defined above.

A. Structure of the Paper

In the next Section we motivate the RVV and describe its mechanism. After the related work, Section IV introduces the blind overwriting in the business context and compares it to the generally known lost update problem, which is typically covered in database literature. Section V presents three SQL patterns, which serve as guideline for the implementations of RVV. Section VI starts with the presentation of a typical use case including SQL statements for its setup on a relational database. Each of the following Sections, VII (JDBC, .NET), VIII (Hibernate, JPA), IX (MS LINQ), and X (JDO) present implementations of RVV using the data access patterns presented in Section V. Section XI concludes the paper with a comparison between these technologies.

II. The Row Version Verification (RVV) Discipline

The *lost update* is a typical phenomenon in multi-user file-based systems without proper concurrency control, i.e., a record x updated by some process A will be overwritten by some other concurrent process B like in the following problematic canonical schedule [7, pp. 62 - 63]: $r_A(x), r_B(x), w_A(x), w_B(x)$, where $r_T(x)$ and $w_T(x)$ denote read and write operations of transaction T on data item x .

With a sufficient isolation level a DBMS would not allow for a lost update and if a LSCC is used x would be locked for the transaction's duration and other transactions are prevented from accessing x. But, if we decide to follow the recommendation of the middleware vendors or the writers of JEE tutorials to use OCC, the DBMS should be configured not to use transactions to avoid locking or to use auto-commit such that every data access statement results in a transaction with locks held as short as possible. With this configuration the DBMS would be unable to prevent the lost update problem within the business transaction's view as it appears only as a blind write to the DBMS. A *blind write* is defined as overwriting an existing data item with a new value that is independent from the old value. RVV, however, as a concurrency control mechanism at the Middleware layer, on top of the DBMS, prevents from the lost update phenomenon.

RVV depends on a technical row version column that needs to be added to every SQL table. Its value is incremented whenever any data in the row is changed. A verification of the row version enables to detect if any concurrent transaction has modified the data meanwhile. If this happened, the validation fails and the transaction has to abort. If the row version is still the same, then the transaction may write the new data. In pseudo-code the mechanism looks like this:

```
(t1)    Read(x, versionX)
        // Input data
        // Process x
        old_versionX := versionX
(t2)    if (old_versionX = Read(versionX))
(t2)    then
(t2)        Write(x, versionX+1)
(t2)    else
(t2)        // Report serialization conflict
(t2)    end if
```

In the time period between t_1 and t_2 other concurrent transactions may access and modify data element x. The if-then-else block is a critical section that must be executed in an uninterruptable manner.

If the row version is cached by the middleware this could lead to stale data. Therefore, it is necessary to circumvent the middleware cache for the row version in order to apply RVV.

RVV is better known under the misleading and ambiguous name *optimistic locking* (see [7, pp. 365 - 367], [8], [9], [10]) even if there is no explicit locking involved.

The motivation to use RVV results from the practice that for example web applications usually split a business transaction into several SQL transactions as previously explained for multi-tier transactional applications. In this case the database concurrency mechanism cannot ensure transactional properties for the business transaction, but RVV helps to avoid at least the blind overwriting. Consider a typical concurrency scenario with two users that try to book the same flight online. First, a list of flights is displayed (Phase 2, in Figure 2), second, a certain flight is chosen (Phase 4), and third, the flight is booked (Phase 6). When a second user books the same seat and commits before the first user proceeds to Phase 6 then the first user would overwrite

the reservation of the second user. This could be avoided by re-reading the seats in Phase 6 and compare it with Phase 4 before storing the new number.

We consider the RVV discipline as a critical reliability criterion in web based transactional application development. The proper way for applications to meet the RVV discipline is to strictly follow the SQL data access patterns presented in Laux and Laiho [11], which will be recapped in Section V. The patterns describe how to implement the RVV protocol for different database programming interfaces. These patterns essentially ensure that the row version is checked before overwriting a data value. The patterns describe how to deal with different concurrency schemes of the underlying DBMS and how to exploit triggers to support RVV from the database side.

In the present paper these data access patterns are applied to the already mentioned generic use case (Figure 2) and code samples show implementations for various technologies.

III. Related Work

Concurrency control is a cornerstone of transaction processing, it has been extensively studied for decades. Namely Gray and Reuter [9] studied locking schemes, whereas Badal [12], Kung and Robinson [3] developed optimistic methods for concurrency control (OCC). OCC distinguishes three phases (see Figure 3) within a transaction:

- read phase
- validation phase
- write phase

The first phase includes user input and thinking time. It may last for an unpredictable period. The following phases are without any user interaction. Validation and write phases are therefore very short in the range of milliseconds. The last two phases are critical in the sense that exclusive access is required. Failing to do so could result in inconsistent data, e.g lost update.

Unland [13] presents OCC algorithms without critical section. He specifically focuses on a OCC solution that solves the starvation problem, increasing the chance for long living and read only transactions to survive. Using these algorithms would allow relaxed locking but involve checking the read set against all concurrent transactions. Even if a full OCC of this type would be implemented at the middleware layer, it could only control applications that use this middleware service. Therefore this algorithms can be ruled out for our approach. Although, OCC mechanisms have been studied already 30 years ago, hardly any commercial DBMS implements algorithms of this type (see Bernstein and Newcomer [14] or Gray and Reuter [9]) except for Multi-Version Concurrency Control (MVCC).

A higher concurrency for query intensive transactions provides MVCC as described by Stearns and Rosenkrantz [15] and Bernstein and Goldman [16]. Comparing locking with MVCC it can be said that a database system with locking holds a single truth of every data item, and if it is locked by others, one needs to wait until the lock is released, whereas a MVCC systems holds a history of the truth. On read committed isolation level it is always possible to read the latest committed truth without waiting, and on serializable isolation level (also called snapshot in some systems), the data that will be read is the latest committed truth at the beginning of the transaction.

In case of MVCC, the middleware has to make sure that caching is not invalidating the multi-versioning system. This problem is discussed by Seifert and Scholl [17] who counteract with a hybrid of invalidation and propagation message. In Web applications the risk of improperly terminating transactions is extremely high (users simply ”click away”). In such cases snapshot data or locks in the case of a locking protocol are held until (hopefully) some timeout elapses. This can severely degrade performance.

With the dissemination of middleware, OCC has been recommended by IT-vendors ([18], [19], [20]) for transactional e-business and m-commerce applications but only little effort has been spent on the realisation using commercial SQL databases. Adya et al [21] recommend to use the system clock as blocking-free protocol for global serialization in distributed database systems. However this approach has to fail if the resolution is not fine enough to produce unique timestamps as we have shown for Oracle [22].

Nock ([8, pp. 395-404] describes an *optimistic lock* pattern based on version information. He points out that this access pattern does not use locking (despite of its misleading name) and therefore avoids orphan locks. His pattern does not address caching. Akbar-Husain [19] believes that demarking the method that checks the version with the required transaction attribute will be sufficient to avoid lost updates. He does not consider that only a strong enough isolation level like REPEATABLE READ or SERIALIZABLE will achieve the desired results.

During decades of development in relational database technologies the programming paradigms in data access technologies have constantly changed. The two mainstream schools in object oriented programming today are the Java [23] and the Microsoft .NET framework [24] camps, both provide call-level APIs and Object-Relational Mappings (ORM) of their own. The problems of using RVV with the older version of Enterprise Java Beans 2.0 are discussed in [25].

We follow in this paper the object persistence abstractions of Hoffer, Prescott, and Topi [26, Chapter 16] and implement the RVV discipline at the middleware level applying the SQL access patterns described in Section V.

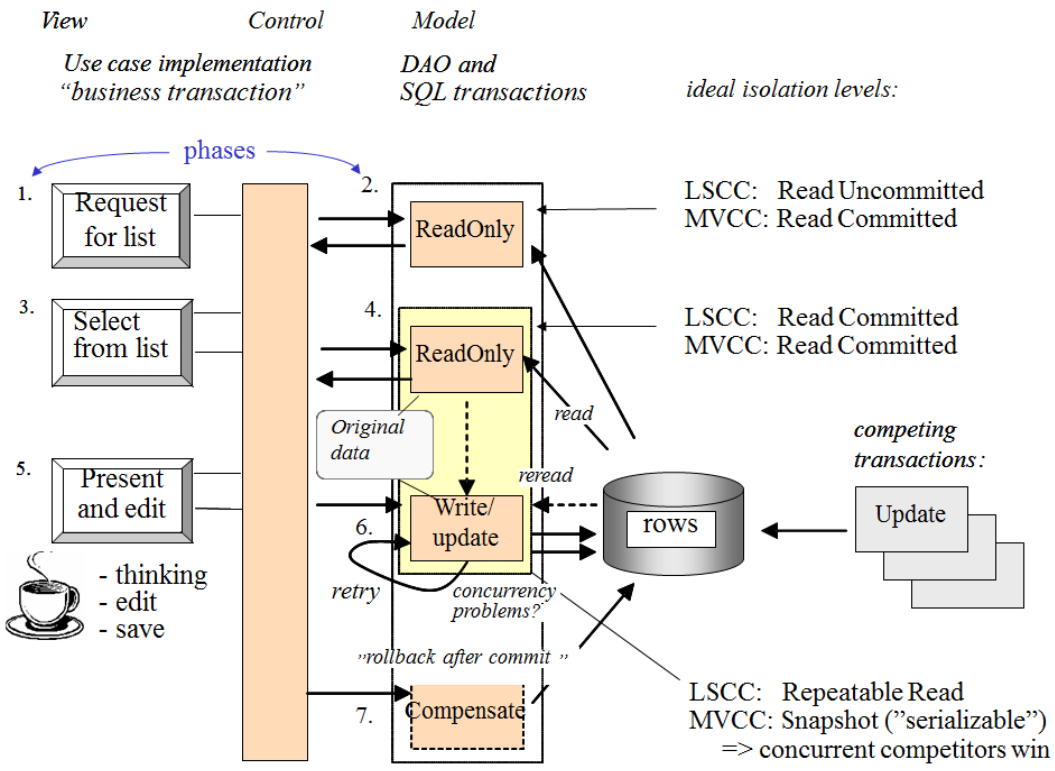


Figure 2. Business transaction with SQL transactions (phases 2, 4, 6, 7) and isolation levels of a sample use case

IV. Blind Overwriting Problem in the Application Context

Let us first consider the following problematic scenario of two concurrent processes A and B, each running a SQL transaction that is updating the balance of the same account, as shown in Table I. In this scenario step 4 of process A is empty.

The withdrawal of 200 € made by the transaction of process B will be overwritten by process A; in other words the update made by B in step 5 will be lost in step 7 when the transaction of A overwrites the updated value by value 900 € which is based on stale data, i.e., an outdated value of the balance, from step 3. If the transactions of A and B are serialized properly, the correct balance value after these transactions would be 700 €, but there is nothing that the DBMS could do to protect the update of step 5, assumed an isolation level of READ COMMITTED which is the default isolation level for most relational DBMS because of performance reasons. In READ COMMITTED isolation level the shared locks provided for the SELECT statement are released immediately after the completion of the statement and therefore it is possible for process B to obtain a lock and update the balance. So, READ COMMITTED does not protect any data read by a transaction of getting outdated right after reading the value. Locking Scheme Concurrency Control (LSCC) could prevent conflicting access to data, but not at READ COMMITTED isolation level. The proper isolation level on LSCC systems to prevent a lost update of process B should be REPEATABLE READ or SERIALIZABLE, which would protect the balance value read in the transaction of process A from getting outdated

Table I
A BLIND OVERWRITING SCENARIO USING SELECT-UPDATE IN TRANSACTION A

step	process A [T_1, T_2]	balance	process B
1	SET TRANSACTION ISOLATION LEVEL READ COMMITTED		
2		1000€	
3	SELECT BALANCE INTO :BALANCE FROM ACCOUNTS WHERE ACCTID = :ID;		
4	[COMMIT WORK;]		
5	NEWBALANCE = BALANCE - 100		UPDATE ACCOUNTS SET BALANCE = BALANCE - 200 WHERE ACCTID = :ID;
6		800€	COMMIT;
7	UPDATE ACCOUNTS SET BALANCE = :NEWBALANCE WHERE ACCTID = :ID;		
8	COMMIT;	900€	

during the transaction by holding shared locks on these rows up to the end of the transaction. As result the shared locks of Process A would block process B from a too early update. The isolation service of the DBMS guarantees that the transaction will either get the requested isolation level, or, in case of a serialization conflict, the transaction will be rejected by the DBMS. The means used for this service and the transactional outcome for the very same application code can be different when using different DBMS systems, and even in using different table structures. The LSCC may wait with granting a lock request until the possible conflict disappears. Usually a transaction rejected due to a serialization conflict should be retried by the application, but we discuss this later in Section VI.

In the second scenario of Table I, process A splits its transaction into two transactions. The first transaction, let it call T_1, consists of steps 1 to 4, including the COMMIT WORK at step 4. After some user interaction and the calculation in step 5, another transaction T_2 continues with steps 7 and 8. In this case, no isolation level can help, but transaction T_2 will make a blind write based on stale data from step 3. But, meanwhile the balance value was updated by transaction T_B of process B in step 5.

From the database perspective, the 3 transactions have been serialized correctly in the order: (T_1, T_B, and T_2). However, the problem is that there is no transaction boundary around T_1 and T_2; they are treated separately by the transaction manager. Hence, it is absolutely correct for T_B to interleave with T_1 and T_2. From a business or user point of view, especially the user that runs $_1$ *and* T_2, this is a semantically wrong behavior. Hence, as soon as a transaction is split into several sessions (e.g., due to connection pooling) or different SQL transactions, a transaction manager at the middleware layer is required that prevents from blind overwrites and provides one transactional context for T_1 and T_2. RVV provides such a context in a transparent way and the row verification ensures only consistent writes.

V. SQL Access Patterns for Avoiding Blind Overwriting

The blind write of the update transaction T_2 of steps 7 - 8 of Table I, which results in the loss of transaction T_B, could have been avoided by any of the following practices. The proposed access patterns assure that either before or during the write phase (step 7), a validation takes place and data will only be updated after a successful validation (see Figure 3). We present the patterns in the canonical form given by Coplien [27] that appears to be more compact than the one used by Gamma et al [28]:

A. Access Pattern: Sensitive UPDATE

Problem: How to prevent a blind overwrite in case of concurrent updates?

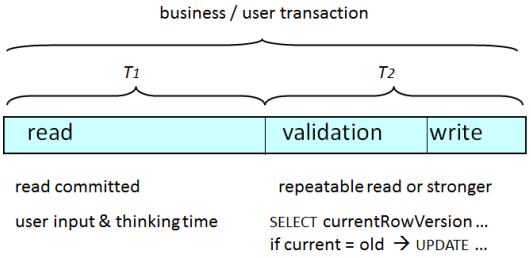


Figure 3. Context of the OCC Access Patterns

Context: Concurrent transaction processing in distributed systems has to deal with temporary disconnected situations, or sequences of SQL transactions belonging to one business transaction and nevertheless ensure correct results.

Forces:

- Using locks (LSCC) to prevent other transactions from changing values can block data items for unpredictable time in case of communication failure or in case of long user thinking time.
- Multiversion concurrency control (MVCC) or OCC do not block data access, but lead to abort conflicting transactions, except for the first one that updates the data.
- OCC is not supported by the mainstream commercial SQL databases, hence we cannot directly rely on the CC mechanism provided by the DBMS.

Solution: There is no risk of blind overwriting if T_2 in step 7 uses the form of the update which is sensitive to the current value, like B uses in step 5 as follows:

```
UPDATE Accounts
SET balance = balance - 100
WHERE acctId = :id;
```

Consequences: Please note, the update of the balance is based on a value unseen by the application. Therefore, the user will not be aware of the changed balance and this access pattern does not provide repeatable read isolation. If the user needs to know about the changed situation the access pattern "RE-SELECT .. UPDATE" could be used (see Subsection V-C) or a SELECT command could be used after the UPDATE.

B. Access Pattern: Conditional UPDATE

Problem: How to prevent a blind overwriting and provide repeatable-read for a business transaction in case of concurrent updates without using locking?

Context: The "Sensitive UPDATE" pattern in concurrent read situations may result in non-repeatable phenomenon.

Forces: Same as for "Sensitive UPDATE" plus:

- The data value on which the update is based, as read and displayed to the user, may no longer be the same (non-repeatable read phenomenon).

Solution: After transaction T_1 first has read the original row version data in step 3, transaction T_2 verifies in step 7, using an additional comparison expression in the WHERE clause of the UPDATE command, that the current row version in the database is still the same as it was when the process previously accessed the account row. For example,

```
UPDATE Accounts
SET balance = :newBalance
WHERE acctId = :id AND
(rowVersion = :old_rowVersion);
```

The comparison expression can be a single comparison predicate like in the example above where rowVersion is a column (or a pseudo-column provided by the DBMS) reflecting any changes made in the contents of the row and :old_rowVersion is a host variable containing the value of the column when the process previously read the contents of the row. In the case that more than one column is involved in the comparison, the expression can be built of version comparisons for all columns used. The comparison will be based on the 3-value logic of SQL.

Consequences: Since this access pattern does not explicitly read data, there is no need to set the isolation level. The result of the concurrency control services is the same for locking scheme concurrency control (LSCC) and multiversion concurrency control (MVCC) based database systems. The result of the update depends on the row version verifying predicate, and the application code needs to evaluate the return code to find out the number of updated rows to verify the result.

C. Access Pattern: Re-SELECT .. UPDATE

Problem: How to provide a repeatable-read isolation for a business transaction in case of concurrent updates? How to inform the user when the read set has changed and take an alternative decision?

Context: In case the result of the "Conditional UPDATE" pattern can not be directly communicated to the user. For example, when an application is using the database via middleware services (see Section VIII).

Forces: Same as for "Conditional UPDATE" plus:

- In the time between the re-SELECT and the UPDATE statement, the data read may be updated again by concurrent transactions. This serialisation conflict would force the transaction to rollback.

Solution: This is a variant of the "conditional UPDATE" pattern in which transaction T_2 first reads the current row version data from the database into some host variable current_rowVersion which allows the application to inform the user of the changed situation and take an alternative decision:

```
SELECT rowVersion
INTO :current_rowVersion
FROM Accounts
WHERE acctId = :id;
if (current_rowVersion = old_rowVersion)
then
   UPDATE Accounts
   SET balance = :newBalance
   WHERE acctId = :id;
else
   // inform the user if desired
   // and/or take an alternative decision
end if
```

To avoid any concurrency problems in this phase, it is necessary to make sure that no other transaction can change the row between the SELECT and the UPDATE. For this purpose, we need to apply a strong enough isolation level (REPEATABLE READ, SNAPSHOT, or SERIALIZABLE) or explicit row-level locking, such as Oracle's FOR UPDATE clause in the SELECT command.

Consequences: Since isolation level implementations of LSCC and MVCC based DBMS are different, the result of concurrency services can be different: in LSCC based systems the first writer of a row or the first reader using REPEATABLE READ or SERIALIZABLE isolation level will usually win, whereas in MVCC based systems the first writer wins the concurrency competition. On server side OCC the first one to commit wins, and in the event of LSCC deadlocks the victim (the transaction that is aborted) is determined by internal rules of the DBMS.

VI. A Basic Use Case Example

In the following scenario we will distinguish *SQL transactions* from *business transactions* (also called user transaction) as defined in Section I. An SQL transaction is known to the DBMS. It starts explicitly with a BEGIN TRANSACTION statement or implicitly with the first SQL statement after the last transaction. The SQL transaction terminates either with COMMIT or ROLLBACK. A business transaction in our case consists of a finite sequence of SQL transaction that are treated as a logical unit of work. The involved database system is unaware of this logical unit. Therefore, the databases can not support the atomicity of a business transaction. If the business transaction maps one-to-one to an SQL transaction as in legacy applications the DBMS can fully support the transactional properties.

In modern, web-based transactional applications a business transaction consists of multiple SQL transactions. This is not only because multiple database systems are involved, there is a technical reason too. Application servers use connection pooling, so even if only one database system is used, different SQL statements may belong to different connections and consequently to different transactions.

We see that the concurrency scope of an application needs to be extended to cover sequences of SQL transactions (or more generally server-side transactions), implementing some business transaction. Figure 2 presents a typical business transaction in 6 phases containing three SQL transactions (list, select, and edit/book) like the flight booking mentioned before plus an optional compensation phase. The ideal isolation levels listed for each SQL transaction depends on the concurrency control provided by the DBMS. For example, the default concurrency control mechanism on SQL Server is locking (LSCC), but it can alternatively be configured to use ”snapshot” isolation (MVCC). With RVV we refer to the sequence of inter-related SQL transactions (phases 4 and 6 in the Figure 2), which may belong to the same SQL connection, but typically in a Web application could belong to different connections as explained above. In this case the locks from Phase 4 cannot be held until Phase 6.

To make sure that no concurrent transaction has changed the contents of the row fetched in Phase 4, we need to verify that the content in the database is still the same when trying to update the row/object in Phase 6. Otherwise, the update will cause a blind write that overwrites the result from other competing transactions, thus loosing data from the database.

If the DBMS reports a concurrency conflict in Phase 6 (the write/update phase), the application may retry the statement because some conflicts are of transient nature. Temporary conflicts that disappear after the conflicting transactions have terminated could result either from an active deadlock prevention, from transactions terminated with ROLLBACK, or from released locks. A RVV validation that fails is unrepeatable as the version value is never decremented.

A committed transaction cannot be rolled back, but some systems provide a compensation transaction that reverses the effects of a previously successful transaction. This is like cancelling an order or contract that in fact results in a new order or contract that reverses the previous one.

For setting up the scenario on the SQL Server, the following Transact-SQL commands could be used:

```
CREATE TABLE rvv.VersionTest(
id INT NOT NULL PRIMARY KEY (id),
s VARCHAR(20), -- a sample data column
rv ROWVERSION -- pseudo-column reflecting updates
) ;
GO
CREATE VIEW rvv.RvTestList (id,s)) -- for Phase 2
AS SELECT id,s FROM rvv.VersionTest ;
GO
CREATE VIEW rvv.RvTest (id,s,rv) --for phases 4 and 6
AS SELECT id,s,CAST(rv AS BIGINT) AS rv
FROM rvv.VersionTest WITH (UPDLOCK) ;
GO
INSERT INTO rvv.RvTest (id,s) VALUES (1,'some text');
INSERT INTO rvv.RvTest (id,s) VALUES (2,'some text');
```

For technical details of the above script the reader is referred to the SQL Server online documentation [29].

VII. Baseline RVV Implementations using Call-Level API

The first open database call-level interface and de facto standard for accessing almost any DBMS is the ODBC API specification which has strongly influenced the data access technologies since 1992. The current specification is almost the same as the SQL/CLI standard of ISO SQL. Many class libraries have been implemented as wrappers of ODBC and many data access tools can be configured to access databases using ODBC. Based on pretty much the same idea, Sun has introduced the JDBC interface for Java applications accessing databases which has become an industry standard in the Java world. Using the previously defined SQL views for accessing table VersionTest and applying the RVV discipline, the following sample Java code for Phase 6 (the update phase) of Figure 2 reveals the necessary technical details:

```
 // *** Phase 6 - UPDATE (Transaction) ***
con.setAutoCommit(false);
// Pattern B update - no need to set isolation level
string sqlCommand = "UPDATE rvv.RvTest " +
"SET s = ? " +
"WHERE id = ? AND rv = ? ";
pstmt = con.prepareStatement(sqlCommand);
pstmt.setString(1, newS);
pstmt.setLong(2, id);
pstmt.setLong(3, oldRv);
int updated = pstmt.executeUpdate();
if (updated != 1) {
throw new Exception("Conflicting row version in the
database! " );
}
pstmt.close();
// Update succeeded -> The application needs to know
the new value of RV
sqlCommand = "SELECT rv FROM rvv.RvTest WHERE id =
?";
pstmt = con.prepareStatement(sqlCommand);
pstmt.setLong(1, id);
ResultSet rs = pstmt.executeQuery();
newRv = rs.getLong(1);
rs.close();
pstmt.close();
```

```
con.commit();
```

In the above, as in all following examples, it is assumed that the version attribute `rv` will be maintained by the database itself, e.g., by a row level trigger or some pseudo-column mechanism as described in the following Subsection VII-A. If the database has no such capability, every application itself has to take care of incrementing the version on every update. If legacy applications do not follow this convention of incrementing the version, they are subject to lose their transactions.

Every 4-5 years Microsoft has introduced a new data access technology after ODBC, and in the beginning of this millennium ADO.NET built on various already existing data providers. Compared to Microsoft's ADO it is a new data access design close to JDBC, but simplified and extended. Instead of providing a universal interface to all kind of data sources it consists of a family of data models which can be generic like OleDb or native providers like SqlClient for SQLServer or OracleClient for Oracle. Each of these implement their own object classes. Without providing details about this rich technology we just show below the Phase 6 of Figure **??** from our baseline implementation of RVV using C# language and the native .NET Data Provider (SqlClient) [30] to access SQL Server 2008:

```
SqlCommand cmd = cn.CreateCommand(); //connection
creates new command object
// Phase 6 - update transaction
txn = cn.BeginTransaction();
cmd.Transaction = txn; // bind cmd to transaction
// Pattern B update including reread of rv using
OUTPUT clause of T-SQL:
cmd.CommandText = "UPDATE rvv.RvTest " +
"SET s = @s OUTPUT INSERTED.rv " +
"WHERE id = @id AND rv = @oldRv ";
cmd.Parameters.Clear();
cmd.Parameters.Add("@s", SqlDbType.Char, 20).Value =
newS;
cmd.Parameters.Add("@id", SqlDbType.Int, 5).Value =
id;
cmd.Parameters.Add("@oldRv", SqlDbType.BigInt,
12).Value = oldRv; // retrieved in step 4
long newRv = 0L;
try { newRv = (long) cmd.ExecuteScalar();
txn.Commit();
}
catch (Exception e) {
throw new Exception("Conflicting row version in
database "+ e.Message);
}
cmd.Dispose();
```

The above code-snippet shows how to bind the SQL command `cmd` to the controlling transaction object `txn`. The SQL Server provides an elegant solution for reading the current row version `rv` at the end of the SQL UPDATE command. Using the OUTPUT clause the Transact-SQL UPDATE command retrieves the new value when the SQL UPDATE is executed with the call to `ExecuteScalar()`.

A. Server-side version management

There are multiple options for verifying the row version. These include the comparison of the original contents of all or some relevant subset of column values of the row, a check-sum of these, some technical pseudo-column maintained by the DBMS, or an additional technical SQL column. In the latter case it is the question how the values of this column are reliably maintained.

A general solution for row version management is to include a technical row version column `rv` in each table as defined in the following example:

```
 CREATE TABLE Accounts (
acctId INTEGER NOT NULL PRIMARY KEY,
balance DECIMAL(11,2) NOT NULL,
rv BIGINT DEFAULT 0); -- row version
```

and using a row-level trigger to increase the value of column `rv` on any row automatically every time the row is updated. The row-level UPDATE trigger is defined in SQL language as follows:

```
 CREATE TRIGGER TRG_VersionStamper
NO CASCADE BEFORE UPDATE ON Accounts
REFERENCING NEW AS new_row OLD AS old_row
FOR EACH ROW
IF (old_row.rv = 9223372036854775807) THEN
   SET new_row.rv = -9223372036854775808;
ELSE
   SET new_row.rv = old_row.rv + 1;
END IF;
```

We call the use of a trigger or a DBMS maintained technical pseudo-column as *server-side stamping* which no application can bypass, as opposed to *client-side stamping* using the SET clause within the UPDATE command - a discipline that all applications should follow in this case. Row-level triggers are affordable, although they lead to lower performance and hence to approximately 2% higher execution time on Oracle and DB2 [22, p. 28], whereas SQL Server does not even support row-level triggers.

Timestamps are typically mentioned in database literature as a means of differentiating any updates of a row. However, our tests [22] show that, for example, on a 32bit Windows workstation using a single processor Oracle 11g can generate up to 115 updates having the very same timestamp. Almost the same problem applies to DATETIME of SQL Server 2008 and TIMESTAMP of DB2 LUW 9, with exception of the new ROW CHANGE TIMESTAMP option in DB2 9.5

which generates unique timestamp values for every update of the same row using the technical TIMESTAMP column.

The native TIMESTAMP data type of SQL Server is not a timestamp but a technical column which can be used to monitor the order of all row updates inside a database. We prefer to use its synonym name ROWVERSION. This provides the most effective server-side stamping method in SQL Server, although, as a side-effect it generates an extra U-lock which will result in a deadlock in the example at step 6 of Figure 2. The remedy for this deadlock is to use the table hint UPDLOCK which will block new U-lock requests and prevents the transactions from running into a deadlock.

In version 10 and later versions, Oracle provides a new pseudo-column, the Oracle Row System Change Number (ORA_ROWSCN) for rows in every table created with the ROWDEPENDENCIES option [31]. This will show the transaction's System Change Number (SCN) of the last committed transaction which has updated the row. This provides the most effective server-side stamping method for RVV in Oracle databases, although as a harmful side-effect, the row-locking turns its value to NULL for the duration of the writing transaction.

VIII. RVV Implementations using ORM Middleware

Data access patterns solving the impedance mismatch between relational databases and object-oriented programming are called Object-Relational Mappers (ORM) [8]. One widely known ORM technology is the Container Managed Persistence (CMP) pattern of the Java Persistence API (JPA) as part of the Java Enterprise Beans 3.0 (EJB3). The JPA specification assumes the use of "optimistic locking" [20].

The JPA stimulated the market for sophisticated persistence managers providing object-relational mappings, such as TopLink [32] and Hibernate [33]. Figure 4 shows our interpretation of the alternatives of the current Hibernate framework which implements the JPA but also allows full control over Hibernate's Core down to the JDBC code level, which we actually need for our RVV implementation when using Hibernate. In terms of RVV we are mainly interested in the object persistence services of ORM frameworks. As examples for these services Hibernate Core and Hibernate JPA providers are tested for their ability to implement RVV.

A. RVV implementation using Hibernate Core

Hibernate provides an optimistic concurrency mechanism called "optimistic locking" (described in the Hibernate Reference Documentation [34]) based on version control. This service can be configured programmatically and may be overwritten by XML-based configuration files.

For instance, the programming paradigm for persistent classes can chose any of the following options

- version checking by the application, e.g., RVV validation

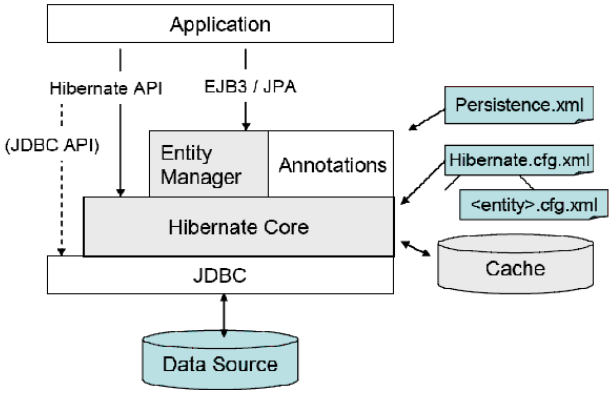


Figure 4. Hibernate architecture

- automatic version checking by an entity manager
- automatic version checking of detached objects by an entity manager

Automatic version checking takes place for every instance of the class during the transaction's COMMIT phase based on a technical version column when the attribute setting is `optimistic-lock="version"`. As alternative, Hibernate provides a validation based on a set of columns when setting the attribute to `optimistic-lock="all"` which will compare the contents of all columns, or by `optimistic-lock="dirty"` which will compare only the contents of columns which have been changed by the transaction.

The single technical column for version validation can be defined by the XML element <version> of the Hibernate object-relational mapping declaration in an entity's cfg.xml file as follows:

```
<class name="rvvtest" table="RVTest" ...
        optimistic-lock="version">
  <id name="id" column="ID" />
  <version column="RV" generated="always" ../>
</class>
```

where the attribute value *generated="always"* means that the value of the technical column is generated by the DBMS on insert and update, whereas the attribute *generated="never"* means that Hibernate will generate the value during synchronizing the contents with the database. The drawback of the validation based on Hibernate's generated technical column is that it is not reliable in case the data gets updated by some other software.

Another configurable behaviour of the data access is the SQL Isolation Level, which unfortunately cannot be changed for a single transaction. But exactly this capability is needed for Phase 6 of our example scenario. The original Hibernate engine, called Hibernate Core, provides a native interface providing the needed low level capability. Hibernate Core

services allow direct JDBC access to the data sources. Switching to the JDBC level involves the creation of a new connection on JDBC level and a subsequent method call to set the isolation level. See comments 1) and 2) in the example Java code of Phase 6.

Hibernate, like Toplink, tries to optimize data access performance using its own cache, that makes row version verification difficult. One must bypass the cache when fetching the current row version from the data source. Switching to the JDBC level allows to reload the RVV entity including the row version which bypasses Hibernate's cache mechanism. This is done in the example Java code using the `refresh(re2)` method on session level.

The use case scenario (see Figure 2) needs at least isolation level REPEATABLE READ in Phase 6. This is available in DB2 and SQL Server, but not in Oracle, which for our purposes can only provide the snapshot isolation, calling it SERIALIZABLE. We see this as a challenge and want to prove that ORA_ROWSCN can be used as row version field managed at server-side without Hibernate's optimistic locking services. It should be pointed out that the maintenance of the ORA_ROWSCN is done by the DBMS and cannot be bypassed by any application (including those not using Hibernate).

Hibernate requires a class definition with member variables matching the table columns of our view RVTEST. Objects of this class act as a wrapper for rows retrieved from or written to the view. The following XML-file defines the mapping between the RvvEntity class and the RVTest table for our scenario:

```
...
<hibernate-mapping>
  <class name="rvvtest.RvvEntity" table="RVTEST">
    <id name="id" column="ID"/>
    <property name="s" column="S" update="true"/>
    <property name="rv" column="RV" update="false"/>
  </class>
</hibernate-mapping>
```

Since the column *RV* is actually the ORA_ROWSCN pseudo column, we don't allow Hibernate to update it. The code portion of the critical Phase 6 shows that special tuning is needed to make Hibernate Core work correctly with Oracle (numbers refer to the comments in the code):

1) The default isolation level of READ COMMITTED suits in other phases of our use case, but it would lead to "blind overwriting" of concurrent transactions during Phase 6. Hibernate does not offer the possibility to change the isolation level dynamically, so we need to switch first to the level of JDBC services.
2) REPEATABLE READ would be the proper isolation level for Phase 6, but Oracle requires SERIALIZABLE, and Hibernate's Oracle dialect adapter does not transform REPEATABLE READ into SERIALIZABLE, so to keep the code portable we stick to SERIALIZABLE.
3) The `save()` method used to store the modified entity back to the database allows no conditional update, i.e., only SQL Pattern C is applicable.

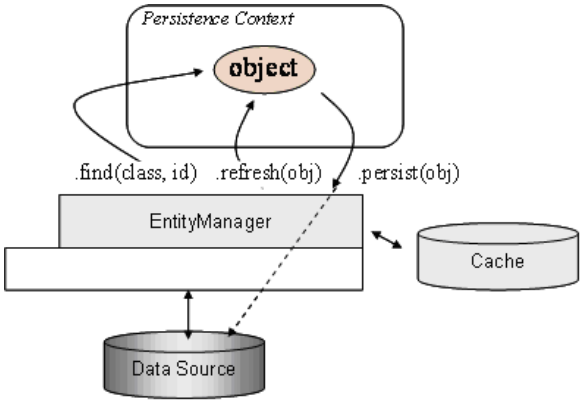


Figure 5. JPA persistence management

```
// Phase 6 - "model"
try {
tx = session.beginTransaction();
Connection conn = session.connection(); // 1) switch
to JDBC
conn.setTransactionIsolation(
      conn.TRANSACTION_SERIALIZABLE); // 2)
RvvEntity re2 = (RvvEntity)
      session.load(RvvEntity.class, id);
session.refresh(re2);
Long newRv = (Long)re2.getRv();
if (oldRv.equals(newRv)) { // 3) Pattern C
   re2.setS(s);
   session.save(re2); // 3)
   /* Programmed breakpoint for concurrency testing:
*/
   tx.commit();
} else
   throw new Exception("StaleObjectState \n" +
      "oldRv=" + oldRv + " newRv=" + newRv);
   System.out.println("persisted S = " + re2.getS() +
      "\n oldRv=" + oldRv + " newRv=" + newRv);
}
catch (Exception ex) {
   System.out.println("Exception: " + ex);
}
```

B. RVV implementation using Hibernate JPA

Hibernate provides now its Java Persistence API (JPA) implementation (EntityManager and Annotations) as a wrapper of Hibernate Core. Figure 5 presents methods of JPA for managing the persistence of an object.

Object properties can be changed only for loaded objects. This means that only *Pattern C* (re-SELECT..UPDATE) updates are possible in ORM frameworks. The caching service of ORM middleware improves performance by buffering objects, but RVV requires the current row version from the database and therefore needs to bypass the cache. ORM frameworks provide automatic "optimistic locking" services based on a timestamp or version column, but according to the JPA specification these are maintained by the ORM middleware itself (persistence provider) at client-side, so any other software can bypass the version maintenance. Therefore, the only generally reliable version control mechanism is the server-side stamping.

The following Java code sample from our RVV Paper [22] shows how to avoid stale data from Hibernate's cache. To set the isolation level via JDBC we first need to switch to Hibernate's core level as in the previous example. This is done from the underlying session object at the JDBC level. The session object creates a new connection, the connection `conn` begins a new transaction and sets the isolation level to SERIALIZABLE. Then, the object is reloaded and the actual *newRv* is read. The used *Pattern C* requires REPEATABLE READ to ensure that the row version will not change during validation and execution of the update. But, for portability reasons we choose the stronger isolation level SERIALIZABLE.

```
// Phase 6 - "model"
em.clear(); //1) clear EntityManager's cache for RVV
try {    Session session = (Session)em.getDelegate();
// JPA => Hibernate Core
   Connection conn = session.connection(); // => JDBC
   Transaction tx6 = session.beginTransaction();
   conn.setTransactionIsolation(
       conn.TRANSACTION_SERIALIZABLE); // Pattern C
   RvvEntity re2 = em.find(RvvEntity.class, id);
// reload the object
   Long newRv = (Long)re2.getRv(); // read current
row version
   if (oldRv.equals(newRv)) { // verifying the
version      re2.setS(s); // update of properties
     em.persist(re2); // Pattern C RVV update
     tx6.commit();
   }
   else
     throw new Exception("StaleObjectState: oldRv=" +
oldRv + " newRv=" + newRv);
}
catch (Exception ex) {
   System.out.println("P 6, catched exception: " +
ex);
}
```

Apart from different method names, Hibernate API and JPA provide approximately the same abstraction level and in both cases it is necessary to use JDBC level access to ensure the appropriate control over the SQL isolation level. To circumvent Hibernate's cache we need to either refresh the object with the session `refresh()` method (Hibernate API) or clear the entity manager's cache with the `clear()` method (Hibernate JPA).

IX. RVV Implementation using LINQ to SQL

Microsoft's answer to the ORM frameworks is Language Integrated Query (LINQ) for the .NET Framework 3.5. The class libraries of LINQ can be integrated as part of any .NET language providing developer "IntelliSense" support during coding time and error checking already at compile-time [35]. So called "standard query operators" of LINQ can be applied to different mappings using LINQ providers, such as LINQ to XML, LINQ to Datasets, LINQ to Objects, and LINQ to SQL. In the following C# code sample the object *myRow* is loaded from the database in Phase 4 and string *newS* contains a new value entered in Phase 5. In Phase 6 our use case enters the update phase, first the string *newS* is assigned to myRow's member variable *S* and then the changes are submitted to the DataContext *dc*. The DataContext object holds the open connection to the database in order to finally synchronize the object *myRow*'s data. The shaded part of the code is just a programmed break allowing concurrent processing for concurrency tests:

```
// Phase 4 - data access
var myRow = (from r in myTable
   where r.ID == id
   select r).First();
// Phase 5 - User interface
Console.WriteLine("Found the row ");
Console.WriteLine("ID={0}, S={1}, RV={2}",
   myRow.ID, myRow.S, myRow.Rv);
long oldRv = myRow.Rv;
Console.Write("Enter new value for column S: ");
string newS = Console.ReadLine();
// Phase 6
TransactionOptions txOpt = new TransactionOptions();
txOpt.IsolationLevel =
System.Transactions.IsolationLevel.RepeatableRead;
using ( TransactionScope txs = new TransactionScope
(TransactionScopeOption.Required, txOpt) ) {
  try { myRow.S = newS;
   // To allow time for concurrent update tests ...
   Console.Write("Press ENTER to continue ..");
   Console.ReadLine();
    dc.SubmitChanges(ConflictMode.FailOnFirstConflict);
    txs.Complete();
  }
  catch (ChangeConflictException e) {
    Console.WriteLine("ChangeConflictException: " +
      e.Message);
  }
```

```
  catch (Exception e) {
   Console.WriteLine("SubmitChanges error: " +
   e.Message + ", Source: " + e.Source +
   ", InnerException: " + e.InnerException);
  }
 }
```

The code shows the use of TransactionScope, the new transaction programming paradigm of .NET Framework which does not depend on LINQ. Setting the isolation level is actually not necessary for the transaction since it uses *Pattern B* (Conditional UPDATE), but we want to show that it can be set programmatically. The test also shows that no stale data was used in spite of LINQ caching.

At run-time, the data access statements of LINQ to SQL are translated into native SQL code which can be traced. The following sample test run trace shows that row version verification is automatic based on *Pattern B* (Conditional UPDATE) and LINQ automatically tries to refresh the updated row version content:

```
Press ENTER to continue ..
Before pressing the ENTER key the contents of column S in row 1 is updated
in a concurrent Transact-SQL session.
UPDATE [rvv].[RvTestU]
SET [S] = @p3
WHERE ([ID] = @p0) AND ([S] = @p1) AND ([RV] = @p2)

SELECT [t1].[RV]
FROM [rvv].[RvTestU] AS [t1]
WHERE ((@@ROWCOUNT) > 0) AND ([t1].[ID] = @p4)
-- @p0: Input Int (Size = 0; Prec = 0; Scale = 0) [1]
-- @p1: Input NVarChar (Size = 9; Prec = 0; Scale =
0) [TestValue]
-- @p2: Input BigInt (Size = 0; Prec = 0; Scale = 0)
[32001]
-- @p3: Input NVarChar (Size = 7; Prec = 0; Scale =
0) [testing]
-- @p4: Input Int (Size = 0; Prec = 0; Scale = 0) [1]
-- Context: SqlProvider(Sql2005) Model:
AttributedMetaModel Build: 3.5.21022.8
ChangeConflictException: Row not found or changed.
```

The SQL UPDATE statement validates the row version with the `([RV] = @p2)` predicate. This is exactly our *Pattern B*. The following SELECT statements reads the new row version and ensures that the row count is greater than 0 `((@@ROWCOUNT) > 0)`. In case of a version conflict no row is updated and a `ChangeConflictException` is returned. There is no need to implement any RVV pattern on the application level as LINQ to SQL applies this pattern automatically.

