International Journal on Advances in Internet Technology

2011 vol. 4 nr. 3&4

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

Sponsored by IARIA
www.iaria.org

International Journal on Advances in Internet Technology
Volume 4, Number 3 & 4, 2011

Communication Theory, QoS and Reliability

- Adrian Andronache, University of Luxembourg, Luxembourg
- Shingo Ata, Osaka City University, Japan
- Eugen Borcoci, University "Politehnica" of Bucharest (UPB), Romania
- Michel Diaz, LAAS, France
- Michael Menth, University of Wuerzburg, Germany
- Michal Pioro, University of Warsaw, Poland
- Joel Rodriques, University of Beira Interior, Portugal
- Zary Segall, University of Maryland, USA

Ubiquitous Systems and Technologies

- Sergey Balandin, Nokia, Finland
- Matthias Bohmer, Munster University of Applied Sciences, Germany
- David Esteban Ines, Nara Institute of Science and Technology, Japan
- Dominic Greenwood, Whitestein Technologies AG, Switzerland
- Arthur Herzog, Technische Universitat Darmstadt, Germany
- Malohat Ibrohimova, Delft University of Technology, The Netherlands
- Reinhard Klemm, Avaya Labs Research-Basking Ridge, USA
- Joseph A. Meloche, University of Wollongong, Australia
- Ali Miri, University of Ottawa, Canada
- Vladimir Stantchev, Berlin Institute of Technology, Germany
- Said Tazi, LAAS-CNRS, Universite Toulouse 1, France

Systems and Network Communications

- Eugen Borcoci, University 'Politechncia' Bucharest, Romania
- Anne-Marie Bosneag, Ericsson Ireland Research Centre, Ireland
- Jan de Meer, smartspace®lab.eu GmbH, Germany
- Michel Diaz, LAAS, France
- Tarek El-Bawab, Jackson State University, USA
- Mario Freire, University of Beria Interior, Portugal / IEEE Portugal Chapter
- Sorin Georgescu, Ericsson Research - Montreal, Canada
- Huaqun Guo, Institute for Infocomm Research, A*STAR, Singapore
- Jong-Hyouk Lee, INRIA, France
- Wolfgang Leister, Norsk Regnesentral (Norwegian Computing Center), Norway
- Zoubir Mammeri, IRIT - Paul Sabatier University - Toulouse, France
- Sjouke Mauw, University of Luxembourg, Luxembourg
- Reijo Savola, VTT, Finland

Future Internet

- Thomas Michal Bohnert, SAP Research, Switzerland
- Fernando Boronat, Integrated Management Coastal Research Institute, Spain
- Chin-Chen Chang, Feng Chia University - Chiayi, Taiwan
- Bill Grosky, University of Michigan-Dearborn, USA

- Sethuraman (Panch) Panchanathan, Arizona State University - Tempe, USA
- Wei Qu, Siemens Medical Solutions - Hoffman Estates, USA
- Thomas C. Schmidt, University of Applied Sciences – Hamburg, Germany

Challenges in Internet

- Olivier Audouin, Alcatel-Lucent Bell Labs - Nozay, France
- Eugen Borcoci, University "Politehnica" Bucharest, Romania
- Evangelos Kranakis, Carleton University, Canada
- Shawn McKee, University of Michigan, USA
- Yong Man Ro, Information and Communication University - Daejon, South Korea
- Francis Rousseaux, IRCAM, France
- Zhichen Xu, Yahoo! Inc., USA

Advanced P2P Systems

- Nikos Antonopoulos, University of Surrey, UK
- Filip De Turck, Ghent University – IBBT, Belgium
- Anders Fongen, Norwegian Defence Research Establishment, Norway
- Stephen Jarvis, University of Warwick, UK
- Yevgeni Koucheryavy, Tampere University of Technology, Finland
- Maozhen Li, Brunel University, UK
- Jorge Sa Silva, University of Coimbra, Portugal
- Lisandro Zambenedetti Granville, Federal University of Rio Grande do Sul, Brazil

International Journal on Advances in Internet Technology
Volume 4, Numbers 3 & 4, 2011

CONTENTS

Peer-to-Peer Virtualized Services

David Bailey and Kevin Vella
University of Malta
Msida, Malta
Email: david@davidbailey.info, kevin.vella@um.edu.mt

***Abstract*—This paper describes the design and operation of a peer-to-peer framework for providing, locating and consuming distributed services that are encapsulated within virtual machines. We believe that the decentralized nature of peer-to-peer networks acting in tandem with techniques such as live virtual machine migration and replication facilitate scalable and on-demand provision of services. Furthermore, the use of virtual machines eases the deployment of a wide range of legacy systems that may subsequently be exposed through the framework. To illustrate the feasibility of running distributed services within virtual machines, several computational benchmarks are executed on a compute cluster running our framework, and their performance characteristics are evaluated. While I/O-intensive benchmarks suffer a penalty due to virtualization-related limitations in the prevailing I/O architecture, the performance of processor-bound benchmarks is virtually unaffected. Thus, the combination of peer-to-peer technology and virtualization merits serious consideration as a scalable and ubiquitous basis for distributed services. A view of some challenges and opportunities that emerge in the design of such frameworks is also offered.**

***Keywords*-Virtualization; distributed systems; peer-to-peer computing; service-oriented computing; cloud computing.**

I. Introduction

This paper describes a framework that enables the dynamic provision, discovery, consumption and management of software services hosted within distributed virtual machines. The framework, Xenos [1][2], uses a decentralised peer-to-peer overlay network for advertising and locating service instances and factories. It also leverages techniques such as live virtual machine migration and replication to enhance operational agility and ease of management, and to lay the foundations for deploying fault-tolerant services. The primary objective is to shift the focus away from managing physical or virtual machines to managing software services.

In recent years, data centre operations have experienced a shift in focus away from managing physical machines to managing virtual machines. Renewed exploration of this well-trodden path is arguably driven by virtualization's mantra of enhanced operational agility and ease of management, increased resource utilisation, improved fault isolation and reliability, and simplified integration of multiple legacy systems. Virtualization is also permeating the cluster and grid computing communities, and we believe it will feature at the heart of future desktop computers and possibly even advance a rethink of general purpose operating system architecture.

The performance hit commonly associated with virtualization has been partly addressed on commodity computers by recent modifications to the x86 architecture [3], with both AMD and Intel announcing specifications for integrating IOMMUs (Input/Output Memory Management Units) with upcoming architectures. While this largely resolves the issue of computational slow-down and simplifies hypervisor design, virtualized I/O performance will remain mostly below par until I/O devices are capable of holding direct and concurrent conversations with several virtual machines on the same host. This generally requires I/O devices to be aware of each individual virtual machine's memory regions and demultiplex transfers accordingly. We assume that this capability or a similar enabler will be commonplace in coming years, and that the commoditization of larger multi-core processors will reduce the frequency of expensive world-switches as different virtual machines are mapped to cores over space rather than time.

The paper is organized as follows. Section II provides an overview of related work, and Section III briefly describes the key topics that underpin this research. Section IV details the proposed framework and the implemented prototype, while Section V presents an evaluation of the framework. Finally, Section VI exposes a number of issues for future investigation, and an overview of this work's contribution can be found in Section VII.

II. Related Work

The ideas presented here are influenced by the Xenoservers project [4], initiated by the creators of the Xen hypervisor. Xenoservers was designed to "build a public infrastructure for wide-area distributed computing" by hosting services within Xen virtual machines. The authors argue that current solutions for providing access to online resources, such as data storage space or an application-specific server, is not flexible enough and is often based on a timeline of months or years, which might not always accommodate certain users. The Xenoserver infrastructure would allow for users to purchase temporary resources for immediate use and for a small time period, for instance a group of players wanting to host a game server for a few hours or even minutes. A global infrastructure can also

aid in exploiting locality by running code on a network location that is close to the entities that it uses, such as data and services, to improve performance. In order to allow untrusted sources to submit their own applications, execution environments need to be isolated; the authors propose to use the Xen hypervisor [5] to provide these isolated and secure environments in the form of virtual machines, which also allows for a high degree of flexibility as users have a wide array of operating system and application environments to choose from. Xenosearch [6] locates Xenoservers using the Pastry peer-to-peer overlay network. A Xenoservers implementation is not generally available, hence our decision to build and conduct experiments with Xenos.

WOW [7] is a "distributed system that combines virtual machine, overlay networking and peer-to-peer techniques to create scalable wide-area networks of virtual workstations for high-throughput computing". Applications and services in the system are provided in virtual machines, which must also contain a virtual network component that is used to register the machine on a peer-to-peer overlay network when the machine boots up. This peer-to-peer overlay network is used to create virtual links between the virtual machines, which are self-organizing and maintain IP connectivity between machines even if a virtual machine migrates across the network. The authors do not provide a mechanism which allows for searching of other services registered on the peer-to-peer network; this is where our approach differs in that we intend to use a peer-to-peer overlay network to advertise the services running within the virtual machines rather than to set up a virtual network to enable communication between virtual machines.

SP2A [8] is a service-oriented peer-to-peer architecture which enables peer-to-peer resource sharing in grid environments, but is not concerned with the uses of virtualization in distributed computing architectures, which is one of our main interests. Several publications have focused on the use of peer-to-peer overlay networks to implement distributed resource indexing and discovery schemes in grids [9][10][11]. Wadge [12] investigates the use of peer groups to provide services in a grid, as well as transferring service code from one node to another for increased fault-tolerance and availability.

The dynamic provisioning of services is a relatively young area of research, and commercial products such as Amazon Elastic Compute Cloud (EC2) have only appeared in the past few years. Virtualization and hardware advancements have had a major impact on the structure of these datacenters, which typically rely on tried-and-tested setups and favour the traditional client-server approach to locating and consuming services. We believe that exploiting the advantages of peer-to-peer networks is the next step in achieving a truly distributed, scalable and resilient services platform.

III. Background

A. Virtualization

In computing, virtualization can be broadly defined as the software abstraction of a set of resources, which enables the sharing of these resources in parallel by higher-level systems. While the actual definition and mechanisms used varies depending on the type of virtualization in question, the concept always remains the same; that of efficiently, securely and transparently multiplexing a set of resources in a manner which allows for higher-level systems to use these resources and allowing them to assume that they are using the real resources instead of the abstraction provided by the mechanism. We are mostly interested in hardware-level virtualization, where the virtualization layer sits on top of the hardware and virtualizes the hardware devices, allowing multiple operating systems to execute within the virtual machine environments presented by the layer. Hardware resources such as the processor, memory and I/O are managed by the virtualization layer and shared by the executing operating systems, although the latter might have no knowledge of the underlying virtualization layer. This layer is often called a virtual machine monitor or a hypervisor.

One of the techniques used in achieving full hardware virtulization is paravirtualization, where the hypervisor provides virtual machines that are not exact copies of the underlying hardware architecture. This implies that the operating system executing in a virtual machine provided by the hypervisor is aware that it is running inside a virtualized environment, and has to issue calls to the hypervisor for certain operations. Legacy operating systems therefore need to be ported in order to run on the hypervisor. Perhaps the most successful paravirtualized hypervisor that has gained widespread use in the industry is the Xen hypervisor, on which a number of commercial solutions are based, such as Citrix XenServer, Oracle VM and Sun xVM, as well as heavily influencing the design of Microsoft's Hyper-V hypervisor. Xen supports existing application binary interfaces, meaning it can support applications written for the x86 architecture without the need for modification; and it exposes certain physical resources directly to guest operating systems, allowing for better performance. The Xen hypervisor aims at supporting legacy operating systems (with minimal porting effort) and existing applications, while leveraging the benefits of a paravirtualized approach, such as high performance and stronger isolation.