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE jdo SYSTEM "jdo.dtd">
<jdo>
  <package name="model">
   <class name="RvvEntity">
    <extension vendor-name="FastObjects" key="index"
        value="IdIndex">
            <extension vendor-name="FastObjects"
key="member"
        value="id"/>
    </extension>
   </class>
  </package>
</jdo>
```

Figure 6. Example XML file defines persistence capable class RvvEntity

X. RVV Implementation using JDO Object Database

Java Data Objects (JDO) is a vendor neutral programming interface that enables the persistent storage of Java objects developed in 2001. Meanwhile the JDO specification and reference implementation is maintained by the JDO Apache project under the Java community process and released its version 3 in April 2010. JDO has strongly influenced the definition of the Java Persistence API (JPA).

The programming model provides a transparent, easy to use object persistence for standard Java objects including transactional support. For this reason, vendors of object oriented databases quickly adopted JDO as programming interface. However, it is possible to implement the JDO API for any persistent data storage (called *datastore* by the JDO specification), in particular for relational database systems. In contrast to ORM software there is no need to define any transformation to tables. The programmer only needs to specify in an XML file the classes that he wants to make persistence capable and optionally - as an vendor extension - the attributes that should have indexed access (as in example Figure 6). This definition will be later used by the class enhancer to make ordinary Java classes persistence capable and the actual mapping - if applicable - is hidden from the programmer.

While in JDO version 1 the mapping was undefined, version 2 allowed different mappings to relational databases. If one starts with the class definition the relational schema can be generated from the optional XML-mapping file.

The JDO Specification 2.2 [36, pp 44-46] distinguishes three types of identity:

- Application identity - based on attribute values, managed by the application, and enforced by the database,
- Datastore identity - based on a system generated identity and managed by the database,
- Nondurable identity - the Java object identity, managed

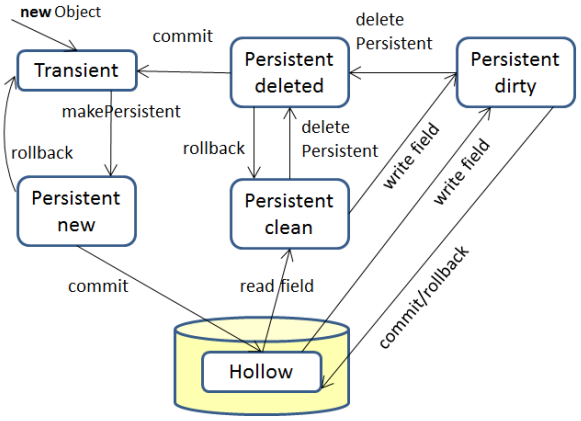


Figure 7. JDO persistence capable object states

by the Java virtual machine.

Only the datastore identity is a persistent identity in the object oriented sense. The representation of the identity is via the JDO object identity that is returned as a copy by method getObjectId(Object po). JDO defines two types of transaction management, the Datastore Transaction Management (DTM) and optimistic transaction management (OTM). DTM ensures transactional properties similar to SQL transactions even if connections to the datastore are dynamically handled by the ConnectionManager. In the case of OTM, persistent objects modified within or outside a transaction are not transactional until during commit. At commit time a short transactional datastore context is established in which all modifications are flushed to the datastore if validation is successful. If an `JDOOptimisticVerificationException` is raised, the transaction fails and in memory modifications are rolled back (the original state is restored). The tested implementation FastObjects J1 does not support the OTM and raised an `JDOUnsupportedOptionException`.

The programming model of JDO version 1 distinguishes between transient and persistent objects. The persistent objects may be in one of five states:

- persistent new - new object prepared to be committed to the datastore
- hollow - object in the datastore
- persistent clean - (partial) object loaded into application
- persistent dirty - (partial) object loaded and modified in application
- persistent deleted - persistent object deleted

The possible states and transitions for transactional objects are shown in Figure 7. Typically an object is created in the application by the Java constructor **new**. The resulting object is *Transient*. If the object's class is persistence capable, calling `makePersitent()` changes its state to *Persistent new*. If the object is committed to the datastore its state is *Hollow*. After reading some fields of an object from the datastore it reaches the state *Persistent clean*. Modifying any attribute value makes the object *Persistent dirty* until it is committed back to the datastore or the values are discarded by the `rollback()` method. In both cases the resulting state is *Hollow* again. A persistent object may be deleted at any time by calling `deletePersistent()`. The deletion is made durable by calling `commit()` or cancelled by `rollback()`.

In JDO version 2, *nontransactional* and *detached* operations were added that are not orthogonal to the above states. This leads to an explosion of states and different life cycles for JDO objects (see JDO 2.2 Specification [36, pp. 50 - 68].

JDO recognizes four isolation levels with identical names as the SQL isolation levels plus the snapshot isolation level. It is unclear if these isolation levels have the same semantics as the isolation levels for relational databases. The tested FastObjects J1 does not support this JDO 2.0 options, but uses strict two-phase-locking that ensures REPEATABLE READ, the isolation level needed for the RVV testing.

The class definition of `RvvEntity` should be stored in the object database as well in order to provide the class definition to all applications. The version attribute `rv` will only be incremented by the setter methods of the other attributes. It is important that all attributes are private and that the `rv` attribute has no setter method. So the only way to access an attribute is via its getter and setter methods. The setter methods ensure that any modification to the object's state, i.e., changing any attribute value, will result in a new row version. The setter method for each attribute should be modified in the following way:

```
package model;
public class RvvEntity {
  private int id;
  private String s;
  private long rv;
  public RvvEntity() {// default constructor
  } //needed for persistence capable classes
  public RvvEntity(int id, String s) {
     this.id = id;
     this.s=s;
     this.rv = 0;
  }
  public void setS(String s) {
     this.s = s;
     this.rv++;
  }
}
```

As in the previous Hibernate examples, only loaded objects can be modified. Therefore, only Pattern C is applicable as the following listing of Phase 4 and 6 shows.

```
PersistenceManager pm;
...
```

```
public void doRVV() throws ExampleException {
  RvvEntity p, p2RVV;
  Object p2oid;
  Extent e;
  Iterator<?> i;

  // Phase 4 of use case
  pm.currentTransaction().begin();
  e = pm.getExtent(RvvEntity.class, true);
  i = com.poet.jdo.Extents.iterator(e, "IdIndex");
  // find object with id = 1
  boolean found = com.poet.jdo.Extents.findKey(i,
1);
  // search for the first key match
  if (found == true && i.hasNext())
    p = (RvvEntity) i.next();
  else {
    System.out.println("RvvEntity #1 not found");
    return;
  }
  p2oid = pm.getObjectId(p); //1)
  long oldRvv = p.getRv(); //2)
  pm.currentTransaction().commit();

  // Phase 6 of use case
  pm.currentTransaction().begin();
  pm.currentTransaction().setIsolationLevel(
  TransactionIsolationLevel.repeatable-read);
  System.out.println("pm: Phase 6 started");
  p2RVV =
  (RvvEntity) pm.getObjectById(p2oid, true); //3)
  if (p2RVV.getRv() == oldRvv){ //4) Pattern C
    p.setS(" pm(Phase 6): NEW TEXT");
    try {
      pm.currentTransaction().commit();
      System.out.println("pm: Phase 6 committed!");
    }
    catch (javax.jdo.JDOException x) {
      pm.currentTransaction().rollback();
      System.out.println("Could not commit! Reason:
"
      + x.getMessage());
    }
  } else {
    System.out.println("pm: RVV serialisation
conflict!");
    pm.currentTransaction().rollback();
  }
}
```

In Phase 4 the RvvEntity #1 is retrieved from the database by its id number. From the programming model the index search could possibly retrieve more than one object with id = 1. It is the responsibility of the application to ensure uniqueness. Later, when the object is loaded, the method `getObjectId()` retrieves its persistent datastore object id (line with comment 1). This object id is later used in Phase 6 to reload the object by its object id (see line with comment 3). The row version from Phase 4 is saved in variable `oldRvv` (comment 2) and used in Phase 6 for RVV (comment 4).

Phase 6 relies on isolation level REPEATABLE READ, but the tested version of FastObjects does not support isolation level setting. From the exception messages we concluded that any read access was protected by a shared lock until the end of the transaction and conflicting implicit lock requests (e.g., by a setter method) resulted in an exception with message:

> write lock for object with id (0:0-1030#0, 1001) on behalf of transaction <unnamed> on database FastObjects://LOCAL/MyRvv_base.j1 not granted

The apparent use of strict two-phase-locking provided reliable isolation for a successful application of the RVV using Pattern C. During Phase 4, no competing transaction could modify the loaded object. Between phases 4 and 6 a concurrent transaction may successfully update objects loaded during Phase 4 but in Phase 6 the changes are discovered and the transaction under test had to abort. During Phase 6 the transaction under test was protected by read locks against any changes from the re-read (line with comment 3) until the end of the transaction.

XI. Conclusion

Table II presents a comparison of the Call-level APIs and ORM Frameworks with RVV practice in mind. It lists the access patterns that can be used in conjunction with different technologies and it depicts the level of control and its limitations. Major findings are the differences when applying the access patterns of Laux and Laiho [11] for different middleware technologies with regard to isolation levels, transaction control, caching, and performance overhead. While we are writing this paper LINQ to SQL is still in its beta phase and it was rather slow in our tests. However, we are impressed about the built-in ”optimistic concurrency control” as Microsoft calls it. Microsoft has the advantage of experiences from the competing technologies. Attributes of LINQ are more orthogonal than the numerous JPA annotations and its object caching did not produce side-effects in concurrency control making LINQ easier to use and manage. It also utilizes server-side version stamping. With the advanced features of the framework - as it proves to do things right - this is a most promising software development extension in the .NET Framework. The native DBMS for LINQ is currently SQL Server, but since IBM and Oracle have already shipped its own ADO.NET data providers, their support for this technology can be expected.

Table II
COMPARISON OF CLI APIS AND ORM FRAMEWORKS

	CLI APIs		**ORM Frameworks**	
	ODBC JDBC ADO.NET	Web Service APIs	Java Hibernate JDO TopLink JPA	.NET LINQ
Access Pattern A	yes	yes	no	no
Access Pattern B	yes	yes	no	yes
Access Pattern C	yes	yes	yes	no
Performance overhead	low	DBMS http: low appl.serv: high	high	high (beta)
Obj. orient. Programming	labor-intensive		yes	yes
Persistence	SQL	SQL	middleware service	middleware service
Use of native SQL	detailed	detailed	limited	limited
– isolation	full control	full control	default	full control
– local transaction	full control	full control	TM 1)	full control
– global transaction	(ADO.NET)	difficult	TM 1)	implicit 2)
2nd level caching			yes	
Optimistic Locking	RVV	RVV	configurable	built-in
Version stamping (default)			client-side	server-side

1) using Transaction Manager (TM), 2) using TransactionScope

As an advantage of ORM Frameworks Hoffer et al [26] lists "Developers do not need detailed understanding of the DBMS or the database schema". We don't share this view. Even if we use these higher abstraction frameworks we need to verify that we understand the low level data access operations so that our applications are transaction safe. For example it was necessary to circumvent middleware caches where possible or when using disconnected data sets we had to explicitly reread the row version from the database in repeatable read isolation (access *Pattern C*). The version stamping of the "optimistic locking" should not be handled at client or middleware side, but on the server side instead to avoid that applications can bypass the RVV mechanism.

The JDO programming interface shielded much of the mapping complexity and the implementation tested used straight forward strict two-phase-locking. So the behaviour was similar to SQL databases with locking scheme. Future tests with products supporting optimistic transaction control and disconnected (called detached by JDO) operations will show if these models can improve performance or facilitate programming.

Some comparisons in Table II are still speculative instead of hard facts. In this respect Table II can be considered as suggestions for further studies.

ACKNOWLEDGEMENTS

This paper is the result of collaborative work undertaken along the lines of the DBTechNet Consortium. The authors participate in DBTech EXT, a project partially funded by the EU LLP Transversal Programme (Project Number: 143371-LLP-1-2008-1-FI-KA3-KA3MP)

REFERENCES

[1] M. Laiho and F. Laux; "Implementing Optimistic Concurrency Control for Persistence Middleware using Row Version Verification," in *Second International Conference on Advances in Databases Knowledge and Data Applications (DBKDA 2010)*, April 11-16, 2010, Les Menuires, France, pp. 45 - 50

[2] T. Härder and A. Reuter, "Principles of Transaction-Oriented Database Recovery," *ACM Computing Surveys*, pp. 287 - 317, Vol. 15, No. 4, December 1983

[3] H. T. Kung and J. T. Robinson; "On Optimistic Methods for Concurrency Control," In *ACM Transactions on Database Systems (TODS)* 6(2), pp. 213 - 226, 1981

[4] M. Crowe; (2011, Jan.), The Pyrrho Database Management System, University of the West of Scotland, [Online], Available: http://www.pyrrhodb.com

[5] VoltDB, Inc. (2011, Jan.), VoltDB Home Page, [Online], Available: http://voltdb.com/

[6] VoltDB, LLC (ed.) (2011, Jan.), VoltDB Technical Overview, [Online], Available: http://www.voltdb.com/_pdf/VoltDBTechnicalOverviewWhitePaper.pdf

[7] G. Weikum and G. Vossen, *Transactional Information Systems*, Morgan Kaufmann Publishers, 2002

[8] C. Nock; *Data Access Patterns*, Addison-Wesley, 2004

[9] J. Gray and A. Reuter; *Transaction Processing: Concepts and Techniques*, Morgan Kaufmann, 1993

[10] U. Halici and A. Dogac; "An Optimistic Locking Technique for Concurrency Control in Distributed Databases," *IEEE Transactions on Software Engineering*, Vol 17, pp. 712 - 724, 1991

[11] F. Laux and M. Laiho; "SQL Access Patterns for Optimistic Concurrency Control," in *Future Computing, Service Computation, Cognitive, Adaptive, Content, Patterns (COMPUTATIONWORLD '09)*, November 15-20, 2009 - Athens/Glyfada, Greece, pp. 254 - 258

[12] D. Z. Badal; "Correctness of Concurrency Control and Implications in Distributed Databases," in *Proc. COMPSAC79*, Chicago, 1979, pp. 588 - 593

[13] R. Unland, U. Prädel, and G. Schlageter; "Ideas on Optimistic Concurrency Control I: On the Integration of Timestamps into Optimistic Concurrency Control Methods and A New Solution for the Starvation Problem in Optimistic Concurrency Control," In *Informatik Fachbericht*; FernUniversität Hagen, Nr. 26. 1982

[14] Ph. Bernstein and E. Newcomer; *Principles of Transaction Processing*, Morgan Kaufmann, 1997

[15] R. E. Stearns and D. J. Rosenkrantz; Distributed Database Concurrency Controls Using Before-Values," in *Proceedings ACM SIGMOD International Conference on Management of Data*, 1981, pp. 74 - 83,

[16] P. A. Bernstein and N. Goodman; "Multiversion Concurrency Control – Theory and Algorithms," *ACM Transactions on Database Systems* **8**, pp. 465 - 483, 1983.

[17] A. Seifert and M. H. Scholl; "A Multi-version Cache Replacement and Prefetching Policy for Hybrid Data Delivery Environments," in *Proceedings of the 28th VLDB Conference*, Hong Kong, China, 2002, pp. 850 - 861

[18] M. Heß; (2010, Oct), Lange Gespräche mit Hibernate, [Online], Available: http://www.ordix.de/ORDIXNews/2_2007/Java_J2EE_JEE/hibernate_long_conversation.html

[19] Y. Akbar-Husain, (2010, Oct.), Optimistic Locking pattern for EJBs, [Online], Available: http://www.javaworld.com/jw-07-2001/jw-0713-optimism.html

[20] L. DeMichiel and M. Keith, *JSR 220: Enterprise JavaBeans™, Version 3.0, Java Persistence API*, Final Release, 8 May, 2006, Sun Microsystems, Inc., Santa Clara, California, USA,

[21] A. Adya, R. Gruber, B. Liskov, and U. Maheshwari; "Efficient Optimistic Concurrency Control Using Loosely Synchronized Clocks," *ACM SIGMOD Record*, Volume 24 , Issue 2 (May 1995), pp 23 - 34, ISSN: 0163-5808

[22] M. Laiho and F. Laux; (2011, Jan.), On Row Version Verifying (RVV) Data Access Discipline for avoiding Blind Overwriting of Data, [Online], Available: http://www.DBTechNet.org/papers/RVV_Paper.pdf

[23] E. Jendrock, J. Ball, D. Carson, I. Evans, S. Fordin, and K. Haase; (2010, Oct.), The Java EE 5 Tutorial, [Online], Available: http://download.oracle.com/javaee/5/tutorial/doc/

[24] N.N.; (2010, Oct.), .NET Framework 3.5, msdn .NET Framework Developer Center, [Online], Available: http://msdn.microsoft.com/ en-us/library/w0x726c2.aspx

[25] M. Laiho and F. Laux; "Data Access using RVV Discipline and Persistence Middleware," in *The Conference for International Synergy in Energy, Environment, Tourism and Contribution of Information Technology in Science, Economy, Society and Education (eRA-3)*, 2008, Aegina/Greece, pp. 189 - 198

[26] J. A. Hoffer, M. B. Prescott, and H. Topi; *Modern Database Management*, 9th ed., Pearson Prentice-Hall, 2009

[27] J. O. Coplien, "A Generative Development-Process Pattern Language," in J.O. Coplien and D.C. Schmidt (eds.), *Pattern Languages of Program Design*, Addison-Wesley, 1995

[28] E Gamma, R Helm, R Johnson, and J Vlissides; *Design Patterns, Elements of Reusable Object-Oriented Software*, Addison-Wesley, 1994

[29] N.N.; (2010, Oct.), SQL Server Books Online, msdn SQL Server Developer Center, [Online], Available: http://msdn.microsoft.com/en-gb/library/ms130214.aspx

[30] N.N.; (2010, Oct.), *SQL Server and ADO.NET*, msdn Visual Studio Developer Center, [Online], Available: http://msdn.microsoft.com/en-us/library/kb9s9ks0.aspx

[31] Diana Lorentz and Mary Beth Roeser; (2011, Jan.), *Oracle SQL Language Reference Manual*, 11g Release 2 (11.1), October 2009, [Online], Available: http://download.oracle.com/docs/cds/E11882_01.zip

[32] Oracle; *TopLink Developers Guide 10g (10.1.3.1.0)*, B28218-01, September 2006

[33] C. Bauer and G. King; *Java Persistence with Hibernate*, Manning, 2007

[34] Gavin King, Christian Bauer, Max Rydahl Andersen, Emmanuel Bernard, and Steve Ebersole; (2010, Oct.), Hibernate Reverence Documentation, Version 3.5.1-Final, [Online], Available: http://docs.jboss.org/hibernate/stable/core/reference/en/pdf/hibernate_ reference.pdf

[35] S. Klein; *Professional LINQ*, Wiley Publishing, 2008

[36] Craig Russell; (2010, Oct.), Java Data Objects 2.2, JSR 243, 10 October 2008, Sun Microsystems, Inc., [Online], Available: http://db.apache.org/jdo/releases/release-2.2.cgi

Efficient Maintenance of all k-Dominant Skyline Query Results for Frequently Updated Database

Md. Anisuzzaman Siddique*
University of Rajshahi
Rajshahi-6205, Bangladesh
Email: anis_cst@yahoo.com

Yasuhiko Morimoto
Hiroshima University
Kagamiyama 1-7-1, Higashi-Hiroshima 739-8521, Japan
Email: morimoto@mis.hiroshima-u.ac.jp

***Abstract*—Skyline queries are useful to multi-criteria decision making as they represent the set of all solutions that the user can safely take without fear that something better is out there. It can act as a filter to discard sub-optimal objects. However, a major drawback of skylines is that, in datasets with many dimensions, the number of skyline objects becomes large and no longer offer any interesting insights. To solve the problem, k-dominant skyline queries have been introduced, which can reduce the number of retrieved objects by relaxing the definition of the dominance. Though it can reduce the number of retrieved objects, the k-dominant skyline objects are difficult to maintain if the database is updated. This paper addresses the problem of maintaining k-dominant skyline objects for frequently updated database. We propose an algorithm for maintaining k-dominant skyline objects. An extensive performance evaluation using both real and synthetic datasets demonstrated that our method is efficient and scalable.**

***Keywords*-Skyline, k-Dominant Skyline, Database Update.**

I. INTRODUCTION

Abundance of data has been both a boon and a curse, as it has become increasingly difficult to process data in order to isolate useful and relevant information. In order to compensate, the research community has invested considerable effort into developing tools that facilitate the exploration of a data space. One such successful tool is the skyline query. The set of skyline objects presents a scale-free choice of data objects worthy for further considerations in many application contexts. Figure 1 shows a typical example of skyline. The table in the figure is a list of hotels, each of which contains two numerical attributes: distance to beach and price, for online booking. A tourist chooses a hotel from the list according to her/his preference. In this situation, her/his choice usually comes from the hotels in skyline, i.e., any of h_1, h_3, h_4 (see Figure 1(b)). Many approaches have been proposed for the efficient computation of skylines in the literature [2], [3], [5], [6], [8], [9], [10], [12].

The skyline query can greatly help user to narrow down the search range. It is always assumed that all the attributes are involved in the skyline queries, that is, the dominating relationship is evaluated based on every dimensions of the dataset. However, a major drawback of skylines is that, in datasets with many dimensions, the number of skyline objects becomes large and no longer offer any interesting insights. The reason is that as the number of dimensions increases, for any object O_1, it is more likely there exists another object O_2 where O_1 and O_2 are better than each other over different subsets of dimensions. If our tourist, cared not just about price and distance to beach, but also about the distance to airport, distance to downtown, rank, and internet charge then most hotels may have to be included in the skyline answer since for each hotel there may be no one hotel that beats it on all criteria.

To deal with this dimensionality curse, one possibility is to reduce the number of dimensions considered. However, which dimensions to retain is not easy to determine, and requires intimate knowledge of the application domain. To reduce the number of dimensions without any intimate knowledge of the application domain, Chan, *et al.* considered k-dominant skyline query [4]. They relaxed the definition of "dominated" so that an object is more likely to be dominated by another. Given an n-dimensional database, an object O_i is said to k-dominates another object O_j $(i \neq j)$ if there are k $(k \leq n)$ dimensions in which O_i is better than or equal to O_j. A k-dominant skyline object is an object that is not k-dominated by any other objects. Note that conventional skyline objects are n-dominant objects.

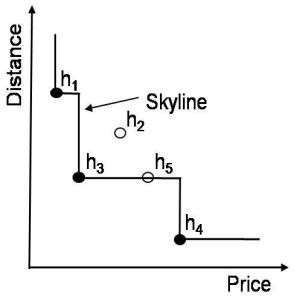
A. Motivating Example

Assume we have a symbolic dataset containing six attributes $(D_1, ..., D_6)$ as listed in Table I. Without loss of generality, we assume smaller value is better in each dimension. Conventional skyline query for this database returns five objects: O_2, O_3, O_5, O_6, and O_7. On the other, if we apply the k-dominant skyline query for this dataset it can control the selectivity by changing k. For example, if $k = 5$, the 5-dominant skyline query returns two objects: O_5 and O_7. Objects O_1, O_2, O_3, O_4, and O_6 are not in 5-

*This work was done when the author was in Hiroshima University.

ID	Price	Distance
h_1	3	8
h_2	5	6
h_3	4	4
h_4	9	2
h_5	7	4

(a) Hotels



(b) Skyline

Figure 1. Skyline example

Table I
SYMBOLIC DATASET

$Obj.$	D_1	D_2	D_3	D_4	D_5	D_6
O_1	7	3	5	4	4	3
O_2	3	4	4	5	1	3
O_3	4	3	2	3	5	4
O_4	5	3	5	4	1	2
O_5	1	4	1	1	3	4
O_6	5	3	4	5	1	5
O_7	1	2	5	3	1	2

dominant skyline because they are 5-dominated by O_7. The 4-dominant skyline query returns only one object, O_7 and the 3-dominant skyline query returns empty.

Though the k-dominant skyline query can control selectivity, but unfortunately, there exist no efficient methods that can handle k-dominant skyline queries under continuous updates. Therefore, we consider continuous queries, that is, queries that are executed when there are some changes in the dataset that affects their result. Continuous k-dominant skyline queries are very useful, since they allow users to monitor the dataset and get informed about new interesting objects. However, the maintenance of k-dominant skyline for an update is much more difficult due to the well-known intransitivity of the k-dominance relation. Assume that "A" k-dominates "B" and "B" k-dominates "C". However, "A" does not always k-dominates "C". Moreover, "C" may k-dominate "A". Because of the intransitivity property, we have to compare each object against every other object to check the k-dominance. To illustrate this problem consider the 5-dominant example again. In the dataset, objects O_5, O_7 are in 5-dominant skyline. If we insert a new object $O_{new} = (6, 4, 4, 2, 2, 3)$ into the dataset, we can compare O_{new} with the 5-dominant skyline objects (i.e., $\{O_5, O_7\}$) to maintain the 5-dominant skyline and after comparisons we may find that O_{new} is in the 5-dominant skyline. But it is not true, because O_{new} is 5-dominated by O_2. Like this example, for each insertion in the dataset, we have to perform domination check of each new object against all k-dominant as well as non-k-dominant skyline objects. This procedure is cost effective and we have to reduce the cost in order to handle frequent updates in a large dataset.

If an update is a deletion, we have to recompute the entire k-dominant skyline from the scratch. Because some objects that are not in the current k-dominant skyline objects may be "promoted" as k-dominant skyline objects by a deletion. Suppose, if we delete object O_2, this deletion will "promote" O_{new} as a 5-dominant skyline object. Again, if we want to revise or update the attributes values of an object. Then similar to deletion operation, we have to recompute the entire k-dominant skyline from the scratch. After this type of modification, three cases can be happened: some k-dominant skyline objects may be "removed", some objects that are not in the current k-dominant skyline objects may be "promoted" as k-dominant skyline objects, and both may be occurred at a time. For example, if we select object O_7 for update and revise the values of D_1 and D_5 from 1 to 6, then the 5-dominant skyline result updated as $\{O_5, O_4, O_7, O_2\}$. The focus of this paper is on developing an efficient algorithm for continuous k-dominant skyline objects.

This paper is the journal version of our paper [1]. The main contributions of this paper are as follows:

- We propose an algorithm to compute k-dominant skyline objects for all k at a time.

- We addresses the problem of continuous k-dominant skyline objects of frequently updated database.

- We develop algorithms for continuous k-dominant skyline objects.

- We conduct the extensive performance evaluation using both real and synthetic datasets and compare our method with the adaptive version of Two-Scan Algorithm (TSA) technique [4], which is currently considered the most efficient k-dominant skyline method. Our evaluation shows that the proposed method is significantly faster than the adaptive version of TSA technique.

The remaining sections of this paper are organized as follows: Section II presents the notions and properties of k-dominant skyline computation, we provide detailed examples and analysis of proposed k-dominant skyline computation and maintenance methods in Section III, Section IV discusses related work, we experimentally evaluate our algorithm in Section V by comparing with other existing algorithms under a variety of settings, finally, Section VI concludes the paper.

II. Problem Definition

Assume there is an n-dimensional database DB and D_1, D_2, $\cdots$, D_n be the n attributes of DB. Let O_1, O_2, $\cdots$, O_r be r objects (tuples) of DB. We use $O_i.D_s$ to denote the s-th dimension value of O_i. An object $O_i \in DB$ is said to dominate another object $O_j \in DB$, denoted as $O_i \prec O_j$, if (1) for every $s \in \{1, \cdots, n\}$: $O_i.D_s \leq O_j.D_s$; and (2) for at least one $t \in \{1, \cdots, n\}$: $O_i.D_t < O_j.D_t$. The skyline of DB is a set of objects $Sky_n(DB) \subseteq DB$ which are not dominated by any other objects. Skyline query for Table I dataset returns five objects: O_2, O_3, O_5, O_6, and O_7. Objects O_1 and O_4 are not in skyline because they are dominated by O_7.

In the remaining sections, we first define the k-dominance relationship using above notation, then introduce k-dominant skyline based on the definition of the k-dominance relationship.

A. k-dominance Relationship

An object O_i is said to *dominate* another object O_j, which we denote as $O_i \prec O_j$, if $O_i.D_s \leq O_j.D_s$ for all dimensions D_s ($s = 1, \cdots, n$) and $O_i.D_t < O_j.D_t$ for at least one dimension D_t ($1 \leq t \leq n$). We call such O_i as *dominant object* and such O_j as *dominated object* between O_i and O_j.

By contrast, an object O_i is said to *k-dominate* another object O_j, denoted as $O_i \prec_k O_j$, if (1) $O_i.D_s \leq O_j.D_s$ in k dimensions among n dimensions and (2) $O_i.D_t < O_j.D_t$ in one dimension among the k dimensions. We call such O_i as *k-dominant object* and such O_j as *k-dominated object* between O_i and O_j.

An object O_i is said to have *δ-domination power* if there are δ dimensions in which O_i is better than or equal to all other objects of DB.

B. k-dominant Skyline

An object $O_i \in DB$ is said to be a *skyline object* of DB if O_i is not dominated by any other object in DB. Similarly, an object $O_i \in DB$ is said to be a *k-dominant skyline object* of DB if O_i is not k-dominated by any other object in DB. We denote a set of all k-dominant skyline objects in DB as $Sky_k(DB)$.

Theorem 1: *Any object in $Sky_{k-1}(DB)$ must be an object in $Sky_k(DB)$ for any k such that $1 < k \leq n$. Any object that is not in $Sky_k(DB)$ cannot be an object in $Sky_{k-1}(DB)$ for any k such that $1 < k \leq n$.*

Proof: Based on the definition, a $(k-1)$-dominant skyline object O_i is not $(k-1)$-dominated by any other objects in DB. It implies that O_i is not k-dominated by any other objects. Therefore, we can say that O_i is a k-dominant skyline object. Similarly, if an object O_j is k-dominated by another object, it must be $(k-1)$-dominated by the object. Therefore, any k-dominated object cannot be a $(k-1)$-dominant skyline object. $\diamondsuit$

The conventional skyline is the k-dominant skyline where $k = n$. If we decrease k, more objects tend to be k-dominated by other objects. As a result, we can reduce the number of k-dominant skyline objects. Using above properties, we can compute $Sky_{k-1}(DB)$ from $Sky_k(DB)$ efficiently. For example, O_1 and O_4 of Table I are not in $Sky_6(DB)$ because they are 6-dominated by O_7. Therefore, they cannot be a candidate of k-dominant skyline object for $k < 6$. We can prune such non-skyline objects for further procedure of the k-dominant query. If we consider 5-dominant query, then O_2, O_3, and O_6 are 5-dominated objects. Therefore, we can prune all of those five objects in 4-dominant query computation. Thus, by decreasing k, more dominated objects can be pruned away.

Theorem 2: *Every object that belongs to the k-dominant skyline also belongs to the skyline, i.e., $Sky_k(DB) \subseteq Sky_n(DB)$.*

Proof: Immediate from theorem 1. $\diamondsuit$

III. k-dominant Skyline Computation and Maintenance

In this section, we present our algorithm for computing k-dominant skyline objects and how to maintain the result when update occurs. We illustrate how to compute k-dominant skyline for all k at a time in section III-A. Next in section III-B, we present three types of maintenance solution. They are insertion, deletion, and update operation.

A. Algorithm for k-dominant Skyline

Chan, *et al.* sort the whole objects with a monotonic scoring function sum in their One-Scan Algorithm (OSA), algorithm for k-dominant query [4]. By using the ordered objects, we can eliminate some of non-skyline objects easily. However, this ordered objects is not effective for k-dominant query computation, especially, when values of each attribute is not normalized. For example, assume $O_i = (1, 2, 3, 3, 3, 2)$ and $O_j = (7, 1, 3, 2, 3, 1)$ are two objects in 6-dimensional space. Although sum of O_i's values is smaller than that of O_j's, O_i does not 5-dominant of O_j. Instead, O_i is 5-dominated by O_j.

In order to prune unnecessary objects efficiently in the k-dominant skyline computation, we compute *domination power* of each object. Algorithm 1 represents the domination power calculation procedure. We sort objects in descending order by domination power. If more than one objects have the same domination power then sort those objects in ascending order of the sum value. This order reflects how likely to k-dominate other objects.

Algorithm 1: *Compute Domination Power, DP*

```
01. for each object O_i(i = 1 ··· r) do
02.     O_i.DP := 0 (initialize DP for each object)
03. for each attribute D_s(s = 1 ··· n) do
04.     minValue := O_1.D_s
05.     for each object O_i(i = 2 ··· r) do
06.       if (O_i.D_s < minValue) then
07.         minValue := O_i.D_s
08.     for each object O_i(i = 1 ··· r) do
09.         if (minValue == O_i.D_s) then
10.           O_i.DP := O_i.DP + 1 (Increment DP)
11. Sort dataset, DB, in descending order by DP and Sum
```

Table II is the sorted object sequence of Table I, in which the column "DP" is the domination power and the column "Sum" is the sum of all values. In the sequence, object O_7 has the highest domination power 4. Note that object O_7 dominates all objects lie below it in four attributes, D_1, D_2, D_5, and D_6.

After computing the sorted object sequence, we compute dominated counter (DC) and dominant index (IDX) by using the algorithm 2. The dominated counter (DC) indicates the maximum number of dominated dimensions by another object in DB. The dominant index (IDX) is the strongest dominator. That means IDX keeps the record of the corresponding strongest dominator for each object.

For example, O_7 is 3-dominated by O_5 and there exist no other object which can 4-dominate O_7. In the algorithm (8-13), we count the number of dominated dimensions for each pair of objects. In the algorithm, we denote the i-th object in the sorted sequence as O_i. During the procedure, we keep the max value and its dominator object in DC and IDX for each object.

The column DC and IDX of Table II shows the result of the procedure. $Sky_k(DB)$ is a set of objects whose DC is less than k. According to the dominated counter, we can see that $Sky_6(DB)$ = $\{O_7, O_5, O_2, O_6, O_3\}$, $Sky_5(DB)$ = $\{O_7, O_5\}$, and $Sky_4(DB)$ = $\{O_7\}$. Since there is no object whose DC value is less than 3, thus $Sky_3(DB)$ = $\{\emptyset\}$.

B. k-dominant Skyline Maintenance

In this section, we discuss the maintenance problem of $Sky_k(DB)$ after an update is occurred in DB. In the maintenance phase, we keep a vector that contains the minimal value for each dimension, which we call the minimal vector. The minimal vector of Table II is $\{1, 2, 1, 1, 1, 2\}$. We also keep an inverted index of the dominant index column (IDX). Table III is the inverted index of Table II. We use a multi-hash data structure for the records in the inverted index so that we can look up efficiently.

Lemma 1: Assume $O_1.D_s \leq O_2.D_s$ for all dimensions D_s $(s = 1, \cdots, n)$ for $O_1, O_2 \in DB$. If O_1 is not deleted

Algorithm 2: *Compute DC and IDX*

```
01. for each object O_i (i := 1 ··· r)
02.     O_i.DC := 0 (initialize DC)
03. for each i := 1 ··· r
04.     if O_i.DC == n then
05.       skip the i-th and continue
06.     for each j := i+1 ··· r
07.         if O_j.DC == n then
08.           skip the j-th and continue
09.         k_i := 0; k_j := 0
10.         for each attribute D_s(s := 1 ··· n)
11.           if O_i.D_s ≤ O_j.D_s then
12.             k_j ++
13.           if O_j.D_s ≤ O_i.D_s then
14.             k_i ++
15.         if k_j > O_i.DC then
16.           O_i.DC := k_j and O_i.IDX := O_j
17.         if k_i > O_j.DC then
18.           O_j.DC := k_i and O_j.IDX := O_i
```

Table II
ORDERED DOMINATION TABLE

Obj.	D_1	D_2	D_3	D_4	D_5	D_6	DP	Sum	DC	IDX
O_7	$\frac{1}{}$	$\frac{2}{}$	5	3	$\frac{1}{}$	$\frac{2}{}$	4	14	3	O_5
O_5	$\frac{1}{}$	4	$\frac{1}{}$	$\frac{1}{}$	3	4	3	14	4	O_7
O_4	$\frac{5}{}$	3	$\frac{5}{}$	$\frac{4}{}$	$\frac{1}{}$	$\frac{2}{}$	2	20	6	O_7
O_2	3	4	4	5	$\frac{1}{}$	3	1	20	5	O_7
O_6	5	3	4	5	$\frac{1}{}$	5	1	23	5	O_7
O_3	4	3	2	3	$\frac{5}{}$	4	0	21	5	O_7
O_1	7	3	5	4	4	3	0	26	6	O_7

Algorithm 3: *Insertion Procedure*

```
01. O_I.DC := 0 (initialize DC for inserted object O_I)
02. for each i := 1 ··· r
03.     if O_i.DC == n then
04.       skip the i-th and continue
05.     k_i := 0; k_I := 0
06.     for each attribute D_s(s := 1 ··· n)
07.       if O_i.D_s ≤ O_I.D_s then
08.         k_I ++
09.       if O_I.D_s ≤ O_i.D_s then
10.         k_i ++
11.     if k_i > O_i.DC then
12.       O_I.DC := k_i and O_I.IDX := O_i
    (update inverted index)
13.   if O_I.DC == n then
14.       break loop
15.   if k_I > O_i.DC then
16.       O_i.DC := k_I and O_i.IDX := O_I
    (update inverted index)
```

Table III
INVERTED INDEX

Obj.	dominated
O_5	O_7
O_7	O_5, O_4, O_2, O_6, O_3, O_1

from DB, O_2 will never be in the k-dominant skyline of DB.

Proof: Since O_2 is n-dominated by O_1, it will also k-dominated by O_1 because $(k \leq n)$. Therefore, while O_2 is in the DB, there will be at least one object, namely O_1, that k-dominate it and consequently cannot become a k-dominant skyline. $\diamondsuit$

The lemma implies that only $Sky_n(DB)$ objects are relevant for the k-dominant skyline maintenance task, since according to theorem 1 other objects can never become part of $Sky_k(DB)$. Because, they are already n-dominated by $Sky_n(DB)$ objects.

Insertion

Assume O_I is inserted into DB. We maintain $Sky_k(DB)$ by using algorithm 3. In the algorithm, we examine the dominated counter (DC) by scanning the sorted object sequence. If the DC value of an object is updated, we also update the inverted index. According to theorem 2, any data object added to k-dominant skyline during the execution of the algorithm is guaranteed to be a skyline object. Thus, during the update procedure, if O_I is n-dominated by another object, then we can stop the procedure immediately.

After updating DC and IDX, we insert O_I in the sorted object sequence. We use a skip list data structure for the sequence so that we can insert efficiently.

Assume we insert $O_8 = (6, 6, 6, 6, 6, 6)$ in the running example. By comparing with the first object O_7, we can find that O_8 is 6 dominated by O_7. Therefore, we immediately complete the update of DC and IDX. Then, we insert O_8 into the last position of the sorted sequence. Next, we insert $O_9 = (3, 4, 4, 2, 2, 3)$. O_9 is not 6 dominated by any object and we can find that the strongest dominator of O_9 is O_2. Then, DC and IDX are updated as in Table IV (left) after inserting O_9. Next, we insert $O_{10} = (2, 2, 3, 1, 2, 3)$. O_{10} is not 6 dominated by any object and the strongest dominator of O_{10} is O_7. Moreover, O_{10} becomes the strongest dominator of O_9. Then, DC and IDX are updated as in Table IV (right) after inserting O_{10}.

Deletion

Assume O_D is deleted from DB. We check the inverted index to examine whether there is O_D's record in the index.

Note that in Table III "Obj." column in each record is the strongest dominator for objects in the "dominated" column.

Table IV
ORDERED DOMINATION TABLE AFTER INSERTIONS

Obj.	DP	Sum	DC	IDX
O_7	4	14	3	O_5
O_5	3	14	4	O_7
O_4	2	20	6	O_7
O_2	1	20	5	O_7
O_6	1	23	5	O_7
O_9	0	18	5	O_2
O_3	0	21	5	O_7
O_1	0	26	6	O_7
O_8	0	36	6	O_7

Obj.	DP	Sum	DC	IDX
O_7	4	14	3	O_5
O_5	3	14	4	O_7
O_{10}	2	13	4	O_7
O_4	2	20	6	O_7
O_2	1	20	5	O_7
O_6	1	23	5	O_7
O_9	0	18	6	O_{10}
O_3	0	21	5	O_7
O_1	0	26	6	O_7
O_8	0	36	6	O_7

Algorithm 4: *Deletion Procedure*