B. High-Performance Computing and Grids

Mergen *et al.* [13] argue that hypervisors offer a new opportunity for high performance computing (HPC) environments to circumvent the limitations imposed by legacy operating systems. Modern hypervisors not only support legacy systems, but they can also simultaneously execute

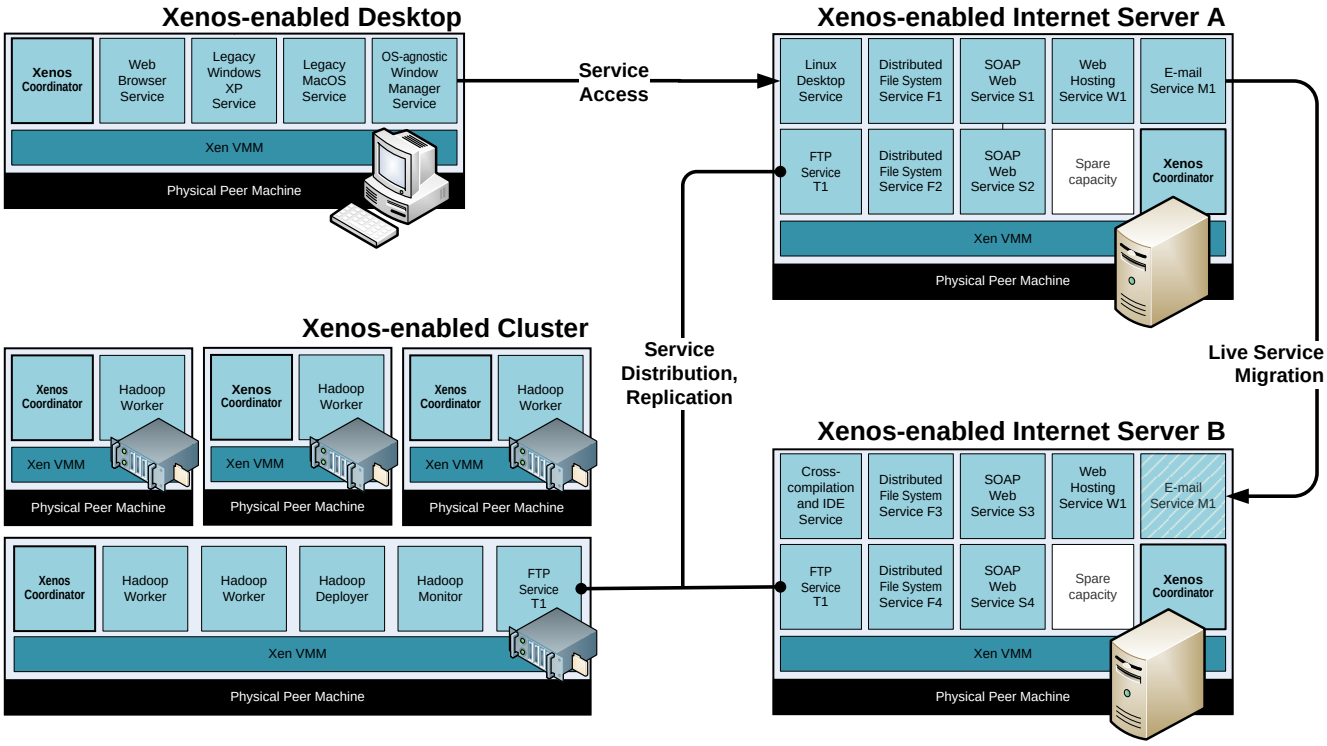

Figure 1. A selection of computing platforms running the Xenos framework and hosting several interacting services.

specialized execution environments in which HPC applications are run; this allows for legacy HPC software and other non-critical operating system services, such as file systems, to be run within the legacy operating system, while the HPC application can exploit the hypervisor directly and implement any optimization opportunities, such as the use of super-pages. These specialized execution environments are also known as *library OS*, since they typically contain only the required software stacks for the application(s) that will be executing within them. Thibault *et al.* [14] implement lightweight Xen domains based on the Mini-OS paravirtualized kernel, which is included with the Xen hypervisor as a demonstration of how to implement a basic guest kernel for Xen – it is able to use the Xen network, block and console mechanisms, supports non-preemptive threads and only one virtual memory address space. A similar approach is taken by Anderson *et al.* [15], although the focus is on security and reliability; the authors argue that partitioning critical services and applications into domains with tight restrictions improves trustworthiness. Falzon [16] implements a lightweight Xen domain to explicitly execute a thread scheduler that supports multiple processors and offers several scheduling policies. This allows for the creation and evaluation of different schedulers that have direct access to the virtualized hardware, and can for instance control the number of kernel threads mapped on a particular virtual CPU, or disable timers and interrupts in the domain.

A number of publications have focused on the use of virtualization and virtual machines in grid computing environments in response to a number of significant issues such as security, administration costs and resource control. Figueiredo *et al.* [17] present a number of tests that show overheads to be minimal under the VMware Workstation hypervisor. The authors also present an architecture for dynamically instantiated virtual machines based on a user's request, where a virtual machine is distributed across three logical entities: image servers that hold static virtual machine states, computation servers that can dynamically instantiate and execute images, and data servers which store user application data. Keahey *et al.* [18] propose a similar architecture, providing execution environments for grid users called Dynamic Virtual Environments (DVEs), as well as implementing DVEs using different technologies such as UNIX accounts, and operating-system and hardware-level virtual machines such as VServer sandboxes and VMware respectively. The different implementations were analyzed to determine their viability for use in a grid infrastructure, and while they provided sufficient in terms of applications without heavy I/O loads, the authors believe that all had shortcomings in Quality of Service (QoS) functionality, and some technologies such as VMware did not expose enough of their functionality for direct use in the grid. Santhanam *et al.* [19] experiment with different sandbox configurations, deployed using the Xen hypevisor, and concluded that jobs with heavy I/O loads take a performance hit when running inside a virtual machine sandbox, although they advocate the use of virtual machines in grid environments where applications often tolerate delays on the order of minutes,

and if the user wishes to benefit from the advantages obtained by using virtual environments.

IV. The Xenos Framework

Xenos is built on top of Xen, a virtualization platform that has gained traction as a stable and mature virtualization solution, but any hypervisor with the appropriate hooks and programming interfaces will suffice in principle, including a hypothetical ROM-based hypervisor. The JXTA framework is currently used to maintain a peer-to-peer overlay network for service advertisement, discovery and, optionally, transport. However, we feel that a more specialized or expressive latter generation peer-to-peer framework would better fit our requirements. The Hadoop map-reduce framework, described in more detail in Section V, is used as a benchmarking tool to evaluate the framework, but it is not an intrinsic part of the Xenos framework itself.

A. Physiology

Figure 1 illustrates a scenario with different hardware platforms running Xenos and a variety of services, which may be any software application that can be encapsulated within a Xen virtual machine. A commodity cluster, typically used for high-performance computing applications, offers users the ability to dynamically create computation services, such as Hadoop map-reduce nodes, while also using other services such as the Hadoop deployer and Hadoop monitor to easily deploy these services on the network, and monitor them for fault-tolerance and load-balancing. Xenos also runs on desktop machines, with the user utilizing several services such as a file system for personal data storage, and a legacy operating system service offering traditional applications. The user interface that the user interacts with is itself a virtualized service, possibly forming part of a distributed operating system made up of several services running on the Xenos framework. If, for instance, the user is transferring files between the file system and the Hadoop cluster, it would be possible for the instance of the file system containing the required files to be migrated (physically moved) to the cluster, thus improving the performance when transferring data to the cluster or retrieving results. Finally, another platform supporting Xenos is a traditional server in a datacentre, where services such as web, FTP and email servers, and web services are executed as virtual machines, and are used by clients or by other services across the Xenos cloud.

From the perspective of the user, the platform provides two major features: the ability to search for services, obtain information about them and make use of them, and the ability to control these services by creating new service instances, migrate running services, manage existing ones and monitor their use. System administrators are responsible for setting up and managing the infrastructure on which Xenos is hosted, providing services packaged in virtual machines, and configuring these services to appear on the Xenos peer-to-peer network. Optionally, users or administrators can also develop custom services that participate on the same peer-to-peer network as the other hosts and services and act as an additional feature to the platform. These services join the peer-to-peer network provided by Xenos and complement the existing features of our framework, or act as support services for a user's existing services. These can include fault-tolerance and load-balancing monitors, which trigger migration and replication of services as required, introspection services that provide useful information about domains, and management services that use JXTA groups to effectively manage a user's services.

B. Architecture

Figure 2 illustrates the architecture of a single physical machine in the framework. Each Xenos-enabled physical machine runs the Xen hypervisor using a paravirtualized Linux kernel in Domain 0, which is a privileged domain capable of controlling the guest domains (virtual machines) that will host services on the same physical machine. The Xenos Coordinator is a Java application that executes in Domain 0 whose primary function is to incorporate the physical machine into Xenos' peer-to-peer overlay network and advertise services running on that physical machine, through the JXTA library. Services running within guest domains do not normally join the overlay network directly, but are registered with the coordinator in Domain 0, which acts as a 'notice board' for all local services. Administrators configure these services through text-based configuration files that are picked up by the Coordinator on startup. It also provides utilities for controlling these domains by making use of the Xen Management API, and other utilities used by other components of the system itself or directly by administrators, such as file management routines and ID generators for quick configuration of hosts and services.

The Xenos API is an XML-RPC programming interface available for users and services to interact with, and is the primary channel through which services are discovered and managed. Users can search for services and/or hosts on the peer-to-peer network by passing in search parameters to the API, which then returns results describing the services or hosts. Services may also be controlled and monitored remotely by passing in identifiers for the services to be acted upon. Migration and replication of services can also be triggered through the API, which implements file transfer and copying features that are required for this functionality. The Xenos API also features an implementation of XML-RPC over JXTA protocols, which enables hosts on the peer-to-peer network to issue XML-RPC calls to each other without requiring a TCP/IP socket connection, but rather use the built-in socket functionality in JXTA. Service delivery itself may be accomplished without the involvement of Xenos, and is not restricted to any particular network protocol or

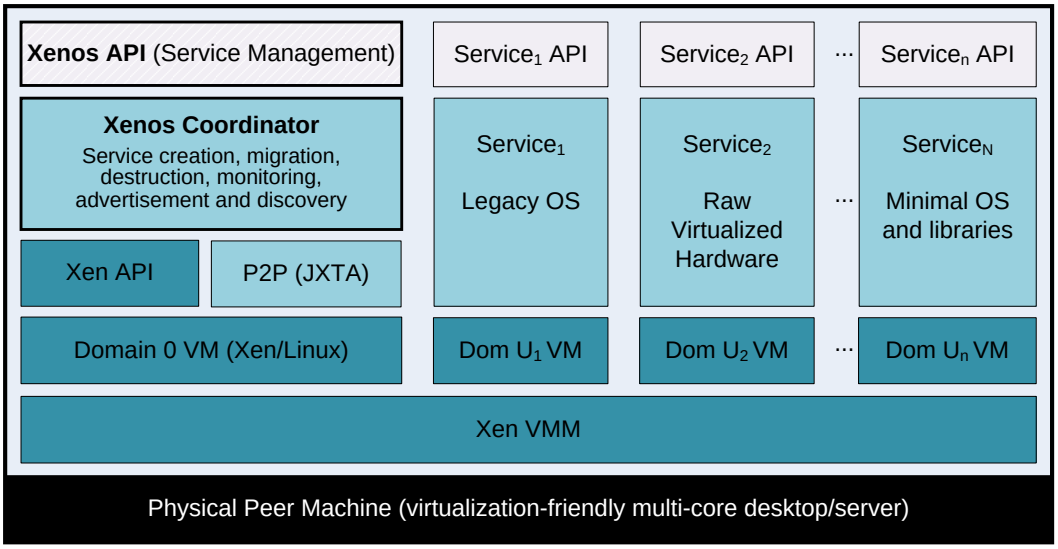

Figure 2. A Xenos-enabled physical machine.

address space. However, the direct use of network protocols beneath layer three (for example, Ethernet) would oblige communicating services to share a physical network or a physical machine.

In order to accommodate multiple instances of the same service and service migration, each service type has a template associated with it that enables the automatic configuration of new service instances and their Xen domains, as illustrated in Figure 3. When replicating a service or creating a new service instance, a new copy of the relevant template is used. Service templates will automatically replicate on other Xenos hosts as required so that service instances can be spawned anywhere on the Xenos cloud. Migration of service instances makes use of Xen's virtual machine migration mechanism with a slight modification to transfer virtual machine disk images along with the virtual machine configuration. Our current implementation inherits a Xen restriction limiting live virtual machine migration to the local area network, though this may be overcome as discussed in Section VI.

C. Design Benefits

The architectural design discussed above leads to several benefits over similar platforms. The use of a peer-to-peer overlay network enables a decentralized approach to registering and discovering services, in contrast with the centralized approach often used within existing web services platforms, such as Universal Description Discovery and Integration (UDDI). By having the Xenos API available on every host on the platform instead of a main server (and possibly some backup servers), users of the platform can make the applications that interact with the API more fault-tolerant by initially searching for a number of Xenos hosts and storing them locally as a backup list. If the host being used by the applications becomes unavailable, another host can be picked from the backup list and communication attempted with it. This can also lead to implementing a load-balancing approach to issuing API calls, so that the workload is spread over multiple hosts instead of a single one.

JXTA provides a grouping facility, where services or peers can be organized into groups that are created by the user. Our framework allows administrators to specify which groups a service should join initially; this can be used, for instance, to group together services that offer the same functionality, or to group together services that belong to the same user. Services can form part of multiple groups, and are always part of the net peer group, which is the global group maintained by JXTA. Additionally, users who build their own applications that form part of the peer-to-peer network can create new groups on the fly and assign services to them. For instance, a custom built load-balancer could create a group and automatically monitor all the services that join it; this scoping can help reduce the amount of messaging going on in the network, since the load-balancer would only need to broadcast into its created group instead of the net peer group.