```
1. for each O_D do
2.   if O_D ∉ IDX
3.     no change in Dominant Counter
4.     break loop
5.   else
6.       for each affected objects do
7.         Compute DC and IDX by using algorithm 2
```

Therefore, if there is no O_D's record in the inverted index, we do not have to change the dominated counter (DC) for other objects. Otherwise, we initialize DC for each object in the O_D's index record. Then, we perform the pairwise comparison like in algorithm 2. In this case, we do not have to examine DC for objects that are not in the O_D's index record and therefore it is not costly.

Consider the running example again. Assume we delete O_{10}. We examine DC of O_9 by scanning Table IV (right). The updated result is Table IV (left).

Update

In this section, we propose maintenance solution for update operation. Although one can handle update operation with consecutive deletion and insertion. However, single update operation is usually faster than consecutive delete and insert operation. Assume O_U is updated from DB. Then, same as delete, we have to check the inverted index to examine whether there is O_U's record in the index. If there is no O_U's record in the inverted index, then we need one scan to revise the dominated counter (DC) for updated object as well as all other data objects. Otherwise, we initialize DC for each object from scratch. However, in this situation we argue that compare with non-IDX objects, the number of IDX objects is very few and therefore update operation also not very costly.

Consider the Table II and select a non-IDX object such as O_3 for update. Assume we update D_5 value of this object from 5 to 1. Then after dominance checking with other objects we find that O_3 becomes the strongest dominator of object O_6. This update procedure is shown in Table V (left). Again from Table II if we select an IDX object such as O_7. In this case, we update the values of D_1 and D_5

Table V
ORDERED DOMINATION TABLE AFTER UPDATES

Obj.	DP	Sum	DC	IDX
O_7	4	14	3	O_5
O_5	3	14	4	O_7
O_4	2	20	6	O_7
O_3	1	17	5	O_7
O_2	1	20	5	O_7
O_6	1	23	6	O_3
O_1	0	26	6	O_7

Obj.	DP	Sum	DC	IDX
O_5	3	14	3	O_4
O_4	2	20	4	O_7
O_7	2	24	4	O_5
O_2	1	20	4	O_5
O_6	1	23	5	O_4
O_3	0	21	5	O_5
O_1	0	26	6	O_4

from 1 to 6. The revised result after this update is shown in Table V (right).

IV. RELATED WORK

Our work is motivated by previous studies of skyline query processing as well as k-dominant skyline query processing, which are reviewed in this section. Section IV-A and IV-B, respectively discuss about the related work on skyline query processing and k-dominant skyline query processing.

A. Skyline Query Processing

Borzsonyi, *et al.* first introduce the skyline operator over large databases and proposed three algorithms: $Block\text{-}Nested\text{-}Loops(BNL)$, $Divide\text{-}and\text{-}Conquer(D\&C)$, and B-tree-based schemes [2]. BNL compares each object of the database with every other object, and reports it as a result only if any other object does not dominate it. A window W is allocated in main memory, and the input relation is sequentially scanned. In this way, a block of skyline objects is produced in every iteration. In case the window saturates, a temporary file is used to store objects that cannot be placed in W. This file is used as the input to the next pass. $D\&C$ divides the dataset into several partitions such that each partition can fit into memory. Skyline objects for each individual partition are then computed by a main-memory skyline algorithm. The final skyline is obtained by merging the skyline objects for each partition. Chomicki, *et al.* improved BNL by presorting, they proposed $Sort\text{-}Filter\text{-}Skyline(SFS)$ as a variant of BNL [3]. SFS requires the dataset to be pre-sorted according to some monotone scoring function. Since the order of the objects can guarantee that no object can dominate objects before it in the order, the comparisons of tuples are simplified.

Among index-based methods, Tan, *et al.* proposed two progressive skyline computing methods Bitmap and Index [13]. Both of them require preprocessing. In the Bitmap approach, every dimension value of an object is represented by a few bits. By applying bit-wise and operation on these vectors, a given object can be checked if it is in the skyline without referring to other objects. The index method organizes a set of d-dimensional objects into d lists such that an object O is assigned to list i if and only if its value at attribute i is the best among all attributes of O. Each list is indexed by a B-tree, and the skyline is computed by scanning the B-tree until an object that dominates the remaining entries in the B-trees is found. Kossmann, *et al.* observed that the skyline problem is closely related to the nearest neighbor (NN) search problem [8]. They proposed an algorithm that returns skyline objects progressively by applying nearest neighbor search on an R*-tree indexed dataset recursively. The current most efficient method is $Branch\text{-}and\text{-}Bound\ Skyline(BBS)$, proposed by Papadias, *et al.*, which is a progressive algorithm based on the best-first nearest neighbor (BF-NN) algorithm [10]. Instead of searching for nearest neighbor repeatedly, it directly prunes using the R*-tree structure. Balke, *et al.* show how to efficiently perform distributed skyline queries and thus essentially extend the expressiveness of querying current Web information systems [14]. Kapoor studies the problem of dynamically maintaining an effective data structure for an incremental skyline computation in a 2-dimensional space [15]. Tao and Papadias studied sliding window skylines, focusing on data streaming environments [16]. Huang, *et al.* studied continuous skyline queries for dynamic datasets [17].

B. k-dominant Skyline Query Processing

Chan, *et al.* introduce k-dominant skyline query [4]. They proposed three algorithms, namely, One-Scan Algorithm (OSA), Two-Scan Algorithm (TSA), and Sorted Retrieval Algorithm (SRA). OSA uses the property that a k-dominant skyline objects cannot be worse than any skyline object on more than k dimensions. This algorithm maintains the skyline objects in a buffer during the scan of the dataset and uses them to prune away objects that are k-dominated. TSA retrieves a candidate set of dominant skyline objects in the first scan by comparing every object with a set of candidates. The second scan verifies whether these objects are truly dominant skyline objects or not. This method turns out to be much more efficient than the one-scan method. A theoretical analysis is provided to show the reason for its superiority. The third algorithm, SRA is motivated by the rank aggregation algorithm proposed by Fagin, *et al.*, which pre-sorts data objects separately according to each dimension and then merges these ranked lists [7].

Another study on computing k-dominant skyline is $k\text{-}ZSearch$ proposed by Lee, *et al.* [18]. They introduced a concept called filter-and-reexamine approach. In the filtering phase, it removes all k-dominant objects and retain possible skyline candidates, which may contain false hits. In the reexamination phase, all candidates are reexamined to eliminate false hits.

For any static dataset in case of insertions and deletions the k-dominant skyline result should be updated accordingly. But in a dynamic dataset insertions and deletions are very frequent and the above schemes [4], [18] are not efficient to solve the frequent update problem. Because they need to recompute k-dominant skyline result from scratch. On the other hand, algorithms developed for skyline maintenance are not easily adapted for the maintenance of k-dominant

skyline, except for the obvious case where $k = n$. This is because existing skyline computation algorithms do not satisfy the requirement of k-dominant skyline computation. Moreover, they can not compute k-dominant skyline for all k at a time. To overcome frequent update problem, our proposed method introduce two new concepts $dominated\ counter$ and $inverted\ index$. From the discussion and experimental results it has been seen that those are very helpful to retrieve the k-dominant skyline query result efficiently.

Recently, more aspects of skyline computation have been explored. Vlachou, *et al.* introduce the concept of extended skyline set, which contains all data elements that are necessary to answer a skyline query in any arbitrary subspace [19]. Fotiadou, *et al.* mention about the efficient computation of extended skylines using bitmaps in [20]. Chan, *et al.* introduce the concept of *skyline frequency* to facilitate skyline retrieval in high-dimensional spaces [5]. Tao, *et al.* discuss skyline queries in arbitrary subspaces [11]. There exist more work addressing *spatial skyline* [21], [22], skylines on partially-ordered attributes [6], *dada cube* for analysis of dominance relationships [23], *probabilistic skyline* [24], skyline search over small domains [9], and *reverse skyline* [25].

V. Performance Evaluation

We conduct a series of experiments to evaluate the effectiveness and efficiency of our proposed methods. In lack of techniques dealing directly with the problem of maintaining k-dominant skyline in this paper, we compare our methods against TSA, which was the most efficient k-dominant skyline search algorithm proposed in Ref. 4). To handle updates, we adapt a variant of the TSA called ATSA (Adaptive Two-Scan Algorithm). Let r be the total number of objects in DB. ATSA takes $O(r^2)$ to compute all k-dominant objects from scratch. If an object is inserted in the DB, ATSA has to perform k-domination check of the inserted object against all objects. Therefore, for each insertion ATSA takes $O(r)$. If an object is deleted from the DB, ATSA has to recompute entire k-dominant skyline objects because some objects that are not in the current k-dominant skyline objects may be "promoted" as k-dominant skyline objects. Therefore, for each deletion ATSA requires $O(r^2)$ time. Moreover, for each update it also requires $O(r^2)$ time for the recomputation of k-dominant skyline.

Though the time complexity of our proposed method is substantially the same, we can drastically reduce comparisons for k-dominant skyline computation. For each new insertion, the time complexity of proposed method varies in between $O(1)$ and $O(r)$ to perform k-domination check. For each deletion, if the deleted object is not in the dominant objects list then the proposed method takes $O(1)$. Otherwise, if we assume the number of dominant objects is x, then it takes $O(x^2)$. We can expect that x is much smaller than r. Finally for each update, if the updated object is not in the dominant objects list then the proposed method takes $O(r)$. Otherwise, it takes $O(r^2)$. However, the number of dominant objects is not large. So it is not costly. From the above analysis we understand that the recomputation of k-dominant skyline is not efficient than proposed maintenance solutions. The results of all experiments support our claim that using proposed techniques we can reduce the number of comparisons drastically.

We conduct simulation experiments on a PC running on MS Windows XP professional. The PC has an Intel(R) Core2 Duo 2GHz CPU and 3GB main memory. All experiments are coded in Java J2SE V6.0. Each experiment is repeated five times and the average result is considered for performance evaluation.

A. Performance on Synthetic Datasets

As benchmark synthetic datasets, we use the datasets proposed in Ref. 2). Objects are generated using one of the following three value distributions:

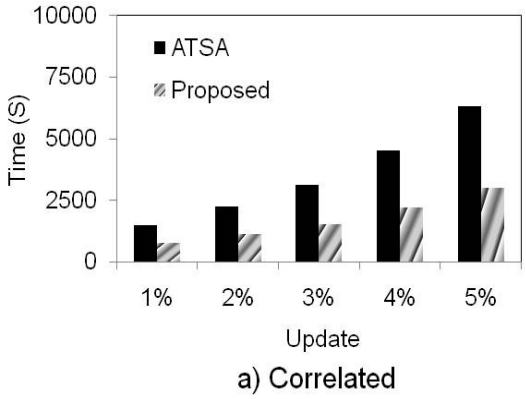
Anti-Correlated: an anti-correlated dataset represents an environment in which, if an object has a small coordinate on some dimension, it tends to have a large coordinate on at least another dimension. As a result, the total number of non-dominating objects of an anti-correlated dataset is typically quite large.

Correlated: a correlated dataset represents an environment in which objects with large coordinate in one dimension are also have large coordinate in the other dimensions. In a correlated dataset, few objects dominate many other objects.

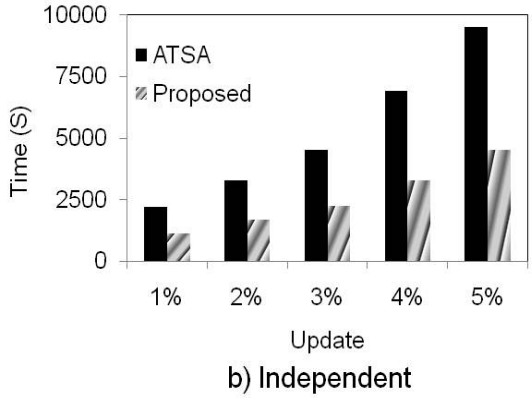
Independent: for this type of dataset, all attribute values are generated independently using uniform distribution. Under this distribution, the total number of non-dominating objects in between that of the correlated and the anti-correlated datasets.

Details of the three distributions can be found in Ref. 1). The generation of the synthetic datasets is controlled by three parameters, n, "Size", and "Dist", where n is the number of attributes, "Size" is the total number of objects in the dataset, and "Dist" can be the any of the three distribution. In addition, we have generated smaller synthetic datasets for all insertion experiments. For example, to conduct insertion experiment on 100k synthetic dataset, we have also generated additional 10k dataset. As for deletion and update experiments, we choose the deleted/updated objects randomly from the experimental dataset.

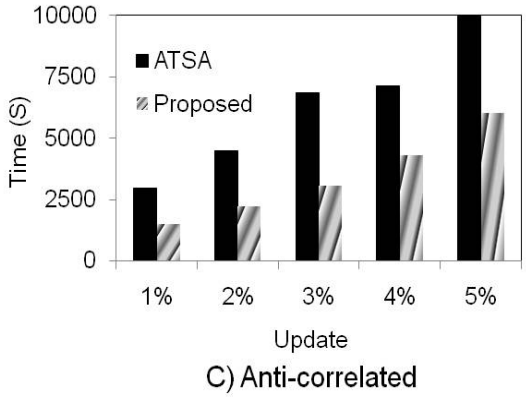
1) Effect of Data Distribution: We first study the effect of data distributions on our techniques. Anti-correlated, independent, and correlated datasets with dimensionality n to 7, cardinality to 100k, and k to 6. Figure 2(a), (b), and (c) shows the time to maintain k-dominant skyline for update ranges from 1% to 5%. In the update experiments, for 100k dataset 1% update implies that we have altered 1000 data objects. Those data objects are randomly selected. However, among 1000 alterations we make sure that there



a) Correlated

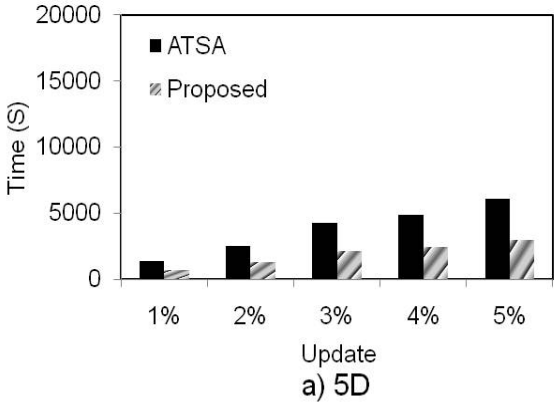


b) Independent

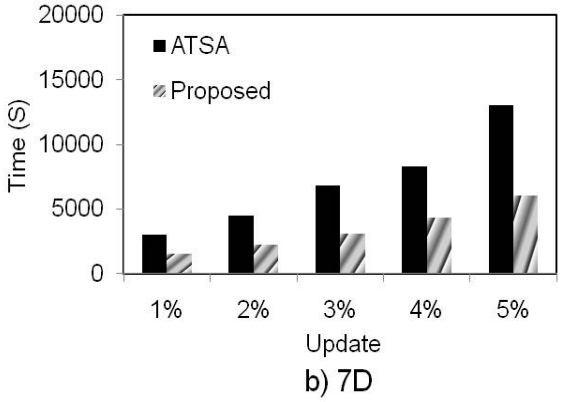


C) Anti-correlated

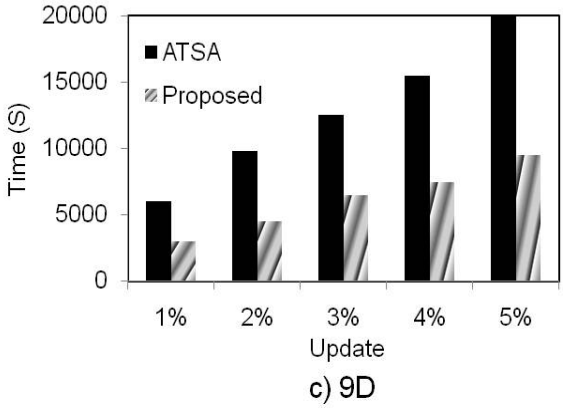
Figure 2. Update performance for different data distributions



a) 5D



b) 7D



c) 9D

Figure 3. Update Performance for different dimensions

are 333 insertions, 333 deletions, and 334 updates (total 1k updates) occurred in the dataset. Figure 2 shows that the performance of both methods deteriorates significantly with the increase value of update size. We further notice that for anti-correlated dataset, many k-dominant skyline objects are retrieved, as a result the maintenance cost of this distribution incurs high computational overheads. On the other hand, for correlated dataset, few objects are retrieved, as a result the maintenance cost of this distribution incurs low computational overheads.

2) Effect of Dimensionality: For this experiment, we use the anti-correlated datasets. We fix the data cardinality to 100k and vary dataset dimensionality n ranges from 5 to 9 and k from 4 to 8. Figure 3(a), (b), and (c) shows the update performance. The ATSA technique is highly affected by the curse of dimensionality, i.e., as the space becomes sparser its pruning power rapidly decreases. The proposed technique also affected but to a lesser degree. The Figure 3 shows that if the dimensionality and the update ratio increase the time grows steadily, which is much less than that of ATSA.

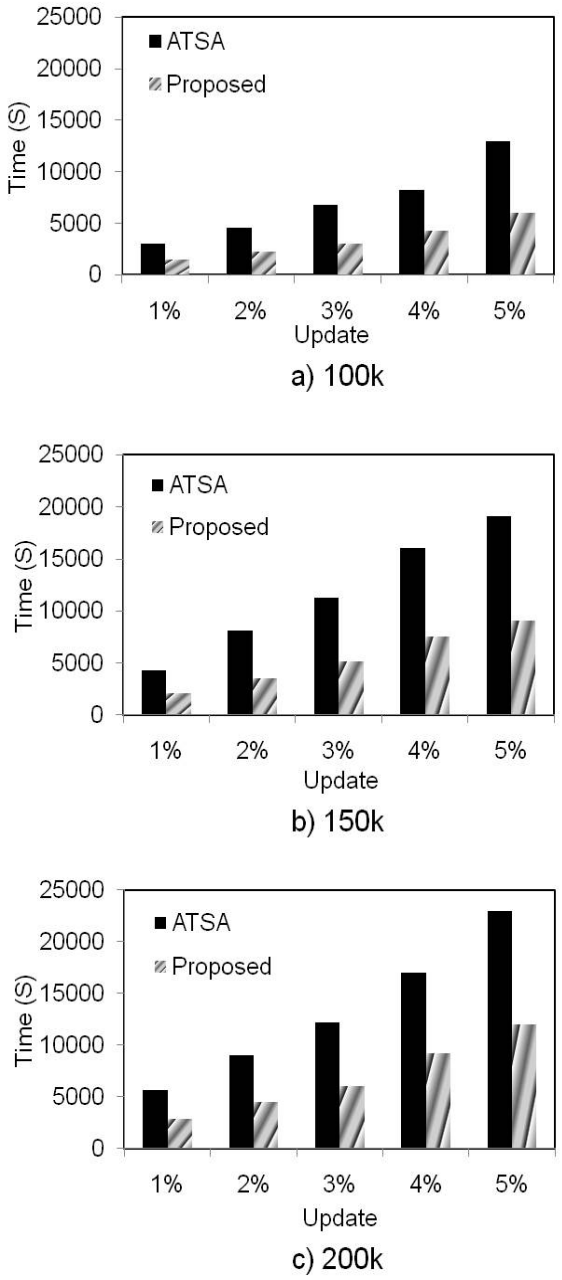


Figure 4. Update performance for different datasize

3) Effect of Cardinality: For this experiment, we use the anti-correlated datasets with varying dataset cardinality ranges from 100k to 200k and set the value of n to 7 and k to 6. Figure 4(a), (b), and (c) shows the time to maintain k-dominant skyline for update ranges from 1% to 5%. The result shows that if the update ratio and data cardinality increases the maintenance time of proposed method also increases. Even though, the time is much smaller than that of ATSA.

B. Performance on Real Datasets

To evaluate the performance for real dataset, we study two different real datasets. The first dataset is NBA statistics. It is extracted from "`www.nba.com`". The dataset contains 17k 13-dimensional data objects, which correspond to the statistics of an NBA players' performance in 13 aspects (such as points scored, rebounds, assists, etc.) and domain have range [0, 4000]. The dataset approximates a correlated data distribution. The second dataset is FUEL dataset and extracted from "`www.fueleconomy.gov`". FUEL dataset is 24k 6-dimensional objects, in which each object stands for the performance of a vehicle (such as mileage per gallon of gasoline in city and highway, etc). For this dataset attribute domain range is [8, 89]. Using both datasets we conduct the following experiment.

1) Experiments on NBA and FUEL datasets: We performed an experiment on NBA dataset. In this experiment, we study the effect of update and set the value of n to 13, and k to 12. Figure 5(a) shows the result. NBA dataset exhibits similar result to synthetic dataset, if the number of updates increases the performance of proposed algorithm becomes slower.

For FUEL dataset, we performed similar experiment like NBA dataset. For this experiment, we set the value of n to 6 and k to 5. Result is shown in Figure 5(b). In this experiment with FUEL dataset, we obtain similar result like NBA dataset that represents the scalability of the proposed method on real datasets. However, in both cases proposed method outperform than ATSA method.

VI. Conclusion

Compared with skyline query processing, k-dominant skyline result maintenance is a relatively new research area. The k-dominant skyline objects are difficult to maintain if the database is updated. However, in lack of techniques dealing directly with the problem of maintaining k-dominant skyline in this paper we propose k-dominant skyline computation and maintenance algorithms for a frequently updated database. As shown later, this technique can produce k-dominant skyline update result for all k at a time. Besides theoretical guarantees, our comprehensive performance study indicate that the proposed maintenance framework is very effective and efficient.

We leave as future work extensions to explore precomputation techniques to further speed up the computation of k-dominant skyline query. Future works should investigate the efficient maintenance of k-dominant skyline for batch updates. To increase the pruning power is another big challenge for continuous k-dominant skyline computation.

Acknowledgements

This work was supported by KAKENHI (19500123). Md. Anisuzzaman Siddique was supported by the scholarship of MEXT Japan.

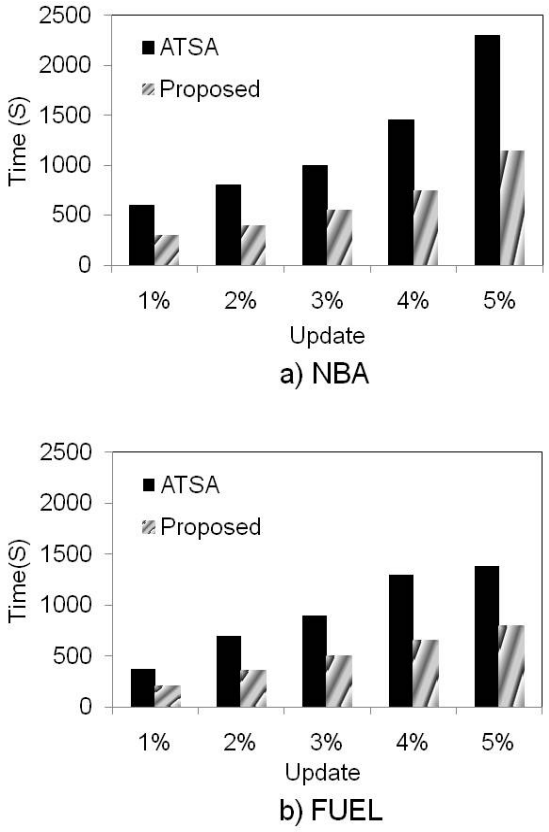


Figure 5. Experiments on NBA and FUEL datasets

REFERENCES

[1] M. A. Siddique and Y. Morimoto, "Efficient Maintenance of k-Dominant Skyline for Frequently Updated ", in: Proceedings of DBKDA, 2010, pp. 107-110.

[2] S. Borzsonyi, D. Kossmann, and K. Stocker, "The skyline operator", in: Proceedings of ICDE, 2001, pp. 421-430.

[3] J. Chomicki, P. Godfrey, J. Gryz, and D. Liang, "Skyline with presorting", in: Proceedings of ICDE, 2003, pp. 717-719.

[4] C. Y. Chan, H. V. Jagadish, K-L. Tan, A-K. H. Tung, and Z. Zhang, "Finding k-dominant skyline in high dimensional space", in: Proceedings of ACM SIGMOD, 2006, pp. 503-514.

[5] C. Y. Chan, H. V. Jagadish, K-L. Tan, A-K. H. Tung, and Z. Zhang, "On high dimensional skylines", in: Proceedings of EDBT, 2006, pp. 478-495.

[6] C.-Y. Chan, P.-K. Eng, and K.-L. Tan, "Stratified computation of skylines with partially-ordered domains", in: Proceedings of ACM SIGMOD, 2005, pp. 203-214.

[7] R. Fagin, A. Lotem, and M. Naor, "Optimal aggregation algorithms for middleware", in: Proceedings of ACM PODS, 2001, pp. 102-113.

[8] D. Kossmann, F. Ramsak, and S. Rost, "Shooting stars in the sky: an online algorithm for skyline queries", in: Proceedings of VLDB, 2002, pp. 275-286.

[9] M. Morse, J. M. Patel, and H. V. Jagadish, "Efficient skyline computation over low-cardinality domains", in: Proceedings of VLDB, 2007, pp. 267-278.

[10] D. Papadias, Y. Tao, G. Fu, and B. Seeger, "Progressive skyline computation in database systems", ACM Transactions on Database Systems, vol. 30(1), pp. 41-82, March 2005.

[11] Y. Tao, X. Xiao, and J. Pei, "Subsky: efficient computation of skylines in subspaces", in: Proceedings of ICDE, 2006, pp. 65-65.

[12] T. Xia, D. Zhang, and Y. Tao, "On skylining with flexible dominance relation", in: Proceedings of ICDE, 2008, pp. 1397-1399.

[13] K.-L. Tan, P.-K. Eng, and B. C. Ooi, "Efficient Progressive Skyline Computation", in: Proceedings of VLDB, 2001, pp. 301-310.

[14] W. T. Balke, U. Guntzer, and J. X. Zheng, "Efficient distributed skylining for web information systems", in: Proceedings of EDBT, 2004, pp. 256-273.

[15] S. Kapoor, "Dynamic Maintenance of Maxima of 2-d Point Sets", in: SIAM Journal on Computing, vol. 29(6), pp. 1858-1877, April 2000.

[16] Y. Tao and D. Papadias, "Maintaining Sliding Window Skylines on Data Streams", in: IEEE Transactions on Knowledge and Data Engineering, vol. 18(3), pp. 377-391, March 2006.

[17] Z. Huang, H. Lu, B. Ooi, and A. Tung, "Continuous skyline queries for moving objects", in: IEEE Transactions on Knowledge and Data Engineering, vol. 18(12), pp. 1645-1658, Dec. 2006.

[18] K. C. K. Lee, B. Zheng, H. Li, and W. C. Lee, "Approaching the Skyline in Z Order", in: Proceedings of VLDB, 2007, pp. 279-290.

[19] A. Vlachou, C. Doulkeridis, Y. Kotidis, and M. Vazirgiannis, "SKYPEER: Efficient Subspace Skyline Computation over Distributed Data", in: Proceedings of ICDE, 2007, pp. 416-425.

[20] K. Fotiadou and E. Pitoura, "BITPEER: Continuous Subspace Skyline Computation with Distributed Bitmap Indexes", in: Proceedings of DaAMaP, 2008, pp. 35-42.

[21] K. Deng, X. Zhou, and H. T. Shen, "Multi-source Skyline Query Processing in Road Networks", in: Proceedings of ICDE, 2007, pp. 796-805.

[22] M. Sharifzadeh and C. Shahabi, "The Spatial Skyline Query", in: Proceedings of VLDB, 2006, pp. 751-762.

[23] C. Li, B. C. Ooi, A-K. H. Tung, and S. Wang, "DADA: A Data Cube for Dominant Relationship Analysis", in: Proceedings of ACM SIGMOD, 2006, pp. 659-670.

[24] J. Pei, B. Jiang, X. Lin, and Y. Yuan, "Probabilistic Skylines on Uncertain Data", in: Proceedings of VLDB, 2007, pp. 15-26.

[25] E. Dellis and B. Seeger, "Efficient Computation of Reverse Skyline Queries", in: Proceedings of VLDB, 2007, pp. 291-302.

Enhancing Availability through Dynamic Monitoring and Management in a Self-Adaptive SOA Platform

Apostolos Papageorgiou, Tronje Krop, Sebastian Ahlfeld, Stefan Schulte, Julian Eckert, Ralf Steinmetz
Technische Universität Darmstadt
Multimedia Communications Lab - KOM
Darmstadt, Germany
apapa@kom.tu-darmstadt.de, tronje.krop@kom.tu-darmstadt.de, ahlfeld@rbg.informatik.tu-darmstadt.de,
stefan.schulte@kom.tu-darmstadt.de, julian.eckert@kom.tu-darmstadt.de, ralf.steinmetz@kom.tu-darmstadt.de

***Abstract*—The availability of Service-oriented Architectures (SOA) depends on two factors. These are firstly the availability of the services that provide a certain business functionality and, secondly, the availability of the components or services that make up the underlying SOA platform. For platforms that are supposed to form the core of mission-critical service-oriented applications, this implicates the need for mechanisms that can regulate the availability levels of the core services in changing conditions. In this paper, we handle open issues about the kind of monitoring functionalities and adaptation mechanisms that should be integrated in SOA infrastructures. In our proposed solution, we integrate concepts of event-based systems to enhance the dynamicity of the SOA platform monitoring, as well as concepts from peer-to-peer computing to achieve an efficient distribution of the SOA platform core. By prototypically implementing the concepts as extensions of Apache Tuscany, which is a realization of the Service Component Architecture standard, we show in an experiment-based evaluation how the availability of the core services of SOA infrastructures has been improved. Additionally, we explain further benefits that can be achieved with adaptation mechanisms other than replication, which are also enabled by our extensions.**

***Keywords*-Service Platform, Adaptation, SOA, Web Services, Availability**

I. Introduction

As has been also described in [1], the success of Service-oriented Architectures (SOA) is normally not credited to the strict technical features or Quality of Service (QoS) levels offered by the underlying technologies. However, along with their established advantages, such as high flexibility, extensibility, and interoperability [2], Service-oriented Architectures are now also expected to achieve performance and availability levels that are as high as these of traditional, platform-dependent solutions. Approaches that aim at improving the availability of SOA are usually built on the assumption that a number of service alternatives can be invoked ad hoc, if a service fails. These approaches use techniques like process replanning with dynamic service substitution (as in [3], [4], and [5]), or dynamic enforcement of governance guidelines [6], and are usually applied at the level of service consumption or business process execution. Still, when the availability of the applications that use these techniques is measured, there is an upper bound that can be achieved. It is the maximum availability level that the used service platform can support. This platform can vary from a simple enabling infrastructure, i.e., a simple service registry with any accompanying components, to a complex Enterprise Service Bus (ESB) [7].

The challenge is that current service platforms can support limited availability levels, because of vulnerabilities or single points-of-failure inside their core. Such a basic vulnerability, which we address with our approach, is the centralized access to functions of the domain and the deployment, i.e., centralized access to interfaces that are used for address resolution, dynamic launching of services, and more. Even if the services are available, the availability experienced by the user declines if the machines that provide these interfaces under-perform. Similar problems exist with service registries and search functions. Furthermore, current solutions use static monitoring approaches (cf. also Section II), which cannot support quick enforcement of healing mechanisms, e.g., replication of overloaded services.

Some techniques, such as Web service replication [8], appeared in order to solve some of the aforementioned problems. These techniques have sometimes high costs and must be supported by monitoring mechanisms and by an adequate decision logic. This monitoring-supported enforcement of such techniques, as well as related research, are normally positioned under the fields of adaptation mechanisms and self-organization. How this can be optimally applied on SOA infrastructures has not been thoroughly examined from a technical perspective, and depends on the nature of the used platform. Different service platforms (e.g., ESBs) are used in different application domains, and each of them presents different challenges concerning its enrichment with adaptation or self-organization capabilities. This work presents a concept which, in its general form, can be used for such enrichment of many different SOA platforms. Its main ideas are the distribution of the core parts of a SOA platform and the employment of event-based monitoring in the platform core for supporting self-adaptation. This general concept is

then implemented as an extension of the Service Component Architecture (SCA [9]). The work is presented and evaluated on the state-of-the-art SCA platform, Apache Tuscany [10].

With this regard, the paper is outlined as follows: Section II examines the related work and states our contributions. Section III identifies some additional challenges that are present in our particular scenario of a mission-critical SCA platform. Sections IV and V form the core of this paper by describing our solution and its evaluation results. As the concept could enhance different platforms, the description of the idea (Subsection IV.A) will be as independent of the implementation as possible. Still, the detailed desription of the different service lifecycle phases (Subsection IV.C) sometimes needs to refer to implementation details in order to better support the reader's understanding. Section V presents the results of a well defined evaluation scenario with our extended Apache Tuscany platform and Section VI offers implementation-related examples of further adaptation mechanisms that can be integrated due to our extensions. Our conclusions and plans for future work are summarized in Section VII.

II. Related Work and Contributions

The decision to use concepts from peer-to-peer (p2p) computing and event-based systems were taken after a careful analysis of all the phases that a self-adaptive SOA platform has to go through. Therefore, the best way to present the related work is to explain where these concepts have been already used or proposed in order to enhance SOA platforms. This way we come to new ideas and propose their usage for further possible enhancements. So, we look into related work in three main directions, where we also identify and position the three partial contributions of our work. First, we look at the research towards third-generation, self-adapting service platforms. Second, we see attempts of enhancing service platforms by using peer-to-peer technologies. Last, we examine monitoring aspects of up-to-date service platforms.

In accordance to the nature of service-oriented software, some tasks exist, which must be fulfilled in almost all SOA solutions. The most important of them are:

- The service registry mechanisms (service advertisement, service look-up, etc.).
- The address resolution (mapping of name-based service calls to exact addresses/endpoints).
- The service deployment (loading, configuration, starting, and stopping of services).
- The management and monitoring, usually in the form of auditing and logging, with focus on QoS parameters such as service response times or hardware metrics such as CPU load.

Depending on the scale at which these tasks are automated or undertaken by middleware components, there are traditionally two approaches for building SOA infrastructures: the point-to-point integration and the hub-and-spoke approach [2]. While the first is simpler and more static, the latter includes a service bus and/or other related middleware that dynamically undertakes the aforementioned tasks, as well as their subtasks, such as the routing and addressing of the used services, or the support and transformation of the used protocols. Other functionalities can also be present, letting the hub-and-spoke approach be considered as more advanced and, in essence, as the successor of the point-to-point integration [2]. Nevertheless, research in the field of SOA self-adaptation ([11], [12]), lets us assume that we are heading for a third generation of SOA infrastructures, in which the service platform, i.e., the service bus with the accompanying middleware, will offer even more automation and further functionalities, namely more sophisticated, integrated monitoring, adaptation mechanisms, and more. As the enrichment of service platforms presents different challenges and opportunities depending on the exact paradigm, we contribute in these attempts towards "third-generation" service platforms by presenting an idea of what these extensions should include, and by showing how it is implemented in the case of SCA. As the SCA paradigm dictates the existence of certain components in the service platform, our contribution is the identification of the exact points where these SCA-specific components could be enriched with self-adaptiveness, as well as our corresponding implementation, performed as an extension of Apache Tuscany.

Main intension of the adaptation mechanisms is to keep the QoS above certain limits. A recent survey [13] already placed peer-to-peer mechanisms among the most highly suitable solutions for the substrate of future service platforms that go in the direction of QoS-guarantee and self-adaptation. Approaches that use peer-to-peer mechanisms for the enhancement of service platforms have focused until now either on special-purpose service orchestration [14], or on service discovery and group collaboration [15]. Believing that the enablement of self-adaptation dictates that these mechanisms lie deeper inside the platform and support all or most of the functionalities of a service bus, we contribute by using peer-to-peer mechanisms to distribute the service bus and enhance the availability of the services of an SCA platform. Furthermore, unlike most of such new frameworks, we provide an evaluation scenario and some measurements to demonstrate the availability enhancement.

Aspects of our integrated platform monitoring can be seen as a further contribution of this work, given that almost all state-of-the-art monitoring components of service platforms are not integrated in the platform logic and cannot serve the goal of supporting self-adaptation optimally. Instead, they normally perform centrally-controlled measurements for hardware modules or service invocations. In the next sections it will be further clarified how this differs from our decentralized, event-based, adaptation-enabling platform monitoring approach. Strengthening our argument, we men-

tion that almost all theoretical SOA Maturity Models (e.g., [16]) define five possible maturity levels for a SOA and they place the feature of event-based platform monitoring in the maturity level 4. Related studies (e.g., [17]) prove that almost no current SOAs achieve that maturity level, but they rather lie on lower levels, usually levels 2 and 3. The referenced study was done among software departments of the german banking industry, which are supposed to be among the leaders of SOA adoption.

III. Further Challenges of our Scenario

The purpose of our extended platform is to serve as the SOA substrate for our project [18], which supports the management of disastrous events. In such a scenario, the availability of the services not only needs to be high when the disaster occurs but it is also expected to be suddenly endangered, because of an "explosion" of the system usage at that point. This system usage pattern will be reflected in the test cases of our evaluation in Section V. The use of our platform in such a scenario indicated a long list of requirements. In the following, we list how these requirements can be summarized or translated to technical challenges for our platform:

- No single point-of-failure is acceptable for any critical core service.
- Control mechanisms must provide the possibility of defining different, application- or situation-dependent algorithms that determine the minimum number of instances of particular services. These algorithms will be designed based on the needed availability levels and the expected usage patterns.
- Consistent and detailed information about the running services is needed in order to provide enhanced control. This means that all services have to be registered with the same procedure before they are started, and there must be mechanisms that find out which services, and how many instances of them are registered / active, and on which nodes.

We have found no service bus, but also no conceptual or architectural approach, which addresses these needs (cf. Section II). As for the implementation, the extensions that will be presented were necessary also because of the following lacking capabilities, which are absent from Apache Tuscany, but also from all other examined platforms:

- The platform enables the development of distributed applications but it is almost impossible to distribute all the core modules in the way that our challenges dictate. Normally, the Tuscany domain and deployment service is centralized and it also lacks many of the desired capabilities and functionalities that we mentioned.
- There are no service monitoring mechanisms that could support self-organization or absolute control of service instances. The monitoring modules can only support static logging and not the dynamic monitoring logic that we will describe in more detail in the next sections.
- There are no replication or maintenance mechanisms for the internal application services.

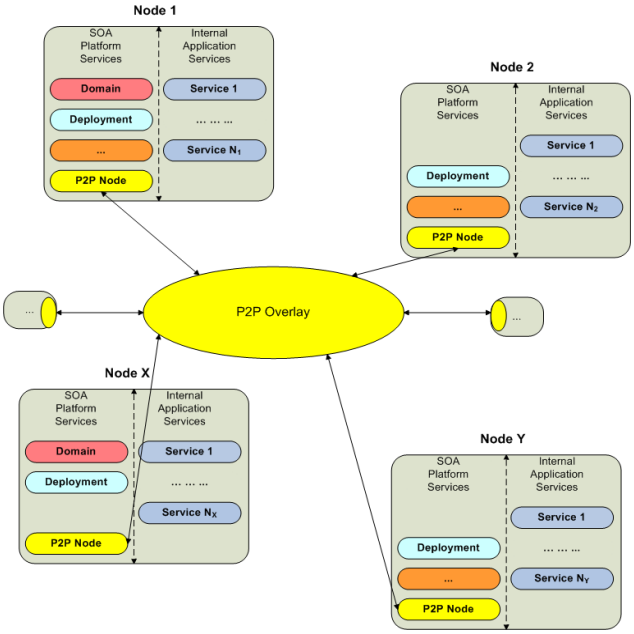


Figure 1. Overview of the p2p-based distribution of the core parts

IV. Service Platform Availability Extensions

With regard to the described challenges, we present in this section a generally applicable idea of how they could be handled inside a service platform, and then we briefly describe how we implemented most parts of the concept by modifying the Apache Tuscany service platform. In the third part, we go into more detail in order to explain how our extensions work. This part mentions implementation details only when this is helpful for the understanding.

A. Concept

We define as core parts of the service platform those parts that are responsible for the main platform functionalities, as we mentioned them in Section II (registry mechanisms, address resolution, service deployment, and monitoring). Our main idea was to re-define these core parts so that:

- They are distributed, consisting of many co-operating instances, supporting fault-tolerance in the classical p2p manner, i.e., being able to operate despite the unavailability of some instances.
- They offer an extended set of functionalities that enable self-organization/adaptation mechanisms and support the fulfillment of our availability requirements.
- All the extended functionalities are offered through the interfaces of a p2p overlay, so that no centralized parts

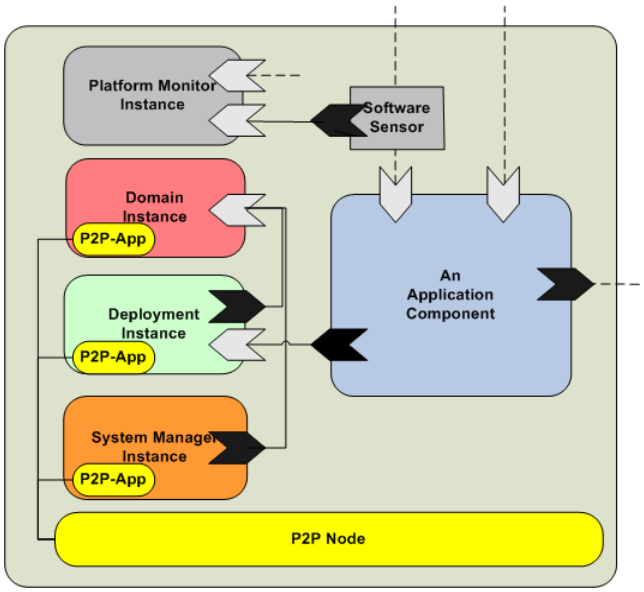


Figure 2. Component interrelations in a node of the modified platform

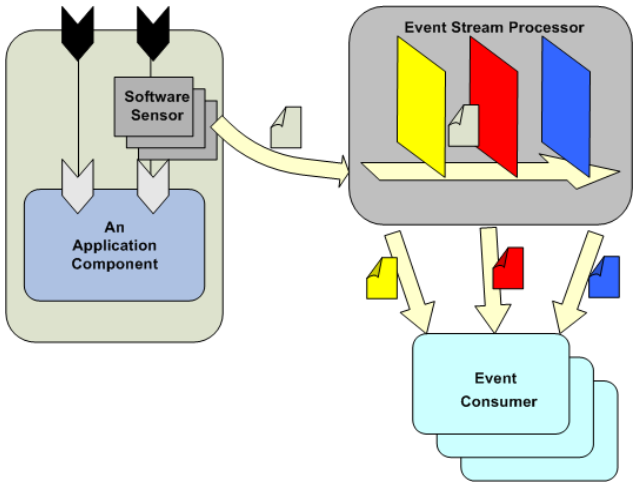


Figure 3. Adaptation-enabling event processing of the platform monitor

of the service bus have to be addressed when core functionalities are requested by any node of the system.

The choice of p2p is driven by our striving for fault-tolerance. The failure of peer nodes, on which instances are running in order to provide core mechanisms, will now not mean that the mechanisms will not be available any more. At the same time, a flexible cooperation of the core part instances is needed. Few technologies can support this fault-tolerance and this cooperation as good as the p2p technology does. On this basis we designed a platform where all participating nodes, i.e., all providers/consumers of application-level services, can also carry instances of core parts, participating in a common p2p network that connects their core part instances (Figure 1). We re-define, extend and distribute four core parts, while a lot of accompanying platform parts/functionalities are abstracted in our concept and taken from the used platform in our implementation. A description of these four core parts follows, focusing on the features that are normally absent in current solutions, like Apache Tuscany.

Our distributed domain service is addressable through the overlay (so that one instance of it might be enough) and offers the extended possibility of returning multiple endpoints to service-lookups. This supports the usage of service replicas that are generated by our self-organization mechanisms, as well as a more reliable address resolution, given that any node may be able to perform this resolution. The service registry can also be seen as part of the domain service and it has the form of a distributed database with its entries being transparently and redundantly distributed among those nodes that carry domain service instances.

Our distributed deployment service enables the local or remote starting/stopping of services. It is assumed that the services store their resources when they are registered in the domain and that these resources are sufficient in order to start/replicate them on other nodes. Nodes also use the deployment service in order to register themselves as capable of hosting particular services.

Our distributed system manager takes care of pre-defined numbers of instances of other core services and offers additional interfaces for system information that is important to other core parts, especially to the platform monitor.

Our distributed platform monitor has major differences from usual service monitoring components or tools. Its goal is to support adaptation, so it engages the Event Stream Processing (ESP) concept [19] and a push-approach for (developer-defined) monitoring events, rather than a database where simple observations are stored. Furthermore, it is integrated in the platform logic, so that no direct or indirect interaction with the monitored services or their “callers” is needed in order to gather information about the service calls. In the evaluation scenario, we will see an exemplary usage of the monitor that would not be achievable with other approaches.

B. Design and Implementation

All these conceptual extensions pose new challenges when it comes to their implementation as extensions of existing service platforms like Tuscany. For example, some features can be added “on-top” while others may present incompatibilities with existing mechanisms. We distinguish three approaches for enhancing the service platform with new features, which are generally valid when it comes to middleware enhancement:

- As new platform modules, i.e., developed and built additionally to the existing modules of the platform.

- As external libraries, which can be either special-purpose libraries, i.e., software developed for these extensions, or ready, possibly third-party, software.
- As modifications in the core of existing platform modules, when incompatibilities appear.

Before listing what we implemented in these three directions, we present in Figure 2 a compact SCA representation of a node of our modified platform, providing a view of the interrelations of the core parts of the service platform, as well as their relation to the p2p overlay and the normal application components.

We had to define a new node type, the *CoreNode*, which merges an SCA node with a p2p node. While the extended domain, deployment, and system manager are based directly on the p2p node, the platform monitor is built on the (modified) service invocation mechanisms of the platform, enabling the binding of queries (posed by any monitoring component) to particular services, in order to retrieve the data that it needs about the corresponding service invocations. This is again compactly depicted in Figure 3.

The API of each core part corresponds to the functionalities described in Section IV-A. We provide here an overview of the implementation with regard to the three categories that we distinguished in this section:

- *New platform modules*: The module that defines the CoreNode and includes the implementations for the deployment and the system manager instances is the *distributed-core*. It is implemented as a new module but depends on some core modifications, as well as on an external library for the p2p overlay. The *platform-monitor* is also a new module, also depending on core modifications and on an external library for the Event Stream Processing.
- *External libraries*: *freepastry* is used for the p2p overlay and *esper* for the Event Stream Processing. Both are third-party, open-source libraries.
- *Core modifications*: The Tuscany module *core* was modified in order to implement our domain instances. Inside the *assembly* module of the core, we had to modify the runtime component implementation. Some other modules, e.g., the *java-runtime-implementation*, also had to be modified in order to support dynamic invocation and other features needed by our modified platform.

C. Service lifecycle phases for redundancy and self-adaptation

With regard to the presented concept, design and implementation of our service platform extension, we present a more detailed view of the lifecycle of a service with the focus on self-adapting and self-organization.

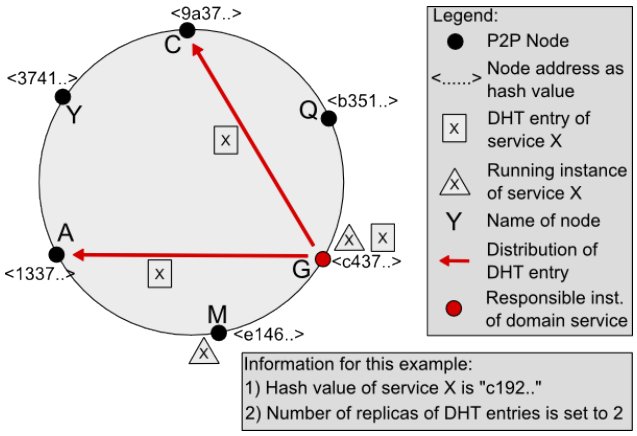


Figure 4. DHT entry distribution

Phase 1 - Service Registration and Distribution of Service Resources:

The most important aspect of the domain service is the transparent distribution of its database, which holds redundantly all the information needed for a service, e.g., its name, its configuration, and its resources. In order to coordinate the self-organization tasks, there is only one instance of the domain service which is responsible for the registration and for any modifications of a particular service X. The same instance is also responsible for returning multiple endpoints to service look-ups of the service X.

To understand the nature of the responsibilty of an instance of the domain service for a service X, it is important to know the structure of the used p2p overlay. Our service platform extension uses *freepastry*, which is an open-source implementation of Pastry [20]. Pastry uses a ring structure for the peer-to-peer overlay, where every node of the ring is identified by an unique address. The address of a node is a hash value randomly allocated by the *freepastry* library during the bootstrap process, i.e., when the node enters the overlay. An example ring structure is presented in Figure 4, where the name and the address of every node are illustrated among other information.

To address a node of the ring it is not necessary to know the accurate address of it, but one hash value part of the range of addresses which "belongs" to the node. For the example of Figure 4, this means that node C with the hash value $\langle 9a37..\rangle$ is addressable by any hash value from $\langle 3741..\rangle + 1$ to $\langle 9a37..\rangle$. Thus, every node of the ring is responsible for the hash values from the address of its predecessor (node Y in the example) to its own. For more details about this kind of hash-based addressing in p2p networks, we refer to [21].

The responsibility of a domain service instance is the range of hash values which belong to the node where the instance is located. To find out which domain service

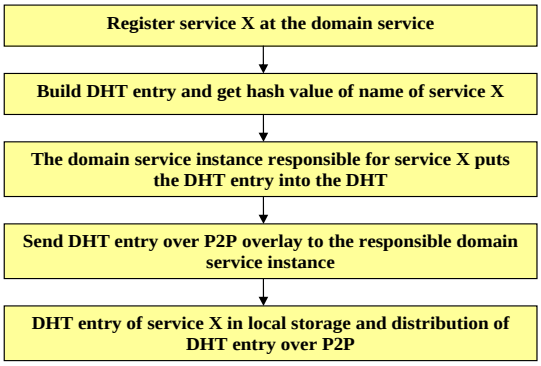


Figure 5. Registration of a service X

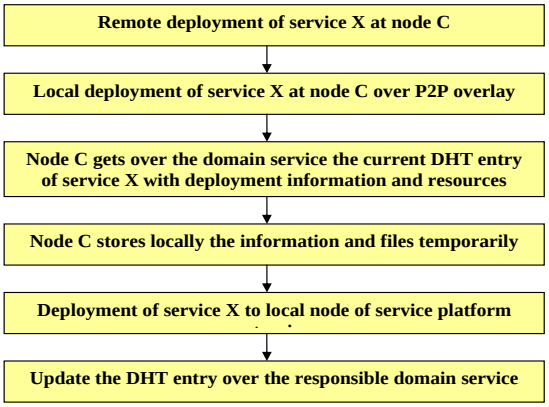


Figure 6. Remote deployment of a service X

instance is responsible for the deployment of a service X, the name of the service will be transformed to a hash value as well. The domain service instance responsible for the hash value obtained from the service name is then addressed in order to deploy the service.

For the registration of an internal service, i.e., for a service that has been created within our platform and is running on it, it is important to know which information must be saved and how the domain service will distribute this information inside the platform. Before explaining the sequence of this information distribution, the Distributed Hash Table (DHT) entry is introduced. A DHT entry contains the name of the service, the data files needed in order to deploy the service on the extended service platform, and information about the nodes where the service is already deployed. In our implementation, the mentioned data files are stored as a JAR file of the service X in its DHT entry. This offers the possibility of remote deployment of the service. This feature will be explained in more detail in the next phase.

The information of the DHT entry includes the status of the service on each node as well. This status can have the values “started”, “running” and “stopped”. “started” stands for the phase when a service is registered on the domain service but not yet deployed. After a successful deployment, the status is changed to “running”, while an undeployment of the service switches the status to “stopped”, until the information about the node where the service was undeployed is deleted.

The different steps of the registration process are presented in Figure 5. The process starts with the registration of service X at the instance of the domain service where the DHT entry of the service was built and the hash value of the service name was calculated. Then the hash value is used to address the responsible instance of the domain service over the peer-to-peer overlay and the DHT entry is transmitted to it. The responsible instance puts the entry into the DHT, which stores the entry locally and also distributes it over the peer-to-peer overlay to other nodes. The number of replicas of the entry is predefined at startup of the peer-to-peer overlay.

The distribution of the DHT entries is illustrated in Figure 4, where the domain service instance which is responsible for the example service X is running on node G. The service itself is running on nodes G and M, while the DHT entry is stored locally on node G and transmitted to nodes C and A as replicates. It is important that the information about a service must not be stored locally where the service is running. The information can be somewhere else inside the peer-to-peer network.

Through this redundant storing of the DHT entries, there is no single point of failure for getting information about services whenever this information is needed. Even more important, the resources that are needed to start the service are also available on more than one node, together with the rest of the information.

Phase 2 - Local or Remote Service Deployment:

The nodes of the extended service platform have an instance of the deployment service running, thus providing the possibility of deploying a service locally and remotely. With the remote deployment function, it is possible to deploy a service X on any other node inside the service platform. This requires the cooperation of the deployment service instance that “wants” to start service X with the domain service instance that is responsible for service X. However, this is performed seamlessly, as the deployment service only addresses its local domain service instance. The latter locates then the domain service instance which is responsible for modifying and re-registering service X.

The procedure of a *local deployment* includes the registration process described in Phase 1. If the service to be deployed is already registered, the information that other nodes have about the status of the service is updated. After

the registration, the deployment continues by adding the service to the platform, using the locally stored data. In the case of Apache Tuscany, the only resource needed for the deployment of a service is a JAR file. Then, the status of the service in the DHT entry is updated to "running". Again, for this update, the deployment service seamlessly addresses the responsible domain service instance.

The detailed procedure of a *remote deployment* for a pre-registered service X is presented in Figure 6. To deploy service X on another node C of the platform, a deploy message will be sent over the peer-to-peer overlay. The node C receiving this message gets the DHT entry from the domain service and saves the information and the resources locally. Then the local resources are used to deploy service X with the local deploy method, including the update of the status of service X over the responsible instance of the domain service.

Phase 3 - Static or Dynamic Service Replication:

Another important additional feature of the extended service platform with regard to self-adaptation is the static or dynamic replication of a service. The part that mainly enables this feature is the system manager. This is performed by a mechanism called Service Instance Control Mechanism (SICM), which works in strong cooperation with the domain service and the deployment service. The dynamic replication, i.e., the replication as a self-adaptation action, is, of course, also supported by the platform monitor.

The SICM can be started for a service X directly after its successful deployment (static replication), or it can be called later by any other node of the extended service platform (dynamic replication). The number of deployed instances of service X with the status "running" is read from the DHT entry. This number is compared to a threshold (minimum number of needed service instances). This threshold has been passed to the SICM as a parameter.

Depending on the result of the comparison, the SICM terminates if enough instances are running. Otherwise, it uses the domain service in order to search for nodes where additional instances of service X could be deployed. If there are not enough nodes, the SICM will be idle for a predefined time and then start again. Otherwise it will start to deploy more instances of service X on nodes with enough resources that had no running instances of service X.

Additional to the SICM, the system manager provides interfaces for getting system information, or other information that support self-adaptation.

Phase 4 - Maintenance and Monitoring of a Deployed Service:

As already mentioned, a distributed, event-driven platform monitor has been added. The modules used by such monitors in order to capture and forward specific information, are called software sensors. The main features of the platform monitor, which will be described in more detail in the following, are the possibility of adding software sensors with a developer-defined focus, as well as the possibility to trigger different adaption actions for different "captured events".

The platform monitor implements the ESP concept. This means that it can gather and preprocess events from remote software sensors. Additionally, it offers the developer the possibility to implement a monitoring logic that may be different for each software sensor. This monitoring logic is defined by writing ESP queries with an SQL-like language, called Event Processing Language (EPL) [19]. With the following example, we give an idea of how such a query looks like:

```
select sender, count(sender)
as sentPackets from
Event.win:time(5 sec)
group by sender
```

The node that submits the query is called *actor*, because it will act (or better, react) upon the event defined in the query. For example, an actor registered at the platform monitor with the above query is interested in all services which transmit packets inside the network of the platform. The platform monitor will store the information that matches the query and will send back to the actor an event for every service that sends packets. This event will contain the packets transmitted by the service within the last five seconds.

To collect the queried information, a software sensor has to be registered at the platform monitor. Then, it sends the collected information back to the platform monitor. The procedure of a registration of a software sensor for service X and the calling of this service by a node B is presented in Figure 7. We refer to this software sensor as sensor S.

After the registration of S at the platform monitor, its creation is registered at the system manager. The system manager inserts S to a proxy of the service that is to be monitored, and not to the service itself.

After this step, S is successfully deployed and will send events to the platform monitor according to the query with which it has been created. In order to present the monitoring process when service X is called by node B, Figure 7 shows the corresponding sequence of actions. Node B perceives the call of service X as a direct call, but inside the extended platform the call is redirected to a proxy. At the proxy, all registered sensors observe the call and act according to their logic. For example, there is the possibility of sending an event just after the service call or after the service execution has been finished.

When the call finishes, the proxy will transmit the result to the calling node (B), which perceives it as a answer from the original service X.

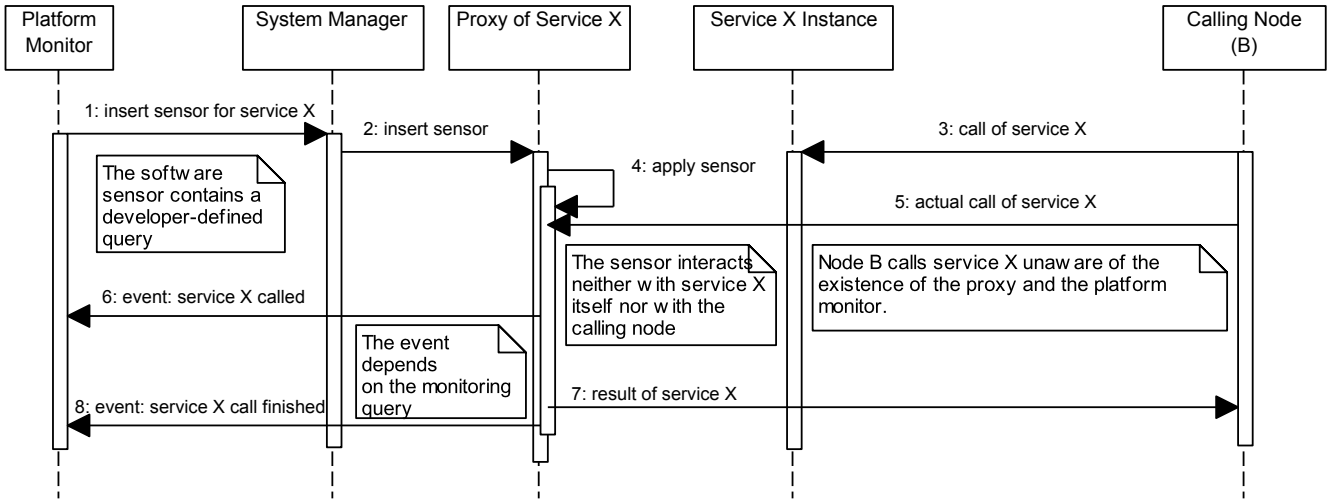


Figure 7. Event-based service monitoring in the extended SCA platform

With the mechanisms of the platform monitor described above, it is possible for a developer to implement own software sensors and event consumers for services and to monitor them according to his needs. This is a completely new feature for Apache Tuscany and is to our best knowledge a highly innovative feature for any service platform.

V. Quantitative Evaluation: Availability Enhancement

In order to evaluate our approach with regard to the availability enhancements, which have been our main concern, we define a specific scenario that was related to our project, and compare our approach with a release version of the used platform. Of course, specific adaptation mechanisms should be compared to related approaches that could potentially enrich the same service platforms. Unfortunately, such general comparisons do not seem to be applicable at the moment, and remain subject of future work. Still, Apache Tuscany is a state-of-the-art SCA platform, and comparisons with it appear to be in our case more interesting than any other scenario. In the next section, we will describe an additional scenario, showing how a node can decide to adapt the protocols used by its services based on monitored information about the types of clients that dominate the system.

A. Evaluation Scenario

The experiments that are based on our modified platform are such that as many new features as possible can be evaluated. Nevertheless, they are limited to include only some capabilities. We condense many functions into two main capabilities that we will use in our experiments. It is necessary to describe now these two capabilities:

- *Interest Registration*: Any component can register itself as "interested" in an SCA service, saving at the same time its queries, determining this way what kind of data the software sensors will be sending to it and when. Such components contain "actors", which enforce reactions under certain circumstances.
- *Service Instance Control Mechanism*: The deployment instances offer to other components the possibility of retrieving the number of running instances of a particular SCA service, as well as the addresses of the nodes that could host further instances. The SICM builds on these capabilities and can be used by any component in order to define a minimum number of instances of a service that should be running. This "requirement" is saved, so that failures of hosting nodes lead to the starting of instances of the service on other candidate nodes.

Internal services of our application are expected to be suddenly invoked with an increasing frequency when a disaster occurs or later when the emergency level of the situation is set higher by the involved organizations. With this regard, we chose an example service, and implemented external clients that invoke it with the pattern shown in Figure 8. There, we see also how a linear increase of users leads to an exponential increase of erroneous service invocations, i.e., to decreased availability levels. The test-clients record errors when no response is received or when a timeout occurs. More details will be analyzed in Section V.B.

With $N_t(x)$ denoting the number of occurrences of x in the last t seconds, we define as *availability* of S for our scenario the value

$$A = \frac{N_{10}(\text{Successful invocations of S})}{N_{10}(\text{Invocations of S})} \times 100\%$$

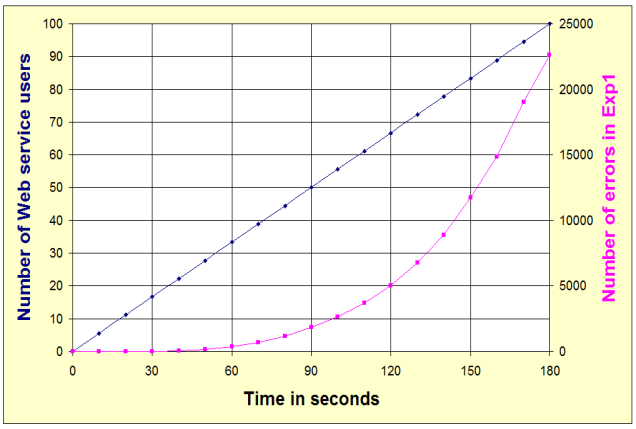


Figure 8. Experimental service invocation pattern

and we measure it over time for the following four experimental cases:

- *Exp1*: An instance of S is running on the Apache Tuscany release platform.
- *Exp2*: Three instances of S are running on the Apache Tuscany release platform and the invocations are equally distributed to them. The number of instances (3) was chosen empirically, so that it could almost always satisfy the given invocations' curve (Figure 8). For this case, as well as for the next two cases, the distribution of the invocations among the instances was simulated. This is safe because the load balancing is irrelevant to the results that we present, though it would, of course, be interesting to test with different balancing of the invocations.
- *Exp3*: An instance of S is running on our extended platform, the deployment instance of a node (more nodes could be used for fail-safety) registers itself as interested in S, with a query for retrieving the number of users of S each second. The deployment instance (more precisely its "actor" upon the retrieved data) has the following simple logic: use the SICM to add an instance every time that the load of S exceeds a limit. This limit was chosen in our case so that, for the given input of Figure 8, the mechanism is started almost every minute.
- *Exp4*: As in *Exp3*, with the difference that the SICM now doubles the number of instances every time it is triggered. With these two different configurations, we show the flexibility of the freely defined adaptation logic, indicating how our framework can easily integrate application-dependent logic in order to be optimally exploited in different systems. Obviously, the choice of this logic affects the results.

B. Comparison Results

Figure 9 and Figure 10 present the evaluation results based on the four experiments that we described. Although the results have been obtained from an example service, which can be either an internal application service or a core platform service (e.g., an instance of the deployment service), it is obvious that this does not harm generality. Similar effects would be noticed for almost any service, maybe with a slightly modified invocation pattern. These evaluation results intend to show some enhancements of a platform in particular scenarios and are not to be seen as a direct and complete comparison. Furthermore, the results only show the benefits of the mechanisms described in Section 5.A, which are based on our extended concept. Further benefits of our solution that we described earlier and relate to the p2p-based fault-tolerance of the core parts are not included in these experiments and are not mirrored in the results.

The results for Exp1 prove that the availability of a service sinks when the number of users increases rapidly. The same effect is slightly noticeable even in the case of the second experiment that is based on the original Tuscany platform, namely Exp2, although the number of service instances was manually chosen in order to satisfy the given input. The decrease of the availability level is in that case much slower than in Exp1, though steady. If the number of users would grow further, then the number of service instances would not be able to satisfy them any more, and an effect similar to that observed in the case of Exp1 would appear. Even if the maximum load that can be expected for a service is known from the beginning, excluding this way the possibility of such effects to appear, the usage of many instances from the beginning can lead to a big waste of resources. In scenarios like ours, where the service usage explosion is expected to happen suddenly but also rarely, this waste will be ongoing during most of the time.

Contrary to Exp1 and Exp2, the number of service instances during the experiments Exp3 and Exp4 is adapted to the service load, maintaining high availability levels without wasting resources. Figure 10 shows the effect of service instance control. The component that uses the extended mechanisms in order to perform this control is (implicitly) informed (in this case every ca. 1 minute) by the platform monitor that the availability is sinking. Accordingly, further service instances are deployed and the service invocations are again distributed among them. So, with an appropriate configuration at the side of the monitoring (and acting) component, the availability can be maintained at the wished levels, as long as this is allowed by the total resources that are available in the system. In a similar manner, the service instances can be adapted to a decreasing number of users, though this is not shown with the present experiments.

During the last minute of the evaluation, Exp4 presents a

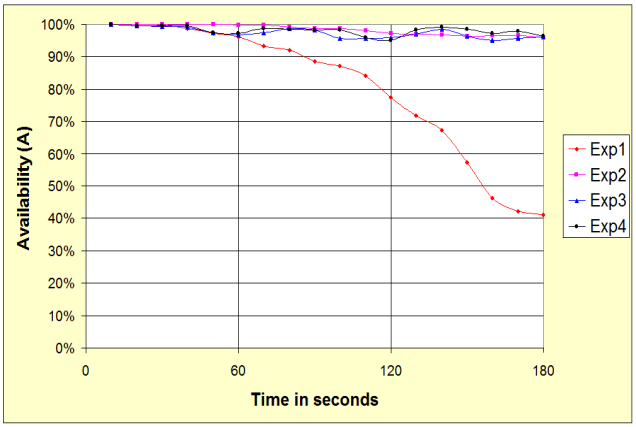


Figure 9. Measured availability

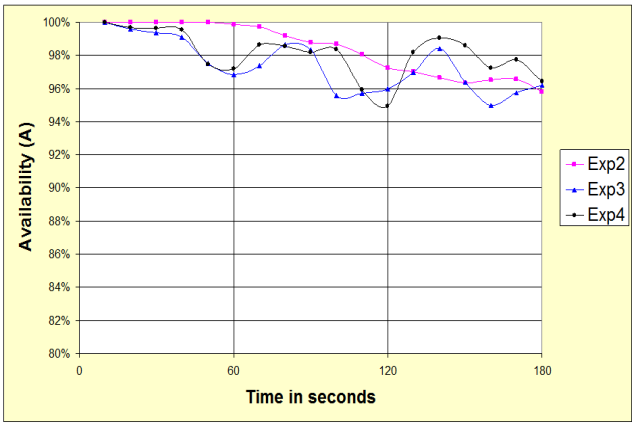


Figure 10. Adaptation effects

higher availability, because the number of service instances is increased more abruptly. With the difference between Exp3 and Exp4, we can understand the configurability of the used mechanisms. The fact that different logics can be used inside these mechanisms offers flexibility in the regulation of the availability levels and of their trade-off with the costs. For example, a logic like the one used in Exp3 would be used in a scenario where service instance adaptations can be performed often, while the logic of Exp4 would rather be applied in scenarios where the frequent adaptation is either impossible or not desired.

VI. Qualitative Evaluation: Dynamic Protocol Adaptation

While a quantitative evaluation has been presented in the previous scenario, the evaluation of this section is characterized as qualitative. A qualitative evaluation does not directly compare approaches in order to mathematically prove which one is the best one, but it rather provides (measurable) hints about how an approach could bring benefits and leaves space for further research. Indeed, we are going to explain on the basis of some experiments, why the dynamic protocol adaptation of services can offer benefits.

In this scenario, the adaptation is triggered based on client types rather than client numbers. More precisely, nodes use our platform extensions in order to extract information about the involvement of mobile service consumers and they adapt their communication protocols accordingly. Mobile usage of services is not the only case that can profit from dynamic protocol adaptation but it is by far the most important one. Thus, our evaluation will refer to this case. Before we discuss the benefits and the limitations of such adaptations based on experimental results, we describe the scenario and explain why it is our platform extension that enables it.

A. Dynamic Protocol Adaptation with Our Apache Tuscany Extensions

Although mobile SOA is pre-mature and the participants of SOA systems are usually stationary computers of IT departments, mobile SOA participants start to appear, usually as simple Web service clients. Mobile SOA participants have many differences to other participants, regarding both the way in which the devices consume the service and the QoS-efficiency of particular communication protocols [22]. For example, service buses, such as Apache Tuscany, normally cannot be used for the development of the client side, if the client is a constrained mobile device. Even more important, the standard communication protocol of Web services (SOAP) causes big delays in some cases of mobile Web service consumption. Although the service platform cannot be run and used on mobile clients, it could trigger service adaptation actions in cases of extensive mobile usage of particular services. Such an important adaptation action could be the dynamic adaptation of the protocol with which a service is offered by the platform. In the following, we describe this dynamic protocol adaptation that can be triggered by our extended platform. After that, we present some experimental results that demonstrate the importance of being able to perform such adaptations dynamically.

In order to explain dynamic protocol adaptation of a service with our extended service platform, a short description of how components and services are configured in SCA is necessary. SCA uses the Service Component Definition Language (SCDL) in order to define inside a configuration file (*composite file*) the components, the services, and their interactions inside the system. So, the architecture of a system, or of a system part, is implicitly defined in this file. Every service deployed to the service platform has to be contained in such a composite file, defining the attributes and settings of the service. The important part related to dynamic protocol adaptation of services is the determination of bindings, with which a service is made available. The bindings determine how a service communicates with other components, with each binding corresponding with a particular protocol. To explain the possibility of binding

modifications in SCDL, we provide the following composite file snippet:

```
...
<service name="ServiceX">
  <binding.ws
    uri="http://www.a.com/serviceX"/>
  <binding.rmi host="www.b.com"
    port="8099" serviceName="serviceX"/>
</service>
...
```

Two main service settings can be seen in the above snippet:

- The name with which the service is defined inside the platform (ServiceX).
- The bindings with which the service is offered, in this case a Web service (SOAP) binding and an RMI binding. The Web service binding only needs the URI of the service, while the RMI binding needs the host address, the port number and the specific name of the service at the location it connects to.

To modify an existing service so that a new binding for it is added, the composite file must be edited and re-deployed. Let us assume that initially only the Web service binding is present for Service X and the usage of this service in the system changes in such a way that the addition of an RMI binding is desired. The first step for the modification is to fetch the current resources of the service from the DHT and to edit the composite file by adding the part written in bold font in the snippet above. The second step is to upload the resources back to the DHT, replacing the old, unedited files with the mechanisms introduced earlier. The last step is the re-deployment of the service which is recognized as a restart of it. After the re-deployment, the additional binding for service X can be used.

SCA supports several protocols and bindings, but it is not recommended to use every binding for every service from the beginning, and, of course, this is never done in the praxis. This is because the existence of many open bindings may lead to increased complexity, unnecessary traffic inside the network, or even security gaps. For this reason, a dynamic and adaptive modification of the bindings of a service is preferrable and is supported by our extended service platform. Furthermore, the whole process is easier with our implementation, because the programmatical re-deployment of a service is simplified in our extended platform, as it can be done through a simple function offered by the overlay.

Another example mechanism, not binding-based this time, for dynamically modifying the way with which a Web service can be accessed, is the activation or de-activation of compression for the SOAP communication. It is not implemented and supported by Apache Tuscany originally, but it is possible through modifications in the configuration of the used Web container. The platform monitor could sense an increase in the number of clients that would profit from compression, e.g., mobile clients, so that the adaptation action of activating compression would be then enforced.

B. Experiments Showing the Potentials of Dynamic Protocol Adaptation

The experiments correspond with the scenario decribed in the previous subsection. Thus, it is assumed that one or more services are offered with a SOAP interface, which means, for our platform, with a Web service binding ("binding.ws"). The existence of more access methods (or bindings) for this service, e.g., over RMI or with data compression, may not be desired from the beginning, for various reasons. This can be, for example, because of system complexity, server costs, or security concerns, caused by the existence of many open endpoints.

Thus, the experiments are meant to answer the following question: "Assuming that the usage of a service changes in such a way that we need to reduce the communication overhead (for example, because more and more mobile clients are consuming it), can our monitored data help us decide which of the re-deployment options that our extensions enable is the most adequate?". It is reminded that Apache Tuscany offers various different types of bindings, but let us abide by our example and prove how the adequacy of RMI and compression depend on other data that could be captured by our platform monitor. In order to demonstrate this, many different experimental setups were possible. However, an interesting comparison is presented, for which no equivalent experimental results were found in literature. This is obviously because the interest in such comparisons is much bigger in the case of self-adaptive SOA platforms than in any other case.

The experimental setup is as follows:

- Two Web services were tested. One sends responses with complex types (a List of complex objects), the other sends responses with single types (a String of varying size).
- The size of the data in the response messages has been varied from 1 to 1000000 bytes (X-axis). In the case of the complex data, the minimum size was ca. 2000 bytes (= size of one Object).
- The two services were called directly with SOAP communication, as well as with the two alternative access methods, i.e., with the RMI protocol and with compression.
- The reduction of the data needed for the transmission of the responses was measured in all cases and was expressed as the size of the "reduced" response divided by the size of the original SOAP response (Y-axis). As mentioned, this overhead reduction is usually unimportant for strong workstations with great connections, but it may be critical for constrained mobile clients [23]. As nicely described in [24], this gap will continue to exist. Even the latest analyses of future technologies

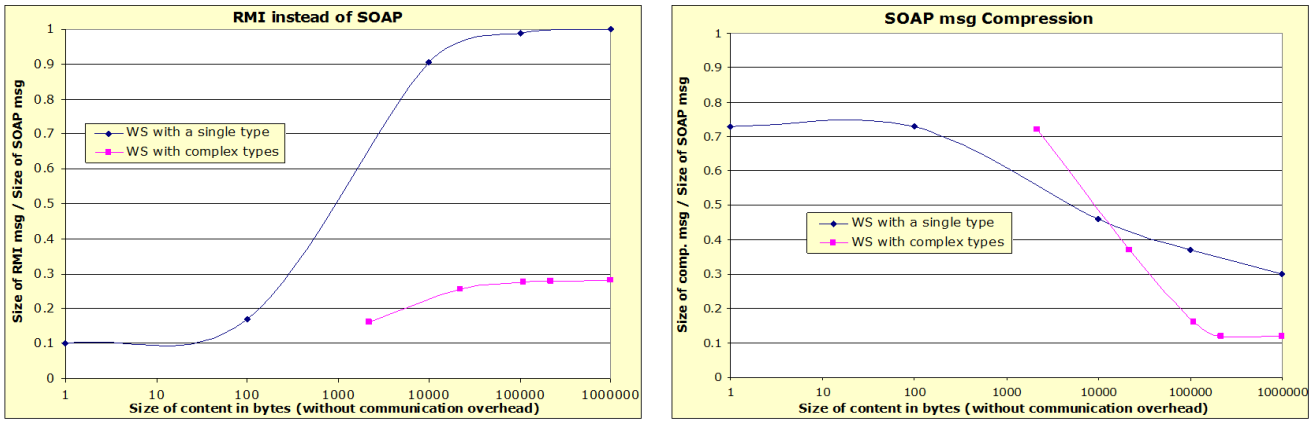


Figure 11. Overhead reduction with different protocol adaptations under varying conditions

for wireless communications strengthen this argument. In the book of Sesia et al. [25] about LTE (Long Term Evolution of 3G mobile networks), 5 categories of user equipment are defined, with smartphones being placed only under the second or third category. According to this categorization, devices of higher categories will be able to use wireless internet connection rates that are up to 6 times bigger. Of course, the wired connections of the future will be even faster than that, not to mention the fact that devices less capable than smartphones will be able to consume Web services. So, the big differences of device capabilities and connection qualities will maintain the need for adaptation and the overhead reduction shown in our experimental results will be always important, as the size of the data that is processed and wirelessly transmitted is growing parallel to all other technological developments.

No detailed analysis of the results shown in Figure 11 is necessary for our qualitative evaluation. The results show unambiguously that the two techniques perform differently under different conditions. For example, compression reduces the overhead significantly for single-typed big data, while the opposite is true for RMI. The conditions (data sizes and data types, in this example) can be perfectly captured by our platform monitor and exploited by a developer-defined adaptation logic. Concerning the exact logic, i.e., in order to answer the question "which adaptation action should be taken under which conditions?", further experiments including all influencing aspects are needed and, of course, the application-specific requirements, as well as the developer preferences, play an important role. A corresponding decision support is an interesting area of research and is a subject of our future work.

VII. Conclusion

In this work, a concept for distributing the core parts of a service platform and enriching them with self-adaptation mechanisms in order to offer fault-tolerance and higher service availability has been presented. Based on a prototypical implementation of our concept, the mentioned enhancements were shown primarily through an evaluation scenario where service availability was measured for the original and the extended platform. The prototypical implementation was done as an extension of the state-of-the-art SCA platform, Apache Tuscany. In addition to the availability measurements, further possible enhancements through different adaptation actions were explained through a qualitative evaluation. In the following, we mention some limitations of our approach, as well as further aspects that we see as subject of future work.

First, security aspects become more critical, because of the further capabilities that simple nodes have now. Lack of control upon them is more dangerous when they carry platform instances than when they simply host applications services. Moreover, the complexity of the distributed implementation, as well as the fact that statefull services cannot be easily replicated or migrated, lead to some limitations concerning the applicability of our mechanisms.

However, the most important incentives for further research can be found in the qualitative evaluation that has been presented. There, it has been explained how the diversity of the users of the platform can lead to the need for different adaptation actions. As an example, mobile clients have been mentioned. On this basis, it must be researched how the different possible adaptation actions match different situations, so that new decision algorithms can be integrated in the logic of a self-adaptive SOA platform, such as the one presented in the work at hand.

VIII. Acknowledgments

This work is supported in part by the E-Finance Lab e. V. (www.efinancelab.de) and the BMBF-sponsored project SoKNOS (www.soknos.de). We would also like to thank Steffen Lortz for his participation in our implementation.

REFERENCES

[1] Apostolos Papageorgiou, Tronje Krop, Sebastian Ahlfeld, Stefan Schulte, Julian Eckert, and Ralf Steinmetz. Enhancing Availability with Self-Organization Extensions in a SOA Platform. In *International Conference on Internet and Web Applications and Services (ICIW 2010)*, pages 161–166. IARIA, 2010.

[2] M. P. Papazoglou and W. J. Heuvel. Service-oriented Architectures: Approaches, Technologies and Research Issues. *The VLDB Journal*, 16(3):389–415, 2007.

[3] Rainer Berbner, Michael Spahn, Nicolas Repp, and Ralf Steinmetz. Heuristics for QoS-aware Web Service Composition. In *International Conference on Web Services (ICWS 2006)*, pages 72–82. IEEE, 2006.

[4] Dieter Schuller, Apostolos Papageorgiou, Stefan Schulte, Julian Eckert, Nicolas Repp, and Ralf Steinmetz. Process Reliability in Service-Oriented Architectures. In *Third IEEE International Conference on Digital Ecosystems and Technologies (IEEE DEST 2009)*, pages 640–645. IEEE, 2009.

[5] G. Canfora, M. Di Penta, R. Esposito, and M. L. Villani. QoS-aware Replanning of Composite Web Services. In *International Conference on Web Services (ICWS 2005)*, pages 121–129. IEEE, 2005.

[6] Apostolos Papageorgiou, Stefan Schulte, Dieter Schuller, Michael Niemann, Nicolas Repp, and Ralf Steinmetz. Governance of a Service-Oriented Architecture for Environmental and Public Security. In *Fourth International ICSC Symposium on Information Technologies in Environmental Engineering (ITEE 2009)*, pages 39–52, 2009.

[7] Dirk Krafzig, Karl Banke, and Dirk Slama. *Enterprise SOA*. Pearson Education, 2005.

[8] Jorge Salas, Francisco Perez-Sorrosal, Marta Patio-Martnez, and Jimnez-Peris. WS-Replication: a Framework for Highly Available Web Services. In *15th international conference on World Wide Web (WWW 2006)*, pages 357–366. ACM, 2006.

[9] OASIS. openCSA Specifications for the Service Component Architecture (SCA), 2007. http://www.oasis-opencsa.org/sca, last accessed on July 2010.

[10] Apache Software Foundation (ASF). Apache tuscany project, 2009. http://tuscany.apache.org/, last accessed on July 2010.

[11] E. Gjorven, R. Rouvoy, and F. Eliassen. Cross-layer Self-adaptation of Service-oriented Architectures. In *Third Workshop on Midleware for Service Oriented Computing (MW4SOC 2008)*, pages 37–42, 2008.

[12] G. Tosi, G. Denaro, and M. Pezze. Towards Autonomic Service-Oriented Applications. *International Journal of Autonomic Computing*, 1(1):58–80, April 2009.

[13] V. Issarny, M. Caporuscio, and N. Georgantas. A Perspective on the Future of Middleware-based Software Engineering. In *IEEE International Conference on Software Engineering (ICSE 2007), Proc. of the Workshop on the Future of Software Engineering (FOSE 2007)*, pages 244–258. IEEE, 2007.

[14] W. Bradley and D. Maher. The NEMO P2P Service Orchestration Framework. In *37th Annual Hawaii International Conference on System Sciences (HICSS 2004)*, page 90290.3. IEEE, 2004.

[15] D. Galatopoulos D. Kalofonos and E. Manolakos. A P2P SOA Enabling Group Collaboration through Service Composition. In *Fifth International Conference on Pervasive Services (ICPS 2008)*, pages 111–120. ACM, 2008.

[16] C. Rathfelder and H. Groenda. iSOAMM: An Independent SOA Maturity Model. In *8th IFIP International Conference on Distributed Applications and Interoperable Systems (DAIS'08)*, pages 1–15. IFIP, 2008.

[17] Julian Eckert, Marc Bachhuber, André Miede, Apostolos Papageorgiou, and Ralf Steinmetz. Readiness and Maturity of Service-oriented Architectures in the German Banking Industry - A Multi-Participant Case Study. In *IEEE International Conference on Digital Ecosystems and Technologies 2010 (IEEE DEST 2010)*. IEEE, 2010.

[18] SoKNOS project. Service-oriented Architectures Supporting Networks of Public Security. http://www.soknos.de, last accessed on July 2010.

[19] D. C. Luckham. *The Power of Events: An Introduction to Complex Event Processing in Distributed Enterprise Systems*. Addison-Wesley Longman Publishing Co., 2001.

[20] Antony Rowstron and Peter Druschel. Pastry: Scalable, Decentralized Object Location and Routing for Large-Scale Peer-to-Peer Systems. In *IFIP/ACM International Conference on Distributed Systems Platforms (Middleware)*, pages 329–350, 2001.

[21] Ralf Steinmetz and Klaus Wehrle. *Peer-to-Peer Systems and Applications*. Springer Verlag, 2005.

[22] Apostolos Papageorgiou, Jeremias Blendin, André Miede, Julian Eckert, and Ralf Steinmetz. Study and Comparison of Adaptation Mechanisms for Performance Enhancements of Mobile Web Service Consumption. In *The 6th IEEE World Congress on Services (SERVICES '10)*, pages 667–670. IEEE, 2010.

[23] M. Tian, T. Voigt, T. Naumowicz, H. Ritter, and J. Schiller. Performance Considerations for Mobile Web Services. *Computer Communications*, 27:1097–1105, March 2004.

[24] C. Canali, M. Colajanni, and R. Lancellotti. Performance Evolution of Mobile Web-based Services. *IEEE Internet Computing*, 13:60–68, March 2009.

[25] Stefania Sesia, Issam Toufik, and Matthow Baker. *LTE, The UMTS Long Term Evolution: From Theory to Practice*. Wiley Publishing, 2009.

Web and Distributed Software Development Risks Management: *WeDRisk* Approach

Ayad Ali Keshlaf
School of Computing Science
Newcastle University
Newcastle, UK
a.a.a.keshlaf@ncl.ac.uk
ayadali2005@gmail.com

Steve Riddle
School of Computing Science
Newcastle University
Newcastle, UK
steve.riddle@ncl.ac.uk

Abstract—**In spite of a variety of software risk management approaches, the software industry is still suffering from associated risks. Web and distributed software development is an example, where there are specific challenges and risk areas, which need to be addressed, considered and managed. In this paper we present a list of potential web and distributed risks, which we have identified based on their challenges and characteristics. We survey a number of software risk management approaches and identify their weaknesses and strengths for managing web and distributed development risks. Examples of weaknesses that we identify include the treatment of cultural issues, geographic location, and process and product perspectives. The identified strengths are quite general and only few of them are targeted to web and distributed developments. Following the review of strengths and weaknesses we present an approach called *WeDRisk*, which we propose in order to tackle the weaknesses of the existing approaches, and to accommodate the continuously evolving challenges to web and distributed software development. *WeDRisk* tries to cover some aspects and perspectives, which have not been covered up to now.**

Keywords-software risk management; web development; distributed development; software reliability; WeDRisk approach

I. INTRODUCTION

Software development projects are, by their nature, a risky, complicated and multi-dimensional endeavor [1][2][3][4]. Software risks have been increasing for as long as the software industry has been growing [5]. Many software development projects miss their goals of delivering acceptable software products within agreed constraints of time, budget and quality, due to a combination of the risks themselves, and absent or poor Software Risk Management (SRM) [6][7]. SRM is still evolving, and many software managers have only a limited understanding of its concepts [4]. Industrial risk management practice tends to lag behind recommended risk management best practice, although there are exceptions [4][8][9]. This lag is clearer with Web and Distributed (W-D) software development, where the level of SRM practice is still low.

This paper investigates the abilities of existing SRM approaches to manage W-D software development risks, and to explore their weaknesses, and proposes a novel approach, *WeDRisk*, in order to address the identified weaknesses. The rest of the paper is structured as follows. Section II provides a background on SRM, Section III explores W-D development challenges and their sources of risks. Section IV provides a list of some potential risks to W-D development. We then review the existing SRM approaches (Section V), comparing them based on specific criteria factors (Section VI) in order to investigate their abilities to manage W-D development risks. Section VII introduces the *WeDRisk* approach, which we propose in order to tackle the weaknesses of existing approaches in managing W-D development risks. We then present our conclusions and propose future work in Section VIII.

II. BACKGROUND

This section gives a background of SRM and its related definitions.

A. Software Risk

The Software Engineering Institute (SEI) defines **risk** as "the possibility of suffering loss [10]" and it defines **loss** in a development project, as "the impact to the project, which could be in the form of diminished quality of the end product, increased costs, delayed completion, loss of market share, or failure [10]".

For each risk there are two aspects: risk probability and risk loss. These aspects are used to estimate the impact or Risk Exposure (RE) [11], as follows:

$$RE = P(UO) \cdot L(UO) \quad (1)$$

where,

RE is the Risk Exposure (or risk impact),
P(UO) is the probability of an unsatisfactory outcome, and
L(UO) is the loss associated with an unsatisfactory outcome.

Risk probability estimation is not a straightforward task and can not be 100% accurate (as otherwise there is no risk). Some probability estimation techniques use qualitative data and then convert it into its equivalent quantitative data using some equations, risk-probability table, checklists or relative

scales [6, 11] where some others use subjective Bayesian approach [12] or other techniques.

The top ten software risk items (listed below), which are introduced by Boehm, are examples of sources of risk for software development projects [11].

- Personnel Shortfall
- Unrealistic Schedules and Budget
- Developing wrong software functions
- Developing wrong user interface
- Gold Plating
- Continuing stream of requirements change
- Shortfalls in externally furnished components
- Shortfalls in externally performed tasks
- Real-time Performance Shortfalls
- Straining Computer-science capabilities

A further list of software risk items includes the following risk items [13]:

- Bad traceability
- Insufficient verification and validation
- System complexity
- Customer unsatisfied at project delivery
- Risk reducing technique producing new risk
- Catastrophe/Disaster

Any list of software risk items will need to be updated from time to time, when there are new changes or challenges in software development technology and environment (e.g., social and culture issues, geographically dispersed, new technologies). The significance and type of risks and their sources will also inevitably evolve over time. As an example a recent review [14] found that different authors have identified or proposed different software risks, which means that the number and items of software risks are not fixed. Therefore, new or improved methodologies, techniques and tools to identify, measure and control them are needed.

B. SRM

Boehm [15] defined SRM as "a discipline whose objectives are to identify, address, and eliminate software risk items before they become either a threat to successful software operation or major sources of software rework". Figure 1 shows the basic steps of SRM [11]

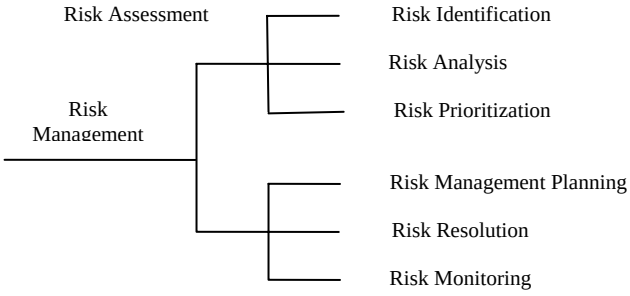


Figure 1. SRM Basic Steps [11]

The main purpose of SRM is to identify potential problems of technical and management aspects before they occur and then take actions to decrease their impact [16].

C. Software Development Perspectives

Software development has three perspectives: project, process and product [17][18]. Looking at these perspectives it is expected that each one of them includes, or could be affected by, different types of risks. For example, the "personnel shortfalls" risk item mainly affects the project perspective, "bad traceability" and "poor testing" affects process whereas "product with wrong functionality" affects product. However, one risk item may affect more than one perspective. Risk management is becoming an important issue from these three perspectives [17][18].

III. CHALLENGES

A number of challenges to traditional software development can be seen in the fields of distributed and web development. The following section focuses on these challenges.

A. Distributed Development Challenges

Distributed Software Development as described by Jimenez and others [19] is a type of development that "allows team members to be located in various remote sites during the software lifecycle, thus making up a network of distant sub-teams". Distributed software projects are usually developed by teams working collaboratively via communication channels (e.g., networks, internet, emails) across many locations. Software developers have adopted distributed software development as a way of reducing the cost and increasing their projects productivity [20]. Developing software across distributed sites presents many challenges, which are summarized in the following points [21][22]:

- Inadequate informal communications
- Lack of trust
- Culture differences (e.g., different language, different corporate culture and different developers' background)
- Time-zone difference (leading to ineffective synchronous communication)
- Development process differences
- Knowledge management challenges (most of the existing management approaches are designed for co-located teams).
- Technical issue: Incompatible data formats and exchanges.
- Security issue (Ensuring electronic transmissions confidentiality and privacy).

All of these challenges could be sources of risk in a variety of development types. In the case of distributed development, they are particularly prevalent challenges and need to be considered by any proposed risk management approach.

B. Web Development Challenges

Web applications are a typical example of web developments, which have become a common type of modern software application. Mendes [23] defines a web application as "an application delivered over the Web that combines characteristics of both Web Hypermedia and Web Software application".

Web applications may be deployed instantly worldwide, without any need for installation or upgrading manuals [24]. They are growing very fast compared with the traditional software, which makes them an important part of the business and software industry. High-performance web sites and applications are used widely in business-to-business ecommerce and many types of services as fully functional systems [25][26].

The development, running and deployment environment of web development need to be considered carefully as well as the significance of associated challenges and risks. Features of the W-D environment, such as diversity and rapid change, present new challenges for the developer, manager, and to traditional project management approaches [26][27[28][29]. More effective risk management methods, models and tools should be introduced to tackle the lack of existing approaches to deal with these challenges [9][30][31].

The importance of web risks is different from others in a number of ways:

- Their impact and significance are different. For example the exposure to security threats is higher in the web [32][33][34]35].
- As web applications may be deployed instantly worldwide [24], their risks can affect a wider range of components and applications simultaneously in a very short period of time.
- Additional risk sources related to W-D environment include communication, culture, diversity and geographical location [36][37][38][39].
- Estimation of risk probability and loss is more difficult because of the involved challenges and relative lack of experience with them.

Ideally, assessment and management of web development risks should be performed during the whole life cycle of the projects [40], but unfortunately, many web developers use a reactive risk strategy (they do not act until something goes wrong). This strategy is insufficient because it makes software projects vulnerable to any type of risks at any time without effective assessment and control [41].

There is no way to avoid risks in W-D development, so (as with other types of risk) the solution is to attempt to manage them. The following section gives an overview on the state of the art of existing SRM approaches and illustrates their strengths and weaknesses.

IV. W-D POTENTIAL RISKS

The challenges and characteristics of W-D development could bring many risks to W-D development. Some potential risks to W-D developments are listed in Table I [19][21][22][23][24][25][27][32][38][39][40][41][42][43][44]. The list of risks is not final, and could be updated when there are any new challenges or environment changes. Any co-located software risks are also considered risks to W-D development, although their impact and significance could be different.

TABLE I. W-D POTENTIAL RISK ITEMS

SN	Risk Item
1	Unfamiliarity with international and foreign contract law
2	Volatile customer requirement
3	Poor documentation
4	Low visibility of project process
5	Inadequate / inappropriate process development
6	Not enough measurement and estimations
7	Lack of security precautions
8	Weaknesses in protection procedures for Intellectual Property rights
9	Vendor feasibility (weaknesses)
10	Insufficient competence
11	Communication failures
12	Poor sites management control
13	Failure to manage user expectations
14	Insufficient project stakeholder involvement
15	Process instability
16	Poor performance
17	Poor UI (rapid changes)
18	Insecure communication channels
19	Inadequate user involvement
20	Difficulties in ongoing support and maintenance
21	Unrealistic estimation of the number of users
22	Differences in the development methodologies and processes
23	Weak or inadequate contracts
24	Complicated development dependencies between project sites
25	Cross cultural differences / influence
26	Poor product functionality
27	Market fluctuations
28	Scalability limitations
29	Poor availability
30	Lack of top management commitment
31	Instability in other project sites
32	Lack of Face-To-Face meetings
33	Lack of Management availability and efficiency
34	Unfamiliarity with customer type
35	Constraints due to time zone differences
36	Culture Influence
37	Not enough experience with the W-D technologies

Another type of risks that could also affect the W-D developments is the atypical risks type. Atypical risks are risks that could not be predicted before they occur.

V. SRM APPROACHES

There are many different SRM approaches. Some of these approaches are named "models" and others are named "frameworks" or "methods", but they have the same target, which is managing software risks.

Existing SRM methods, models and tools are reviewed in this section. Each of the approaches uses some steps, components or techniques, which may be different or have some similarities with other approaches.

A. Existing Approaches

Nine of the existing approaches have been selected for detailed comparison in this study. The selected approaches are the ones that we expect to satisfy the needs of risk management for software industry in the W-D development environment. The approaches were selected because they are dedicated to manage W-D development risks, or related aspects. The compared approaches are described hereafter.

a) DS-RM-Concept:

Distributed Software - Risk Management Concept (DS-RM -Concept) has been designed based on the idea that communication and continuous risk assessment play a vital role in managing the risks. Risk assessment in this approach uses three concepts: reviews for risk identification; snapshots for analysis; and reports for assessment [45].

b) EBIOS Methodology:

Originally the EBIOS (In French: Expression des Besoins et Identification des Objectifs de Sécurité) method was introduced by Central Directorate of Security of Information Systems (DCSSI) in the French government. It is a risk management methodology concentrating on Information Systems Security (ISS) risks. It consists of a set of guidance steps and it is supported with a free open source software tool. The methodology has five phases: Context Study; Security Requirements Checklist; Threats Study; Identification of Security Objectives and Determination of Security Requirements [33][46]. W-D developments are highly vulnerable to security risks and EBIOS methodology is widely used in government and private sectors to manage such type of risks, as it is supported by an open source tool.

C) ProRisk Framework:

ProRisk is an open system where the users can develop, calibrate a choice from published models (templates) or use different models to accommodate their project need. It is a risk management framework for small and large software projects. However, in order to provide project risk factor a detailed analysis of the project is required [47].

d) Riskit Method:

Riskit method is a SRM method introduced by Jyrki Kontio [48]. Figure 2 shows the process diagram of the method, which is designed to provide organized SRM process and to support involvement of all relevant stakeholders in risk management process [49]. The method is provided with analysis graph and it uses a specific ranking technique called Riskit Pareto Ranking Technique, which uses probability and utility loss ranking [50][51].

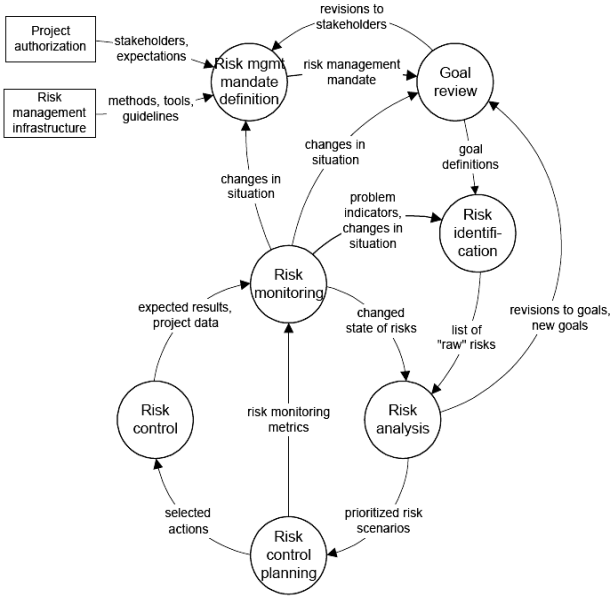


Figure 2. The Process Diagram of Riskit Method [48]

e) SoftRisk:

SoftRisk is model to manage software development risks introduced by the author and others [6]. Figure 3 shows the main steps of SoftRisk model [6][31][52].

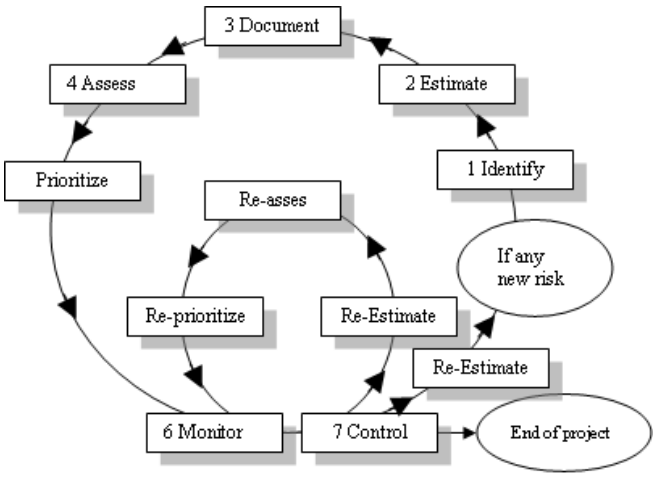


Figure 3. The Main Steps of SoftRisk Model [6]

The model is designed based on the idea of documenting and using historical risk data and focusing on top risks in order to reduce the effort and time in managing software risks. The model has been supported with a prototype tool.

f) CMMI-RSKM:

Capability Maturity Model Integration (CMMI) is an approach for improving processes within organization. The guidance, which is provided by CMMI consists of a group of steps to improve development management, services, and maintenance of products. CMMI has RiSK Management (RSKM) process area and it has been adopted worldwide by many organizations. Its models cover development, acquisition, and services in projects [51][53][54][55].

g) PMBOK RM Process:

Project Management Body of Knowledge (PMBOK) is a process introduced by Project Management Institute (PMI). Its third edition was published in 2004. The PMBOK combines nine areas of knowledge (Integration, scope, time, cost, quality, human resource, communications, purchase and risk). It consists of four process phases - Initiating, Planning, Executing, and Closing. It can be considered as standard for Project Management [51][56][57].

h) GDSP RM Framework:

Geographically Distributed Software Projects (GDSPs) is an integrated framework to manage risks in distributed software projects. It emphasizes many aspects, which are shared between GDSPs and web application developments. The idea behind this framework was based on synthesizing some known risks and risk techniques into integrated approaches. GDSPs links resolution techniques into project risk areas [39]. Elements of the framework are illustrated in Figure 4.

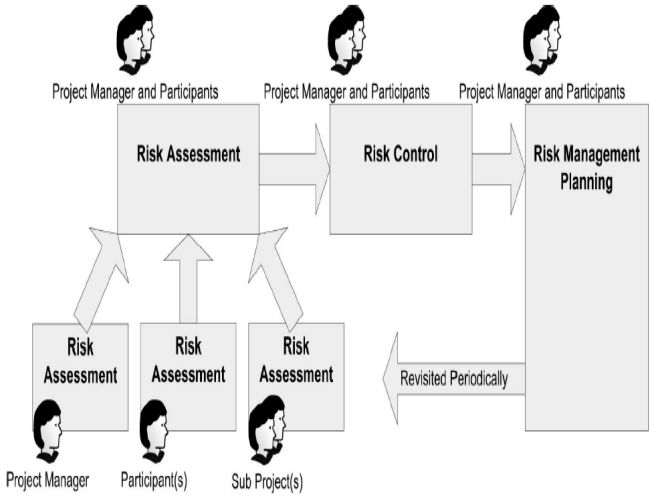


Figure 4. Elements of GDSP's Risks Management Framework [39]

i) Risk and Performance Model:

This model is designed to inspect the relationship between risk and project performance. This includes product and process performance. For this purpose six dimensions (*Organizational Environment*, *User*, *Requirements*, *Project Complexity*, *Planning & Control and Team risk*) of software risks are used by the model [58].

VI. ANALYSIS

The approaches were reviewed for their ability to manage risks of modern software development under the W-D environment and how they can deal with their challenges. In order to see their weaknesses and strengths, a comparison between them has been conducted based on our predefined criteria factors.

The criteria factors were prepared after the challenges, risk areas and characteristics of W-D development were identified, by conducting a risk management practice survey and literature search [16][17][18][20][21][22][23][24][25][28][29][30][31][32][33][34][35][36][37][38][39][40][41]. In order to get a consistent list of criteria factors initially, a list of all criteria factors has been created and then the most related ones to W-D software development were filtered. Meanwhile, some other factors are specified in order to cover aspects that we felt that were not touched before.

The factors cover important risk management aspects (e.g., Perspectives, Communications, Geographically Dispersed, Evolving Environment, Risk Management Evolution, culture issue and Interoperability tracking).

The comparison has been conducted based on available literature such as papers, reports, previous comparison, formal websites of the approaches and related technical reports (references are mentioned in Section V). Table II shows the result of the comparison.

In Table II there are three options for each criteria factor:

- ✔ *when the factor is supported or agreed by the approach.*
- ✕ *if the factor is not supported or not agreed by the approach.*
- **P** *if it is partially supported or partially agreed by the approach.*

Table II can be read either horizontally or vertically. If it is read horizontally then the numbers on the table represent the total of points that each criteria factor has got from all of the approaches for each one of the above three options. If the table is read vertically then the numbers represent the total of points each approach has got for each one of the above three options.

TABLE II. SRM APPROACHES COMPARISON RESULT

Criteria Factors \ Approaches	DS- RM Concept	EBIOS	ProRisk	RisKit	SoftRisk	CMMI-RSKM	PMBOK	GDSP-RM	Risk&Performance	Sub Totals: ✔	✘	P
Perspectives:												
- Project	✔	✘	✔	✔	✔	✔	✔	✔	✔	8	1	
- Process	✘	P	P	P	✘	✔	✔	✘	✔	3	3	3
- Product	✘	P	✘	✘	✘	P	✘	✘	P		6	3
Stakeholder :												
- Involved Stakeholder	✔	P	✔	✔	P	✔	✔	P	P	5		4
- Stakeholder Roles in SRM	P	P	P	P	P	✔	P	P	P	1		8
SRM & Product Quality Link	✘	✘	P	✔	✘	P	✔	✘	✔	3	4	2
Remote SRM	P	✘	P	✘	P	✘	✘	✔	✘	1	5	3
Estimating SRM Cost	✘	✘	P	P	✘	P	P	P	✘		4	5
Provided/Suggested Options :												
- Communications	✔	✔	P	✘	✘	✔	✔	✔	✘	5	3	1
- Collaboration	P	✘	✘	P	✘	✘	✘	P	✘		6	3
Consideration of:												
- Geographically dispersed	✔	✘	✘	✘	P	✘	✘	✔	✘	2	6	1
- Social and legal issues	✘	✔	✘	✘	✘	✘	✘	P	P	1	6	2
- Intellectual property	✘	✘	✘	✘	✘	✘	✘	✘	✘		9	
- Ethical issues	✘	✔	✘	✘	✘	✘	✘	✔	✘	2	7	
- Multicultural environment	✘	✘	✘	✘	✘	✘	✘	✔	✘	1	8	
- Evolving environment	✘	✘	✘	✘	✘	✘	✘	✘	✘		9	
Preparedness to Atypical Risk	✘	✘	✘	✘	✘	✘	✘	✘	✘		9	
Provided SRM Types:												
- Plain	✘	✘	✘	✘	✘	✘	✘	✘	✘		9	
- Deep / Ordinary	✔	✔	✔	✔	✔	✔	✔	✔	✔	9		
SRM Evolution Ability	✘	✘	P	✘	✘	✘	✘	✘	✘		8	1
SRM Effect Evaluation	P	✘	P	✔	P	✘	P	P	P	1	2	6
Learning from Mistakes	✔	✘	✘	✘	✔	✘	P	P	✘	2	5	2
Performance Evaluation	P	✘	P	✔	P	✔	✘	P	✔	3	2	4
Acceptable Levels	✘	✘	P	P	✔	✔	✔	✔	✘	4	3	2
Risks of SRM Exploration	✘	✘	✘	✘	✘	✘	✘	✘	✘		9	
Prediction Techniques	✔	P	P	✔	P	✘	✘	P	P	2	2	5
Side Affect Absorber	✘	✘	✘	✘	✘	✘	✘	✘	✘		9	
Interoperability Tracking	✘	✘	✘	P	✘	✘	✘	P	✔	1	6	2
Dependences Tracking	P	✘	P	P	✘	✘	P	P	P		3	6
Virtual SRM support	P	✘	✘	✘	P	✘	✘	✔	✘	1	6	2
Standard Operation Procedures	✘	✘	✘	✘	✘	✘	✘	✘	✘		9	
Risk Source Tracing	✘	✘	✘	✘	✘	P	✔	✘	✘	1	7	1
Totals :												
✔ Supported or agree	7	4	3	7	4	8	8	9	6	56		
✘ Not Supported or not agree	18	23	17	18	20	20	19	12	19	166		
P Partially Supported or partially agree	7	5	12	7	8	4	5	11	7	66		
Total:										288		

From the numbers that appear in Table II it can be noticed that the total number of criteria factors that are supported or agreed by the approaches has got 56 points from the total of points, which is 288 (with percentage 19%) The ones that are partially supported or partially agree have got 66 points (with percentage from the total of points 23 %) whereas the factors that have got the lowest support by the existing approaches have got the highest number of points, 166 (with percentage 58%). The criteria factors that have got the lowest support are:

- Covering of process and product perspectives
- Consideration of: Geographically dispersed, Social and legal issues, Intellectual property, Ethical issues, Multicultural environment and Evolving environment
- Preparedness for atypical risks
- Plain risk management type
- Evolution of SRM processes
- Exploration of SRM Risks itself

- Risks side affects absorber mechanism
- Risks interoperability tracking
- Standard Operation Procedures

As can be seen in Table II, the points are different from one approach to another. This means that a weak aspect in one approach could be a strong aspect in another one. This is clear from the total points at the end of each approach. On the other hand there are many similarities between many approaches in many aspects as they have the same selections for some criteria factors. Table III summarizes the main strengths and weaknesses of the approaches, from the W-D point of view.

TABLE III. SOME STRENGTHS AND WEAKNESSES OF EACH APPROACH

Approach	Strengths	Weaknesses
DS-RM-Concept	• Targeting of distributed software. • Focusing on communications role. • Supports the use of risks database. • Supported with Risk Guide tool. • It has an effective identification technique.	• It does not consider some aspects such as social, multicultural, and evolving environment. • Lack of risk controlling. • It does not link risk management to development processes and product.
EBIOS Methodology	• Supported with an open source tool. • Its consideration of technical entities and non-technical entities. • Compliance with some standards (ISO27001:2005).	• It is dedicated and limited to Information Systems Security (ISS) risks only. • It has a very limited ability to consider aspects of W-D development environment.
ProRisk Framework	• Can be applied to small and complex projects. • It is open system. • It links business domain to risk management. • It is partially considers the cost in risk management.	• It requires detailed risk analysis. • It depends on other models to perform the risk analysis, which sometimes are not validated enough or not available to the users. • It does not consider most of aspects that are related to W-D environment.
Riskit Method	• It provides conceptual and graphical tool. • It defines project goals based on certain steps.	• It is not supported with risk communication channel. • Other weaknesses can be seen in Table II.
SoftRisk	• It supports risk documentation. • It switches between qualitative and quantitative data. • It is provided with checklist for risks estimation.	• It does not support risk communication. • It does not provide management for product perspective. • Other weaknesses can be seen in Table II.
CMMI-RSKM	• It supports the standardizations in risk management. • It is provided with a sort of guidelines.	• It supports only heavy risk management. • Project managers play most of risk management role. • Many aspects that are related to W-D environment are not considered.
PMBOK RM Process	• It considers the processes of software development. • It includes risk management as a part of project management.	• It is generic to meet some special needs of software projects. • Project managers play most of risk management role. • It does not support many features related to W-D development like consideration of remote risk management, social issues.
GDPS RM Framework	• Consideration of geographically dispersed • It supports categorization of risk areas, risk factors and resolution techniques.	• It uses a predefined list of risk areas and factors, which limits risk identification process. • No integration between risk management and overall project plan. • It does not consider process and product perspectives. • It provides only one type of management. It does not provide plain management.
Risk and Performance Model	• It comes with six dimensions of software risks. • It treats the relation between risk and performance.	• It does not give guidelines for managing risks. • It considers only internal risks. • Other weaknesses can be extracted from Table II.

In general, the associated weaknesses of existing approaches that have resulted from the comparison can be summarized in the following points:

- The existing approaches concentrate on project perspective of software development and they do not pay enough attention to other perspectives (Process and Product).
- They do not accommodate the continuous evolvement and changes issues of software industry and they do not consider aspects related to web, and distributed development environment

(e.g., geographical difference, time zones differences, intellectual property, culture issues, evolving environment etc.).
- Lack of preparedness to atypical risks (No absorbing mechanism for side affects of atypical risks).
- They do not suggest any effective mechanisms to monitor or trace risks interoperability and dependences.
- They are not flexible enough and they offer only deep type of risk management. Plain risk management is not offered.
- Not enough monitoring to SRM performance and its associated risks.
- Most of the approaches are focused on theoretical aspects and do not provide clear guidelines for practicing.

VII. WEDRISK APPROACH

WeDRisk is an approach we propose in order to tackle the weaknesses of existing SRM approaches with more emphasis on W-D development. While the approach is particularly aimed towards W-D development, it should be applicable to modern software developments in general. The general principles of this approach are:

- It is built to tackle the weaknesses of existing approaches, with some new improvements.
- It focuses on W-D software development, but it can be used for others.
- The approach is supposed to be flexible and able to evolve if need be.
- It considers risks from three perspectives (project, process, and product) and uses a modular approach structure of components, phases and layers to manage the complexity in the range of different weaknesses identified.

A. WeDRisk Structure:

The *WeDRisk* approach consists of five layers (Project Layer, Stakeholder Layer, Risk Management (RM) Customization Layer, RM Implementation Layer and Evaluation & Auditing Layer) and two supporter components (Communication & Plug-In Controller and RM Evolution Regulator).

The layers consist of components, which contain steps, techniques and guidelines. The supporter components provide the necessary support to the other *WeDRisk* components. Figure 5 illustrates the main architecture design of *WeDRisk* approach.

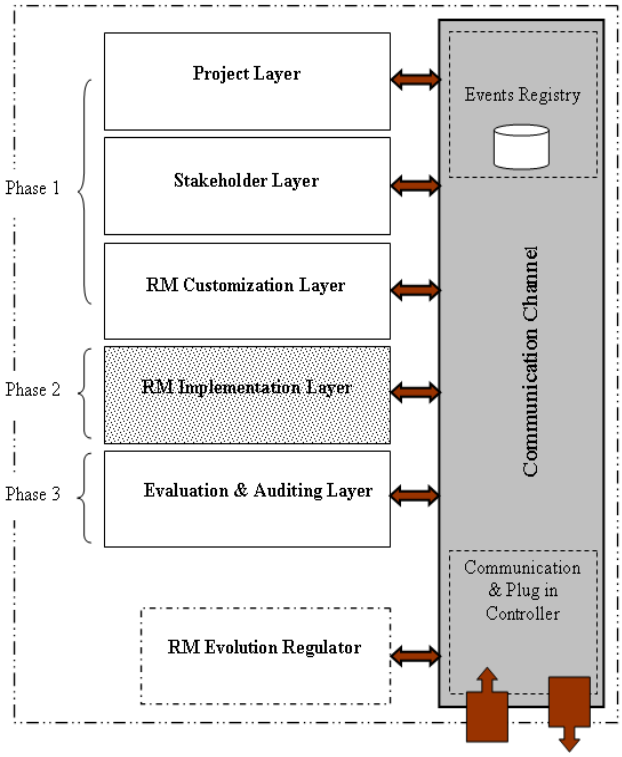


Figure 5. *WeDRisk* Main Architecture

This modular approach structure simplifies the *WeDRisk* design and makes it ready for evolving and integrating.

B. WeDRisk Run Phases & Layer Descriptions

Running phases of the *WeDRisk* consist of three main phases (see Figure 6). They are briefly described below with the appropriate layers that work under them.

First Phase: Establishing RM set-up:

This is an essential phase for RM establishment (set-up). It produces projects' and stakeholders' cards. As well as it customizes the type of RM (deep or plain type). The following layers work under this phase:

Project Layer	**P**roduces /updates Project Card
Stakeholder Layer	**P**roduces Stakeholders Cards
RM Customization Layer	**S**pecifies Management Type

Table IV shows an example of the project card. The data that is shown in the example is dummy data (not real).

TABLE IV. PROJECT CARD EXAMPLE (DUMMY)

Project ID	WP-09-001					
Opening Type	New Project (*may be updating an old one*)					
Project Name	Billing System for van hiring system					
Type	Web Application					
Customer	Newcastle Group					
Project Developer	Advanced SoftGroup Ltd.					
Project Manager	ALI					
Development Sites	One site ; Main Site (Newcastle)					
Development Team	3 Programmers + Editor + Graphics Designer					
Dev. Team Leader	John					
Planned Starting Date	01/04/2009	Planned Finishing Date:	30/04/2009			
Actual Starting Date	05/04/2009	Actual Finishing Data:	25/05/2009			
Initial Contract Cost	£100,000	Actual Cost at Delivery:	£177,000			
Requirement Specification Doc. File	WP-09-001-Req.Pdf					
Events Registry Ref. No	WP-09-001EventReg					
Dependency or Linked Projects	WP-09-201; DP-09-30					
All Project's Identified Risks						
Risk ID	Associated Loss	Responsible	Attack Date	Resolve Date	Attack TREV	Resolve TREV
R-Cu-011	5 days delay	Project Secretary	01/04/2009	04/04/2009	5.7	20.2
~	~	~	~	~	~	~
~	~	~	~	~	~	~
~	~	~	~	~	~	~
R-Cu-034	£500	Site 2 manager	12/05/2009	11/05/2009	4.5	10.5
Project's Current Identified Risks (Prioritized based on TREV)						
Risk ID	Associated Loss	Responsible	Attack Date	Prob.	Mag.	Attack TREV
R-Cu-011	Extra Cost	Programmer No. 1	01/04/2009	0.3	200	60.7
R-Te~	~	~	~	~	~	~
~	~	~	~	~	~	~
~	~	~	~	~	~	~
R-Ge-231						45.5

Second Phase: Implementation of RM Cycles:

The main RM operation/steps are implemented at this phase. The operations include the estimation, evaluation, planning and controlling of the risks.

At this phase risk cards (Table V shows a dummy example of the risk card) are produced for new risks. These cards contain all important identification data of the risks. The identified risks are clustered from their perspectives (Project, Process and Product). Project cards are continuously updated with current risks data. In case of any attack from atypical risks the absorber mechanism will be triggered. Extracting Learned Lessons and tracing dependencies and interoperability are also operations implemented under this phase using special components. The layer that works under this phase is the implementation layer.

Implementation Layer	**P**roduces Risks Cards; Estimates, evaluates, plans for and controls the risks; Deals with atypical risks and cluster risks from the three perspectives (Project, Process and Product), Extracts Learned Lessons traces risks dependencies & interoperability

Third Phase: Evaluation and Auditing

This phase is concerned with RM performance and RM cost evaluation. This required data is periodically collected about RM progression during RM cycle. Collected data cover RM Establishing cost, RM Running cost, Risks Consequences cost, RM durations time and RM efficiency. These data are used to monitor cost and performance of RM operations and it is used to produce RM performance report. It is also used to support evolution of the approach. The responsible layer for this phase is the Evaluation and Auditing Layer.

Evaluation & Auditing Layer	**M**onitors RM progress **P**roduces Performance Report

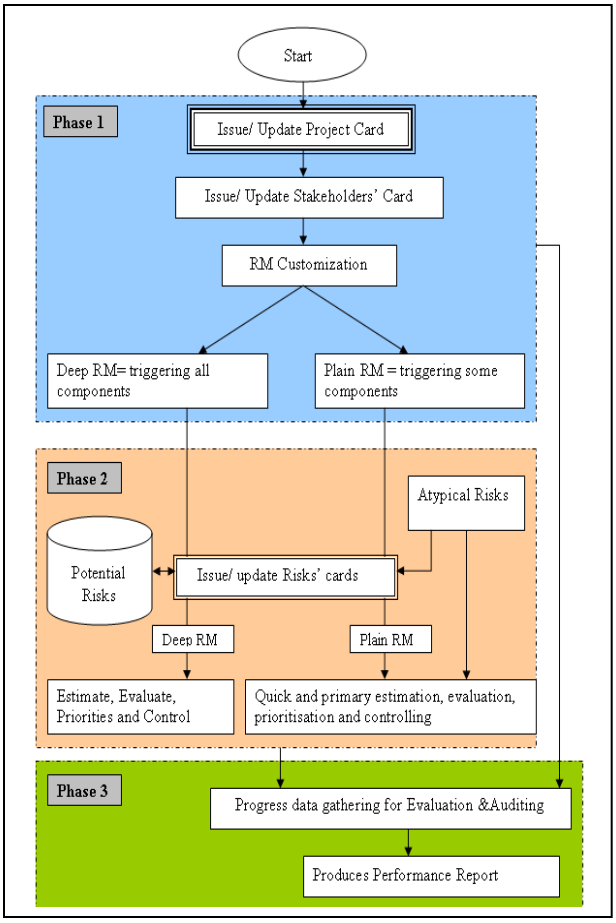


Figure 6: *WeDRisk* Running Phases

C. *WeDRisk Supporter Units*

The role of these units is to provide required support and services to *WeDRisk* components. There are two main units (Communication & Plug-in Controller and RM Evolution Regulator), which are described briefly hereafter:

Communication & Plug-in Controller**:**

The communication and Plug-in Controller works with all layers and at all phases. It consists of the following components:

Communications channel is a component used to ensure all needed communications between all RM layers during the RM cycle. Furthermore, this channel also ensures the exchanges of the RM data between all project sites.

Events Registry is a component used to record some important data about all RM events and actions during the RM cycle. The recorded data are considered as the history of RM implementation cycles, which could be used for statistics, performance monitoring or taking corrective actions.

Plug-in Components provides the support for connecting *WeDRisk* with other approaches. For that it provides standard format for data exchange and checks the permissions and authentications.

Communications Channel	**E**nsures communication for RM operations
Events Registry	**R**egisters RM process Events
Plug-in Component	**L**ink / communicate with others

RM Evolution Regulator**:**

The *Evolution Regulator* is responsible for making any evolutions (improvement or changes) to the RM process. The evolutions are based on needs, enhancement or in some cases as part of corrective actions, which are collected in a special repository called the Evolution Box, and then implemented after they get approval from Evolutions Approval Board. Evolution Regulator components and their roles are summarized below:

Evolution Box	Repository collects data about needs, problems and any evolution suggestions during the RM process
Evolution Approving Board	Evaluates the needs and take evolution decisions; specifies the modifications, priorities, responsibilities and schedules

TABLE V. RISK CARD EXAMPLE (DUMMY)

R. ID	R-Te-011
R. Name	Not enough experience with the W-D technologies
R. source	Programmer
Aspect	Technical Risks
Perspective	Process
Risk Description	The programmer supposed to have enough experience with Java and web services, but he has got stuck with some critical web services aspects.
Risk Factors	- The time is too short to learn web services. - Not enough time/budget to hire programmers - Not enough experience
Potential Impact	Extra Cost (e.g., it cost £3000 per a day for any delays)
Potential Affected Areas	Web related aspects
Dependability of Risks	Testing phase, product perspective
Mitigation Steps	Fast training course, postponing web service part, changing the type of the application or hiring programmer
Primary Precautions	Allocate some funds for hiring extra programmers
Controlling Steps	Hire extra programmers if the time is short, but if there is enough time and less dependency train the existing programmers.
Card Issue Date	18/11/2009
Relation Pointer (Linker)	In our case is null, which means that there is no any risk linked to this risk

D. WeDRisk Distinctiveness

WeDRisk tackles the existing approaches weaknesses by providing new components and covering new aspects to improve RM in W-D development. Although the *WeDRisk* approach is mainly designed for W-D development, it can also be used in the rest of software developments in general. The main contributions of *WeDRisk* are:

- It considers the three W-D perspectives (Project, Process and Product) as it clusters the risks from these three perspectives. This saves time and effort and increases the effectiveness of RM in W-D developments by making the concentration only on the risks of the appointed perspective.
- It provides an absorption mechanism to deal with W-D atypical risks.
- It considers the challenges and characteristics of W-D development since it provides a list of the potential risks that are associated with these characteristics and challenges. This helps to identify current risks faster and easier. This list of potential risks is updateable based on the current challenges and environment.
- The nature of W-D developments needs a flexible RM, therefore *WeDRisk* approach offers two types of RM (plain and deep).
- *WeDRisk* has been provided with an Events Registry component, which works as a log file, recording important events data during RM operation progression.
- Communication plays a vital role in managing W-D development risks. Therefore, the approach has a Communication and Plug-in Controller to ensure the internal communication (between approach components) via a communication channel, and external communication with other approaches via Plug-in unit.
- The approach includes W-D factors as a part of the risk estimation equation.
- Risks network is very complicated in W-D development projects. Combination of some risks could produce new risks or increase their severity. Meanwhile as many projects are multisite projects there is a dependency among them. *WeDRisk* treats this with a special component called Dependencies & Interoperability Tracer.
- *WeDRisk* is an evolutional approach as it has been designed to accommodate the evolutions in W-D developments.

E. Benchmarking

Comparing with other approaches, *WeDRisk* maintains the strengths of existing approaches and tackles their weaknesses in managing W-D risks. It designed to be an evolutionary approach. Table VI illustrates how *WeDRisk* approach comes with new features to improve the RM in W-D development.

TABLE VI. BENCHMARKING TABLE

Current Approaches	*WeDRisk* Approach
Perspectives Consideration	
The consideration is mainly on Project Perspective	It considers all perspective (project Product Process) and clusters the risks from all perspectives.
Evolution Ability	
They are fixed approaches	It is flexible to accommodate the W-D evolutions. It has a special component to handle that.
Offered RM types	
Usually they offer one type of RM, which is Deep RM type.	*WeDRisk* offers two types of RM (Deep and Plain). RM can be customized based on the situation needs, availability of resources and criticality of time.
Preparedness to atypical Risk	
None of them can deal with to atypical risk	*WeDRisks* has a mechanism to deal with atypical risks
W-D risk estimation and assessment	
Not enough consideration to W-D factors	It includes W-D factors at risks estimation equations
Dependencies & Interoperability	
Very limited and indirect ability	*WeDRisk* maps risks Dependencies and Interoperability
Auditing and Evaluation	
Limited in some of them	It has components for RM cost and performance evaluation
Learning from Mistakes	
Somewhat some of them have databases that can be used to learn from previous cycles	*WeDRisk* Extracts Learned Lessons from RM cycles
Communication	
Some of them has good communication channels	It has a communication channel supported with events registry and plug-in components

F. WeDRisk Evaluation

Currently, we are in the stage of evaluating the *WeDRisk* approach. For the evaluation purpose we have planned for two options, which are:

- Evaluating the whole approach (all components together) using one or more case studies.
- Dividing the approach into 'chunks' (representing the novel aspects in the *WeDRisk* approach) and then evaluating them one by one using case studies or experiments.

The preparation for the two options is currently in progress. A case study has been designed for the first option whereas, for the second option an experiment has been designed to evaluate the first 'chunk', which covers:

- Proposed list of W-D potential risks
- The usefulness/effectiveness of clustering the risks from the three perspectives (project, process product) and clustering criteria
- Potential of atypical risks in W-D development

Due to some difficulties in getting suitable projects where a case study can be executed, we plan to start with the second option (the experiment). The subjects for the first experiment are PhD and MSc students at School of Computing Science, Newcastle University, UK. The experiment was conducted in July/August 2010.

Other experiments or case studies will be designed to evaluate other novel aspects of *WeDRisk* approach and the evaluation results will be presented in the forthcoming papers.

G. The Prototype

It is expected that *WeDRisk* approach will lead to the development of a risk management tool. The tool will be targeted for use by W-D software development houses to manage W-D development risks. The tool functions are intended to comply with the proposed *WeDRisk* components and techniques. Currently the prototype of the tool is under construction. There are some challenges for the prototype implementation, which include:

- The prototype should cover important novel aspects of *WeDRisk*.
- It should be a web application and be able to deal with W-D multisite developments.
- It should be able to cover the three perspectives (project, process and product).
- It should be supported with a database for risks, projects and stakeholders cards data.

Implementing the prototype early could help in *WeDRisk* evaluation. However, in order to reduce rework in the implementation, commencing work on the prototype is dependent upon the completion of the first phase of the *WeDRisk* evaluation. We expect that the prototype could accelerate the rest of evaluation and validation phases and saves the time and effort. Moreover the result of evaluation can be considered as evaluation for both *WeDRisk* and the prototype, and can be used to improve both of them and to build a reliable tool based on the prototype. The work finished in the prototype implementation includes the design and creation of the supported database and building some main components. The prototype is expected to be ready in the middle of 2011.

VIII. Conclusion and Future Work

The paper has identified the challenges of W-D development and shown how the importance of risk in this context is different from others. A list of potential risks to W-D has been presented. The list is just an initial one, and should be updated from time to time when there are any new challenges or changes in the development environment. In order to investigate the weaknesses and strengths of existing approaches in managing the risks in W-D, the related existing SRM approaches have been reviewed and compared. The comparison is based on special criteria factors, which are prepared carefully in order to examine the ability of the approaches to manage the risks of W-D software development. The weaknesses and strengths of the compared approaches are identified in this paper. In general, most of the identified strengths are related to co-located development software and they are spread among the approaches.

It can be concluded that though there are many SRM approaches there is still a large gap between the existing approaches and actual practicing in software industry practice. This is due to the associated weaknesses in the approaches (e.g., not enough consideration to: difference in geographical locations, culture issues, process perspective and product perspective).

From Table II and Table III the following points can be concluded:

- There is no single approach that is able to manage software risks in W-D environments alone, unfortunately the strengths of the approaches are dispersed between them. In the current situation the developers either have to use more than one approach or miss some aspects and support.
- Tackling the weaknesses of the approaches and combining the strengths of them in a new approach is a step toward improving risk management in W-D environment.

For effective risk management in W-D development all challenges, characteristics, risk areas, development and running environment and development perspectives (project, process and product) and other related aspects must be considered.

The reviewed approaches have added significant value to traditional software development projects, but it is clear that the W-D developments are not yet well covered.

As a part of ongoing PhD research at School of Computing Science, Newcastle University, UK, the *WeDRisk* approach to manage W-D development projects risks has been presented in this paper. The approach aims to tackle the weaknesses in existing approaches and to propose new management concepts in order to improve the level of practicing of SRM in the field. While the approach is particularly aimed towards W-D development, it should be applicable to modern software developments in general.

The *WeDRisk* approach has been designed to satisfy the needs of risk management for W-D development. *WeDRisk* provides some contributions to manage W-D risks such as: the consideration of W-D risks from three perspectives (Project Process and product); involving specific factors for W-D as a part of risks estimation equations; providing a mechanism to deal with atypical risks; ability to evolve; mapping the dependencies and interoperability of the risks; managing risks across multisite projects; and reflecting W-D risks by providing an updateable list of W-D potential risks. In addition to W-D development *WeDRisk* is thought to be ready for serving other software development. The future work in this project includes more evaluation of *WeDRisk* and completing the prototype tool.

REFERENCES

[1] A. Keshlaf and S. Riddle, "Risk Management for Web and Distributed Software Development Projects," in 2010 Fifth International Conference on Internet Monitoring and Protection (ICIMP 2010), IEEE Computer Society, 2010, pp. 22-28.

[2] Y. Kwak and J. Stoddard, "Project Risk Management: Lessons Learned from Software Development Environment," Technovation, vol. 24, November 2004, pp. 915–920.

[3] H. Yong, C. Juhua, R. Zhenbang, M. Liu, and X. Kang, "A Neural Networks Approach for Software Risk Analysis," Proc. of Sixth IEEE International Conference on Data Mining - Workshops (ICDMW'06), IEEE, 2006, pp. 722-725.

[4] P. Bannerman, "Risk and Risk Management in Software Projects: A Reassessment," Journal of Systems and Software - Elsevier, vol. 81, December 2008, pp. 2118 - 2133.

[5] A. Tiwana and M. Keil, "Functionality Risk in Information Systems Development: An Empirical Investigation," IEEE Transactions on Engineering Management, vol. 53, AUGUST 2006, pp. 412- 425.

[6] A. Keshlaf and K. Hashim, "A Model and Prototype Tool to Manage Software Risks," Proc. of First Asia–Pacific Conference on Quality Software, IEEE Computer Society, 2000, pp. 297–305.

[7] S. Islam, "Software Development Risk Management Model – A Goal Driven Approach," Proc. of ESEC/FSE Doctoral Symposium'09, ACM, 2009, pp. 5-8.

[8] J. Esteves, J. Pastor, N. Rodriguez, and R. Roy, "Implementing and Improving the SEI Risk Management Method in a University Software Project," Latin America Transactions, IEEE (Revista IEEE America Latina), vol. 3, March 2005, pp. 90-97.

[9] M. Kajko-Mattsson and J. Nyfjord, "State of Software Risk Management Practice," IAENG International Journal of Computer Science - On-line Issue, vol. 35, November 2008.

[10] R. Williams, G. Pandelios, and S. Behrens, "Software Risk Evaluation (SRE) Method Description (Version 2.0)," Software Engineering Institute (SEI) December 1999.

[11] B. W. Boehm, "Software Risk Management: Principles and Practices," IEEE Software, vol. 8, 1991, pp. 32-41.

[12] J. Moses, "Bayesian Probability Distributions for Assessing Measurement of Subjective Software Attributes," Information and Software Technology, vol. 42, 15 May 2000, pp. 533-546.

[13] K. Hashim and A. Keshlaf, "An Approach to Sharing Solutions to Software Project Management Problems," Proc. of International Conference on Information Management and Engineering (ICIME '09), IEEE Computer Society, 2009, pp. 694-697.

[14] W. Han and S. Huang, "An Empirical Analysis of Risk Components and Performance on Software Projects," Journal of Systems and Software, vol. 80, January 2007, pp. 42-50.

[15] B. W. Boehm, Software Risk Management, IEEE Computer Society Press, 1989.

[16] IEEE Std. 1540-2001, "IEEE Standard for Software Life Cycle Processes – Risk Management," IEEE 2001.

[17] B. Boehm, J. Kwan, D. Port, A. Shah, and R. Madachy, "Using the WinWin Spiral Model: A Case Study," IEEE Computer, 1998, pp. 33 - 44.

[18] S. Misra, U. Kumar, V. Kumar, and M. Shareef, "Risk Management Models in Software Engineering," International Journal of Process Management and Benchmarking (IJPMB), vol. 2, 2007, pp. 59-70.

[19] M. Jiménez, M. Piattini, and A. Vizcaíno, "Challenges and Improvements in Distributed Software Development: A Systematic Review," Advances in Software Engineering, vol.2009, 2009, pp. 1-14.

[20] M. Malarvannan, "Managing Offshore Software Teams." vol. 2009: Outsource Portfolio, 2009.

[21] B. Sengupta, S. Chandra, and V. Sinha, "A Research Agenda for Distributed Software Development," Proc. of 28th International Conference on Software Engineering (ICSE'06), ACM, 2006, pp. 731 - 740.

[22] K. Nidiffer and D. Dolan, "Evolving Distributed Project Management," IEEE Software, vol. 22, September/October 2005, pp. 63-72.

[23] E. Mendes and N. Mosley, Web Engineering, Berlin Heidelberg: Springer-Verlag, 2006.

[24] A. Taivalsaari, "Mashware: The Future of Web Applications." vol. 2009: Sun Microsystems Laboratories, 2009.

[25] J. Offut, "Quality Attributes of Web Software Applications," IEEE Software, vol. 19, March / April 2002, pp. 25-32.

[26] F. Donini, M. Mongiello, M. Ruta, and M. Totaro, "A Model Checking-based Method for Verifying Web Application Design," Electronic Notes in Theoretical Computer Science, vol. 151, 31 May 2006, pp. 19 - 32.

[27] C. Beise, "IT Project Management and Virtual Teams", Proc. of (SIGMIS'04) Conference Arizona, USA,Tucson, 2004, pp. 129-133.

[28] B. Behkamal, M. Kahani, and M. Akbari, "Customizing ISO 9126 Quality Model for Evaluation of B2B Applications," Information and Software Technology, vol. 51, March 2009, pp. 599-609.

[29] J. Tian, S. Rudrarjuand, and Z. Li, "Evaluating Web Software Reliability Based on Workload and Failure Data Extracted from Server Logs," IEEE Transaction on Software Engineering, vol. 30, November 2004, pp. 754 - 769.

[30] J. Kontio, M. Hoglund, J. Ryden, and P. Abrahamsson, "Managing Commitment and Risks: Challenging in Distributed Agile Development," Proc. of 26th International Conference on Software Engineering (ICSE '04), 2004, pp. 732- 733.

[31] M. Rabbi and K. Mannan, "A Review of Software Risk Management for Selection of Best Tools and Techniques," Proc. 9th ACIS International Conference on Software Engineering, Artificial Intelligence, Networking, and Parallel / Distributed Computing, IEEE Computer Society, 2008, pp. 773 - 778.

[32] W. Glisson and R. Welland, "Web Development Evolution: The Assimilation of Web Engineering Security," in Proc. of Third Latin American Web Congress (LA-WEB'05), IEEE Computer Society, 2005, p. 5.

[33] B. Romero, H. Haddad, and J. Molero A, "A Methodological Tool for Asset Identification in Web Applications Security Risk Assessment," in Proc. of Fourth International Conference on Software Engineering Advances, IEEE Computer Society, 2009, pp. 413-418.

[34] Y. Huang, C. Tsai, D. Lee, and S. Kuo, "Non-Detrimental Web Application Security Scanning," in Proc. of the 15th International Symposium on Software Reliability Engineering (ISSRE'04): IEEE Computer Society, 2004, pp. 219-239.

[35] X. Ge, R. Paige, F. Polack, H. Chivers, and P. Brooke, "Agile Development of Secure Web Applications," Proc. of 6th International Conference on Web Engineering (ICWE'06), ACM, 2006, pp. 305 -312.

[36] CA/Wily, "White Paper: Effectively Managing High-Performing Business-Critical Web Application," http://zones.computerworld.com/ca/ accessed on 16 Jan. 2011.

[37] G. Kappel, B. Proll, S. Reich, and W. Retschitzegger, Web Engineering the Discipline of Systematic Development of Web Application, John Wiley & Sons, Ltd, 2006.

[38] V. Bruno, A. Tam, and J. Thom, "Characteristics of Web Applications that Affect Usability: A Review," in Proc. of the 17th Australia conference on Computer-Human Interaction: Citizens Online: Considerations for Today and the Future (OZCHI 05), vol. 122 Canberra, Australia, 2005, pp. 1- 4.

[39] J. Presson, L. Mathiassen, B. Jesper, T. Madsen, and F. Steinson, "Managing Risks in Distributed Software Projects: An Integrative Framework," IEEE Transaction on Software Engineering, vol. 56, 2009, pp. 508-532.

[40] S. Willis, Using QA for Risk Management in Web Projects, Software Quality and Software Testing in Internet Times, New York,USA: Springer-Veriag Inc., 2002.

[41] P. Pressman and D. Low, Web Engineering A Practitioner's Approach. International Edition, Mc Graw Hill, 2009.

[42] C. Iacovou and R. Nakatsu, "A risk profile of offshore-outsourced development projects," Communications of the ACM, vol. 51, 2008, pp. 89-94.

[43] R. Prikladnicki, J. Audy, and R. Evaristo, "Global Software Development in Practice Lessons Learned ," Software Process Improvement and Practice, vol. 8, October/December 2003, pp. 267-281.

[44] C. Ebert, B. Murthy, and N. Jha, "Managing Risks in Global Software Engineering: Principles and Practices," in Proc. of 2008 IEEE International Conference on Global Software Engineering. 2008, pp. 131-140.

[45] J. Gorski and J. Miler, "Towards an Integrated Environment for Risk Management in Distributed Software Projects," Proc. of 7th European Conference on Software Quality (ECSQ02), Helsinki, Finland, 2002.

[46] ENISA, "Ebios Product Identity Card," ENISA, http://rm-inv.enisa.europa.eu/methods_tools/m_ebios.html accessed on 28 December 2010.

[47] G. Roy, "A Risk Management Framework for Software Engineering Practice," in Proc. of the 2004 Australian Software Engineering Conference (ASWEC'04), IEEE Computer Society, 2004, pp. 60-67.

[48] J. Kontio, "The Riskit Method for Software Risk Management, Version 1.00 CS-TR-3782 / UMIACS-TR- 97-38," University of Maryland, Maryland 1997.

[49] J. Kontio and V. R. Basili, "Empirical Evaluation of a Risk Management Method," in SEI Conference on Risk Management Atlantic City, Nj, USA, 1997.

[50] B. Freimut, S. Hartkopf, P. Kaiser, J. Kontio, and W. Kobitzsch, "An Industrial Case Study of Implementing Software Risk Management," in Proc. of 8th European Software Engineering Conference held jointly with 9th ACM (SIGSOFT) International Symposium on Foundations of Software Engineering, ACM, 2001, pp. 277- 287.

[51] J. Dhlamini, I. Nhamu, and A. Kachepa, "Intelligent Risk Management Tools for Software Development," in Proc. of the 2009 Annual Conference of the Southern African Computer Lecturers' Association (SACLA 09), ACM, 2009, pp. 33 - 40.

[52] J. Smith, S. Bohner, and D. McCricard, "Project Management for the 21st Century Supporting Collaborative Design Through Risk Analysis," Proc. of 43rd ACM Southeast Conference, ACM, 2005, pp. 300 - 305.

[53] C. Pan and Y. Chen, "An Optimization Model of CMMI-Based Software Project Risk Response Planning," International Journal of Applied Mathematics and Computer Sciences, vol. 1, 2005, pp. 155 - 159.

[54] SEI-CMMI, "CMMI-SVC,V1.2." Software Engineering Institute, http://www.sei.cmu.edu/reports/09tr001.pdf accessed on 30 December 2010.

[55] R. Williams, "The CMMI RSKM Process Area as a Risk Management Standard," in Proc. of Sixteenth Annual International Symposium of the International Council On Systems Engineering (INCOSE): INCOSE, 2006.

[56] D. Callegari and R. Bastos, "Project Management and Software Development Processes: Integrating RUP and PMBOK," in Proc. of 2007 International Conference on Systems Engineering and Modeling, IEEE, 2007, pp. 1- 8.

[57] W. R. Duncan, "A Guide to the Project Management Body of Knowledge PMBOK- PMI," Project management Institute, Boulevard -Newtown Square, USA 1996.

[58] L. Wallace, M. Keill, and A. Rai, "How Software Project Risk Affects Project Performance: An Investigation of the Dimensions of Risk and an Exploratory Model," Decision Sciences vol. 35, 2004, pp. 289 – 321.

Privacy by Flexible Parameterization with Erlang Active Objects

Andreas Fleck
Software Engineering Group
Technische Universität Berlin, Germany
Email: andreasfleck@web.de

Florian Kammüller
Middlesex University, London UK and
Technische Universität Berlin, Germany
Email: f.kammueller@mdx.ac.uk

***Abstract*—Functional active objects are a new paradigm for the implementation of services. They offer safe distributed evaluation with futures and immutable objects guaranteeing efficient implementation of privacy while offering verified quality assurance based on the functional paradigm and a development in an interactive theorem prover. In this paper, we present a novel and highly performant implementation of functional active objects in Erlang. Besides outlining the guiding principles of the interpreter, we show how secure services can be realized based on the classical service triangle and prove its security based on a formal definition of information flow security for functional active objects.**

***Keywords*-Active object, future, Erlang, privacy, service computing**

I. Introduction

The free lunch is over – as Sutter describes so vividly in his famous paper [24]. In all realms of modern computing, we need to distribute to keep up performance. Active objects combine the successful concept of object-orientation with the necessary concepts of concurrent processes to make them fit for this distributed world.

We present an implementation of the novel language ASP_{fun} for *functional active objects* in the programming language Erlang. ASP_{fun} is an computation model that can be seen as a descendant of the actor model, or more precisely active objects. Its main specialty is the use of *futures* to avoid blocking states while invoking asynchronous methods. Since no data is shared between active objects, concurrent method invocation can be simply used without fear of racing. These features are very similar to the features of services. Hence, it is possible to formalize complex services in ASP_{fun} and this fact allows us to transfer ASP_{fun} properties to services. The Erlang implementation of ASP_{fun} enables a prompt transfer from an ASP_{fun} configuration to executable code and so the "real behavior" can be tested. Besides the highly performant parallelization of Erlang, this approach supports privacy enhancing implementations for services. The main contributions presented in this paper are as follows.

- Functional active objects enable a deadlock free evaluation that implies service invocation in a higher order fashion. That is, a customer can invoke a service without needing to provide his private data.
- The use of *futures* as the machinery behind method invocation enables a flexible reply to method requests. In particular, this reply mechanism supports the privacy enhancing result acquisition described in the previous point.
- Using Erlang as implementation language we present a novel future implementation concept where each future is represented as a process. Thereby, we can abstract from different future update strategies; the Erlang ASP_{fun} interpreter stays close to the original semantics (see Section II-A): since it is functional it is not forced to sacrifice generality for the sake of operationality.
- We offer a formal definition of information flow security and illustrate its use for the proof of security on the service triangle – our running example.

In this paper we first provide the prerequisites of this project: brief introductions to the language ASP_{fun}, currying in ASP_{fun}, and Erlang (Section II). From there, we develop the concepts of our implementation of active objects in Erlang (Section III). We then illustrate how the language can be efficiently used to implement secure services on three examples from privacy reflecting our contribution (Section IV). A formal security definition enables the proof of privacy for flexible parameterization (Section V). We finally offer conclusions, position our work, and give an outlook on further plans (Section VI). This paper extends the original conference contribution [13] by further details on the implementation concepts and a formal definition of security and proof of privacy for the service triangle using flexible parameterization.

II. Prerequisites

In this section, we present the formal definitions of the language ASP_{fun} and a brief introduction in the concepts behind Erlang.

A. Functional Active Objects

The language ASP_{fun} [12] is a computation model for functional active objects. Its local object language is a simple ς-calculus [1] featuring method call $t.l(s)$, and method update $t.l := \varsigma(x,y)b$ on objects (ς is a binder for the self x and method parameter y). Objects consist of a set

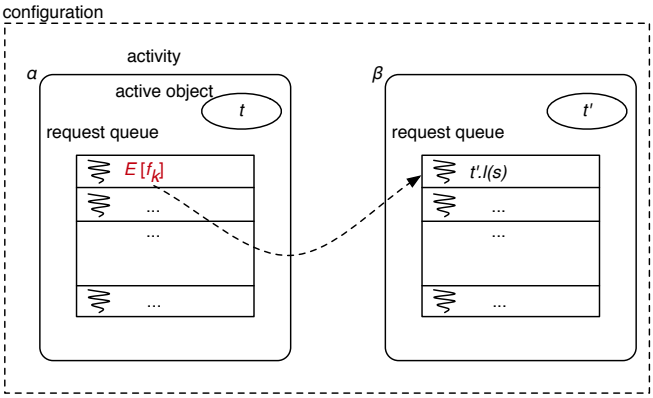


Figure 1. ASP_{fun}: a configuration

of labeled methods $[l_i = \varsigma(x,y)b]^{i\in 1..n}$ (attributes are considered as methods with no parameters). ASP_{fun} now simply extends this basic object language by a command *Active(t)* for creating an activity for an object t. A simple configuration containing just activities α and β within which are so-called active objects t and t' is depicted in Figure 1. This figure also illustrates *futures*, a concept enabling asynchronous communication. Futures are promises for the results of remote method calls, for example in Figure 1, f_k points to the location in activity β where at some point the result of the method evaluation $t'.l(s)$ can be retrieved from. Futures are first class citizen but they are not part of the *static* syntax of ASP_{fun}, that is, they cannot be used by a programmer. Similarly, activity references, e.g. α, β, in Figure 1, are references and not part of the static syntax. Instead, futures and activity references constitute the machinery for the computation of configurations of active objects. Syntactically, we write configurations as $\alpha[R_\alpha, t_\alpha] \parallel \beta[R_\beta, t_\beta] \parallel \ldots$. For example, the configuration of Figure 1 would be syntactically expressed as $\alpha[f_0 \mapsto E[f_k] :: R_\alpha, t] \parallel \beta[f_k \mapsto t'.l(s) :: R_\beta, t']$.

B. Informal Semantics of ASP_{fun}

Local (ς-calculus) and *parallel* (configuration) semantics are given by a set of reduction rules informally described as follows.

- LOCAL: the local reduction relation $\rightarrow_\varsigma$ is based on the ς-calculus.
- ACTIVE: $Active(t)$ creates a new activity $\alpha[\varnothing, t]$ for new name α, empty request queue, and with t as active object.
- REQUEST: *method call* $\beta.l$ creates new future f_k in future-list of activity β (see Figure 1).
- REPLY: *returns result*, i.e. replaces future f_k by the referenced result term s (possibly not fully evaluated).
- UPNAME-AO: *activity upname* creates a copy of the activity and upnames the active object of the copy – the original remains the same (functional active objects are *immutable*).

C. Formal ASP_{fun} semantics

We use a concise contextual description with contexts E defined as usual. Classically we define contexts as expressions with a single hole ($\bullet$).

$$E ::= \bullet \mid [l_i = \varsigma(x,y)E, l_j = \varsigma(x_j,y_j)t_j^{j\in(1..n)-\{i\}}] \mid E.l_i(t) \mid s.l_i(E) \mid E.l_i := \varsigma(x,y)s \mid s.l_i := \varsigma(x,y)E \mid Active(E)$$

$E[s]$ denotes the term obtained by replacing the single hole by s. The semantics of the ς-calculus for (local) objects is simply given by the following two reduction rules for calling and updating a method (or field) of an object.

CALL

$$\frac{l_i \in \{l_j\}^{j\in 1..n}}{E\left[[l_j = \varsigma(x_j,y_j)b_j]^{j\in 1..n}.l_i(b)\right] \rightarrow_\varsigma E\left[b_i\{x_i \leftarrow [l_j = \varsigma(x_j,y_j)b_j]^{j\in 1..n}, y_j \leftarrow b\}\right]} \quad (1)$$

UPDATE

$$\frac{l_i \in \{l_j\}^{j\in 1..n}}{E\left[[l_j = \varsigma(x_j,y_j)b_j]^{j\in 1..n}.l_i := \varsigma(x,y)b\right] \rightarrow_\varsigma E\left[[l_i = \varsigma(x,y)b, l_j = \varsigma(x_j,y_j)b_j^{j\in(1..n)-\{i\}}]\right]} \quad (2)$$

The semantics of ASP_{fun} is built over the local semantics of the ς-calculus as a reduction relation $\rightarrow_\parallel$ that we call the parallel semantics (see Table I).

D. A Running Example from Service Computing

In the following example (an extension of the motivating example of [11]), a customer uses a hotel reservation service provided by a broker. This simple example is representative for service oriented architectures; we refer to it also as the *service triangle*. In this triangle, the three activities hotel, broker, and customer are composed by $\parallel$ into a configuration. To simplify this example the broker's search for a hotel is omitted and we always consider the same hotel and in addition we abstract from the computation in hotel that produces the booking reference for the customer. We concentrate on the message passing implemented in futures to highlight the actual flows of information in the following evaluation sequence.

$$\begin{aligned} &\text{customer}[f_0 \mapsto \text{broker.book}(\textit{date}), t] \\ \parallel\ &\text{broker}[\varnothing, [\text{book} = \varsigma(x,(\textit{date}))\text{hotel.room}(\textit{date}), \ldots]] \\ \parallel\ &\text{hotel}[\varnothing, [\text{room} = \varsigma(x,\textit{date})\text{bookingref}, \ldots]] \end{aligned}$$

The following step of the semantic reduction relation $\rightarrow_\parallel^*$ creates the new future f_1 in broker following rule REQUEST. According to LOCAL, this call is reduced and the original call in the customer becomes f_1.

$$\begin{aligned} &\text{customer}[f_0 \mapsto f_1, t] \\ \parallel\ &\text{broker}[f_1 \mapsto \text{hotel.room}(\textit{date}), \ldots] \\ \parallel\ &\text{hotel}[\varnothing, [\text{room} = \varsigma(x,\textit{date})\text{bookingref}, \ldots]] \end{aligned}$$

The parameter x representing the *self* is not used but the call to hotel's method room with parameter *date* creates again by rule REQUEST a new future in the request queue of the

$$\text{LOCAL}\quad \frac{s \to_\varsigma s'}{\alpha[f_i \mapsto s :: Q, t] :: C \to_\| \alpha[f_i \mapsto s' :: Q, t] :: C} \tag{3}$$

$$\text{ACTIVE}\quad \frac{\gamma \notin (\mathrm{dom}(C) \cup \{\alpha\}) \qquad \mathit{noFV}(s)}{\alpha[f_i \mapsto E[\mathit{Active}(s)] :: Q, t] :: C \to_\| \alpha[f_i \mapsto E[\gamma] :: Q, t] :: \gamma[\varnothing, s] :: C} \tag{4}$$

$$\text{REQUEST}\quad \frac{f_k \text{ fresh} \qquad \mathit{noFV}(s)}{\alpha\,[f_i \mapsto E[\beta.l(s)] :: Q, t] :: \beta[R, t'] :: C \to_\| \alpha\,[f_i \mapsto E[f_k] :: Q, t] :: \beta\,[f_k \mapsto t'.l(s) :: R, t'] :: C} \tag{5}$$

$$\text{SELF-REQUEST}\quad \frac{f_k \text{ fresh} \qquad \mathit{noFV}(s)}{\alpha\,[f_i \mapsto E[\alpha.l(s)] :: Q, t] :: C \to_\| \alpha\,[f_k \mapsto t.l(s) :: f_i \mapsto E[f_k] :: Q, t] :: C} \tag{6}$$

$$\text{REPLY}\quad \frac{\beta[f_k \mapsto s :: R, t'] \in \alpha[f_i \mapsto E[f_k] :: Q, t] :: C}{\alpha[f_i \mapsto E[f_k] :: Q, t] :: C \to_\| \alpha[f_i \mapsto E[s] :: Q, t] :: C} \tag{7}$$

$$\text{UPDATE-AO}\quad \frac{\gamma \notin (\mathrm{dom}(C) \cup \{\alpha\}) \qquad \mathit{noFV}(\varsigma(x,y)s) \qquad \beta[Q, t'] \in (\alpha[f_i \mapsto E[\beta.l := \varsigma(x,y)s] :: Q, t] :: C)}{\alpha[f_i \mapsto E[\beta.l := \varsigma(x,y)s] :: Q, t] :: C \to_\| \alpha[f_i \mapsto E[\gamma] :: Q, t] :: \gamma[\varnothing, t'.l := \varsigma(x,y)s] :: C} \tag{8}$$

Table I
$\mathrm{ASP}_{\mathrm{FUN}}$ SEMANTICS

hotel activity that is immediately reduced due to LOCAL to bookingreference where the index indicates that *date* has been used.

$$\begin{array}{l} \quad \mathrm{customer}[f_0 \mapsto f_1, t]] \\ \| \ \mathrm{broker}[f_1 \mapsto f_2, \ldots] \\ \| \ \mathrm{hotel}[f_2 \mapsto \mathrm{bookingref}_{\langle \mathit{date} \rangle}, \ldots] \end{array}$$

Finally, the result bookingreference is returned to the client by two REPLY-steps: first the future f_2 is returned from the broker to the customer and then this client receives the bookingreference via f_2 directly from the hotel.

$$\begin{array}{l} \quad \mathrm{customer}[f_0 \mapsto \mathrm{bookingref}_{\langle \mathit{date} \rangle}, t] \\ \| \ \mathrm{broker}[f_1 \mapsto f_2, \ldots] \\ \| \ \mathrm{hotel}[f_2 \mapsto \mathrm{bookingref}_{\langle \mathit{date} \rangle}, \ldots] \end{array}$$

This configuration can be considered as the final one; at least the service has been finished. From the perspective of privacy, it is actually here that we would like to end the evaluation. Unfortunately, the future f_2 is also available to the broker. So, in an final step the broker can serve himself the bookingreference as well.

$$\begin{array}{l} \quad \mathrm{customer}[f_0 \mapsto \mathrm{bookingref}_{\langle \mathit{date} \rangle}, t] \\ \| \ \mathrm{broker}[f_1 \mapsto \mathrm{bookingref}_{\langle \mathit{date} \rangle}, \ldots] \\ \| \ \mathrm{hotel}[f_2 \mapsto \mathrm{bookingref}_{\langle \mathit{date} \rangle}, \ldots] \end{array}$$

The abstract general semantics of $\mathrm{ASP}_{\mathrm{fun}}$ allows this privacy breach.

We introduce now a general way of enforcing privacy by not disclosing private data in the first place. We show that relying on the $\mathrm{ASP}_{\mathrm{fun}}$ paradigm guarantees that flexible parameterization can be used to use services in a private manner.

E. Currying for $\mathrm{ASP}_{\mathit{fun}}$

The contribution of this paper is a concept more generally useful for privacy: *flexible parameterization* – enabling the use of service functions while not supplying all parameters. For example, in the European project SENSORIA the COWS calculus has been designed as an extension to the Pi-calculus to realize *correlation*, a similarly dynamic service concept [4].

We have implemented this technique in our Erlang prototype for $\mathrm{ASP}_{\mathrm{fun}}$ (see Section III) as a pragmatic extension of the base language. However, as we will show now, this feature on flexible parameterization can be constructed conservatively in $\mathrm{ASP}_{\mathrm{fun}}$ using *currying*.

Currying is a well known technique in functional programming to render functions with several parameters partially applicable. That is, the parameters of a curried function may be supplied one after the other, in each step returning a new function.

Recall the definition of curry and its inverse uncurry in the λ-calculus.

$$\begin{array}{rcl} \mathsf{curry} & \equiv & \lambda\ f\ p.\ f(\mathsf{fst}\ p)(\mathsf{snd}\ p) \\ \mathsf{uncurry} & \equiv & \lambda\ f\ a\ b.\ f(a,b) \end{array}$$

Here, (a, b) denotes the product and fst and snd the corresponding projections on a and b respectively. This datatype is itself definable in terms of simpler λ-terms as follows.

$$\begin{array}{rcl} (a,b) & \equiv & \lambda\ f.\ f\ a\ b \\ \mathsf{fst}\ p & \equiv & (\lambda\ x\ y.\ x) \\ \mathsf{snd}\ p & \equiv & (\lambda\ x\ y.\ y) \end{array}$$

We recall these classic definitions in order to prepare the less intuitive definition of currying for the ς-calculus and hence for ASP_{fun}. In comparison to the ς-calculus, the base objects of ASP_{fun} differ in that we explicitly introduce a second parameter to each method in addition to the *self*-parameter x. Therefore, when we emulate functions in our version of the ς-calculus we profit from this parameter and avoid roundabout ways of encoding parameters.[1] As a prerequisite for having several parameters, we need products. Compared to the above presented encoding of pairs in the λ-calculus, pairs in the ς-calculus can make use of the natural *record* structure of objects thus rendering a more intuitive notion of product as follows.

$$\begin{aligned} (a,b) &\equiv [\,\mathsf{fst} = \varsigma(x,y)a,\ \mathsf{snd} = \varsigma(x,y)b\,] \\ \mathsf{fst}\ p &\equiv p.\mathsf{fst} \\ \mathsf{snd}\ p &\equiv p.\mathsf{snd} \end{aligned}$$

We simulate currying of a method f of an object o that expects a pair p of type $\alpha \times \beta$ as second parameter, i.e.

$$o = [\, f = \varsigma(x,p).t\,]$$

by extending this object o with a second method f_C as follows.

$$\begin{aligned} \mathsf{curry}\ o \equiv\ & [\, f = \varsigma(x,p)o.f(p), \\ & f_C = \varsigma(x,a)[f' = \varsigma(y,b)x.f(a,b)]\,] \end{aligned}$$

F. Erlang

Erlang is a concurrent-oriented functional programming platform for open distributed telecommunication (OTP) systems developed by Ericsson corporation. It implements the actor paradigm by providing message passing as strategy for communication between several actors implemented as processes. Processes run fully parallel in Erlang. Each process has a mailbox where arriving messages are stored. The programmer can use pattern matching for message selection. Hence, the behavior of an actor is controllable. If a process needs an answer its process identifier (PID) has to be passed through the message. Since memory sharing does not exist, neither locks nor mutexes are necessary. The code is grouped in modules which are referred to by their name. So `modulname:functionname(args).` starts a function from a specific module.

A process is created by the `spawn`-command supplying it with the process' function and initial arguments. Erlang supports also named processes. Using `register(Name,PID)` the PID is registered in a global process registry and the process can be called by its name.