Existing commercial cloud solutions, such as Amazon EC2, often provide computing instances that are fixed and feature large amounts of memory, processor resources and storage space, which are not always necessary when dealing with lightweight or specialized services. We have already discussed the benefits of running certain services inside specialized execution environments in Section III; the majority of publications that we review have used Xen as the hypervisor, as it is based on paravirtualization, which performance significantly better that other virtualization techniques and allows for modifications or development of custom operating systems for running specific services. Our platform allows administrators to create services that are based within lightweight Xen domains, and assign to them

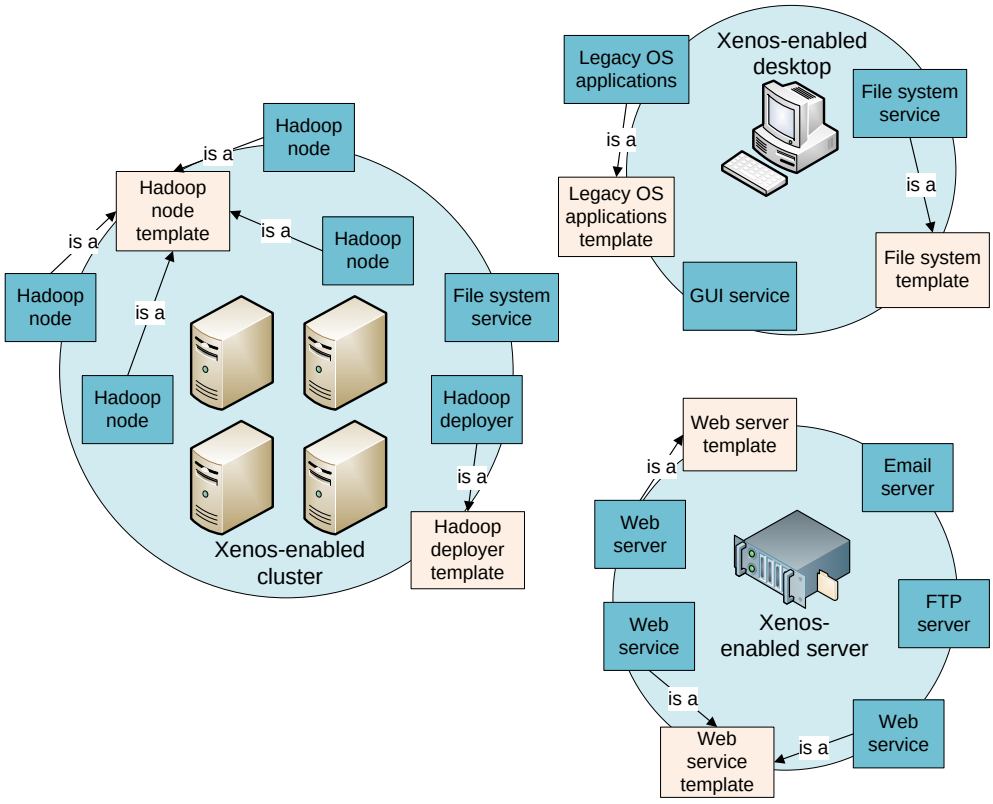

Figure 3. The relationship between Xenos templates and services.

resources as needed. This is beneficial for certain services that, for instance, would not require large amounts of storage space but benefit from multiple virtual CPUs and large amounts of memory due to their computation being mostly processor bound. Conversely, certain services might deal with data storage and processing, and thus require large amount of storage space but can do with a single virtual CPU and a small amount of memory. Although our current prototype does not have the ability for users to upload their own virtual machine images and configurations, this is trivial to add to our existing infrastructure, and would be a powerful feature that gives users even more flexibility.

V. Hadoop Case Study and Performance Analysis

A series of preliminary tests were conducted in order to assess the viability of our approach. The test cases all involve deploying multiple instances of a Hadoop map-reduce wrapper service using a separate distributed coordination service. We aim to explore three principal avenues, namely (1) the automatic and dynamic deployment of the Hadoop service to Xenos hosts and the migration of the master Hadoop node from a failing physical machine; (2) the performance of file I/O within virtual machines, which is crucial for services with large-volume data processing requirements (this is particularly relevant since Xenos requires virtual machine images to exist in files rather than as physical disk partitions); and (3) the performance of a series of virtualized Hadoop map-reduce processing jobs.

A similar evaluation of running the Hadoop map-reduce framework within a virtualized cluster is carried out by Ibrahim *et al.* [20]. They argue that a virtual machine-based Hadoop cluster can offer compensating benefits that overshadow the potential performance hit, such as improved resource utilization, reliability, ease of management and deployment, and the ability to customize the guest operating systems that host Hadoop to increase performance without disrupting the cluster's configuration.

A. Map-Reduce and Hadoop

In our experiments we used the Hadoop Distributed File System (HDFS) and MapReduce components of the Apache Hadoop framework. The map-reduce programming model, introduced by Dean *et al.* [21], is aimed at processing large amounts of data in a distributed fashion on clusters. HDFS is a distributed file system suitable for storing large data sets for applications with heavy data processing, such as typical map-reduce jobs. The Hadoop map-reduce implementation involves a master node that runs a single *JobTracker*, which accepts jobs submitted by the user, schedules the job across worker nodes by assigning map or reduce tasks to them, monitors these tasks and re-executes failed ones. Each worker (or slave) node runs a single *TaskTracker*, which is responsible for executing the tasks assigned to it by the job

tracker on the master node.

B. Deploying Hadoop Services

When setting up a computing cluster with the aim of running the Hadoop map-reduce framework, each node needs to be configured with specific settings, such as the hostname, SSH certificates and the hosts that it has access to, and the HDFS and map-reduce settings that are common throughout the cluster. When setting up a non-virtualized environment, administrators typically configure a single node, and then manually clone the hard disk to all the other nodes, resulting in an identical installation across the cluster, which would then require node-specific settings on each machine. Setting up Hadoop on a more general cluster can be done by setting up an installation on single node, and then distributing the installation to the other cluster nodes, typically via shell scripts and *rsync*. Another alternative is to use existing deployment frameworks that manage the configuration, deployment and coordination of services such as Hadoop, and do much of the work.

One of the issues that we identify with deploying any sort of service on an existing cluster environment is the potential to disrupt the configuration or execution of other services when configuring the new one. If one were to use a virtualized cluster, services could be supplied within pre-packaged virtual machines that would not interfere with other services running within their own virtual machines, since the hypervisor guarantees isolation between them. The configuration of the physical node would therefore never need to be modified when adding new services; of course, the initial setup of the virtualized cluster still needs to be done manually by administrators cloning an initial setup to all the cluster nodes, but this is inevitable. One can always set up a physical cluster with a single service in mind, which would not require frequent re-configuration, but this often leads to wasted resources that virtual machines could fully exploit if the cluster were to be virtualized.

We can identify several other benefits in using a virtualized cluster for Hadoop services. Since services would be packaged within their own virtual machine, we can easily modify the installation and configuration of the operating system running within the virtual machine to accommodate Hadoop map-reduce and the HDFS and tweak its performance, without having to modify the configuration of the operating system running on the physical node, which is Domain 0 in the case of Xen. Since the master nodes are potential single-points-of-failure both in the HDFS and Hadoop map-reduce, the master node can also be packaged inside a virtual machine, which can be checkpointed regularly, thus saving the whole virtual machine state, or migrated to another physical host if the current host is malfunctioning or needs to be shut down.

If we opt for a virtualized cluster on which to deploy Hadoop, we are still faced with the task of deploying the virtual machine containing the Hadoop map-reduce workers on the nodes of the cluster. Deployments methods similar to the ones when running a non-virtualized cluster can be used, such as setting up shell scripts to transfer virtual machine images and then issuing remote commands on the nodes to start the virtual machines. However, a more appropriate solution would be to use an existing platform, which can deploy virtual machine images to the cluster's nodes, and allows users to administer these images remotely, typically from the same node that acts as the map-reduce master. The Xenos framework that we have implemented is a perfect candidate on which to build a Hadoop deployer that allows users and administrators to provide their own Hadoop installation as a service within a domain, register this service with the Xenos coordinator, and then use the framework's replication, migration and service control features to deploy these services on the virtualized cluster. This requires a small application to be developed that oversees this task, since by itself the framework has no capabilities of deploying services automatically, but simply provides the mechanisms that allow this. We have therefore developed a Java application that uses the JXTA framework to connect to the Xenos network, and use the Xenos API to deploy the Hadoop service supplied by the administrator. Although we have tailored this application for the Hadoop map-reduce and HDFS service, we feel that it can be generalized rather easily to support any service that is registered within our system; in fact, only a small portion of the application is Hadoop-specific, as the rest simply deals with services that are defined by the user in a separate configuration file. This would provide users with a service deployer with which they can deploy their services on the Xenos framework.

C. Evaluation Platform and Results

Our evaluation platform consists of a thirteen-host commodity cluster, connected over a 1 Gigabit Ethernet connection through a D-Link DGS-1224T switch. Each physical host in the cluster runs an Intel Core 2 Duo E7200 CPU, with 3MB of L2 cache and clocked at 2.53GHz, 2GB of DDR2 RAM, and a 500GB SATA2 hard disk. All the hosts were configured with Xen and the Xenos framework. One of the hosts, which we refer to as the master host, was configured with a template of the Hadoop slave service as well as an instance of the Hadoop master node, from where we issue commands to deploy services and execute tests; however, it was also configured not to accept service instances of the Hadoop slave service, meaning that we have 12 hosts on which to instantiate Hadoop slave services. The master host was also set as a JXTA rendezvous server, and all the Xenos hosts configured to use it. All physical hosts were assigned fixed IP addresses, and a DHCP server was configured on the master host to allocate addresses to spawned domains.

In all of our tests except where stated, the domain that we use as the Hadoop slave template which is replicated to

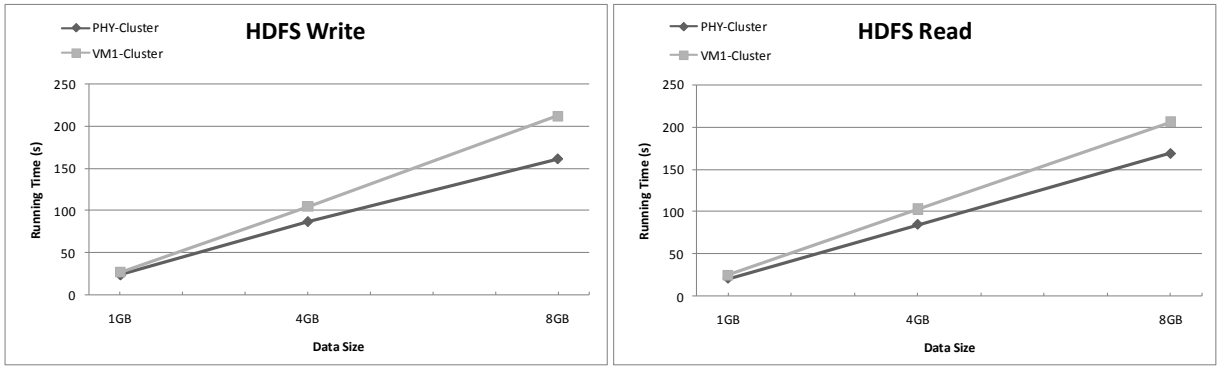

Figure 4. PHY-Cluster *vs* VM1-Cluster with varying data sizes.

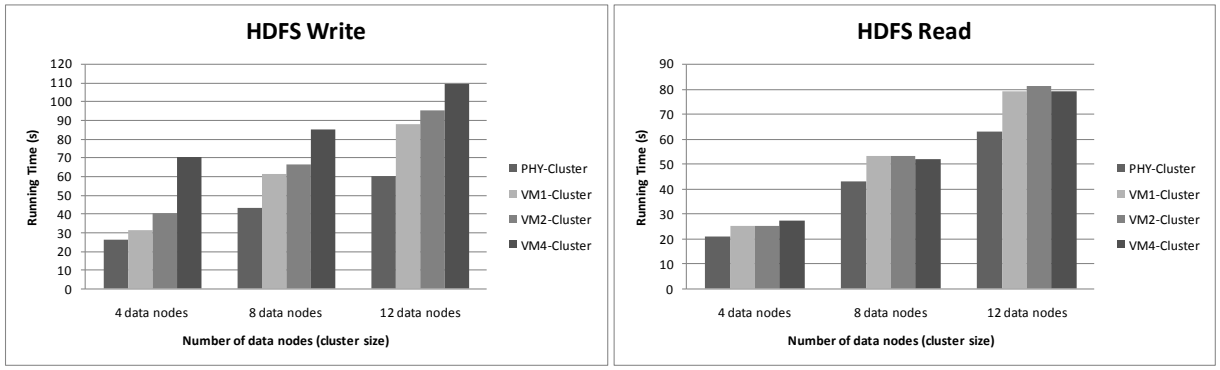

Figure 5. PHY-Cluster *vs* VM clusters with varying data nodes (cluster size) and virtual machines per physical machine.

all the hosts is configured with a 10GB disk image, 1GB swap image, and the vmlinuz-2.6.24-27-xen kernel. Default settings for the domain are 1 virtual CPU (VCPU), 384MB of RAM, and set to use DHCP to obtain addresses. Domain 0 is set to use 512MB of memory, leaving the rest to be allocated to domains, and has no restrictions on the physical CPUs it can use. For the Hadoop tests run on the native, non-virtualized Linux distribution, the same cluster and same Linux installation that is used as Domain 0 is used, but without booting into Xen so that the operating system runs natively. In all our tests, the HDFS replication factor is set to 2, and we do not use rack awareness since our network only has one switch. Each task tracker is set to execute a maximum of 2 map-tasks and 2 reduce-tasks at any given time. No optimizations to Hadoop or any other software component were made to suit this particular cluster.