```
PID = spawn(Func,Args),
PID!Message,
Func(Args)...
receive
 Pattern1 [when Guard1] -> Expression1;
 Pattern2 [when Guard2] -> Expression2;
 ...
end.
```

Above, we show the basics of distribution in Erlang. First, we start a new process which runs the function `Func`. Then, we send a `Message` to the new process which is identified by `PID`. The function `Func` implements several patterns for incoming messages. Now, the system tries to match the arrived message against `Pattern1` (and the guard if it exists). In case of success, `Expression1` is evaluated. If the first pattern fails, the second will be used and so on. Another fundamental feature of Erlang is the single assignment, as in algebra, meaning that Erlang variables are immutable.

The main data types are (untyped) lists and records, called tuple, for example {`green, apple`} and atoms which represent different non-numerical constant values. Any lower case name is interpreted as an atom, any higher case name is a variable. In addition, there are modules for interoperability to other programming languages like C, Java or databases.

III. An ASP_{FUN} Interpreter in Erlang

Active objects bridge the gap between parallel processes and object-orientation. Intuitively, we want an object to be a process at the same time; unfortunately the two concepts are not identical. Hence, *activities* are introduced as a new notion of process containing an active object together with its current method execution(s).

In this section we describe how the concepts of activities, active objects and futures are realized on the infrastructure of Erlang; each concept resides in a separate module.

A. Activity

The first module describes the functionality of an *activity*. An *activity* encapsulates a functional active object to prevent direct access to it and manage requests simultaneously. In a functional language there is no need to make a sequential plan for execution in contrast to imperative active objects [6]. All requests are executed in parallel and run in individual processes. In our Erlang interpreter, the *activity* is implemented as a separate process. Following the ASP_{fun} semantics, an activity contains a request-queue and the functional active object which are dedicated to the process. We use the Erlang built-in functionality for send and receive. This comes in quite naturally to model asynchronous communication of activities. In fact – as we will see when implementing futures (see Section III-C) – message passing is the correct foundation for asynchronous communication with futures.

To keep the activity alive, the process is called again after each receive. Any request has to be sent as a tuple {`Caller_PID,request,RequestFunction,Args`} where

[1] In the ς-calculus the parameter has to be simulated by updating a separate field in an objects and that consequently needs to be attached to each object.

`Caller_PID` is the `PID` or registered name of the calling process (see Section III-C), the constant `request` which is used as pattern in the receive evaluation, the name of a requested function, and optionally the arguments as tuple or nil. An activity is now started in Erlang by

```
ActiveObject_PID = activeObject:start(),
activity:start(Identifier,ActiveObject_PID).
```

where `ActiveObject_PID` is the process identifier of the functional active object to be introduced next.

B. Functional Active Object

The second module specifies the functionality of *functional active objects*. The active object stores the ς-calculus methods in an list, where they can identify by a given name. Deviating from the original notion of immutable objects of ASP_{fun}, our Erlang implementation is a dynamic ASP_{fun}-interpreter: ς-calculus methods can be added or deleted on the fly. Methods can also be declared at runtime or even be specified in separate modules. In our implementation, a functional active object is also a process which communicates with its activity by message passing. The activity requests a function using its name and the functional active object replies the function to the activity if it exists. This fact allows additionally separate distribution of the activity and the active object. To start an empty functional active object in our Erlang active object interpreter, we just call the following function.

```
ActiveObject_PID = activeObject:start().
```

To add functionality to this initial functional active object one can define own functions or use existing Erlang modules.

```
Foo = fun(Self,{arg1,arg2,...})
  -> some calculation, return value
end.
ActiveObject_PID!{add_func,functionname, Foo}.
```

where `functionname` is the name which one can use in other activities. To enable functions as return values, it is important to add a `Self`-Parameter. This parameter is set automatically by our system when distributing functions.

C. Future

The last module represents the implementation of *futures*. Futures act traditionally as placeholder for later results calculated in parallel [6]. In the ASP_{fun} computation model there is no need to describe the fashion of updating a future with the calculated value. Furthermore, the evaluation of a future is possible at any time. This means, the result can be an value or the current state of the function evaluation. These facts have to be considered while implementing ASP_{fun}. For example, the "on demand evaluation" can be implemented in Erlang by a watching process for each calculation which stores the current state. This is not very efficient and in most cases unnecessary. For the future updating process there exist several approaches, such as message-based or forward-based updates [16]. These strategies have in common that they need to store the relations between activities and futures [27]. We decided to expand the functionality by implementing the future as a process which also stores the final value. This concept makes allows a complete separation from the activity in a parallel manner and presents several advantages. First, the future is more active and allows the use of message passing and the distribution by its network identifier. So a future is unique in the entire configuration and therefore, there is no need to plan the update-process because other activities can call the future directly by its network identifier to get the value. The second advantage is the location of future creation. In our implementation, the future is created by the enquiring activity and not by the requested activity. This augments the privacy of activities by using the future as a kind of "proxy for communication" between activities. No activity has to announce itself to others when remote calculation is needed. The future asks the remote activity for the requested calculation and waits for the response: it is a "proxy for communication". In Figure 2, we illustrate this communication flow. In our opinion, this approach is the consequent continuation of an asynchronous communication concept (cf [5]). In some existing approaches [6], a future is created and immediately returned by the called activity (in pseudo-code).

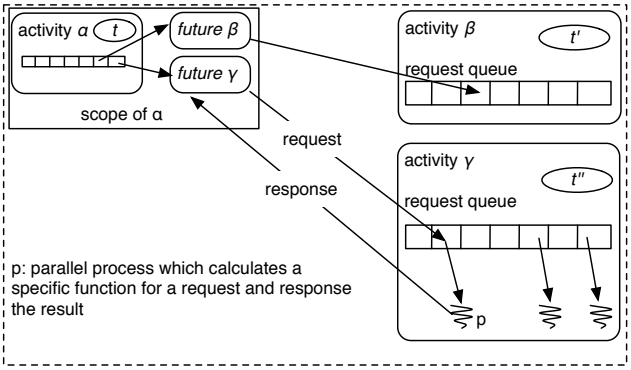


Figure 2. ASP_{fun}-Erlang: communication flow

```
Future localfuture = activity_anywhere_in_www.foo()
```

However, this call is not really asynchronous because the remote activity might not respond immediately or the message is lost. As a result, the local activity also blocks. A really asynchronous solution must therefore use messages instead of return-values [5]. Our approach works as follows.

```
Future localfuture = new Future,
localfuture.start(activity_anywhere_in_www.foo())
```

First, the future is created in the scope of a calling activity and, then, the communication with the remote activity starts through the future by messages in an asynchronous manner. A new future-activity-request in Erlang may be started as

follows.

```
newfuture = future:start(activity,functionname,args)
```

This function call creates a new future, sends a request message with the identifier of the new future, the function's name and the arguments to `activity` (see Section III-D). The final value of this local function is the process identifier of the future. A next advantage is the enforcement of security policies at the point of communication. The circumstance that the future is created by the calling activity and stores the final value (see Section IV-D) allows to enforce security policies of this activity for requests and responses at one single point.

D. Function Execution and Evaluation

All functions which are invoked in activities are running completely parallel in their own processes. By contrast, imperative active object systems like ProActive or others [9] use a sequentializing process and the execution runs in the thread of the activity. A further benefit is that activities do not freeze in case a function execution blocks. If a function blocks, only the future evaluation blocks. Therefore, we have built a second argument into our evaluation function representing the maximal evaluation time. After that time span the evaluation returns nil. If and when the result is ready, the requested activity, that is, the calculating process, sends a message with the result to the calling future. The future get the result, stores it and waits for the evaluation by the activity which starts by

```
Result = future:evaluate(Future, 10).
```

Since the evaluation can occur at any time, we have implemented two different cases:

- if the result is finished, it will be returned,
- and, if the result is not ready, the evaluation-process blocks until the future is updated (wait-by-necessity and finished after the update).

The result for self can be an ordinary value (tuple, atom, variable, etc.), a function (higher-order), an activity (a `PID` or name) or again a future. In the first three cases, the result is returned. If the result is itself a future, the evaluation function evaluate this future and returns this result.

IV. Secure Services in ASP_{FUN}

In this section, we will now come back to the running example of a hotel-broker-service and show that our Erlang active object interpreter can model different possible scenarios. Note, that these scenarios are consistent with the ASP_{fun} semantics given in Section II. They define just different strategies corresponding to various privacy goals. We first show the classical evaluation order where service results flow back via the invocation structure to the customer. We then additionally sketch two refinements, where first the actual service is passed to the customer so he can communicate directly with the hotel without passing private data through the broker. Next, we show that our Erlang active object interpreter makes full use of the functional support: a customer can use a service by only providing partial information. Thereby, he can guard private information and still get some (information about the) service.

Thus, our Erlang active object interpreter[2] represents an implementation of ASP_{fun} in its broadest sense. Various different more "operational" semantics corresponding to different security policies can be easily implemented in our Erlang active object framework. For professional use, the basic machinery presented in this paper needs to be equipped with a mechanism for a simplified control over the different strategies.

A. Classic Service Evaluation

First, we show how the ASP_{fun} example from Section II looks in "standard" form. Therefore, we define a function `Room` where we use the ordinary Erlang syntax including the named specifications.

```
Room=fun(Self,{Date})->
BookingRef= database:any_database_call(Date),
BookingRef
end.
```

This function calls a function at a local database module. This can take some time. Next, we create a new active object, add the created function, and instantiate an activity named `hotel` which encapsulates the active object.

```
AO_Hotel = activeObject:start(),
AO_Hotel!{add_func,room,Room},
activity:start(ActHotel,AOHotel),
```

Thereafter, we define the `broker` in the same manner, with the exception that the `book` function uses the `Room` function of `hotel` and returns a future.

```
Book = fun(Self,{Date})->
... find a hotel by Date -> return an activity hotel
FutBookingRef=future:start(hotel,room,{Date}),
FutBookingRef
end,
```

The newly created future sends a message to `hotel` and waits for an answer with result. Finally, we define the `customer`'s wish.

```
FutBookHotelRoom =
future:start(broker,book,{Mydate}),
BookingRef = future:evaluate(FutBookHotelRoom)
```

The arguments is the date on which he wants to book a room. The created future `FutBookHotelRoom` sends a message to the activity `broker` which runs the function `book`. The function `book` also creates a future `FutBookingRef` to communicate with `hotel` and returns this to the future of `customer` – similar to the first application of the rule REPLY in the ASP_{fun} example. After the `Room` function has finished, the future `FutBookingRef` is updated. The evaluation of `FutBookHotelRoom` additionally

[2]For the sources `http://www.users.cs.tu-berlin.de/~flokam`

calls an evaluate at FutBookingRef which returns the BookingRef. These two steps represent the second application of the rule REPLY in the ASP_{fun} example. In the case that FutBookingRef is not updated yet, a wait-by-necessity occurs until FutBookingRef is ready. As in the original ASP_{fun} example, the broker can evaluate the future FutBookingRef too because it is in the same scope. This means that the result is not passed directly to the broker but there is a potential risk that he can retrieve it.

B. Private Customer Negotiation

In the first extension, the customer has an additional parameter Name, which should not be shared with the broker. So in the relation between this special customer and the untrusted broker our goal is to prevent the untrusted broker to read the private data. For another customer, the same broker could be a trusted partner. This behavior can be defined in individual security policies. To make these scenarios possible we change the book function and add a case analysis.

```
Book = fun(Self,{Date,Name})->
case (Name == nil) of
 true ->
  whereis(hotel);
 false ->
 ...find a hotel by Date -> return an activity hotel
  FutBookingRef=future:start(hotel,room,{Date,Name}),
  FutBookingRef
end
end,
```

So, if the argument Name is missing, the function returns the network identification of the activity hotel. Now, the customer can communicate directly with the services of the hotel.

```
FutBookHotelRoom =
    future:start(broker,book,{mydate,nil}),
ActHotel =future:evaluate(FutBookHotelRoom),
FutBookingRef =
    future:start(ActHotel,room,{mydate,myname}),
BookingRef = future:evaluate(FutBookingRef),
```

The evaluation of the future FutBookHotelRoom returns now an activity. The customer uses this activity to call the function Room directly with his private data. In this example, the broker cannot read the private argument of the customer. This strategy is simple and intuitive but needs the knowledge of the inner structure of the web-service. So the programmer needs to know the interfaces of actually hidden services. The difficulty grows with the complexity of the web-service.

C. Privacy by Partial Services

In the second extension, we show another way to implement privacy, now with distributed functions. This time, the customer also shares the date with the broker and the date and the name with the hotel. However, the function book is not modified and calls the function Room with one argument missing. In the definition of Room, we use a local database function. This fact does not allow to distribute this function. To make it again possible, we change the code slightly using our Self-operator and add a case analysis because Erlang does not implement currying. Using existing implementations of currying functors, the following code could be further improved.

```
Room=fun(Self,{Date,Name})->
case Name == nil of
true ->
    NewFun = fun(MissingName) ->
            Args ={Date,MissingName},
            future:start(Self,room,Args)
            end;
false ->
   BookingRef= database:any_database_call(Date,Name),
   BookingRef
end
end.
```

In case argument Name is missing, a new function is defined which uses the existing argument Date and needs the missing Name. This function returns a new future which communicates also with hotel and uses both arguments the private Name and the public Date. The customer's wish looks now as follows.