1) Replication and Migration of Hadoop Service Templates and Services: The unoptimized replication process took around 45 minutes to deploy a template and a single slave service instance to each of the twelve cluster hosts, which included a network transfer of 132GB as well as another 132GB in local data copying; this translates to a network throughput of around 40MB/s and a disk throughput of around 25MBs/s. Since the process mostly involves transferring domain files over the network and copying them locally, its performance depends on the hardware platform that the services are being deployed on, as well as the size of the domains that contain the service. Other operations performed during replication, such as issuing Xenos API calls and updating local configuration files are typically sub-second operations that do not affect the overall performance. Further optimizations such as using LAN broadcasts on the cluster to transfer the service to the hosts can be implemented to minimize the time required for deployment.

In order to test the migration of the Hadoop master instance, a Hadoop Wordcount benchmark was initiated on the master, using a single slave instance deployed on each cluster. About half-way through the job, we issue a migration request from a small application in the master host to the Xenos API in the same host, instructing Xenos to migrate the master instance to another host on the cluster, which is automatically located through a peer-to-peer search. This causes the master to be paused while its files are moved and its state migrated, inevitably causing some map tasks on the slave services to fail, since they are not able to report back to the master. Once the master has migrated, it is un-paused and resumes executing the job, and the failed map tasks are re-executed by Hadoop itself. The job finishes successfully, although as expected it takes more time than if it were not migrated, due to the re-execution of failed map tasks. The migration itself takes less than 5 minutes, which is practically the time needed to transfer the domain files and the memory from the original host to the target host, and to resolve the new address of the host. Once again,

the operations issued by Xenos are lightweight and have a negligible affect on the overall duration.

2) HDFS Performance: To evaluate the performance of the HDFS on which Hadoop map-reduce relies, we designed a series of tests to measure its performance when reading and writing data in both a physical and virtualized cluster setup. Our main objective is to determine the performance penalty suffered by I/O operations in virtualized environments, using different data set sizes, cluster configurations and number of HDFS datanodes. All read and write operations were issued using the *get* and *put* operations provided by Hadoop, which allows reading from the HDFS to the local filesystem and writing from the local filesystem to the HDFS respectively. The data written into the HDFS in all tests is a large text file automatically generated by scripts beforehand. In all results, PHY-Cluster refers to a Hadoop cluster on native Linux, while VM1-Cluster, VM2-Cluster and VM4-Cluster refer to Xenos-enabled virtualized clusters with one, two and four virtualized Hadoop slave services deployed per physical host respectively.

We first evaluate the performance of the HDFS when reading and writing different data sizes (1GB, 4GB and 8GB) under a physical and a virtualized environment with only one service instance per host (VM1-Cluster). 12 cluster hosts are used in all these tests, resulting in 12 datanodes being made available to the Hadoop master. As shown in Figure 4, PHY-Cluster performs better than VM1-Cluster in both reading and writing, which is expected due to the overheads typical in virtual machines. While the performance gap is marginal for the 1GB data set, which translates to around 85MB per datanode, the gap increases with bigger data sets that involve more data per node.

Another evaluation carried out for HDFS is to identify whether the number of virtualized service instances on each physical host affects read and write performance. For each test, we read and write 256MB for each datanode, as in the previous test, meaning 1GB, 2GB and 3GB for 4, 8 and 12 datanodes respectively. As shown in Figure 5, PHY-Cluster once again outperforms all the virtualized setups as expected. However, we note an interesting difference between reading and writing on virtualized datanodes; when writing, the performance gap grows significantly larger as the number of datanodes increases, but remains stable when reading. Ibrahim *et al.* [20] also make this observation in one of their tests, indicating that the write performance gap increased markedly but it increased only slightly when reading.

3) Hadoop Benchmarks: One of the possible benefits of running Hadoop jobs within virtual machines is increasing the amount of computation nodes thus using the physical processing resources available more efficiently. To evaluate this, we execute several benchmark jobs that are provided as examples by Hadoop. In all of the evaluations presented below, we execute Hadoop jobs on the physical (native) cluster and three other virtualized cluster setups, as shown in Table I. We use 12 physical hosts throughout all tests, but since the number of service instances (VMs) per host changes, we have a different amount of Hadoop nodes available in certain setups. We again refer to these setups as PHY-Cluster, VM1-Cluster, VM2-Cluster and VM4-Cluster. Note that since the domain of each service instance is set to use 1 VCPU, when deploying four domains on each physical host, the total number of VCPUs on the host is larger than the number of physical CPU cores available, which can have a detrimental effect on the performance of the domains on the host, due to CPU contention. The best VCPU to CPU core ratio is in the VM2-Cluster case, where each VCPU is mapped to a CPU core, as shown in Table I.

The Wordcount benchmark counts the occurrence of each word in a given text file, and outputs each word and its associated count to a file on the HDFS. Each mapper takes a line as input, tokenizes it and outputs a list of words with the initial count of each, which is 1. The reducer sums the counts of each word and outputs the word and its associated count. We execute the benchmark varying the input data size, using 1GB and 8GB data files. As shown in Figure 6, the performance of the VM2-Cluster and VM4-Cluster is better than the VM1-Cluster, indicating that the extra computation nodes being made available are providing a performance benefit. However, the performance of PHY-Cluster is still better than all the virtualized clusters. In the Wordcount benchmark, Ibrahim *et al.* [20] achieved better performance on their virtualized clusters with 2 and 4 VMs per physical host than their physical (native) cluster; however each host in their evaluations was equipped with 8 cores, and their VCPU to CPU core ratio is always less than 1.

The Sort benchmark sorts an input file containing <key,value> pairs and outputs the sorted data to the file system. The input data is usually generated using the RandomWriter sample application provided with Hadoop, which can be used to write large data sets to the HDFS, consisting of sequence files. The mappers reads each record and outputs a <key, record> pair, sorting them in the process, and the reducer simply outputs all the pairs unchanged. We execute the benchmark varying the input data size, using 1GB and 8GB data files. As shown in Figure 7, increasing the number of computation nodes does not result in a performance benefit for the VM2-Cluster and VM4-Cluster; their performance actually degrades significantly. During the tests we observed that while the mappers started executing at a quick rate, once the reducers started executing, the whole job execution slowed down considerably; this was also observed by Ibrahim *et al.* [20] in their evaluation of the Sort benchmark. The authors argue that this can be attributed to the large amount of data transferred from the mappers to the reducers when they start, causing the virtualized Hadoop nodes on the same physical host to compete for I/O resources. The performance of the HDFS is also a factor,

	PHY-Cluster	VM1-Cluster	VM2-Cluster	VM4-Cluster
Services (VMs)	-	1 VM/host	2 VM/host	4 VM/host
Hadoop nodes	12	12	24	48
VCPU : CPU core ratio	-	1:2	1:1	2:1

Table I
THE PHYSICAL AND VIRTUALIZED CLUSTERS SETUPS ON WHICH HADOOP JOBS ARE EXECUTED.

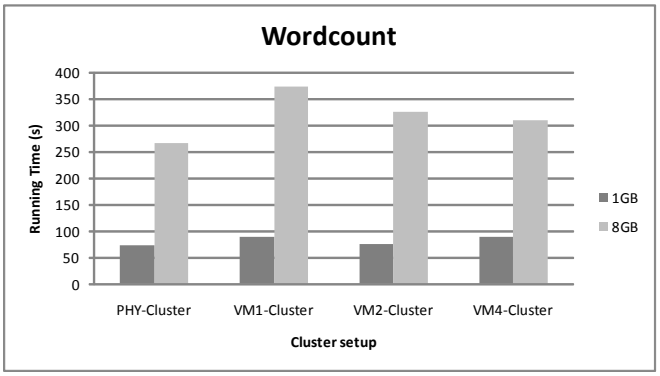

Figure 6. Wordcount execution on PHY-Cluster and VM clusters with varying data input size and virtual machines per physical machine.

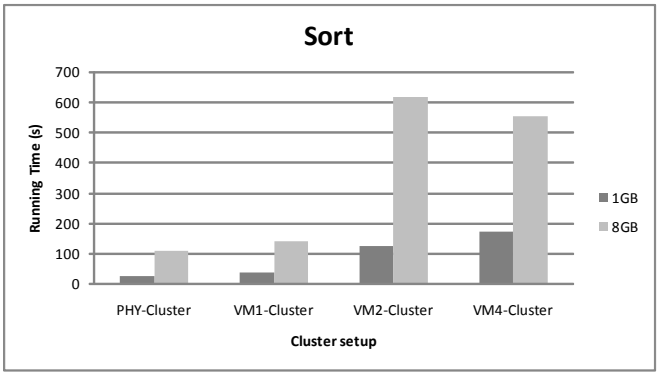

Figure 7. Sort execution on PHY-Cluster and VM clusters with varying data input size and virtual machines per physical machine.

since large amounts of data are being read and written at the same time.

The PiEstimator application included with Hadoop uses a quasi-Monte Carlo method to estimate the value of Pi. The mappers generate points in a unit square, and then count the points inside and outside of the inscribed circle of the square. The reducers accumulate these points from the mappers, and estimate the value of Pi based on the ratio of inside to outside points. The job takes as input the number of mappers to start, and the number of points to generate for each mapper; 120 mappers and 10,000 points were used in all tests. As shown in Figure 8, the performance of VM2-Cluster and VM4-Cluster shows a decisive improvement over VM1-Cluster, since more processing nodes are available, and very little I/O operations are done on the HDFS. In order to verify whether the ratio of VCPUs to physical cores has an effect on performance, we setup a VM1-Cluster with each VM assigned 2 VCPUs instead of 1; this resulted in the same amount of Hadoop nodes available, but each node has an extra VCPU compared to the standard VM1-Cluster. We ran the PiEstimator tests on this cluster and noticed a considerable performance improvement, although not as high as the VM2-Cluster and VM4-Cluster. Interestingly enough, we performed a test on PHY-Cluster where we restricted the Linux kernel to use only a single core, expecting to see a performance degradation when compared with a dual-core setup. However there was no degradation; this could be a deficiency with this particular Hadoop job or Hadoop itself, although it does not explain how VM1-Cluster with two VCPUs achieved better performance than with a single VCPU. It would be an interesting exercise to perform more tests, varying parameters such as the number of map-tasks and reduce-tasks allowed per node, to identify the reasons for this observation.

D. Summary of Results

Using the Hadoop map-reduce framework and the HDFS as a test case, we evaluated the Xenos framework in terms of the functionality and features that it provides, and whether

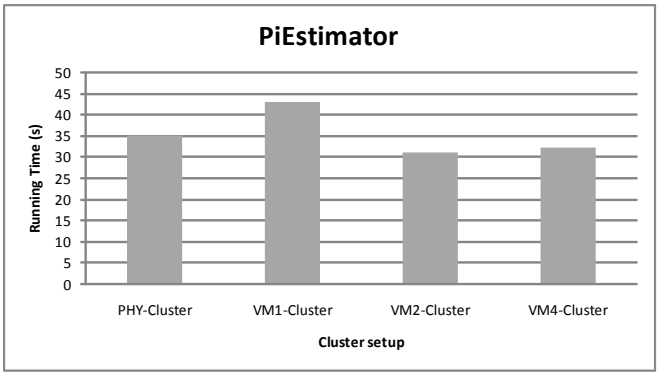

Figure 8. PiEstimator execution on PHY-Cluster and VM clusters with varying virtual machines per physical machine.

the use of virtualization introduces a performance penalty that might not make it feasible to use such a framework for certain data intensive applications such as Hadoop. Using the functionality exposed by the Xenos API, we successfully developed an application that automatically deploys Hadoop map-reduce services over a cluster, allowing the user to specify the number of cluster hosts to use, and the number of services per host. We also successfully migrated the Hadoop master node from its original host to another; the node resumed without issues and eventually the job was completed.

As expected, reading and writing operations on the HDFS when run on a virtualized cluster suffers a performance penalty when compared to a physical (native) cluster setup. For small data transfers and cluster setups, the gap is negligible, but increases steadily when involving large data sets or a large number of cluster nodes. While we acknowledge that these I/O performance penalties can be a barrier when adopting virtualization, a significant amount of ongoing work and research is aiding in reducing this cost and increasing hardware support for virtual I/O.

Increasing the number of computation nodes by adding more virtualized service instances per physical host benefits certain Hadoop jobs that are processor bound, since more efficient use of the physical processing resources is being made. However, jobs that are more I/O bound and that deal with large data sets tend to suffer a performance hit due to the performance degradation of the HDFS. For this reason, an interesting experiment would be to separate the HDFS from the service instances, which become computation nodes that execute the Hadoop tasks but use a non-virtualized HDFS, and evaluate whether any performance benefits are obtained.

To summarize, we have shown that any negative performance effects arising from using Xenos are related to the deficiencies in current virtualized I/O systems, and not due to the overhead imposed by Xenos, which is kept to a minimum. A peer-to-peer virtualized services platform similar to Xenos allows for rapid deployment of services, with the additional benefit that it does not require applications to be modified. We have also shown that leveraging the superior search capabilities of peer-to-peer networks and virtualization features such as migration allows for a more scalable and resilient approach to dynamic service provisioning. Once a platform like Xenos is in place, we can focus on managing services instead of managing physical machines.

VI. Topics For Further Investigation

Xenos can fill the role of a test-bed to facilitate experimentation with a variety of emerging issues in distributed virtualized services, some of which are briefly discussed here.

A. A Library of Essential Services

The core functionality provided by the Xenos framework can be further extended and abstracted away through additional services. Examples include service wrappers for load-balancing and fault-tolerance (virtual machine checkpointing is invisible to the service(s) hosted within), virtual machine pooling and replication, service deployers such as the Hadoop deployer discussed previously, platform emulators, legacy services supporting a range of operating systems, and a Xenos-UDDI adapter that can be used to search for Xenos services via UDDI. Xenos does not impose a single method for actual service delivery, thus web services, Sun RPC, and even services using raw Ethernet may be advertised.

B. Seamless Wide-Area Service Migration

The issue of live virtual machine migration over WANs has been addressed by several authors and a number of prototypes are available. Travostino *et al.* [22] approach the problem of preserving TCP connections by creating dynamic IP tunnels and assigning a fixed IP address to each virtual machine, which communicates with clients via a virtual gateway interface that is set up by Xen. After migration, a virtual machine retains its address, and the IP tunnels are configured accordingly to preserve network routes – this is completely transparent to TCP or any other higher level protocol. Bradford *et al.* [23] combine the IP tunneling approach with

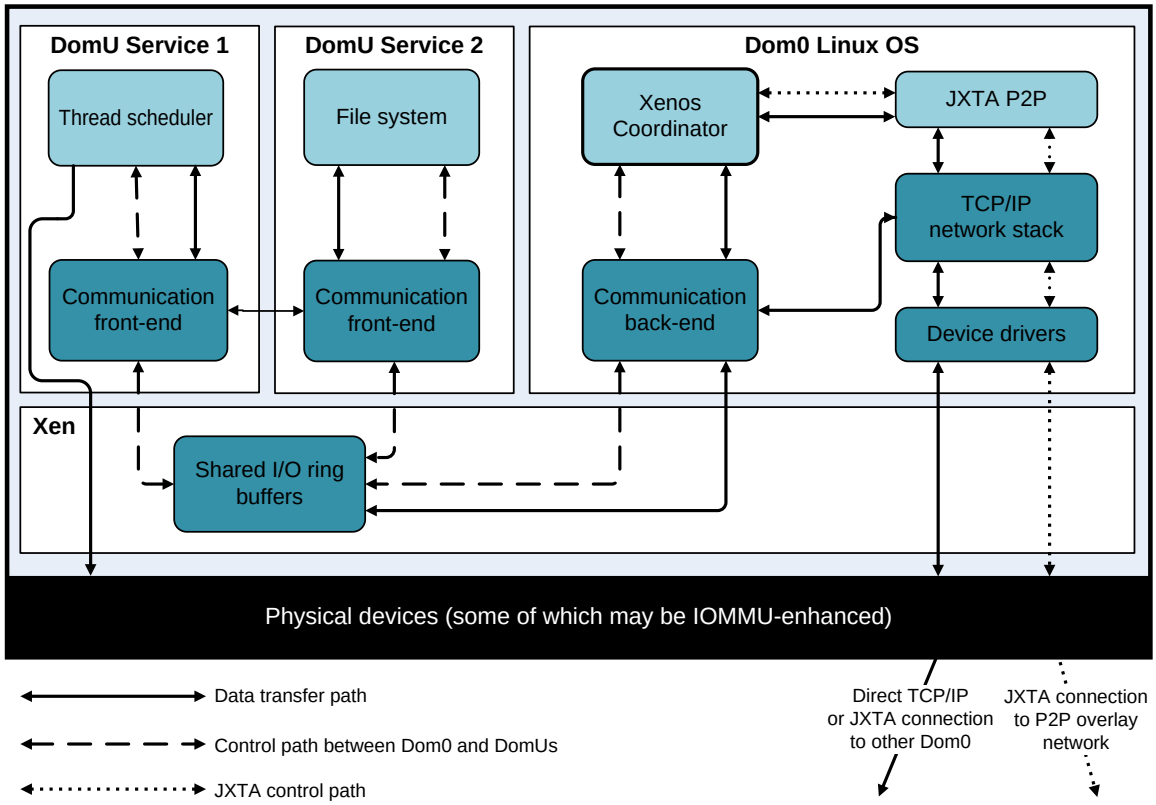

Figure 9. An alternate architecture allowing for various transport methods.

Dynamic DNS to address the problem of preserving network connections. More importantly, the authors also implement a pre-copy approach for transferring the disk image attached to a virtual machine, using a mechanism similar to that used by Xen when live migrating the state of a virtual machine. This greatly minimizes downtime even if the actual migration takes long due to poor network performance. Harney *et al.* [24] suggest using the mobility features in the IPv6 protocol to preserve network communication sessions, an approach that is viable in the long-term.

C. Alternative Transport Methods For Service Delivery

Applications featuring fine grained concurrency spanning across virtual and physical machines stand to gain from inter-virtual machine communication path optimizations such as shared memory communication for services residing on the same physical machine, and hypervisor-bypass network communication for distributed services. In both instances, the secure initialization of each communication path would be delegated to Xenos, allowing the data to move directly between the participating virtual machines and virtualization-enabled I/O devices. In some cases, an I/O could be permanently and exclusively bound to a specific service for low-latency dedicated access. Another enhancement is to allow services to piggy-back their communication over the JXTA protocol, which would allow communication between services that cannot reach one another outside of the peer-to-peer network. Figure 9 illustrates this concept.

D. Security, Authentication and Service Provisioning

A number of underlying mechanisms could be inherited from the Xen hypervisor and the JXTA peer-to-peer framework or their respective alternatives. To our benefit, JXTA provides several security and authentication features, as discussed by Yeager *et al.* [25]; these include TLS (Transport Layer Security), and support for centralized and distributed certification authorities. Xen provides a basis for automated accounting and billing services that track service consumption as well as physical resource use. However, Xenos should at least provide unified and distributed user, service and hierarchical service group authentication and permission mechanisms, a non-trivial undertaking in itself.

E. The Operating System-Agnostic Operating System

Software architectures in the vein of Xenos could fit the role of a distributed microkernel in a virtualization-embracing operating system that consists of interacting light-weight services hosted within virtual machines, including a multi-core thread scheduler, file systems (a stripped down Linux kernel), and device drivers. Each operating system service would run within its own light-weight Xen domain and expose itself through Xenos services (reminiscent of system calls). Xenos services would also host legacy operating systems and applications, presented to users through an

operating system-agnostic window manager hosted in a separate virtual machine. Applications with particular resource requirements or requiring isolation, such as computer games or web browsers, may easily be hosted in their own virtual machines, supported by a minimal application-specific kernel or library or even executing on 'bare virtualized metal'. Xen, and virtual machine monitors in general, have been described as "microkernels done right" [26], although others have argued that the drawbacks that muted the adoption of microkernels [27] still apply.

VII. Conclusion

An approach to building distributed middleware where services are hosted within virtual machines interconnected through a peer-to-peer network has been presented through the Xenos framework. Xenos extends well-established solutions for virtualization hypervisors and peer-to-peer overlay networks to deliver the beginnings of a fully decentralized solution for virtualized service hosting, discovery and delivery.

Using the Hadoop map-reduce framework and the HDFS as a test case, it was established that minimal performance overheads are associated with using the Xenos framework itself, and that the overheads introduced through the use of virtual machines are principally linked with the incidence of I/O operations. It is expected that forthcoming hardware support for virtualization will further reduce the gap between virtualized and native I/O performance pinpointed in the results, while simplifying hypervisors. This will further consolidate the virtual machine's position as a viable alternative for hosting both computation- and I/O-intensive tasks.

In practice, Xenos automated to a large degree the deployment of jobs while enabling the seamless migration of live Hadoop nodes. We thus believe that the combination of peer-to-peer technology and virtualization merits serious consideration as a basis for resilient distributed services.

References

[1] D. Bailey and K. Vella, "Towards peer-to-peer virtualized service hosting, discovery and delivery," in *AP2PS '10: Proceesings of the The Second International Conference on Advances in P2P Systems*, 2010, pp. 44–49.

[2] D. Bailey, "Xenos: A service-oriented peer-to-peer framework for paravirtualized domains," Master's thesis, University of Malta, 2010.

[3] P. Willmann, S. Rixner, and A. L. Cox, "Protection strategies for direct access to virtualized I/O devices," in *ATC'08: USENIX 2008 Annual Technical Conference on Annual Technical Conference*. Berkeley, CA, USA: USENIX Association, 2008, pp. 15–28.

[4] K. A. Fraser, S. M. Hand, T. L. Harris, I. M. Leslie, and I. A. Pratt, "The XenoServer computing infrastructure," University of Cambridge Computer Laboratory, Tech. Rep., 2003.

[5] P. Barham, B. Dragovic, K. Fraser, S. Hand, T. Harris, A. Ho, R. Neugebauer, I. Pratt, and A. Warfield, "Xen and the art of virtualization," in *SOSP '03: Proceedings of the nineteenth ACM symposium on Operating systems principles*. New York, NY, USA: ACM, 2003, pp. 164–177.

[6] D. Spence and T. Harris, "XenoSearch: Distributed resource discovery in the XenoServer open platform," in *HPDC '03: Proceedings of the 12th IEEE International Symposium on High Performance Distributed Computing*. Washington, DC, USA: IEEE Computer Society, 2003, p. 216.

[7] A. Ganguly, A. Agrawal, P. O. Boykin, and R. Figueiredo, "WOW: Self-organizing wide area overlay networks of virtual workstations," in *In Proc. of the 15th International Symposium on High-Performance Distributed Computing (HPDC-15*, 2006, pp. 30–41.

[8] M. Amoretti, F. Zanichelli, and G. Conte, "SP2A: a service-oriented framework for P2P-based grids," in *MGC '05: Proceedings of the 3rd international workshop on Middleware for grid computing*. New York, NY, USA: ACM, 2005, pp. 1–6.

[9] V. March, Y. M. Teo, and X. Wang, "DGRID: a DHT-based resource indexing and discovery scheme for computational grids," in *ACSW '07: Proceedings of the fifth Australasian symposium on ACSW frontiers*. Darlinghurst, Australia, Australia: Australian Computer Society, Inc., 2007, pp. 41–48.

[10] Q. Xia, R. Yang, W. Wang, and D. Yang, "Fully decentralized DHT based approach to grid service discovery using overlay networks," *Computer and Information Technology, International Conference on*, pp. 1140–1145, 2005.

[11] D. Talia, P. Trunfio, J. Zeng, and M. Hgqvist, "A DHT-based peer-to-peer framework for resource discovery in grids," Institute on System Architecture, CoreGRID - Network of Excellence, Tech. Rep. TR-0048, June 2006.

[12] W. Wadge, "Providing a grid-like experience in a P2P environment," Master's thesis, University of Malta, 2007.

[13] M. F. Mergen, V. Uhlig, O. Krieger, and J. Xenidis, "Virtualization for high-performance computing," *SIGOPS Oper. Syst. Rev.*, vol. 40, no. 2, pp. 8–11, 2006.

[14] S. Thibault and T. Deegan, "Improving performance by embedding HPC applications in lightweight Xen domains," in *HPCVirt '08: Proceedings of the 2nd workshop on System-level virtualization for high performance computing*. New York, NY, USA: ACM, 2008, pp. 9–15.

[15] M. J. Anderson, M. Moffie, and C. I. Dalton, "Towards trustworthy virtualisation environments: Xen library OS security service infrastructure," Hewlett-Packard Laboratories, Tech. Rep. HPL-2007-69, April 2007. [Online]. Available: http://www.hpl.hp.com/techreports/2007/HPL-2007-69.pdf

[16] K. Falzon, "Thread scheduling within paravirtualised domains," Bachelor of Science (Hons) Dissertation, University of Malta, 2009.

[17] R. J. Figueiredo, P. A. Dinda, and J. A. B. Fortes, "A case for grid computing on virtual machines," in *ICDCS '03: Proceedings of the 23rd International Conference on Distributed Computing Systems*. Washington, DC, USA: IEEE Computer Society, 2003, p. 550.

[18] K. Keahey, K. Doering, and I. Foster, "From sandbox to playground: Dynamic virtual environments in the grid," in *GRID '04: Proceedings of the 5th IEEE/ACM International Workshop on Grid Computing*. Washington, DC, USA: IEEE Computer Society, 2004, pp. 34–42.

[19] S. Santhanam, P. Elango, A. Arpaci-Dusseau, and M. Livny, "Deploying virtual machines as sandboxes for the grid," in *WORLDS'05: Proceedings of the 2nd conference on Real, Large Distributed Systems*. Berkeley, CA, USA: USENIX Association, 2005, pp. 7–12.

[20] S. Ibrahim, H. Jin, L. Lu, L. Qi, S. Wu, and X. Shi, "Evaluating MapReduce on virtual machines: The Hadoop case," in *CloudCom*, 2009, pp. 519–528.