```
FutBookHotelRoom =
    future:start(broker,book,{Mydate,nil}),
FunctionRoomByName=future:evaluate(FutBookHotelRoom),
FutBookingRef = FunctionRoomByName(myname),
BookingRef = future:evaluate(FutBookingRef),
```

The evaluation of FutBookHotelRoom returns the function NewFun which is defined in Room and awaits Name as argument. The execution of this function returns a new future which is evaluated by customer and returns the bookingreference.

D. Subsumption

The three different examples show how privacy can be implemented by using futures and active objects. As shown above there are the possibilities to use intermediary activities, which return futures of others requests. Furthermore, the result can be an activity allowing to break up the communication flow. In the last example, we show how functionality can be transferred/delegated. These basic concepts allow – in the context of web-services – to implement private services in a new manner. The concept of partial services is the base of a new kind of using complex partially trusted services. The fact that the name is just shared with hotels (or one of them) and that the date is public can be specified by the customer in an individual security policy. The implementation of an active future, the evaluation behavior of futures and the idea of currying allow to enforce these policies by using flexible parameterization in the future. The future knows the current communication partner and can share the arguments on the basis of the actual security policy. If the current call is untrusted, private data will not be shared.

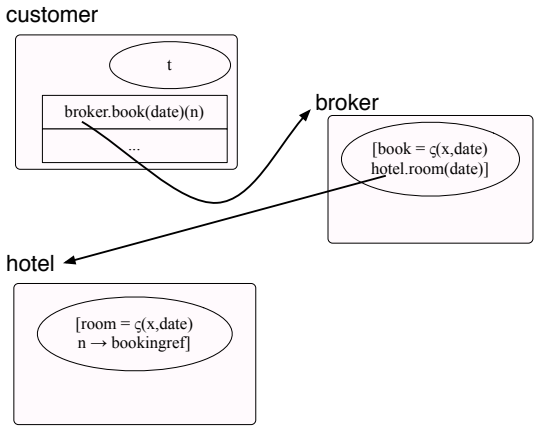


Figure 3. Private service scenario using currying

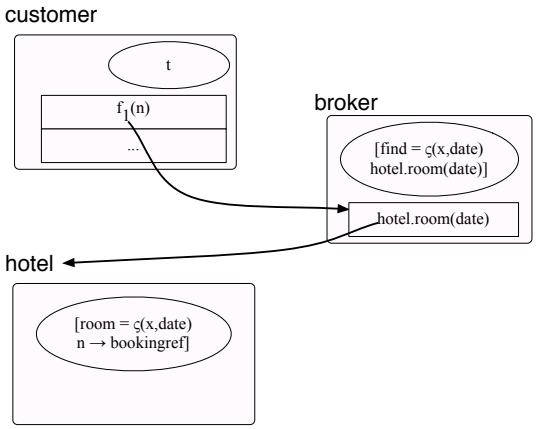


Figure 4. Partially instantiated request is delegated.

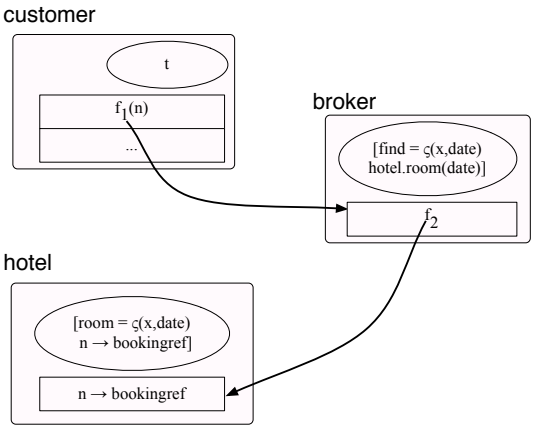


Figure 5. Customer may retrieve "bookingref"-function.

Then the behavior of currying will take effect and the result is a new function of a new communication partner and the description of them. So, the evaluation will run again and call the function with the required arguments, depending on the policy. For the case of the evaluation of a future by another activity, the future can also check the policy before returning the result. Another challenge is to avoid an information flow between requests inside an activity. This is prevented by Erlang through the no data sharing concept and in ASP_{fun} through the rule UPDATE. For example, the use of one generic service which can be cloned and loaded with private data by clients allows to create a one-to-one relation between a client and his "private" web service. The generic service and other private services are then excluded from the communication of one specific service client relationship [11].

E. Service Triangle with Currying

To prepare a rigorous privacy analysis, we generalize the service triangle in an abstract fashion summarizing the various possible extensions seen above to a "bare bones" privacy scenario. Now, the function room in hotel has one additional parameter n for *name* besides date. This is to emphasize the privacy issue; wishing to keep your name n private seems natural; date, by contrast, can be considered as irrelevant, i.e. low. See Figure 3 for the setup configuration of this privacy scenario.

Now in the curried version, room can be called just supplying the first date parameter. The broker still delegates the partially instantiated request to the hotel (see Figure 4). Thereby, the customer can then directly access a function in hotel – via the futures f_1 and f_2 – that calculates his bookingref on supplying the missing parameter name (see Figure 5).

This intuitive idea of a curried version looks in the ASP_{fun} representation as follows.

$$
\begin{aligned}
&\text{data}[\varnothing, [\text{id} = \text{hugo}]]\\
\|\ &\text{customer}[f_0 \mapsto \text{broker.book}_C(d).\text{room'}(\text{data.id}), t]\\
\|\ &\text{broker}[\varnothing, [\text{book}_C = \varsigma(x,d)\text{hotel.room}_C(d), \ldots]]\\
\|\ &\text{hotel}[\varnothing, [\text{room} = \varsigma(x,(d,n))\text{bookingref},\\
&\qquad \text{room}_C = \varsigma(x,d)[\text{room'} = \varsigma(x',n)x.\text{room}(d,n)]]]
\end{aligned}
$$

It can reduce according to the semantics as shown in the subsequent configurations.[3] First, f_1 is created in data.

$$
\begin{aligned}
&\text{data}[f_1 \mapsto \text{hugo}, [\text{id} = \text{hugo}]]\\
\|\ &\text{customer}[f_0 \mapsto \text{broker.book}_C(d).\text{room'}(f_1), t]\\
\|\ &\text{broker}[\varnothing, [\text{book}_C = \varsigma(x,d)\text{hotel.room}_C(d), \ldots]]\\
\|\ &\text{hotel}[\varnothing, [\text{room} = \varsigma(x,(d,n))\text{bookingref},\\
&\qquad \text{room}_C = \varsigma(x,d)[\text{room'} = \varsigma(x',n)x.\text{room}(d,n)]]]
\end{aligned}
$$

The call to book_C creates future f_2,

[3]Clearly, there are also other possibilities to reduce, e.g. instead of evaluating data.id in the first step we could first reduce the call to book_C, thereby duplicating data.id. Since ASP_{fun} is confluent up to identical copies, we always end with the same result.

$$\begin{array}{l} \quad \text{data}[f_1 \mapsto \text{hugo}, [\text{id} = \text{hugo}]] \\ \| \ \text{customer}[f_0 \mapsto f_2.\text{room'}(f_1), t] \\ \| \ \text{broker}[f_2 \mapsto \text{hotel.room}_C(d), \\ \qquad\qquad [\text{book}_C = \varsigma(x,d)\text{hotel.room}_C(d), \ldots]] \\ \| \ \text{hotel}[\varnothing, [\text{room} = \varsigma(x,(d,n))\text{bookingref}, \\ \qquad \text{room}_C = \varsigma(x,d)[\text{room'} = \varsigma(x',n)x.\text{room}(d,n)]]] \end{array}$$

which in turn produces a future f_3 in hotel where A is an abbreviation for the active object of hotel.

$$\begin{array}{l} \quad \text{data}[f_1 \mapsto \text{hugo}, [\text{id} = \text{hugo}]] \\ \| \ \text{customer}[f_0 \mapsto f_2.\text{room'}(f_1), t] \\ \| \ \text{broker}[f_2 \mapsto f_3, [\text{book}_C = \varsigma(x,d)\text{hotel.room}_C(d), \ldots]] \\ \| \ \text{hotel}[f_3 \mapsto [\text{room'} = \varsigma(x',n)\text{A.room}(d,n)], \\ \qquad [\text{room} = \varsigma(x,(d,n))\text{bookingref}, \\ \qquad \text{room}_C = \varsigma(x,d)[\text{room'} = \varsigma(x',n)x.\text{room}(d,n)]]] \end{array}$$

Now, we choose to reply first f_3 to customer,

$$\begin{array}{l} \quad \text{data}[f_1 \mapsto \text{hugo}, [\text{id} = \text{hugo}]] \\ \| \ \text{customer}[f_0 \mapsto f_3.\text{room'}(f_1), t] \\ \| \ \text{broker}[f_2 \mapsto f_3, [\text{book}_C = \varsigma(x,d)\text{hotel.room}_C(d), \ldots]] \\ \| \ \text{hotel}[f_3 \mapsto [\text{room'} = \varsigma(x',n)\text{A.room}(d,n)], \\ \qquad [\text{room} = \varsigma(x,(d,n))\text{bookingref}, \\ \qquad \text{room}_C = \varsigma(x,d)[\text{room'} = \varsigma(x',n)x.\text{room}(d,n)]]] \end{array}$$

whereby the customer can serve himself by f_3 the resulting object containing the room' method from hotel.

$$\begin{array}{l} \quad \text{data}[f_1 \mapsto \text{hugo}, [\text{id} = \text{hugo}]] \\ \| \ \text{customer}[f_0 \mapsto [\text{room'} = \\ \qquad\qquad \varsigma(x',n)\text{A.room}(d,n)].\text{room'}(f_1), t] \\ \| \ \text{broker}[f_2 \mapsto f_3, [\text{book}_C = \varsigma(x,d)\text{hotel.room}_C(d), \ldots]] \\ \| \ \text{hotel}[f_3 \mapsto [\text{room'} = \varsigma(x',n)\text{A.room}(d,n)], \\ \qquad [\text{room} = \varsigma(x,(d,n))\text{bookingref}, \\ \qquad \text{room}_C = \varsigma(x,d)[\text{room'} = \varsigma(x',n)x.\text{room}(d,n)]]] \end{array}$$

Finally, the customer receives by the rule REPLY the id hugo contained in f_1 (see Table I),

$$\begin{array}{l} \quad \text{data}[f_1 \mapsto \text{hugo}, [\text{id} = \text{hugo}]] \\ \| \ \text{customer}[f_0 \mapsto [\text{room'} = \\ \qquad\qquad \varsigma(x,n)\text{A.room}(d,n)].\text{room'}(\text{hugo}), t] \\ \| \ \text{broker}[f_2 \mapsto f_3, [\text{book}_C = \varsigma(x,d)\text{hotel.room}_C(d), \ldots]] \\ \| \ \text{hotel}[f_3 \mapsto [\text{room'} = \varsigma(x',n)\text{A.room}(d,n)], \\ \qquad [\text{room} = \varsigma(x,(d,n))\text{bookingref}, \\ \qquad \text{room}_C = \varsigma(x,d)[\text{room'} = \varsigma(x',n)x.\text{room}(d,n)]]] \end{array}$$

and we locally reduce the future f_0 in several steps with the rule for CALL of the ς-reduction (see Section II-A).

$$\begin{array}{l} \quad \text{data}[f_1 \mapsto \text{hugo}, [\text{id} = \text{hugo}]] \\ \| \ \text{customer}[f_0 \mapsto \text{bookingref}_{\langle\text{hugo}\rangle}, t] \\ \| \ \text{broker}[f_2 \mapsto f_3, [\text{book}_C = \varsigma(x,d)\text{hotel.room}_C(d), \ldots]] \\ \| \ \text{hotel}[f_3 \mapsto [\text{room'} = \varsigma(x',n)\text{A.room}(d,n)], \\ \qquad [\text{room} = \varsigma(x,(d,n))\text{bookingref}, \\ \qquad \text{room}_C = \varsigma(x,d)[\text{room'} = \varsigma(x',n)x.\text{room}(d,n)]]] \end{array}$$

It is crucial to observe that, even if the broker would have served himself the content of future f_3 he would not be able to produce the result $\text{bookingref}_{\langle\text{hugo}\rangle}$. Thus, the program with currying is secure with respect to a security assignment that marks broker as low, or an outsider, and thus protects the privacy of customer. Although the acquired privacy effect might now seem obvious, it is necessary to find a way to ascertain it rigorously. Therefore, we introduce in the following section a formal notion of security for ASP_{fun} and use it to analyze the security of the server triangle.

V. Formal Security Proof

In this section, we present a formal analysis of security for the service triangle. We show that the classical solution is insecure while the one with currying is secure. These formal proofs apply a formal security definition for ASP_{fun} to show that no information flows from the private domain of the client to the public server.

A. Noninterference Definition for ASP_{fun}

Intuitively, noninterference [26] means that an attacker cannot learn anything about private data by regarding public parts of a program. To arrive at a formal expression of this idea for ASP_{fun}, we first define a relation of indistinguishability, often also called L-equivalence because in this relation L-terms have to be equal.

We use here the notion of types informally because this suffices to disambiguate the following bijection. Indeed, ASP_{fun} has a safe type system [11] that can serve here but is omitted for brevity.

Definition 5.1 (Typed Bijection): A typed bijection is a finite partial function σ on activities α (or futures f_k respectively) such that

$$\forall\, a : \text{dom}(\sigma).\ \vdash a : T \ \Rightarrow \vdash \sigma(a) : T$$

(where T is given by an activity type $\Gamma_{act}(\alpha)$ or a future type $\Gamma_{fut}(f_k)$ respectively).

The intuition behind typed bijections is that $\text{dom}(\sigma)$ designates all those futures or activity references that are or have been visible to the attacker. We cannot assume the names in different runs of programs, even for low elements, to be the same. Hence, we relate those names via a pair of bijections. These bijections are typed because they relate activities and futures that might need to be structurally equivalent, in case they are low. The following definition of indistinguishability uses the typed bijection in this sense.

We define (low)-indistinguishability as a relation $\sim_{\sigma,\tau}$ parameterized by two typed bijections one over activity names and one over futures. It is a heterogeneous relation as it ranges over elements of different types, for example activities and request queues. We leave out the types as they are indicated by our notational convention. By $t =_{\sigma,\tau} t'$ we denote the equality of terms up to replacing all occurrences of activity names α or futures f_k by their counterparts $\tau(\alpha)$

or $\sigma(f_k)$, respectively. The local reduction with $\rightarrow_\varsigma$ of a term t to a value t_e (again up to future and activity references) is written as $t \Downarrow t_e$.

Definition 5.2 (Indistinguishability): An indistinguishability relation is a heterogeneous relation $\sim_{\sigma,\tau}$, parameterized by two isomorphisms σ and τ whose differently typed subrelations are as follows.

$$\begin{aligned}
t \sim_{\sigma,\tau} t' &\equiv t \Downarrow t_e \wedge t' \Downarrow t'_e \\
&\quad \wedge\ t_e =_{\sigma,\tau} t'_e \\
\alpha \sim_{\sigma,\tau} \beta &\equiv \tau(\alpha) = \beta \\
f_k \sim_{\sigma,\tau} f_j &\equiv \sigma(f_k) = f_j \\
[R_\alpha, t_\beta] \sim_{\sigma,\tau} [R_\alpha, t_\alpha] &\equiv R_\beta \sim_{\sigma,\tau} R_\beta \wedge t_\alpha \sim_{\sigma,\tau} t_\beta \\
R_\alpha \sim_{\sigma,\tau} R_\beta &\equiv \begin{array}{l}\mathrm{dom}(\sigma) \subseteq \mathrm{dom}(R_\alpha) \wedge \mathrm{ran}(\sigma) \subseteq \mathrm{dom}(R_\beta) \\ \wedge \forall\, f_k \in \mathrm{dom}(\sigma).\ R_\alpha(f_k) \sim_{\sigma,\tau} R_\beta(\sigma(f_k))\end{array} \\
C_0 \sim_{\sigma,\tau} C_1 &\equiv \begin{array}{l}\mathrm{dom}(\tau) \subseteq \mathrm{dom}(C_0) \wedge \mathrm{ran}(\tau) \subseteq \mathrm{dom}(C_1) \\ \wedge \forall\, \alpha \in \mathrm{dom}(\tau).\ C_0(\alpha) \sim_{\sigma,\tau} C_1(\tau(\alpha))\end{array}
\end{aligned}$$

The high part of the program is ignored for the above L-indistinguishability. That is, it is not part of the typed bijections σ and τ. Indistinguishability is for H-elements really something like "indistinguishability undefined".

Using indistinguishability we define now noninterference as *preservation* of "low"-indistinguishability between pairs of configurations. This is equivalent to saying that the indistinguishability relation is a (weak low)-*bisimulation* [19] over the configuration semantics. "Low" is the set of all elements (activities and futures) identified as low by the security assignment and hence in the domain of σ and τ. The definition of security of an $\mathrm{ASP}_{\mathrm{fun}}$ configuration is given with the following definition of noninterference.

Definition 5.3 (Noninterference): Two $\mathrm{ASP}_{\mathrm{fun}}$ configurations C_0 and C_1 are called *non-interfering* with respect to a security assignment *sp* represented by σ, τ, if whenever they are indistinguishable, i.e. $C_0 \sim_{\sigma,\tau} C_1$ and $C_0 \rightarrow_\parallel C'_0$ there exists a configuration C'_1 that $C_1 \rightarrow^*_\parallel C'_1$ and $C'_0 \sim_{\sigma,\tau} C'_1$. A configuration C is now called *secure* for *sp* if C and C_1 are non-interfering for all configurations C_1 with $C \sim_{\sigma,\tau} C_1$.

B. Classical Service Triangle is Insecure

Let us now show how the formal definition of information flow security, i.e. noninterference, is applied by reconsidering our running example of a service triangle.

$$\begin{array}{rl}
 & \mathrm{data}[\varnothing, [\mathrm{id} = \mathrm{hugo}]] \\
 & \parallel \mathrm{customer}[f_0 \mapsto \mathrm{broker.book}(d, \mathrm{data.id}), t], \\
C_0 \equiv & \parallel \mathrm{broker}[\varnothing, [\mathrm{book} = \varsigma(x, (d, n)) \\
 & \qquad\qquad \mathrm{hotel.room}(d, n), \ldots]] \\
 & \parallel \mathrm{hotel}[\varnothing, [\mathrm{room} = \varsigma(x, (d, n))\mathrm{bookingref}, \ldots]]
\end{array}$$

We want to protect the customer's privacy against the broker as reflected in the following security assignment *sp*.

$$sp \equiv (\{\mathrm{data}, \mathrm{customer}, \mathrm{hotel}\} \mapsto H, \mathrm{broker} \mapsto L\}, \{f_0 \mapsto L\})$$

According to the above assumption, the name `hugo` in `data` is thus confidential. To show that the above configuration, say C_0, is secure with respect to *sp*, we have to prove according to Definition 5.3, that all other configurations C_1 that are indistinguishable with respect to σ, τ containing *sp*, remain so under evaluation – or – fail in the attempt. In fact, as this example is insecure, we must fail. Let us consider an arbitrary configuration C_1 with $C_0 \sim_{\sigma,\tau} C_1$ as follows.

$$\begin{array}{rl}
 & \delta[\varnothing, [\mathrm{id} = \mathrm{ianos}]] \\
 & \parallel \gamma[g_0 \mapsto \beta.\mathrm{book}(d, \delta.\mathrm{id}), t], \\
C_1 \equiv & \parallel \beta[\varnothing, [\mathrm{book} = \varsigma(x, (d, n)) \\
 & \qquad\qquad \alpha.\mathrm{room}(d, n), \ldots]] \\
 & \parallel \alpha[\varnothing, [\mathrm{room} = \varsigma(x, (d, n))\mathrm{bookingref}, \ldots]]
\end{array}$$

Since C_1 is low-indistinguishable to C_0 with respect to *sp*, we can define σ and τ as a bijection of the low future and activity references of *sp*.

$$\begin{aligned}
\tau &\equiv \{\mathtt{broker} \mapsto \beta\} \\
\sigma &\equiv \{f_0 \mapsto g_0\}
\end{aligned}$$

Now, in three steps of evaluation of C_0 we reach the following configuration C'_0.

$$\begin{array}{rl}
 & \mathrm{data}[f_1 \mapsto \mathrm{hugo}, [\mathrm{id} = \mathrm{hugo}]] \\
 & \parallel \mathrm{customer}[f_0 \mapsto f_2, \mathrm{hugo}), t], \\
C'_0 \equiv & \parallel \mathrm{broker}[f_2 \mapsto \mathrm{hotel.room}(d, \mathrm{hugo}), \\
 & \qquad [\mathrm{book} = \varsigma(x, (d, n))\ \ \alpha.\mathrm{room}(d, n), \ldots]] \\
 & \parallel \mathrm{hotel}[\varnothing, [\mathrm{room} = \varsigma(x, (d, n))\mathrm{bookingref}, \ldots]]
\end{array}$$

A configuration C'_1 can be reached similarly from C_1.

$$\begin{array}{rl}
 & \delta[g_1 \mapsto \mathrm{ianos}, [\mathrm{id} = \mathrm{ianos}]] \\
 & \parallel \gamma[g_0 \mapsto g_2, t], \\
C'_1 \equiv & \parallel \beta[g_2 \mapsto \alpha.\mathrm{room}(d, \mathrm{ianos}), \\
 & \qquad [\mathrm{book} = \varsigma(x, (d, n))\alpha.\mathrm{room}(d, n), \ldots]] \\
 & \parallel \alpha[\varnothing, [\mathrm{room} = \varsigma(x, (d, n))\mathrm{bookingref}, \ldots]]
\end{array}$$

Since the future f_0 and g_0 are low, the new futures f_2 and g_2 are low, while the call to the high object data and δ has created the new futures f_1, g_1 as high. Now, σ can only be extended to $\sigma' = \sigma \cup (f_2 \mapsto g_2)$ since it must remain a bijection of the low futures. However, differing from the definition of indistinguishability we have for the request queues

$$R_{C'_0}(f_2) = \mathrm{hotel.room}(d, \mathrm{hugo}) \neq R_{C'_1}(g_2) = \alpha.\mathrm{room}(d, \mathrm{ianos})$$

whereby $\neg(C'_0 \sim_{\sigma,\tau} C'_1)$. Since no further reduction of C'_1 can remedy this, we know by Definition 5.3 that the configuration C_0 is *not secure*.

C. Curried Service Triangle is Secure

Now, we reconsider the same example but this time using the curried call to the booking function in the broker activity. We use the same security assignment $sp = (\{\text{data}, \text{customer}, \text{hotel}\} \mapsto H, \text{broker} \mapsto L\}, \{f_0 \mapsto L\})$ and abbreviate again *date* by d.

$$
\begin{array}{ll}
 & \text{data}[\varnothing, [\text{id} = \text{hugo}]] \\
 & \| \ \text{customer}[f_0 \mapsto \text{broker.book}_C(d).\text{room'}(\text{data.id}), t] \\
C_0 \equiv & \| \ \text{broker}[\varnothing, [\text{book}_C = \varsigma(x,d)\text{hotel.room}_C(d), \ldots]] \\
 & \| \ \text{hotel}[\varnothing, [\text{room} = \varsigma(x,(d,n))\text{bookingref}, \\
 & \qquad \text{room}_C = \varsigma(x,d)[\text{room'} = \varsigma(x',n)x.\text{room}(d,n)]]]
\end{array}
$$

Now, any other indistinguishable configuration, say C_1, would have at least elements corresponding to the low elements f_0 and broker of C_0 because otherwise the bijections σ and τ with $f_0 \in \text{dom}(\sigma)$ and broker $\in \text{dom}(\tau)$ were undefinable. In addition, the methods that are called in the low parts of C_1 must be defined even if they are contained in its high part. So, the least structure we have in an indistinguishable configuration C_1 is up to renaming as follows, unknown parts marked by dots.

$$
\begin{array}{ll}
 & \delta[\ldots, [\text{id} = \ldots]] \\
 & \| \ \ldots [g_0 \mapsto \beta.\text{book}_C(d).\text{room'}(\delta.\text{id}), \ldots] \\
C_1 \equiv & \| \ \beta[\varnothing, [\text{book}_C = \varsigma(x,d)\alpha.\text{room}_C(d), \ldots]] \\
 & \| \ \alpha[\ldots, [\ldots, \text{room}_C = \varsigma(x,d)[\text{room'} = \varsigma(x',n)\ldots]]
\end{array}
$$

The typed bijections are now $\sigma = \{f_0 \mapsto g_0\}$ and $\tau = \{\text{broker} \mapsto \beta\}$. The configuration C_0 can now make the steps we have seen in Section IV-E arriving at the following configuration C_0'.

$$
\begin{array}{ll}
 & \text{data}[f_1 \mapsto \text{hugo}, [\text{id} = \text{hugo}]] \\
 & \| \ \text{customer}[f_0 \mapsto [\text{room'} = \\
 & \qquad \varsigma(x',n)\text{A.room}(d,n)].\text{room'}(\text{hugo}), t] \\
C_0' \equiv & \| \ \text{broker}[f_2 \mapsto f_3, [\text{book}_C = \varsigma(x,d)\text{hotel.room}_C(d), \ldots]] \\
 & \| \ \text{hotel}[f_3 \mapsto [\text{room'} = \varsigma(x',n)\text{A.room}(d,n)], \\
 & \qquad [\text{room} = \varsigma(x,(d,n))\text{bookingref}, \\
 & \qquad \text{room}_C = \varsigma(x,d)[\text{room'} = \varsigma(x',n)x.\text{room}(d,n)]]]
\end{array}
$$

For any arbitrary C_0-indistinguishable configuration C_1 described above we know that we can arrive in a similar configuration C_1', where $\hat{\alpha}$ denotes α's active object.

$$
\begin{array}{ll}
 & \delta[g_1 \mapsto \text{“value of id”} :: \ldots, [\text{id} = \ldots]] \\
 & \| \ \ldots [g_0 \mapsto [\text{room'} = \varsigma(x',n)\ldots].\text{room'}(g_1), \ldots] \\
C_1' \equiv & \| \ \beta[g_2 \mapsto g_3, [\text{book}_C = \varsigma(x,d)\hat{\alpha}.\text{room}_C(d), \ldots]] \\
 & \| \ \alpha[g_3 \mapsto [\text{room'} = \varsigma(x',n)\ldots] :: \ldots, \\
 & \qquad [\ldots, \text{room}_C = \varsigma(x,d)[\text{room'} = \varsigma(x',n)\ldots]]
\end{array}
$$

This parallel reduction $C_1 \to_{\|} C_1'$ might have taken some more steps than $C_0 \to_{\|} C_0'$ but this is legal for a weak bisimulation. The bijection τ is updated to $\tau' \equiv \tau \cup \{f_2 \mapsto g_2\}$; $\sigma' = \sigma$. Since

$$R_{C_0'}(f_2) = f_3 \sim_{\sigma',\tau'} g_3 = R_{C_1'}(g_2)$$

we see that the resulting configurations are low-bisimilar, i.e.

$$C_0' \sim_{\sigma',\tau'} C_1'.$$

That is, the service triangle based on flexible parameterization using currying is secure with respect to the security assignment sp, i.e. preserves the privacy of customer's identity from the service broker.

D. Discussion

Concerning the enforcement of privacy policies in distributed application, the most successful and well-known approach is the Decentralized Label Model (DLM) of A. C. Myers [21]. It enables role based enforcement of program security. In the DLM, explicit labels are used to annotate elements of programs. These labels specify the *principals* that own those program elements as well as who has access to them. The DLM model is founded as our approach on the idea of information flow control; the labels serve as types in a noninterference type systems for static analysis of the allowed information flows. The main criticism to the DLM is that it *assumes that all principals respect the DLM*. We also consider this as a weakness in particular in distributed applications where assumptions about remote parties seems inappropriate. To illustrate this difference: in our example above the DLM would have assumed that the customer's call of book to the broker would also be high and thus be treated confidentially. Contrarily to this strong assumption of the DLM, we do *not make any assumptions about the low site*. In particular the customer can see everything in his request queue, be it marked high or low.

Following the earlier paper [15], we follow the philosophy that private information must not leave the trusted site. However, this is often too strong an assumption. Consider again our example as analyzed in this section. The additional parameter d for the *date* is considered to be low, thus not relevant to the security analysis. However, to avoid invalidating our security analysis, *date* needs to be constant in all applications! Otherwise, the broker would note a difference between changes on the high data. It might seem from this example that our notion of low-indistinguishability is too strict. However, in principle, it is correct to reject the application example if *date* can differ: since it is a parameter set by high, its visibility in broker represents a flow of information from high to low. To overcome this problem, we should consider a possibility to mark certain terms from high as “don't care” in our security model. That is, they are treated as low values, but their actual value is abstracted to make them admissible to low-equality. This represents a kind of down-grading of high values; in our example a “real” *date* of arbitrary value would be downgraded by anonymizing it to a default constant d before passing it on to the broker (this only for the analysis, the program remains unchanged). Then the call would pass as before as indistinguishable.

VI. Conclusions

We have introduced functional active objects, their implementation in Erlang, and how the Erlang active object framework can be employed to support privacy in web-services: using flexible parameterization by currying, we could prevent illegal information flows of private data. This claim has been formally justified using a formal definition for noninterference for ASP_{fun} functional active objects.

A. Related work

In earlier work, we have used Erlang to support privacy for data enquiries [15]. The current work follows the same basic philosophy used in this earlier paper as a naïve solution to the privacy problem in distributed settings: never let the private data leave the trusted home environment, instead import all data and search at home. We already discussed in [15] the implications for the more general setup of distributed active objects – now finally treated here.

Our implementation of futures is – in comparison – the most natural as we base it on message passing. Similar to the ideas recently expressed in Ambient Talk [5], the future is created by the asynchronous send. In other implementations, the future is the result of a remote method invocation and therefore not completely asynchronous: blocking can occur. The next difference to other future implementations is the fact that our future is more active. This means that the future is the active communicator between activities. In addition, this augments privacy: in the example above, the customer is always invisible for broker and hotel. In our current implementation, we decided to declare the future explicitly to show the concrete communication and information flows. Although possible for little examples, it represents a source of fault for complex programs. The idea to hide the complete asynchronous communication can be implemented as a further step. For the time being, it should be seen as a playground for evaluating different strategies. We believe our functional parallel approach even allows us to run activities with circular references without deadlock (because the circle is formed to a helix).

As already discussed in Section IV, when presenting our different strategies to support privacy, there is need to enable users to specify these strategies and consequently to enforce these privacy requirements based on our implementation. Concepts similar to Myers' Decentralized Label Model (DLM) [20] and the related Java implementation JIF based on type systems for information flow control are an adequate means to specify security policies for Java programs and make them amenable for (mostly) static analysis. However, as a prerequisite, a formal proof that typing implies the semantical notion of security, usually noninterference, is necessary. Myers has only lately come up with (partial) proof of soundness of JIF [25].

B. Positioning

For open communication systems, such as web-application and web-services where data and the computation are commonly distributed, there exist several approaches to enforce privacy, in particular the protection of private data. In the scope of local computation, *information flow control* [8], [7] – an approved method to protect data sharing in operating systems – is used more an more. This method, originally a centralized form of mandatory access control, has more recently been extended by decentralized aspects to concern distributed data. A practical implementation is Myer's JIF [21] that tags variables by labels representing owners and readers. The information of a variable is governed by the security policy that is expressed by these labels. A special compiler enforces the security policy by verifying that a program respects its policy. Between verified programs privacy is secured. As pointed out in Section V, we doubt that in a distributed setting it is reasonable to assume that activities stick to the rules. Therefore, our security model uses much weaker assumptions, and still enforces privacy using computational methods, like currying, to enforce data protection. The formal model of currying presented in Section II-E has already been presented in [14] but the notion of security presented there has been further refined in this paper. Here, it is a precise general bisimulation.

Another approach, which goes one step further, is the use of cryptographic protocols such as blind signatures and zero knowledge proof to hide or masquerade real private data. So data can be protected not just against outsider attacks but also against attacks caused by the communication partners. To confirm the cryptographic token a centralized verifier is needed. We consider this technique an important step forward because it can be used in addition to techniques as we use them to support the kind of "downgrading" we consider as necessary based on the security analysis in Section V. In that respect, work on integration of typing techniques for a static security analysis with cryptographically masked flows, e.g. [18], [17] serves the same purpose as our approach of protecting confidential parameters via flexible parameterization.

Complex distributed systems and the provided services are characterized by being often not completely verified. A centralized instance to enforce policies is not always possible or desired and sometimes sharing of private data is necessary. In the scope of intercommunication between the parts of a service, there are trusted and untrusted parts. Of course, programmers can differentiate between trusted and untrusted parts and split the communication process. But with the complexity level of the system, this task becomes more complicated. Our approach tries to implement privacy without a centralized policy instance, splitting communication processes but sharing private data with trusted

parties. For the enforcement of these policies *inside* the local computation we can rely on existing components like JIF.

Nevertheless, being based on a formal computation model, i.e. ASP_{fun}, even distributed systems might be amenable to statically provable security as illustrated in a first approximation in this paper. With this goal in mind, our further plans are to define a security type system enabling static analysis of privacy of an ASP_{fun} program as seen in this paper. The semantic security definition for ASP_{fun} presented in this paper can be used as the basis for proving the correctness of this type system.

References

[1] M. Abadi and L. Cardelli. *A Theory of Objects.* Springer, New York, 1996.

[2] J. Armstrong. *Programming Erlang – Software for a Concurrent World.* The Pragmatic Bookshelf, 2007.

[3] H. Baker and C. Hewitt. The Incremental Garbage Collection of Processes. *Symposium on Artificial Intelligence Programming Languages.* SIGPLAN Notices 12, 1977.

[4] J. Bauer, F. Nielsen, H. Ries-Nielsen, and H. Pilegaard. Relational analysis of correlation. In *Static Analysis, 15th International Symposium, SAS'08*, volume 5079 of *LNCS*, pages 32–46. Springer, 2008.

[5] E. Boix, T. Van Cutsem, J. Vallejos, W. De Meuter, and T. D'Hondt. A Leasing Model to Deal with Partial Failures in Mobile Ad Hoc Networks *TOOLS, 2009.*

[6] D. Caromel and L. Henrio. *A Theory of Distributed Objects.* Springer-Verlag, 2005.

[7] D. E. Denning and P. J. Denning. Certification of programs for secure information flow. *Communications of the ACM*, 20(7), 1977.

[8] D. E. Denning. Lattice model of secure information flow. *Communications of the ACM*, 19(5):236–242, 1976.

[9] T. Gurrock. *A Concurrency Abstraction Implemented for C# and .NET.* Bachelor Thesis. Universität. Paderborn, 2007.

[10] L. Henrio and F. Kammüller. Functional active objects: Noninterference and distributed consensus. Technical Report 2009/19, Technische Universität Berlin, 2009.

[11] L. Henrio and F. Kammüller. Functional Active Objects: Typing and Formalisation. *8th International Workshop on the Foundations of Coordination Languages and Software Architectures, FOCLASA'09*. Satellite to ICALP'09. ENTCS **255**:83–101, Elsevier, 2009.

[12] L. Henrio, F. Kammüller, and B. Lutz. ASP_{fun}: A Functional Active Object Calculus. *Science of Computer Programming*, Elsevier. In print, 2011.

[13] A. Fleck and F. Kammüller. Implementing privacy with Erlang active objects. *The 5th International Conference on Internet Monitoring and Protection, ICIMP'10*. IEEE, 2010.

[14] F. Kammüller. Privacy Enforcement and Analysis for Functional Active Objects. *Fifth International Workshop on Data Privacy Management, DPM'10.* Satellite to ESORICS'10. LNCS 6514, Springer, 2011.

[15] F. Kammüller and R. Kammüller. Enhancing Privacy Implementations of Database Enquiries. *The Fourth International Conference on Internet Monitoring and Protection.* IEEE, 2009. Extended version: Security Analysis of Private Data Enquiries in Erlang. *Int. Journal on Advances in Security*, **2**(2+3), 2009.

[16] M. Uzair Khan and L. Henrio. First class futures: a study of update strategies. Research Report RR-7113, INRIA, 2009.

[17] P. Laud. On the computational soundness of cryptographically masked flows. *35th Symposium on Principles of Programming languages, POPL'08*. ACM 2008.

[18] P. Laud and V. Vene. A type system for computationally secure information flow. *FCT'05*, volume 3623 of *LNCS*, Springer, 2005.

[19] Robin Milner. *Communication and Concurrency*. International Series in Computer Science. Prentice-Hall, Englewood Cliffs, New Jersey, 1989. SU Fisher Research 511/24.

[20] A. C. Myers. Jflow: Practical mostly-static information flow control. In *26th ACM Symposium on Principles of Programming Languages, POPL'99*, 1999.

[21] A. C. Myers and B. Liskov. Protecting Privacy using the decentralized label model. *Transaction on Software Engineering and Methodology, TOSEM* **9**:410–442, IEEE 2000.

[22] L. Paulson. *ML for the Working Programmer*. Cambridge University Press, 1995.

[23] R. G. Lavender and D. C. Schmidt. *An Object Behavioral Pattern for Concurrent Programming* [Online] Available: http://www.cs.wustl.edu/∼schmidt/PDF/Act-Obj.pdf

[24] H. Sutter. The Free Lunch Is Over – A Fundamental Turn Toward Concurrency in Software. *Dr. Dobb's Journal*, **30**(3), 2005.

[25] L. Zheng and A. C. Myers. Dynamic security labels and static information flow control. *International Journal of Information Security*, 6(2–3), 2007.

[26] J. Goguen and J. Meseguer. Security Policies and Security Models. *Proceedings of Symposium on Security and Privacy, SOSP'82*, pages 11-22. IEEE Computer Society Press, 1982.

[27] Muhammad Uzair Khan. *A Study of First Class Futures: Specification, Formalisation, and Mechanised Proofs* University of Nice, PhD Thesis, 2011.

Automatic Tagging of Art Images with Color Harmonies and Contrasts Characteristics in Art Image Collections

Krassimira Ivanova
Institute of Mathematics and Informatics – BAS
Sofia, Bulgaria
kivanova@math.bas.bg

Peter Stanchev
Kettering University
Flint, MI, 48504, USA
Institute of Mathematics and Informatics – BAS
pstanche@kettering.edu

Koen Vanhoof
IMOB
Hasselt University
Hasselt, Belgium
koen.vanhoof@uhasselt.be

***Abstract* – In this paper we present a classification of color harmonies and contrasts, which is consistent with human perceiving of visual expression. It is conformed to the possibilities of automatic extraction of visual information from digitalized copies of art images. The classification is done on the base of the three main characteristics of the color most closed to the human perception – hue, saturation and lightness. Functions for automatic features extraction from digital images are defined. These functions are realized as part of a virtual laboratory "Art Painting Image Color Aesthetic and Semantic" (APICAS). The system can be used by designers and art students for searching images, having certain harmonies or contrasts in image collections as well as a for examining specifics of artists or movements. In future we will use the system as a Web 2.0 service, which could be included in a virtual learning environment.**

Keywords – Content-Based Image Retrieval; Image content; Color; Harmonies; Contrasts

I. INTRODUCTION

One of the most felicitous analogies for presenting the existing semantic gap in area of Content-Based Image Retrieval (CBIR) can be found in "The Hitch-Hiker's Guide to Galaxy" by Douglas Adams. In this story, a group of hyper-intelligent pan-dimensional beings demand to learn the "Answer to Life, the Universe, and Everything" from the supercomputer Deep Thought, specially built for this purpose. It takes Deep Thought 7½ million years to compute and check the answer, which turns out to be "42" [2]. The efforts of covering the semantic gap in CBIR are turned to avoid these misunderstanding between human perceiving and the ways of communications and computer manner of low-level representations.

As it is mentioned in [3], the user questions in image search are partitioned into three main levels:

Low level – this level includes basic perceptual features of visual content (dominant colors, color distribution, texture pattern, etc.). Low-level queries and analysis can support the retrieval of art images in order to seek some specifics or common characteristics between artists, schools or movements.

Intermediate level – this level forms next step of extraction from visual content, connected with emotional perceiving of the images, which usually is difficult to express in rational and textual terms. The visual art is an area, where these features play significant role. Typical features in this level are color harmonies and contrasts, because one of the goals of the painting is to produce specific psychological effects in the observer, which are achieved with different arrangements of colors.

High level – this level includes queries according to rational criterions. In many cases the image itself does not contain information, which would be sufficient to extract some of the characteristics. For this reason current high-level semantic systems still use huge amount of manual annotation.

Different features' levels imply different ways for communication between the user and the CBIR system. When a system uses low-level properties such as color percentages, color layout, and textures (see the pioneer of the area QBIC, developed by IBM [4]), the queries do not need to be described in words. When working with such systems, the user can select a sample image and the system returns all images that are "similar" to it. For systems, which operate with high level features, only choosing a sample or drawing a sketch and search similar characteristics is not sufficient, even because such system has to "know" which of characteristics are targeted by the user. There are two mutually connected tasks in this area:

- Defining features and terms, which present certain effect or criterion and describing correlation between defined concepts;
- Finding appropriate algorithms for generating metadata, which alone or in combination with present terminal features and terms will allow improved image search as well as proposing adequate methods and tools for establishing belonging of a sample to same concept.

This paper presents an experimental software system for intermediate semantic image search based on color harmonies and contrasts. These ideas were firstly introduced in [1]. Section 2 stops the attention on the different sides of the image content. In Section 3 we make an analysis of the phenomenon of the impact of one color on the perception of others. In Section 4 we present a hierarchical classification of

different types of harmonies and contrasts in order to be used as base for further analysis and extraction tools from image databases. In Section 5 we describe an experimental software system, which integrates the proposed tools. Section 6 contains experimental results made by the realized system. Finally, conclusion and future work are presented.

II. TAXONOMY OF ART IMAGE CONTENT

In the last years several efforts have been devoted to the application of image processing and digital imaging techniques in order to facilitate museums activities. Numerous applications have to consider fully or partially some artworks analysis techniques: e.g., virtual restoration of artworks, artistic practices studies, art history investigation, authentication, watermarking, expressive rendering, etc. [5]. From point of view of universal citizen, taking into account that artwork brings a specific authors' message to the viewer the computer should provide the ability to present history, context, and relevance in order to enrich education, enhance cross-cultural understanding, and sustain one's heritage and cultural diversity.

In the field of image retrieval we have faced with obvious difference between human vision system, which has evolved genetically over many millenniums, and computer possibilities, which is limited to processes of capturing and analyzing pixels. Even in this first step of image recognition we have a hard task to find appropriate machine algorithms to represent the picture, which are different of human ways of perceiving, but that can give similar results for interpreting the aesthetic and semantic content in the pictures. Naturally, the interpretation of what we see is hard to characterize, and even harder to teach a machine. Over the past decade, ambitious attempts have been made to make computers learn to understand, index and annotate pictures representing a wide range of concepts, with much progress.

The unique specific of visual pieces of arts is that they are created by a cognitive process. It can therefore be instructive not to only understand the way we look at an artistic image, but also to understand how a human being creates and structures his artwork. Each touch to the artwork causes building the bridge between cultures and times. As was mentioned in [6] "research on significant cultural and historical materials is important not only for preserving them but for preserving an interest in and respect for them".

Different styles in art paintings are connected with used techniques from one side and aesthetic expression of the artist from other side. The process of forming artist style is very complicated process, where current fashion painting styles, social background and personal character of the artist play significant role. All these factors lead to forming some common trends in art movements and some specific features, which distinguish one movement to another, one artist style to another, one artist period to another, etc. From other side the theme of the paintings also stamp specifics and can be taken into account. The compositions in different types of images (portraits, landscapes, town views, mythological and religious scenes, or everyday scenes) also set some rules, aesthetically imposed for some period.