[21] J. Dean and S. Ghemawat, "MapReduce: simplified data processing on large clusters," *Commun. ACM*, vol. 51, no. 1, pp. 107–113, 2008.

[22] F. Travostino, P. Daspit, L. Gommans, C. Jog, C. de Laat, J. Mambretti, I. Monga, B. van Oudenaarde, S. Raghunath, and P. Y. Wang, "Seamless live migration of virtual machines over the MAN/WAN," *Future Gener. Comput. Syst.*, vol. 22, no. 8, pp. 901–907, 2006.

[23] R. Bradford, E. Kotsovinos, A. Feldmann, and H. Schiöberg, "Live wide-area migration of virtual machines including local persistent state," in *VEE '07: Proceedings of the 3rd international conference on Virtual execution environments*. New York, NY, USA: ACM, 2007, pp. 169–179.

[24] E. Harney, S. Goasguen, J. Martin, M. Murphy, and M. Westall, "The efficacy of live virtual machine migrations over the internet," in *VTDC '07: Proceedings of the 3rd international workshop on Virtualization technology in distributed computing*. New York, NY, USA: ACM, 2007, pp. 1–7.

[25] W. Yeager and J. Williams, "Secure peer-to-peer networking: The JXTA example," *IT Professional*, vol. 4, pp. 53–57, 2002.

[26] S. Hand, A. Warfield, K. Fraser, E. Kotsovinos, and D. Magenheimer, "Are virtual machine monitors microkernels done right?" in *HOTOS'05: Proceedings of the 10th conference on Hot Topics in Operating Systems*. Berkeley, CA, USA: USENIX Association, 2005.

[27] G. Heiser, V. Uhlig, and J. LeVasseur, "Are virtual-machine monitors microkernels done right?" *SIGOPS Oper. Syst. Rev.*, vol. 40, no. 1, pp. 95–99, 2006.

Aggregation Skip Graph: A Skip Graph Extension for Efficient Aggregation Query over P2P Networks

Kota Abe, Toshiyuki Abe, Tatsuya Ueda, Hayato Ishibashi and Toshio Matsuura
Graduate School for Creative Cities, Osaka City University
Osaka, Japan
Email: {k-abe,t-abe,tueda,ishibashi,matsuura}@sousei.gscc.osaka-cu.ac.jp

***Abstract*—Skip graphs are a structured overlay network that allows range queries. In this article, we propose a skip graph extension called *aggregation skip graphs*, which efficiently execute aggregation queries over peer-to-peer network. An aggregation query is a query to compute an aggregate, such as MAX, MIN, SUM, or AVERAGE, of values on multiple nodes. While aggregation queries can be implemented over range queries of conventional skip graphs, it is not practical when the query range contains numerous nodes because it requires the number of messages in proportion to the number of nodes within the query range. In aggregation skip graphs, the number of messages is reduced to logarithmic order. Furthermore, computing MAX or MIN can be executed with fewer messages as the query range becomes wider. In aggregation skip graphs, aggregation queries are executed by using periodically collected partial aggregates for local ranges of each node. We have confirmed the efficiency of the aggregation skip graph by simulations.**

***Keywords*—aggregation query; peer-to-peer networks; skip graphs**

I. Introduction

P2P (Peer-to-Peer) systems have attracted considerable attention as technology for performing distributed processing on massive amounts of information using multiple nodes (computers) connected via a network. In P2P systems, each node works autonomously cooperating with other nodes that constitute a system that can be scaled by increasing the number of nodes.

Generally, P2P systems can be grouped into two major categories: unstructured and structured P2P systems. Structured P2P systems is able to look up data efficiently (typically in logarithmic or constant order), by imposing restrictions on the network topology.

Regarding structured P2P systems, DHT (Distributed Hash Table)-based systems, such as Chord [2], Pastry [3], and Tapestry [4], have been extensively researched. DHTs are a class of decentralized systems that can efficiently store and search for key and value pairs. DHTs also excel at load distribution. However, DHTs hash keys to determine the node that will store the data, and hence a value cannot be searched for if the correct value of the key is not known. Therefore, it is difficult with DHT to search for nodes whose key is within a specified range (range query).

As a structured P2P system which supports range queries, the skip graph [5] has attracted considerable attention. A skip graph is a distributed data structure that is constructed from multiple skip lists [6] that have keys in ascending order. The skip graph is suitable for managing distributed resources where the order of the keys is important.

Aggregation queries can be considered a subclass of range queries. An aggregation query is a query to compute an aggregate, such as the MAX, MIN, SUM, or AVERAGE, from the values that are stored in multiple nodes within a specified range. Aggregation queries are useful and sometimes essential for P2P database systems. Aggregation queries have a wide variety of applications. For example, across a range of nodes in a distributed computing system such as a grid, an aggregation query can be used to obtain the average CPU load, the node with the maximum CPU load, or the total amount of available disk space. An aggregation query can also be used to compute the average or maximum value from sensor data within a specified range on a sensor network. Other possible usage of aggregation queries can be found in [7][8].

While aggregation queries can be implemented by using range query over skip graphs, this is not efficient because every node in the specified range must process the aggregation query message; thus, this method stresses network bandwidth and CPU especially when aggregation queries having a wide range are frequently issued.

In this paper, we propose the **aggregation skip graph**, a skip graph extension that efficiently execute aggregation queries. In the aggregation skip graph, the expected number of messages and hops for a single aggregation query is $O(\log n + \log r)$, where n denotes the number of nodes and r denotes the number of nodes within the query range. Furthermore, computing MAX or MIN can be executed with fewer messages as the query range becomes wider.

We discuss related work in Section II and present the algorithm of the aggregation skip graph in Section III. In Section IV, we evaluate and discuss the aggregation skip graph. Lastly, in Section V, we summarize this work and discuss future work.

II. Related Work

A. Aggregation in P2P Network

Some research has focused on computing aggregations on P2P networks, to name a few, in the literature [7]–[11].

Most of the existing methods construct a reduction tree for executing aggregation queries, as summarized in [10]. However, this approach incurs a cost and complexity because

constructing a reduction tree over a P2P network is equal to adding another overlay network layer over an overlay network.

In addition, to our knowledge, none of the existing methods support computing aggregations in a subset of nodes; they compute aggregates only on all nodes.

As we discuss later, the aggregation skip graph does not require constructing a reduction tree, nor even maintaining additional links to remote nodes; aggregation queries are executed utilizing the data structure of underlying skip graphs. Furthermore, it can compute aggregates within a subset of nodes, by specifying a key range.

B. Skip Graph and Skip Tree Graph

A skip graph [5] is a type of structured overlay network. The skip graph structure is shown in Fig. 1. The squares in the figure represent nodes, and the number within each square is the key. Each node has a membership vector, which is a uniform random number in base w integer. Here, we assume $w = 2$.

Skip graphs consist of multiple levels, and level i contains 2^i doubly linked lists. At level 0, all of the nodes belong to only one linked list. At level $i(> 0)$, the nodes for which the low-order i digit of the membership vector matches belong to the same linked list. In the linked list, the nodes are connected by the key in ascending order. We assume that the leftmost node in the linked list and the rightmost node are connected (i.e., circular doubly-linked list). To maintain the linked lists, each node has pointers (IP address, etc.) to the left and right nodes at each level.

In a skip graph, when the level increases by 1, the average number of nodes for one linked list decreases by $1/2$. We refer to the level at which the number of nodes in the linked list becomes 1 as the *maxLevel*. The maxLevel corresponds to the height of the skip graph. In the skip graph for n nodes, the average maxLevel is $O(\log n)$, and the number of hops required for node insertion, deletion and search is also $O(\log n)$.

With skip graphs, aggregation queries can be easily implemented over range queries, in which one is asked all keys in [x, y]. Range queries require all nodes within the range receive a message. If we denote the number of nodes within the target range of the aggregation query by r, then range queries requires $O(\log n + r)$ messages and hops on average.

The skip tree graph [12] is a variant of skip graph, which allows fast aggregation queries by introducing additional pointers called *conjugated* nodes at each level. Skip tree graphs run range queries in $O(\log n + r)$ messages and $O(\log n + \log r)$ hops.

In either skip graphs or skip tree graphs, the number of messages for range queries increases in proportion to r; thus, these methods are not practical for aggregation queries, especially when aggregation queries with a wide range are frequently issued.

III. Proposed Method

In this section, we describe the detail of the aggregation skip graph.

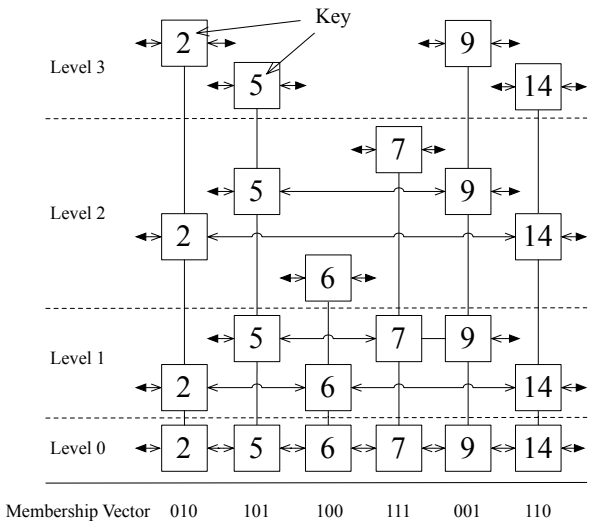

Fig. 1. Example of skip graphs

In the following sections, at first, we focus on aggregation queries to find the largest value in a specified range (i.e., MAX). We discuss other general aggregates (such as AVERAGE, SUM, etc.) later in Section III-D.

A. Data Structure

In aggregation skip graphs, each node stores a key–value pair. In the same manner as conventional skip graphs, the linked lists at each level are sorted by key in ascending order. The *value* is not necessarily related to the order of the key, and may change, for example, as in the case of sensor data.

The data stored in each node of an aggregation skip graph are shown in Table I. The key, membership vector, left[] (pointer to the left node at each level), right[] (pointer to the right node at each level), and maxLevel are the same as in conventional skip graphs. Hereinafter, we use the notation P.key to denote the key of node P. Also, we use the notation "node x" to denote the node whose key is x.

In addition to the skip graph, each node of a aggregation skip graph stores agval[] and keys[]. The *value* of the node is stored in agval[0]. P.agval[i] ($0 < i$) stores the MAX value within the nodes between P (inclusive) to P.right[i] (exclusive) in the linked list at level 0, where P.right[i] denotes the right node of node P at level i. P.keys[i] ($0 < i$) stores the key set that corresponds to the P.agval[i]. (A set is used because multiple keys may share the same MAX value.) P.keys[0] is not used. We describe the method to collect agval[] and keys[] later in Section III-C.

In skip graphs, the pointers for the left and right nodes point to more distant nodes as the level increases. Therefore, as the level increases, agval[] stores MAX values in a wider range, and agval[maxLevel] stores the MAX value for all nodes.

Fig. 2 shows an example of an aggregation skip graph. The squares in level 0 show the *value* (agval[0]) of a node, and $\mathrm{agval}[i]_{/\mathrm{keys}[i]}$ in level i ($0 < i$).

TABLE I
DATA MANAGED BY EACH NODE

Variable	Description
key	Key
m	Membership vector
right[]	Array of pointers to right node
left[]	Array of pointers to left node
maxLevel	First level at which the node is the only node in the linked list
agval[]	Array of collected MAX values for each level (agval[0] is the *value* of the node)
keys[]	Array of key sets that correspond to agval[]

Let us consider, as an example, the square at level 2 of node 5. The square contains $5_{/6}$ because the MAX value in the nodes between node 5 (inclusive) and node 9 (exclusive), which is the right node of node 5 at level 2, is 5 and the corresponding key is 6. Note that all of the nodes contain $9_{/2}$ at the highest level because the MAX value for all nodes is 9 and the corresponding key is 2.

In an aggregation skip graph, insertion and deletion of nodes can be accomplished by using the conventional skip graph algorithm; thus, this is not discussed here.

B. Query Algorithm

Here, we describe the algorithm for aggregation queries.

The algorithm gives the MAX value (and the corresponding key set) within all values stored by nodes whose key is in the specified key range $r = [r.\min, r.\max]$.

In the following, we provide a brief overview of the algorithm first (Section III-B1) and the details next (Section III-B2).

1) Algorithm overview: An aggregation query proceeds, starting from a node on the left side of the specified range (having a smaller key), to a node on the right side of the range.

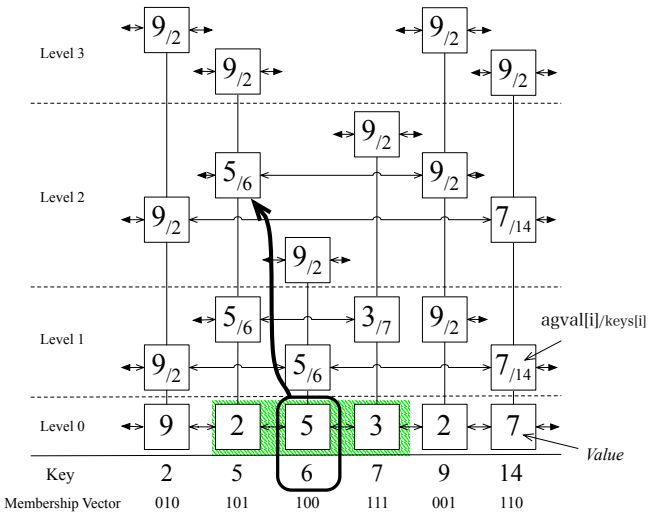

Fig. 2. Example of aggregation skip graph

Let us consider the case where node P issues an aggregation query for range r. If P is within r, P forwards the query message to node Q, where Q is a node known to P, which is outside of range r and the closest to $r.\min$. (If P is outside range r, then read the Q below as P instead.)

Let i denote the current level, starting from $\mathrm{maxLevel} - 1$. Let s denote the range from Q to Q.right[i], and x the MAX value in s. Node Q executes one of the following steps, determined by the relation between range r, range s and x. This is depicted in Fig. 3 (1)–(4). In the figure, the arrow represents the pointer from Q to Q.right[i].

(1) **Range s includes range r, and the key of x is within r.**
It is clear that x is also the MAX value for r; thus, x is returned to P as the MAX value and the query terminates.

(2) **Range s has a common area with range r, and the key of x is within r.**
The value x is the MAX value for the common area between range s and range r. However, because an even larger value may exist in the remaining range r, the query is forwarded to Q.right[i]. The value of x and the corresponding key are included in the forwarded message.

(3) **Range s has a common area with range r, but the key of x is not within r**
In this case, no information is obtained about the MAX value within range r; thus, the current level (i) is decreased by 1 and this process is repeated again from the beginning.

(4) **Range s and range r do not have any common areas**
The MAX value in range r does not exist in range s; thus, the query is sent to Q.right[i].
In this case, Q.right[i] acts as the new Q and the process is repeated again in the same manner.

Next, we describe the algorithm for a node that receives a forwarded query message from node Q in case (2). We denote such a node by R. Again, let i denote the current level, starting from $\mathrm{maxLevel} - 1$. Also, let t denote the range from R to R.right[i], and let y denote the MAX value in t. Node R executes one of the following steps. This is depicted in Fig. 3 (5)–(7).

(5) **Range t has a common area with range r, and the key of y is within r**
The $\max(x, y)$ is returned to P and the query terminates.

(6) **Range t has a common area with range r, but the key of y is not within in r**
If $x > y$, x is returned to P as the MAX value and the query terminates. Otherwise, decrease the current level (i) by 1 and repeat this process.

(7) **Range t is included in range r**
The query is forwarded to R.right[i]. The $\max(x, y)$ and the corresponding key are included in the for-

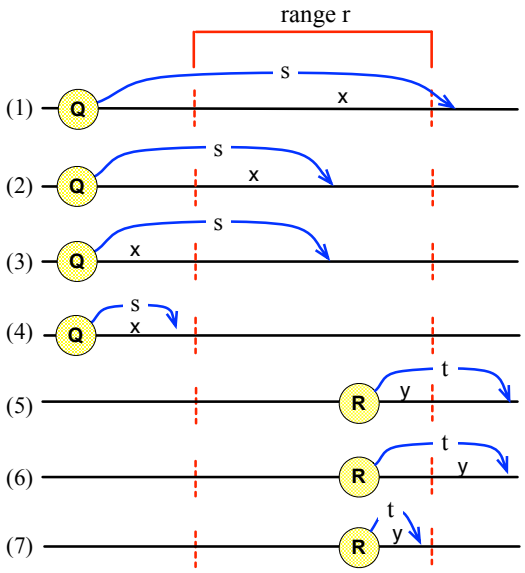

Fig. 3. Relationship between node and range

warded message.

2) Algorithm details: Here, we present the detailed algorithm in pseudocode.

Node P initiates an aggregation query by calling `agQuery(r, d, P, −∞, ∅)`. The parameter d indicates the direction of the query. The initial value of d is, LEFT if P.key is included in range r, otherwise RIGHT.

In the description, RPC (Remote Procedure Call) is used for communication. The notation $a \prec b \prec c$ means $(a < b < c \ \vee \ b < c < a \ \vee \ c < a < b)$. $(a \prec b \prec c = \text{true}$ if node a, b, c appear in this order in a sorted circular linked list, following the right link starting from node a.)

```
// r: key range [r.min, r.max]
// d: forwarding direction (LEFT or RIGHT)
// s: query issuing node
// v: MAX value that has been acquired thus far
// k: set of keys that corresponds to v
P.agQuery(r, d, s, v, k)
  if elements of P.keys[maxLevel] are included in r then
    send P.agval[maxLevel] and P.keys[maxLevel] to node s
    return
  end if
  {When forwarding to the left (trying to reach the left side of range r)}
  if d = LEFT then
    search for the node n that is closest to r.min and satisfies
    (r.max ≺ n.key ≺ r.min) from the routing table of P (i.e.,
    P.left[] and P.right[])
    if such node n exists then
      call agQuery(r, RIGHT, s, v, k) on node n
    else
      let n be the node that is closest to r.min and satisfies (r.min
      ≺ n.key ≺ P.key), found in the routing table of P
      call agQuery(r, LEFT, s, v, k) on node n
    end if
    return
  end if
  {When forwarding to the right}
  if d = RIGHT then
    if P.key is included in r and a level j exists that satisfies (P.key
    ≺ r.max ≺ P.right[j]) and v > P.agval[j] then
      {Corresponds to the case where x > y in SectionIII-B1 case (6)}
      send v and k to node s
      return
    end if
    {The process for finding i that satisfies the conditions in the next if
    statement corresponds to (3) or (6)}
    if level k exists such that elements of P.keys[k] are included
    within r (let i be the largest value for such k) then
      {Update the v and k}
      if P.agval[i] > v then
        v = P.agval[i], k = P.keys[i]
      else if P.agval[i] = v then
        k = k ∪ P.keys[i]
      end if
      {Terminate if the query exceeds the rightmost end of r}
      if r.min ≺ r.max ≺ P.right[i].key then
        {Corresponds to (1) or (5)}
        send v and k to node s
        return
      end if
      {Corresponds to (2) or (7)}
      call agQuery(r, RIGHT, s, v, k) on P.right[i]
      return
    else
      if P.key ≺ r.max ≺ P.right[0].key then
        {No node exists within the range}
        send null to node s
      else if P.right[0].key is included in r then
        {The case that P is the node directly to the left end of r}
        call agQuery(r, RIGHT, s, v, k) on P.right[0]
      else
        {Corresponds to (4)}
        search for the node n that is closest to r.min and satisfies
        (P.key ≺ n.key ≺ r.min) from the routing table of P
        call agQuery(r, RIGHT, s, v, k) on n
      end if
      return
    end if
  end if
```

C. Aggregates Collecting Algorithm

Because nodes may join or leave, and the *value* of nodes may change, we periodically collect and update the agval[] and keys[] of each node. We next explain the algorithm used to accomplish this.

1) Algorithm overview: When a node joins an aggregation skip graph, all entries of agval[] and keys[] of the node are respectively set to the value and the key of the node. Then, each node periodically sends an *update message* to collect agval[] and keys[] for each level. This is shown in Fig. 4. The thick line in the figure indicates the message flow when node 5 sends an update message. (It is assumed that node 5 has newly joined the aggregation skip graph.) The update message is forwarded to the left node, starting from level $\text{maxLevel} - 1$. If the left node is the originating node of the update message (node 5), then the level is decreased and the forwarding continues. Finally at level zero, the message returns to the originating node (node 5).

The update message contains v[] and k[] to store the MAX value and the corresponding key set for each level. While the message is being forwarded to the left at level i, the

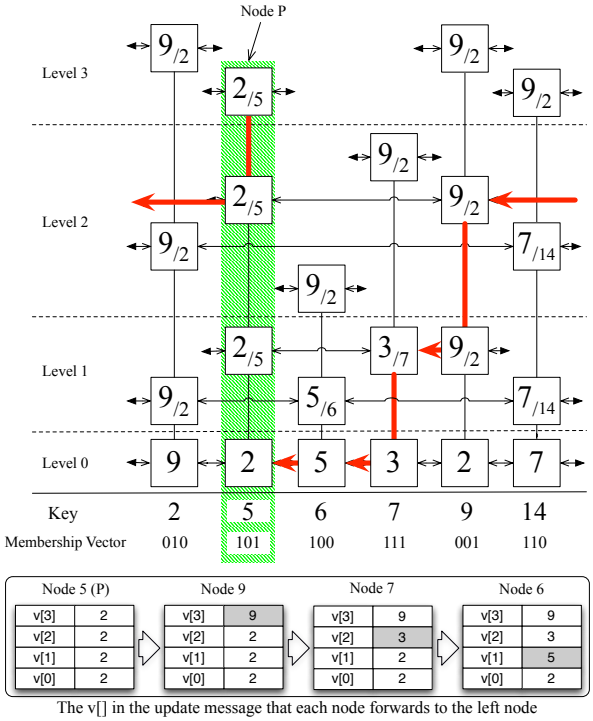

Fig. 4. Message flow of an update message

MAX value of agval[i] on the node that the message passes is collected in v[$i+1$]. When the message returns to the originating node, agval[i] ($i > 0$) is calculated using the following formula:

$$\text{P.agval}[i] \leftarrow \max\{v[j] \mid 0 \leq j \leq i\}.$$

The lower part of Fig. 4 shows v[] in an update message that each node forwards to the left node. When the originating node (node 5) receives the update message, its agval[] is set to $\{2, 5, 5, 9\}$.

To obtain the correct agval[] and keys[] for every node, each node must execute the update procedure described above maxLevel times because the accuracy of the information stored in an update message level i (i.e., v[i] and k[i]) depends on the accuracy of agval[$i-1$] and keys[$i-1$] of each node that the message passes.

2) Algorithm details: Here, we present the detailed algorithm using pseudocode.

Each node periodically execute `update(maxLevel - 1, P.agval[], P.keys[], P)` for updating its agval[] and keys[] .

```
// lv: level
// v[]: array of aggregated value
// k[]: array of key set
// s: originating node of update message
P.update(lv, v[], k[], s)
  {When the message returns to the originating node}
  if lv = 0 and P = s then
    for i = 1 to maxLevel do
      find j where v[j] is the MAX value from v[0] to v[i]
      P.agval[i] ← v[j]
      P.keys[i] ← k[j]
      {If multiple j's correspond to the MAX value, k[j] is a union set of them}
    end for
    return
  end if
  {A larger value is found}
  if v[lv + 1] < P.agval[lv] then
    v[lv + 1] ← P.agval[lv]
    k[lv + 1] ← P.keys[lv]
  else if v[lv + 1] = P.agval[lv] then
    k[lv + 1] ← k[lv + 1] ∪ P.keys[lv]
  end if
  {Find a level from the top where the left node is not s}
  for i = lv downto 1 do
    if P.left[i] ≠ s then
      call update(i, v, k, s) on P.left[i]
      return
    end if
  end for
  call update(0, v, k, s) on P.left[0]
  return
```

D. Computing General Aggregates

While the algorithm described so far targets the MAX as the aggregates, it is trivial to adapt to the MIN. To compute other general aggregates such as the AVERAGE, SUM or COUNT, the following modification is required.

Instead of storing the MAX value, agval[] stores both the sum of values and number of nodes within each range. The aggregates collecting algorithm should be modified accordingly. In the case of computing the MAX value, the result may be obtained in an early step (i.e., it is not always necessary to reach the nodes at each end of the region.). However, when computing the AVERAGE, SUM or COUNT, it is always necessary to reach the node at both ends; first route a query message to the leftmost node of the query range and then route the message to the rightmost node of the range. This requires total $O(\log n + \log r)$ messages and hops.

In general, this method can compute any aggregation function f that satisfies a property $f(S) = f(S_1) \circ f(S_2)$ where S, S_1 and S_2 are a set of data that satisfies $S = S_1 + S_2$ and $\circ$ is a binary operator. In addition, a function consisting of combination of such f can be computed. For example, the variance ($\frac{1}{N}\sum_{i=1}^{N} x_i^2 - \overline{x}$) can be computed using COUNT, SUM of squares and AVERAGE.

IV. Evaluation and Discussion

In this section, we give some evaluation and discussion of the aggregation skip graph. We take n as the total number of nodes, r as the target range for the aggregation query.

A. Cost of the Aggregation Query

Here, we examine the number of messages and hops required for an aggregation query. The algorithm proposed in this paper does not send messages to multiple nodes simultaneously, so the required number of hops and number of messages are equal.

1) MAX and MIN: When computing the MAX (or MIN), in the worst case, the query algorithm in Section III-B gives the maximum number of hops when the closest nodes on both sides outside of range r store values that are larger than the largest value within r. In such cases, the aggregation query message (1) first arrives at the node immediately to the left of r, and then (2) reaches the rightmost node within r. This requires $O(\log n + \log r)$ hops on average.

However, in an aggregation skip graph, as the target range of the aggregation query becomes wider, the probability becomes higher that the MAX (or MIN) value stored in agval[] of each node falls within the query range. Therefore, on average the aggregation query is able to be executed in fewer hops. To evaluate the number of hops, we ran the following simulation.