Trying to put some basis for bridging the gaps between interpreting the information from human and from computers several taxonomies of image content as extracted by the viewer of an image are suggested. Alejandro Jaimes and Shih-Fu Chang [7] are focused on two aspects of image content – the received visual *percepts* from the observed images and underlying abstract idea, which corresponds to *concepts*, connected with the image content. In his brilliant survey for 2D artistic images analysis Tomas Hurtut [5] expands taxonomy given by Bryan Burford, Pam Briggs and John Eakins [8]. He gives profiling of extraction primitives and concepts accounting the specific of artworks, splitting image categories into three groups: *image space*, *object space* and *abstract space*.

In our investigation we consent Hurtut's proposition with slightly changes of distribution of features in the groups. We examine *image space*, *semantic space* and *abstract space*. Image space contains visual primitives, needed to record an image through visual perception. Image space includes perceptual primitives (color, textures, local edges), geometric primitives (strokes, contours, shapes) and design constructions (spatial arrangement, composition). Semantic space is related to the meaning of the elements, their potential for semantic interpretation. Semantic space consists of semantic units (objects), 3D relationship between them (scene, perspective, depth cues) and context (illumination, shadow). Abstract aspects are specific to art images and reflect cultural influences, specific techniques as well as emotional responses evoked by an image.

Several big projects addressed the description of the high-level semantic and abstraction concepts in the art domain:

- *The Getty vocabulary databases* [9] are produced and maintained by the Getty Vocabulary Program. They contain terms, names, and other information about people, places, things, and concepts relating to art, architecture, and material culture. The vocabularies in this program are: The Art and Architecture Thesaurus (AAT), the Union List of Artist Names (ULAN), the Getty Thesaurus of Geographic Names (TGN), and finally the Cultural Objects Name Authority (CONA), which expects to be introduced in 2011;
- *WordNet* [10] is a large lexical database of English, developed under the direction of George A. Miller. WordNet is freely and publicly available for download. Although it is not domain-specific, it is a useful tool for computational linguistics and natural language processing especially for English-language texts;
- *Iconclass* [11] is a hierarchical system designed for art and iconography, developed by the Netherlands Institute for Art History. It includes the following main divisions: Abstract, Non-representational Art; Religion and Magic; Nature; Human being, Man in general; Society, Civilization, Culture; Abstract Ideas and Concepts; History; Bible; Literature; Classical Mythology and Ancient History.

In order to present properly concepts and their correlation between low and intermediate levels as well as the connections to the high level, every system usually creates its own dataset. This allows implementing the specific elements of the used methods and tools. Some examples are:

- The "Pictorial Portrait Database" [12] uses a hierarchical database indexing method based on Principal Component Analysis. Its description model is based on the eyes as the most salient region in the portraits;
- An approach for extraction of low level color characteristics and their conversion into high level semantic features using Johannes Itten theory of color, Dempster-Shafer theory of evidence and fuzzy production rules is suggested in [13];
- Hering theory of complementary colors is in the ground of the approach for extracting high level concepts, proposed by [14];
- The team, headed by R. Jain uses annotation of paintings based on brushwork, where brushwork is modeled as part of the annotation of high-level artistic concepts such as the artist name using low-level texture [15].

III. HUMAN PERCEPTION OF THE COLOR

From all the senses that connect us to the world – vision, hearing, taste, smell, and touch – vision is the most important. More than 80% of our sensory experiences are visual [16]. When the brain receives a light stimulus it first interprets form as distinct from background. Figure-ground separation or pattern recognition is the first cognitive step in the process of perception. Color plays an important, but secondary role in recognition. Color responses are more tied to human emotions than to his intellect. Just this property makes the colors very powerful source of influence of human perception. The presence of one or more colors in different proportions conveys different messages, which can increase or suppress the perception of the observed objects.

A. Color

The nature of color is in the focus of research by different science disciplines – Physics studies the power essence of the color, Physiology is interested in the process of human eyes perception of specific wavelengths and theirs transformation to color, Psychology examines the problems of colors' perception and theirs influence on the mentality, Mathematics suggests methods for color measurement. The enormous growth of the number of digital images and videos in different application areas explains the extensive interest in developing computer science methods in this area.

Different models for presenting the color have been created from Antiquity. A detailed survey of color models was made by the team of Urs Baumann [17]. Different models serve various domains – from Physics and Colorimetry; through Painting, Architecture, and Design; to Digital coding for printers, monitors and TV. The history and practice show that a perfect color model cannot be created: one is suitable to supply compact coding and transmitting of the color characteristics, another is easy perceived from humans, etc.

From human point of view, it is most easy to define the color as composition of three components – hue, saturation and lightness. Hue means the name of the color – red, orange, etc. Black, grays and white are called achromatic. Saturation measures the hue intensity or brilliance of a sample, its dullness or vividness. Lightness refers to relative light and dark in a sample [16]. Such point of view to the color facilitates the structuring of color harmonies and contrasts are evinced in art images.

B. Harmonies and Contrasts

The contrasts are experienced when we establish differences between two observed effects. When these differences reach maximal values we talk about diametrical contrast. Our senses perceive only on the base of comparison. For instance one segment is short when lays near long segment and vice versa. In similar way color effect becomes strong or weak thorough contrasts.

The color combinations called "harmonious" in common speech usually are composed of closely similar hues, or else of different colors in the same shades. They are combination of colors that meet without sharp contrast. As a rule, the assertion of harmony or discord simply refers to an agreeable-disagreeable or attractive-unattractive scale.

Many people are observed and examined the influence of the color each other. Aristotle in his "De meteorologica" posed questions about different looking of violet near to white wool and black wool [18]. His questions were systematically examined and explained later by Michel Eugène Chevreul.

In 1772 – the same year that Johann Heinrich Lambert constructed his color pyramid and demonstrated for the first time that the complete fullness of colors can only be reproduced within a three dimensional system [19], another color circle was published in Vienna by Ignaz Schiffermüller. He was one of the first, who arranged the complementary colors opposite one another: blue opposite orange; yellow opposite violet; red opposite green [18].

Leonardo da Vinci (1452-1519) had probably been the first to notice that when observed adjacently, colors will influence each other. Goethe, however, was the first who specifically draw attention to these associated contrasts.

Michel Eugène Chevreul (1786-1889) had continued resolving the questions for contrast with establishing a law of "Simultaneous Contrast" [18]. When colors interact, they are capable of changing in appearance, depending on particular relationships with adjacent or surrounding colors. Simultaneous contrast is strongly tied to the phenomenon of afterimage, also known as "Successive contrast", when the eye spontaneously generates the complementary color even when the hue is absent. The explanation of successive contrast is given in opponent color vision theory, which acquired its integral view in the works of Ewald Hering in 1872 [18]. Successive and simultaneous contrast suggest that

the human eye is satisfied, or in equilibrium, only when the complemental relation is established.

The great contribution in revealing effects of color interactions has Josef Albers (1888-1976). His book "The Interaction of Color" [20] has proven key to understanding color relationships and human perception. Albers stated that one color could have many "readings", dependent both on lighting and the context in which the color is placed. He felt that the comprehension of color relationships and interactions was the key to gaining an eye for color. According to Albers, we rarely see a color that is not affected by other colors. Even when a color is placed against a pure neutral of black, white, or gray, the color is influenced by that neutral ground. Colors interact and are modified in appearance by other colors in accordance with three guiding rules: *Light/dark value contrast*, *Complementary reaction*, and *Subtraction*.

Johannes Itten (1888-1967) continued theories of Albers. He was one of the first to define and identify strategies for successful color combinations [21]. Through his research he devised seven methodologies for coordinating colors utilizing the hue's contrasting properties. These contrasts add other variations with respect to the intensity of the respective hues; i.e., contrasts may be obtained due to light, moderate, or dark value. He defined the following types of contrasts: *Contrast of hue*, *Light-dark contrast*, *Cold-warm contrast*, *Complementary contrast*, *Simultaneous contrast*, *Contrast of saturation*, and *Contrast of extension (proportion).*

C. Artists' color wheel

Usually, in accordance of Johannes Itten proposition, the color wheel, which represents relations between hues, is divided in twelve sections. Centers of three equidistance sections correspond to primary colors. Between them secondary colors are posed, which from one side are middle points of two primary colors, and from other side are complementary to the third color. The quantization is expanded with the intermediate colors, which lays at midpoint to adjacent primary and secondary hues.

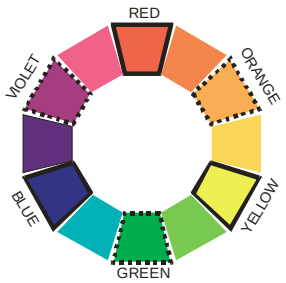


Figure 1. Standard artists' color wheel

In Figure 1 the position of the hues in standard artists' color wheel is shown. This order and correlations between hues is described in RYB (Red-Yellow-Blue) color model, used by the artists. Let us mention that this arranging of hues differs from many of contemporary color models – RGB (Red-Green-Blue), CMY (Cyan-Magenta-Yellow), HSL (Hue-Saturation-Luminance), HSV (Hue-Saturation-Value), based on the defining of colors as primary or secondary in accordance with trichromatic theory [22].

IV. Classification of Harmonies and Contrasts

We present one classification of different types of harmonies and contrasts, from the point of view of the three main characteristics of the color – hue, saturation and lightness.

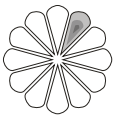
A. Harmonies/contrasts from point of view of hue

There can be examined two different types of harmonies/contrast: ones that take into consideration only disposition of hues each other and others that account exact hue values and their influence on the human perceiving.

1) Hue harmonies/contrasts based on the disposition of hues

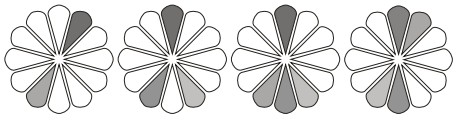
The figures below shows only relatively disposition of the colors, not the absolute meaning of the color. Some of these combinations are discussed in [16] and [23].

a) Monotone compositions: These compositions use one hue, and image is built on the base of varying of lightness of color. These images are used to suggest some kind of emotion since every hue bears specific psychological intensity;



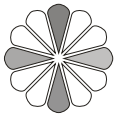
b) Analogous hues: Analogous hues can be defined as groups of colors that are adjacent on the color wheel; contain two, but never three primaries and have the same hue dominant in all samples;

c) Complementary contrasts: Complementary colors are hues that are opposite one another on the color wheel. When more than two colors take part in the composition the harmonic disposition suggests combination between analogous and complementary hues;



d) Triads: Three colors that are equidistance on the color wheel form triad. This means that all colors are primary or secondary, or intermediate;

e) Tetrads: The tetrad includes four colors in equidistance on the color wheel. This contrast produces very complicated scheme and can lead to disharmony;



f) Achromatic compositions: As a special case, images composed by black, grays and white tones or contain colors with very small saturation.

2) Harmonies/contrasts based on the group of hues (Warm-cold contrast)

Warm and cold are two opposing qualities of hue. Warm colors are hues around red and orange; cold colors are these around blue. The terms warm and cold are helpful for describing families of colors. They can be defined as follows:

a) Warm: The image is warm when composition is built from family of warm colors;

b) Cold: By analogy – the image is cold when it is composed only (or predominantly) with cold colors;

c) Neutral: The composition contains colors mainly from neutral zones;

d) Warm-cold: The composition lays in this category when the percentage of cold family is in some proportion to the percentage of warm family;

e) Warm-neutral: In such compositions there is proportion between warm colors and neutral ones;

f) Cold-neutral: The image contains cold and neutral tones in some proportion.

Unlike of hue, which is circular and continuous, saturation and lightness are linear. That difference determines different definitions of harmonies for these characteristics.

B. *Harmonies/contrasts from point of view of saturation*

This harmony appears together with the hue ones. It is used to give different perception when the color is changed. As a whole we can define three big groups of harmonies and contrasts:

a) Dull: An image can be classified as dull when composition is constructed mainly from desaturated colors;

b) Clear: Clear images have been build mostly from clear (spectral and near to spectral, respectively only with varying in lightness) colors.

c) Different proportion of saturations: Usually in composition of clear colors in combination of dull ones. Depending on content of different saturation and of distance between predominate quantities harmonies can be defined such as *smooth*, *contrary*, etc.

C. *Harmonies/contrasts from point of view of lightness*

The whole effect of the lightness of the image as well as light-dark contrast is a very powerful tool in art mastering. Mainly, an artwork can not contain light-dark contrast – at that case the image has one integral vibration of the lightness. In other case sharp light-dark contrast is used to focus the attention in exact points of the image.

a) Dark: Dark compositions are built mainly from dark colors;

b) Light: Light images contain mostly colors near white;

c) Different proportion of lightness: Light colors combined with dark ones compose the image. Depending on content of different lightness and of distance between predominate quantities contrasts can be defined as*: smooth*, *contrary*, etc.

V. EXPERIMENTAL SOFTWARE SYSTEM FUNCTIONALITY

An experimental software system for automatic image descriptor annotation, which corresponds to defined harmonies and contrasts, was created in the frame of a virtual laboratory for semantic image retrieval APICAS.

We have used analyses of the images and artists' styles, made in [18][21][24][25][26] to tune up our algorithms and parameters.

For the purposes of the system we convert RGB-values of color of each pixel to values in non-uniformly quantized HSL-feature space – twelve hues plus one value for achromatic color, five levels of saturation and five levels of luminance are identified. The numbers of the layers are chosen on the base of Itten's color theory.

The system allows user definitions of the quantization of the space. Figure 2 shows the screen where the user can set up quantization for the purposes of further defining of color harmonies or contrasts. The screenshot is made when quantization of hue is in accordance with artists' color wheel. The displacement between correlation of hues in two color models – RYB and HSL is clearly seen. Current realization of defining hue dispositions is based on RYB color space.

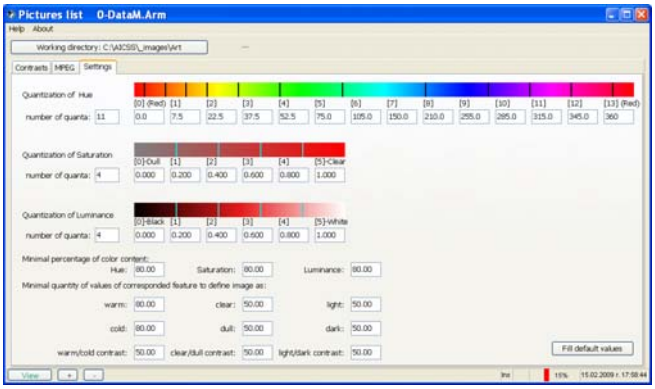


Figure 2. Screen for set up the quantization parameters and boundaries

The saturation and luminance are quantized in five levels. The boundaries can be set up by the user. By default, equal quantization is proposed.

In previous version [1] we have used exact function of defining the belonging of the color characteristic to quantizing segment. Now, quantization of colors is made using fuzzy calculating of belonging of color to corresponded index (Figure 3). If the position of examined value is in inner part of one defined segment (more than one half from the left bound and less than three half from the right bound) the characteristic is considered to belong to this segment. In other case (except the endmost parts for saturation and lightness), part of the characteristic is considered to belong to this segment and the rest part is considered to belong to the adjacent segment. For receiving that part a linear function, which reflects the decreasing of belonging of that characteristic to the segment, is used.

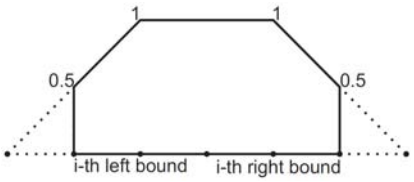


Figure 3. Fuzzy function for calculating quantization part of color characteristic

Taking into account our earlier examination of the distribution of color components in art paintings [27] we make normalization of the colors in respect of hue distribution (Figure 4). This normalization allows simplifying of comparing presence of color characteristic values in further stages.

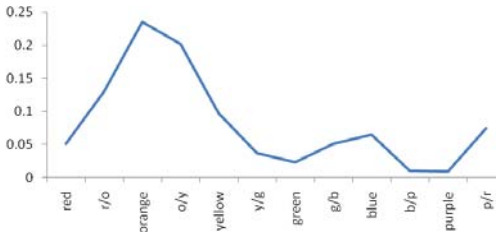


Figure 4. Average distribution of hue component in art paintings

As a result, every picture is represented with three dimensional array containing coefficients of participation of colors with correspondingly measured characteristics of the picture.

A={A(ih,is,il)|ih=-1,...,NH-1; is=0,...,NS-1; il=0,...,NL-1}

Here NH=12 and corresponds to the number of quantized colors in Ittens' circle. "-1" index percentage of achromatic tones; "0" to "NH-1" points percentage of colors, ordered as it is shown on Figure 1, starting from reds and ending to purples.

In our examination NS=5. Index "0" holds percentage of grays and almost achromatic tones, and "4" contains percentage of pure (in particular – spectral) tones.

For indexing of luminance we use NL=5 also. "0" holds percentage of very dark colors, and "4" contains percentage of very light colors.

On the base of this array for simplification of further calculation in some cases three arrays, containing percentage values of corresponding characteristics in the picture is calculated. These arrays are:

- H (h_{-1}, h_0, … , h_{NH-1}) for hues;
- S (s_0, …, s_{NS-1}) for saturation;
- L (l_0, …, l_{NL-1}) for lightness.

A. *Hue order vector*

This vector contains number of dominant hues *nh*, and positions of dominant hues, ordered in decreasing percentage. *nh* can vary from zero for achromatic paintings, to maximal defined dominant colors. For the purposes of defining hue harmonies maximal dominant colors are restricted in this example to 5. When image is not achromatic the value of *nh* is defined as the number of ordered hues, which sum of the percentages exceed some (expert-defined) value *x*.

(nh; p_1, p_2, …, p_{nh}),

$nh \in \{0, \ldots, 5\}$

$p_i \in \{-1, \ldots, NH-1\}$ *and* $h_{p_i} \geq h_{p_{i+1}}$, $i \in \{1, \ldots, nh-1\}$

$$nh: \begin{cases} nh = 1 \ \ if \ \ h_{p_1} \geq x \\ nh = n \ \ if \ \ \sum_{i=1}^{n-1} h_{p_i} < x \ \ and \ \ \sum_{i=1}^{n} h_{p_i} \geq x \end{cases}$$

B. *Hue harmony/contrast, based on disposition*

For defining hue harmonies/contrasts first we define:

$$opposite(p) = \begin{cases} p + NH \ div \ 2 & if \quad p \leq NH \ div \ 2 \\ p - NH \ div \ 2 & if \quad p \geq NH \ div \ 2 \end{cases}$$

$$l\_neighbour(p) = \begin{cases} NH - 1 \ if \ \ p = 0 \\ p - 1 \ if \ \ p \ in \{1, ..., NH - 1\} \end{cases}$$

$$r\_neighbour(p) = \begin{cases} 0 \ if \ \ p = NH - 1 \\ p + 1 \ if \ \ p \ in \{0, ..., NH - 2\} \end{cases}$$

$$l\_triad(p) = (NH + p - NH \ div \ 3) \bmod NH$$

$$r\_triad(p) = (p + NH \ div \ 3) \bmod NH$$

$$l\_tetrad(p) = (NH + p - NH \ div \ 4) \bmod NH$$

$$r\_tetrad(p) = (p + NH \ div \ 4) \bmod NH$$

The values of the hue harmony depend from the number of dominant hues *nh*:

- *nh*=0:

Achromatic: the composition is constructed by black, white and gray tones. This construction can be examined as special case of monochromatic harmony;

- *nh*=1:

Monochromatic: only one hue predominates in image;

- *nh*=2:

Analogous: when p_2=*l_neighbour*(p_1) or p_2=*r_neighbour*(p_1);

Complementary: when p_2=*opposite*(p_1);

Partial Triad: when p_2=*l_triad*(p_1) or p_2=*r_triad*(p_1);

- *nh*=3:

Analogous: if for one of dominant hues p_i ($i \in \{1,\ldots,nh\}$) is fulfilled that the other two colors are *l_neighbour*(p_i) and *r_neighbour*(p_i) respectively;

Split complementary: if for one of dominant hues p_i ($i \in \{1,\ldots,nh\}$) is fulfilled that the other two colors are *l_neighbour*(*opposite*(p_i)) and *r_neighbour*(*opposite*(p_i));

Triad: if for one of dominant hues p_i ($i \in \{1,\ldots,nh\}$) the other two colors are *l_triad*(p_i) and *r_triad*(p_i);

- *nh*=4:

Analogous: if for one of dominant hue p_i ($i \in \{1,\ldots,nh\}$) is fulfilled that one of the other three colors p_j ($j \in \{1,\ldots,nh\}$, $j \neq i$) p_j=*l_neighbour*(p_i) or p_j=*r_neighbour*(p_i) and other two colors are *l_neighbour*(p_j) and *r_neighbour*(p_j);

Double Complementary: if for one of dominant hue p_i ($i \in \{1,\ldots,nh\}$) is fulfilled that one of the other three colors p_j ($j \in \{1,\ldots,nh\}$, $j \neq i$) p_j=*opposite*(p_i) and other two colors are *l_neighbour*(p_i) and *l_neighbour*(p_j) or *r_neighbour*(p_i) and *r_neighbour*(p_j);

Split Complementary: if for one of dominant hue p_i ($i \in \{1,\ldots,nh\}$) is fulfilled that one of the other three colors p_j ($j \in \{1,\ldots,nh\}$, $j \neq i$) p_j=*opposite*(p_i) and other two colors are *l_neighbour*(p_j) and *r_neighbour*(p_j);

Tetrad: if for first hue p_1 the other hues are *l_tetrad*(p_1), *opposite*(p_1), *r_tetrad*(p_1) respectively;

- *nh*=5:

Multicolor: here can be searched the presence of defined combinations discarding one of the colors.

C. Cold/warm contrast

For defining cold/warm contrast the system compares percentage values of families of colors p_{warm}, p_{cold}, and $p_{achromatics}$. We have take into account the fact of changing the type of some color in dependency of its saturation and lightness [28]. Because of this we calculate the values of p_{warm}, p_{cold}, and $p_{achromatics}$ on the base of three-dimensional array A . The strongest contrasts points is the warmest "red-orange" (ih=1) and the coolest "blue-green" (ih=7). We use semi-linear function of including colors in warm, respectively cold family, whit following properties:

- all achromatic values (ih=-1) and very desaturated colors (is=0) are added to achromatic family;
- increasing the lightness in desaturated colors (is=1,2) leads to increasing of coldness. For instance dark desaturated colors is added in warm family from magenta to orange-yellow (ih in {11, 0, 1, 2, 3}), but from light ones only red and red-orange are added (ih in {0, 1}). Conversely, dark colors added to cool family are only these near blue-green (ih in {6, 7, 8}); increasing the light expands the range and in cool family from lightest yellow-green to lightest blue-magenta (ih in {5, 6, 7, 8, 9}) and half of neighbors are included (ih in {4, 10});
- colors with middle saturation (is=3) include stable families of warm colors (ih in {0, 1, 2, 3}) and cold colors (ih in {6, 7, 8});
- for saturated colors (is=4) increasing the lightness cause expanding of both families of warm and cold colors. For instance for dark saturated colors in warm family belongs from magenta to orange-yellow (ih in {11, 0, 1, 2, 3}), while in light spectrum half of their neighbors also are included (ih in {10, 4}).

The image is defined as *warm*, *cold*, or *neutral* if corresponding value is greater than some threshold. If none of these values exceeds given parameters, the image is *warm-cold*, *warm-neutral*, *cold-neutral* according to order of decreasing of corresponded values.

D. Saturation order vector

This vector contains number of dominant saturations *ns* ($ns \in \{1,\ldots, NS\}$), and positions of dominant saturations, ordered in decreasing percentage. The value of *ns* is defined as the numbers of ordered saturations, which sum of the percentages, exceed some value *y*.

(*ns*; p_1, p_2, …, p_{ns}),

$ns \in \{1, \ldots, NS\}$

$p_i \in \{0, \ldots, NS\text{-}1\}$ *and* $s_{p_i} \geq s_{p_{i+1}}$, $i \in \{1, \ldots, ns\text{-}2\}$

$$ns: \begin{cases} ns = 1 \ \ if \ \ s_{p_1} \geq y \\ ns = n \ \ if \ \ \sum_{i=1}^{n-1} s_{p_i} < y \ \ and \ \ \sum_{i=1}^{n} s_{p_i} \geq y \end{cases}$$

E. Saturation combinations

If *ns*=1 the picture is defined as *monointense*. If *ns*>1 some combinations of presence of dominant saturations can be outlined. For instance, if p_0 and $p_{NS\text{-}1}$ are dominant saturations, the image can be defined as *contrary*; if saturations are adjoining – the feature is *smooth*, etc.

F. Clear/dull contrast

Depending of the global lightness of the image the saturation distribution of the image is possessed in another attribute, which can receive values as *soft* or *sharp* for light images, *ground* or *spectral* for images with medium lightness and *dull* or *clear* for dark images.

G. Lightness order vector

This vector (nl; p_1, p_2, …, p_{nl}) is defined in the same way as the saturation order vector. It contains number of dominant lighting values nl ($nl \in \{1,\ldots, NL\}$), and their positions, ordered in decreasing percentage.

H. Lightness combinations

These values are defined in the equal manner as saturation ones – the same function are used; only corresponding parameters are changed.

I. Light/dark contrasts

The attribute, which receives values for light-dark contrast depends of user defined threshold of darkness and lightness. The images, which hold l_0 more than given dark threshold, are identified as *very dark*. *Dark* images are these for which l_0+l_1 exceed this threshold. Similarly, the images with l_4 receive value *very light* and these for which l_3+l_4 exceed the threshold are *light*. Depending of distribution of lightness, images can be categorized as *dark-light*, *light-dark*, *middle*, etc.

VI. Experimental Software System Realization

The proposed tools for automatic annotation of the images with harmonies' and contrasts' descriptors are realized as part of a virtual laboratory for image retrieval "Art Painting Image Color Aesthetic and Semantic" (APICAS).

A. Data entry

The system operates with images in JPEG-format. Images, stored in one directory, form a collection.

The user can choose the specific collection by changing the working directory. The system automatically scans the collection and extracts features. The user can refine setting of some parameters or boundaries (see Figure 2), which provoke recalculating of the corresponded descriptors.

The files, used in these collections, contain in their names the information about the artist and the name of the stored painting. The system extracts the names of the picture and the artist and, using a small thesaurus with information about the artists (dates of birth and death; countries; movement; periods), connects these metadata, extracted by the context, to information for the pictures. This way for automatic metadata extraction is applied in order to ensure easy way for making the experiments and analyzing the results.

B. Access

The extracted descriptors (from the content and from the context) can be observed in a grid. The user can sort it by any selected feature. Pointing on the exact image, the user can see all extracted metadata, connected to this image – an example is given on Figure 5.

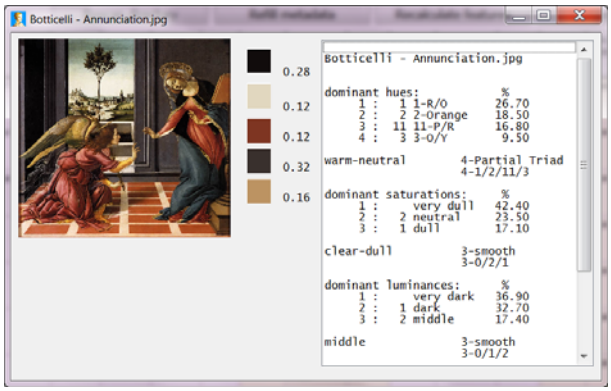


Figure 5. Results of calculating of types of harmonies/contrasts for the picture "Annunciation" by Botticelli

The user can set different conditions on the extracted descriptors and receive the images that satisfy these conditions. The results can be obtained in two forms:

- in thumbnail form, where the images can be seen. An example of such result is shown on Figure 6;
- in a file, where selected images can be additionally batched using other features, selected by user.

The system allows searching within a collection of images, which has specific combination of the colors, defined by some harmony or contrast.

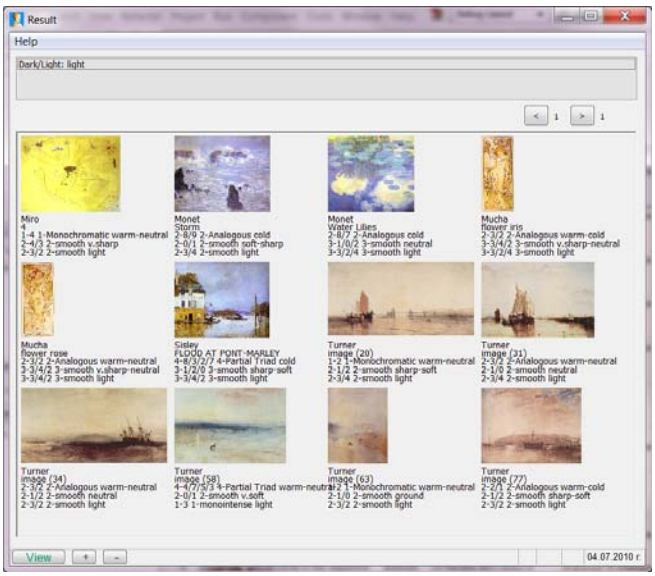


Figure 6. Result of retrieval from the image base with parameter: "Dark/light contrast = Light"

Another part of the system allows creating a datasets, containing extracted attributes or selected part of them labeled with chosen profile such as artist name, movement, scene-type. These datasets can be used for further analysis by data mining tools for searching typical combinations of characteristics, which form profiles of artists or movements, or reveal visual specifics, connected to the presented thematic in the images.

VII. EXPERIMENT RESULTS

For our experiments we have used a dataset that includes 600 paintings of 18 artists from different movements of West-European fine arts and one group, which represent Orthodox Iconographic Style from Eastern Medieval Culture (Table 1). The paintings were chosen by an art expert reviewer. He has included in the collection the most valuable paintings for every movement. The pictures were obtained from different web-museums sources using ArtCyclopedia as a gate to the museum-quality fine art on the Internet [29].

TABLE I. LIST OF THE ARTISTS, WHICH PAINTINGS WERE USED IN EXPERIMENTS, CLUSTERED BY MOVEMENTS

Movement	Artist
Icons (60)	Icons (60)
Renaissance (90)	Botticelli (30); Michelangelo (30); Raphael (30)
Baroque (90)	Caravaggio (30); Rembrandt (30); Rubens (30)
Romanticism (90)	Friedrich (30); Goya (30); Turner (30)
Impressionism (90)	Monet (30); Pissarro (30); Sisley (30)
Cubism (90)	Braque (30); Gris (30); Leger (30)
Modern Art (90)	Klimt (30); Miro (30); Mucha (30)

A. *Distribution of some harmonies/contrasts in art paintings*

Here some examples of distribution of defined features by movements or artists styles are presented.

In these experiments we have used HSL-artist color model with fuzzy calculating of belonging of color to corresponded index.

Figure 7 shows the distribution of images from different movements, based on cold/warm contrast. The high predominance of warm paintings in ICON style can be explained with the orthodox tradition for using gold paints as well as red color, which is main symbol of sacrificing and martyrdom. The big presence of dark warm colors is specific for the Baroque. Presenting the nature in paintings is typical for the Romanticism, which leads to forcing the presence of cold (green and blue) tones. This tendency increases in the Impressionism. Intensive study of nature led the Impressionists to an entirely new color rendition. Study of sunlight, which alters the local tones of natural objects, and study of light in the atmospheric world of landscape, provided the Impressionist painters with new essential patterns [21].

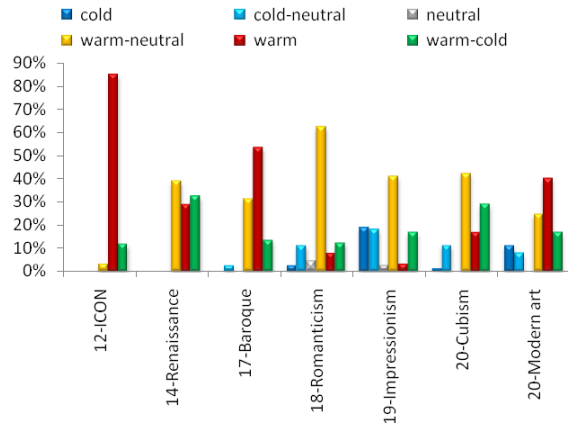


Figure 7. Distribution of paintings, grouped by movements, based on cold/warm contrast

Figure 8 shows the distribution of lightness in paintings from different movements. The big presence of dark colors and dark-light contrast is typical for Baroque. This is connected with using the techniques of oil-paints, which gives very deep dark effects in the paintings from one side and with typical using of light-dark contrast in this movement. This fact is connected not only with searching of maximal expression with applying this tool in the paintings, but also with the practice of this epoch to paint in the candle lights in studios [18].

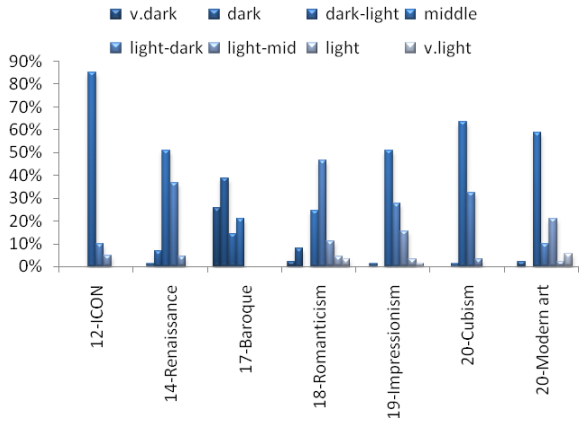


Figure 8. Lightness distribution of paintings, grouped by movements

Figure 9 shows the distribution of images in different movements, based on the first dominant hue. As we have observed in our previous work [27] the colors around orange are frequently dominant colors in the paintings in classic art. More modern movements tend to use different colors as dominant.

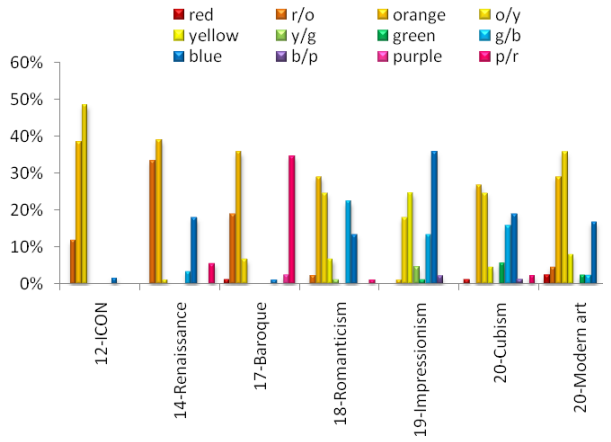


Figure 9. Distribution of paintings, grouped by movements, based on first dominant hue

Figure 10 shows the distribution of hue contrasts in the paintings, clustered by authors.

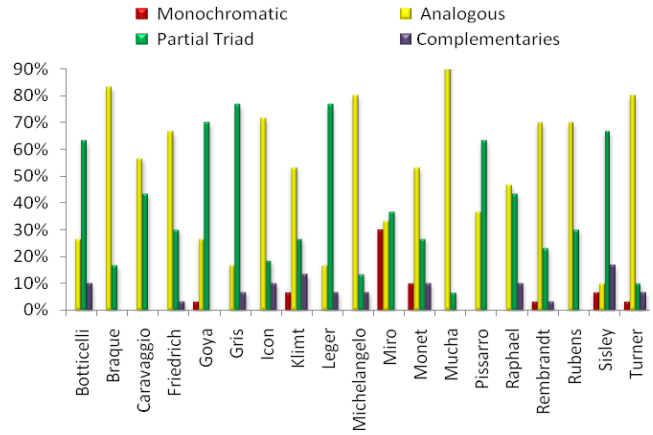


Figure 10. Percentage of different hue contrasts in the paintings of examined movements

As we can see partial triads are used in a lot of cases of natural paintings, for instance Pissarro and Sisley. The triads exist in paintings with scene presentation from authors, which techniques are based mainly on hue contrasts, such as Botticelli and Goya. Monochromaticity and analogous harmonies are presented in artworks of painters, where other key expressions are used, for instance light-dark contrast in Baroque artists, gradient expressions in Braque style, Miro's abstract paintings, etc. [26].

B. Analysis of combination of defined features

The collective of Institute of Mathematics and Informatics in Bulgarian Academy of Sciences has created data mining environment system PaGaNe [30]. Using the associative rule miner, realized in PaGaNe we have made more complicated analysis for extracting combinations of extracted features typical for examined artists. For instance, for more than one third of paintings, combinations of four attributes are presented in Table II.

TABLE II. COMBINATIONS OF FOUR ATTRIBUTES, WITH MORE THAN 33.33% SUPPORT FOR EXAMINED ARTISTS

Artist	4-items combinations	support
CARAVAGGIO	Sat. Harmony =3-SMOOTH Lum. Harmony =2-SMOOTH Warm-cold contrast =WARM-NEUTRAL Clear-dull contrast =CLEAR-DULL	33.33
GRIS	Hue Harmony =PARTIAL TRIAD Sat. Harmony =4-VARIETY Lum. Harmony =3-SMOOTH Dark-light contrast =MIDDLE	36.67
GRIS	Hue Harmony =PARTIAL TRIAD Lum. Harmony =3-SMOOTH Clear-dull contrast =SPECTRAL-GROUND Dark-light contrast =MIDDLE	33.33
ICON	Hue Harmony =ANALOGOUS Lum. Harmony =3-SMOOTH Warm-cold contrast =WARM Dark-light contrast =MIDDLE	35.00
ICON	Hue Harmony =ANALOGOUS Sat. Harmony =3-SMOOTH Warm-cold contrast =WARM Dark-light contrast =MIDDLE	31.67
MUCHA	Hue Harmony =ANALOGOUS Sat. Harmony =3-SMOOTH Lum. Harmony =3-SMOOTH Warm-cold contrast =WARM	36.67
MUCHA	Hue Harmony =ANALOGOUS Sat. Harmony =3-SMOOTH Lum. Harmony =3-SMOOTH Clear-dull contrast =SPECTRAL-GROUND	33.33
MUCHA	Hue Harmony =ANALOGOUS Lum. Harmony =3-SMOOTH Warm-cold contrast =WARM Clear-dull contrast =SPECTRAL-GROUND	33.33
RUBENS	Hue Harmony =ANALOGOUS Sat. Harmony =3-SMOOTH Lum.Harmony =2-SMOOTH Warm-cold contrast =WARM	33.33
REMBRANDT	Hue Harmony =ANALOGOUS Warm-cold contrast =WARM Clear-dull =CLEAR-DULL Dark-light =DARK	40.00
REMBRANDT	Warm-cold contrast =WARM Hue harmony =ANALOGOUS Clear-dull contrast =CLEAR-DULL Sat. Harmony =1-MONOINTENSE	36.67
REMBRANDT	Warm-cold contrast =WARM Clear-dull contrast =CLEAR-DULL Lum. harmony =1-MONOINTENSE Dark-light =DARK	33.33

Such approach of extracting rules from frequent datasets as well as their extension in the direction of class association algorithms can be used for defining semantic profiles of observed phenomena – movement, artists style or thematic, connected with abstract space of the taxonomy of the art image content, discussed by T. Hurtut [5].

VIII. Conclusion and Future Work

In this article we presented a novel and more complete classification of color harmonies by three main characteristics of the color, which is most close to the human perception. We used this classification in a designated software tool, which extracts the defined features from an image.

The next step will be to extend these concepts, adding texture features, which will allow us to address additional definitions of contrasts, presented in Ittens' theory.

One of the directions for future work will be to conduct experiments for the educational use of the resource and build a service, which could be included in a virtual learning environment and will allow student to search for similar images.

Another future application of the method and tool presented in the article is to integrate it in image databases as a service for generation of tags for use within Web 2.0 services. The huge amount of digital objects and the metadata bottleneck are well known; such a system will not produce the human social tagging but could generate image-characteristics tags like ‘greenish’, ‘scarlet’, ‘pale’, ‘dark’, which will be useful in image searches.

This makes the presented tool a natural component within the virtual laboratory for semantic image retrieval. The work presented here provides a good basis for these subsequent developments.

Acknowledgment

This work is partially financed by Bulgarian National Science Fund under the project D002-308/19.12.2008 "Automated Metadata Generating for e-Documents Specifications and Standards".

References

[1] Kr. Ivanova, P. Stanchev, "Color harmonies and contrasts search in art image collections", First International Conference on Advances in Multimedia MMEDIA, 20-25.07.2009, Colmar, France, pp. 180-187.

[2] D. Adams, The Hitch-hiker’s Guide to Galaxy, Pan Macmillan, 1979.

[3] V. Castelli, L. Bergman (eds.), Image Databases: Search and Retrieval of Digital Imagery, John Wiley & Sons, 2002.

[4] M. Flickner, H. Sawhney, W. Niblack, J. Ashley, Q. Huang, B. Dom, M. Gorkani, J. Hafner, D. Lee, D. Petkovic, D. Steele, P. Yanker, "Query by image and video content: the QBIC system", Computer, 1995, pp. 23-32.

[5] T. Hurtut, 2D artistic images analysis, a content-based survey, 2010, http://hal.archives-ouvertes.fr/hal-00459401_v1/, 10.01.2011.

[6] C.-C. Chen, H. Wactlar, J. Wang, K. Kiernan, "Digital imagery for significant cultural and historical materials – An emerging research field bridging people, culture, and technologies", International Journal Digital Libraries, 5(4), 2005, pp. 275–286.

[7] A. Jaimes, S.-F. Chang, "Concepts and techniques for indexing visual semantics", Image Databases: Search and Retrieval of Digital Imagery. V. Castelli and L. D. Bergman, John Wiley & Sons, 2002, pp. 497-565.

[8] B. Burford, P. Briggs, J. Eakins, "A taxonomy of the image: on the classification of content for image retrieval", Visual Communication, 2(2), 2003, pp. 123-161.

[9] Getty Vocabulary Program, http://www.getty.edu/research/conducting_research/vocabularies/, 10.01.2011.

[10] WordNet: a lexical database for the English language, http://wordnet.princeton.edu/, 10.01.2011.

[11] Iconclass, http://www.iconclass.nl/, 10.01.2011.

[12] C. Saraceno, M. Reiter, P. Kammerer, E. Zolda, W. Kropatsch, "Pictorial portrait indexing using view-based eigen-eyes", D. Huijsmans and A. Smeulders (eds), Visual Information and Information Systems, Lecture Notes in Computer Science, vol. 1614, 1999, pp. 649-656.

[13] P. Stanchev, D. Green Jr., B. Dmitrov, "High level color similarity retrieval", International Journal on Information Theories and Applications, vol.10, Num.3, Sofia, 2003, pp 283-287.

[14] J. Lay, L. Guan, "Retrieval for color artistry concepts", IEEE Trans. on Image Processing 13, 3, 2004, pp. 326-339.

[15] Y. Marchenko, T. Chua, R. Jain, "Semi-supervised annotation of brushwork in painting domain using serial combinations of multiple experts", Technical Report, NUS, Singapore, 2006.

[16] L. Holtzschue, Understanding Colors, John Wiley & Sons, 2006.

[17] Color museum, ©Echo Productions, http://www.colorsystem.com/, 10.01.2011.

[18] J. Gage, Colour and Culture: Practice and Meaning from Antiquity to Abstraction, Thames and Hudson, London, 1993.

[19] W. Spillmann, "Color systems", Color Consulting, H. Linton, New York, 1992, pp. 169-183.

[20] J. Albers, Interaction of Color, Yale University Press; Revised edition, 1975.

[21] J. Itten, The Art of Color: the Subjective Experience and Objective Rationale of Color, Reinhold Publishing Corporation of New York, 1961.

[22] A. Colman, A Dictionary of Psychology, 2nd ed., Oxford University Press, Oxford, 2006.

[23] L. Eiseman, Color: Messages and Meanings. A PANTONE Color Resource, Hand Books Press, 2006.

[24] R. Arnheim, Art and Visual Perception: A Psychology of the Creative Eye, University of California Press, Berkeley, 1974.

[25] M. Walch, A. Hope, Living Colors – The Definitive Guide to Color Palletes through the Ages, Chronicle Books, San Francisco, 1995.

[26] B. Koenig, Color Workbook, Prentice Hall, Third edition, 2010.

[27] Kr. Ivanova, P. Stanchev, B. Dimitrov, "Analysis of the distributions of color characteristics in art painting images", Serdica Journal of Computing, vol.2, num.2, Sofia, 2008, pp. 111-136.

[28] K. Clark, Civilisation – a Personal View. British Broadcasting Corporation, London and John Murray, London. 1969.

[29] ArtCyclopedia: The Guide to Great Art on the Internet, http://www.artcyclopedia.com/, 10.01.2011.

[30] I. Mitov, K. Ivanova, K. Markov, V. Velychko, K. Vanhoof, P. Stanchev, "PaGaNe – a classification machine learning system based on the Multidimensional numbered information spaces", In Intelligent Decision Making Systems, WSPS on Computer Engineering and Information Science, No:2, pp. 279-286, 2009.

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
issn: 1942-2679

International Journal On Advances in Internet Technology
ICDS, ICIW, CTRQ, UBICOMM, ICSNC, AFIN, INTERNET, AP2PS, EMERGING
issn: 1942-2652

International Journal On Advances in Life Sciences
eTELEMED, eKNOW, eL&mL, BIODIV, BIOENVIRONMENT, BIOGREEN, BIOSYSCOM, BIOINFO, BIOTECHNO
issn: 1942-2660

International Journal On Advances in Networks and Services
ICN, ICNS, ICIW, ICWMC, SENSORCOMM, MESH, CENTRIC, MMEDIA, SERVICE COMPUTATION
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
issn: 1942-2628

International Journal On Advances in Systems and Measurements
ICQNM, ICONS, ICIMP, SENSORCOMM, CENICS, VALID, SIMUL
issn: 1942-261x

International Journal On Advances in Telecommunications
AICT, ICDT, ICWMC, ICSNC, CTRQ, SPACOMM, MMEDIA
issn: 1942-2601