We set the number of nodes (n) as 1,000 for the simulation. We assigned a key and value to each node using a random number between 0 and 9,999. We also assigned the membership vector using a random number.

Next, we registered each node in the aggregation skip graph. We also set the agval[] and keys[] of each node using the algorithm discussed in Section III-C.

We executed the simulation varying the size of the aggregation query range, from 1% to 95% of the key value range (0 to 9,999). (We used 1% steps from 1% to 10%, and 5% steps beyond 10%.) We measured the maximum, the average, and the 50th and 90th percentiles, of number of hops.

The trial was performed 1,000 times for each range size. For each experiment, the initiating node of the aggregation query and the range of the aggregation query were selected using random numbers.

The results are shown in Fig. 5. The x-axis shows the width of the aggregation query range, and the y-axis shows the number of hops.

From the graph, we can confirm that as the target range of the aggregation query becomes wider, the number of average hops decreases. In addition, according to the 50th percentile value, we can see that the query often terminates in 0 hops when the range size is large.

The maximum number of hops is not stable. This is because there is no tight upper bound of hops in skip graphs; it is affected by distribution of membership vectors.

2) General Aggregates: We also executed a simulation for computing general aggregates (See Section III-D) in the same condition (Fig. 6). It confirms that computing general aggregates is executed in $O(\log n + \log r)$ messages.

B. Cost of Collecting Aggregates

As discussed in Section III-C, each node must periodically collect aggregates into its agval[] and keys[]. Here, we discuss the cost of this procedure.

On average, the number of hops that the message is forwarded to the left node in one level is about 1 hop, assuming uniformity of membership vectors. Considering the average maxLevel is in $O(\log n)$, an update message sent from a particular node will be forwarded $O(\log n)$ times on average before it returns to the node.

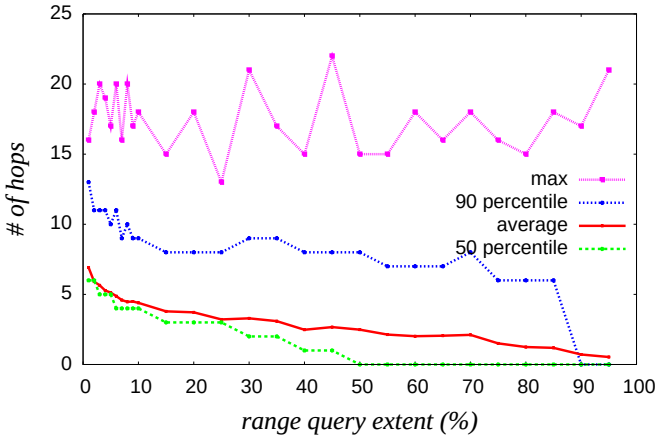

Fig. 5. Number of hops vs query range size for computing MAX

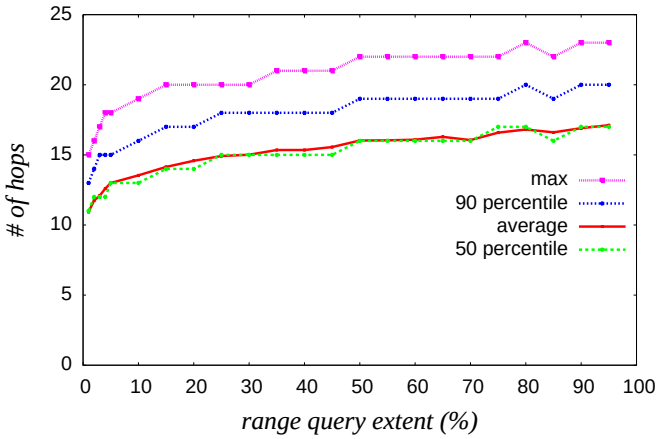

Fig. 6. Number of hops vs query range size for computing general aggregates

Each node executes this procedure periodically. If the period of this procedure is t, then $O(n \log n)$ update messages are forwarded by all nodes in t. Because these messages are scattered over n nodes, each node processes $O(\log n)$ messages in t on average.

Let us investigate this additional cost is worth paying by comparing the estimated number of messages for an aggregation skip graph with that for a conventional skip graph using simple range queries.

We assume that each node issues q aggregation queries in period t and each query targets $\lceil pn \rceil$ nodes ($0 < p \leq 1$). In the conventional method, $qn(c_1 \log_2 n + \lceil pn \rceil)$ messages are issued in t, where c_1 (and c_2, c_3 below) is a constant factor. In the aggregation skip graph, $nc_2 \log_2 n$ messages are issued for collecting aggregates and $qn(c_1 \log_2 n + c_3 \log_2 \lceil pn \rceil)$ messages are issued for aggregation queries, which sum up to $nc_2 \log_2 n + qn(c_1 \log_2 n + c_3 \log_2 \lceil pn \rceil)$ messages in t. Note that this calculation can be applied both to the worst case of computing MAX or MIN and to the average case of computing general aggregates (see Section IV-A).

The aggregation skip graph is more efficient than the conventional method with regard to the number of mes-

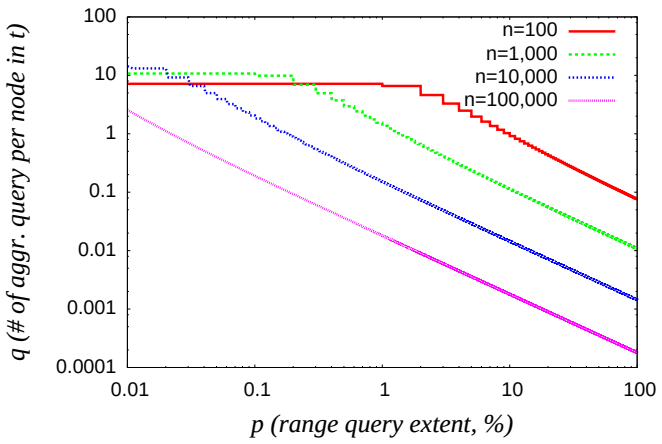

Fig. 7. Graph to determine which, the aggregation skip graph or the conventional skip graph, is efficient. Both axes are in logarithmic scale. The area under the curve is the region where the conventional skip graph is more efficient.

sages if $qn(c_1 \log_2 n + \lceil pn \rceil) > nc_2 \log_2 n + qn(c_1 \log_2 n + c_3 \log_2 \lceil pn \rceil)$, which is equivalent to $q > (c_2 \log_2 n)/(\lceil pn \rceil - c_3 \log_2 \lceil pn \rceil)$.

Fig. 7 is a plot of function $(c_2 \log_2 n)/(\lceil pn \rceil - c_3 \log_2 \lceil pn \rceil)$ in several n, varying p from 0 to 1. The x-axis shows p and the y-axis shows the q, both in logarithmic scale. We use $c_2 = 1.08, c_3 = 0.91$, that are obtained from a preliminary experiment (c_1 is eliminated in the function). The area under the curve is the region where the conventional skip graph is more efficient. For example, if $n = 100,000$ and each node issues 0.1 queries in t, the aggregation skip graph is more efficient when the query covers more than about 0.18% of nodes, or 180 nodes. As one can see from the graph, the aggregation skip graph is in general more efficient than the conventional skip graph when the query frequency is high. In addition, even if the query frequency is low, the aggregation skip graph is more efficient unless both the query range and the number of nodes are very small.

C. Recovering from Failures

Due to a node failure or ungraceful leaving, the link structure of (aggregation) skip graphs might be temporarily broken. In that case, agval[] and keys[] of some nodes might have out-of-date values. However, because these values are periodically updated, this situation is eventually resolved as the link structure of the skip graphs is recovered. (We assume that some repair algorithm for skip graphs is engaged.)

V. Conclusion and Future Work

In this paper, we have proposed aggregation skip graphs, which allows efficient aggregation queries. The structure of aggregation skip graphs is quite simple; utilizing the structure of skip graphs enables eliminating construction of a reduction tree. Thus, it can be easily implemented over skip graphs. The aggregation skip graph supports computing aggregates on a subset of nodes by specifying key ranges, which cannot be accomplished with conventional aggregation algorithms in P2P networks.

Computing aggregates with aggregation skip graphs requires only $O(\log n + \log r)$ messages, which is a substantial improvement over the $O(\log n + r)$ messages required of range queries over conventional skip graphs (n denotes the number of nodes and r denotes the number of nodes within the query range). In addition, computing the MAX or MIN is quite fast; it requires fewer messages as the query range becomes wider.

The aggregation skip graph has the following drawbacks: (1) additional costs are incurred because each node collects aggregates periodically; and (2) the aggregation query results do not reflect the up-to-date situation because results are based on periodically collected aggregates. However, we believe that aggregation skip graphs are useful for P2P systems that execute aggregation queries over a wide target range or that have large number of nodes.

One of our future work is to devise a method to reduce the cost of collecting aggregates. The following methods should be considered. (1) adaptively adjust the collection period, or (2) update (and propagate) aggregates only when necessary. Another future work is to give an exhaustive comparison between the aggregation skip graph with the existing techniques such as in [7]–[11].

Acknowledgments

We would like to express our gratitude to Dr. Mikio Yoshida at BBR Inc. for his insightful comments and suggestions. This research was partially supported by the National Institute of Information and Communications Technology (NICT), Japan.

References

[1] Kota Abe, Toshiyuki Abe, Tatsuya Ueda, Hayato Ishibashi, and Toshio Matsuura. Aggregation skip graph: An extension of skip graph for efficient aggregation query. In *AP2PS '10: Proceedings of the 2nd International Conference on Advances in P2P Systems*, pages 93–99, 2010.

[2] Ion Stoica, Robert Morris, David Karger, M. Frans Kaashoek, and Hari Balakrishnan. Chord: A Scalable Peer-to-peer Lookup Service for Internet Applications. *ACM SIGCOMM Computer Communication Review*, 31(4):149–160, 2001.

[3] Antony Rowstron and Peter Druschel. Pastry: Scalable, decentralized object location, and routing for large-scale peer-to-peer systems. *Lecture Notes in Computer Science*, 2218:329–350, 2001.

[4] Ben Y. Zhao, John Kubiatowicz, and Anthony D. Joseph. Tapestry: An infrastructure for fault-tolerant wide-area location and routing. Technical Report UCB/CSD-01-1141, UC Berkeley, April 2001.

[5] James Aspnes and Gauri Shah. Skip graphs. *ACM Trans. on Algorithms*, 3(4):1–25, 2007.

[6] William Pugh. Skip lists: A probabilistic alternative to balanced trees. *Communications of the ACM*, 33:668–676, 1990.

[7] Carlos Baquero, Paulo Sérgio Almeida, and Raquel Menezes. Fast estimation of aggregates in unstructured networks. In *International Conference on Autonomic and Autonomous Systems (ICAS)*, pages 88–93. IEEE Computer Society, 2009.

[8] Márk Jelasity, Alberto Montresor, and Ozalp Babaoglu. Gossip-based aggregation in large dynamic networks. *ACM Transactions on Computer Systems*, 23(3):219–252, 2005.

[9] Mayank Bawa, Hector Garcia-Molina, Aristides Gionis, and Rajeev Motwani. Estimating aggregates on a peer-to-peer network. Technical Report 2003-24, Stanford InfoLab, 2003.

[10] Norvald H. Ryeng and Kjetil Nørvåg. Robust aggregation in peer-to-peer database systems. In *Proceedings of the 2008 international symposium on Database engineering & applications (IDEAS'08)*, pages 29–37. ACM, 2008.

[11] Ji Li, Karen Sollins, and Dah-Yoh Lim. Implementing aggregation and broadcast over distributed hash tables. *SIGCOMM Compututer Communication Review*, 35(1):81–92, 2005.

[12] Alejandra González Beltrán, Peter Milligan, and Paul Sage. Range queries over skip tree graphs. *Computer Communications*, 31(2):358–374, 2008.

Naming, Assigning and Registering Identifiers in a Locator/Identifier-Split Internet Architecture

Christoph Spleiß, Gerald Kunzmann[1]
Technische Universität München
Department of Communication Engineering
Munich, Germany
{*christoph.spleiss, gerald.kunzmann*}*@tum.de*

***Abstract*—Splitting the IP-address into locator and identifier seems to be a promising approach for a Future Internet Architecture. Although this solution addresses the most critical issues of today's architecture, new challenges arise through the necessary mapping system to resolve identifiers into the corresponding locators. In this work, we give an overview of a scheme how to name identifiers not only for hosts, but basically for anything that needs to be identified in a future Internet. Our approach is based on the HiiMap locator/ID split Internet architecture and supports user-friendly identifiers for hosts, content, and persons and does not rely on DNS. We show how the registration and assignment for identifiers is handled and which modifications in the network stack are necessary. Furthermore, a possible solution for a lookup mechanism that can deal with spelling mistakes and typing errors in order to improve the quality of experience to the user is provided.**

***Keywords*-Locator/ID-split, Future Internet, Naming schemes, Content Addressing.**

I. Introduction

Today's Internet architecture has been developed over 40 years ago and its only purpose was to interconnect a few single nodes. At that time, no one expected that the Internet and the number of connected devices would grow to the current size. Measurements show that the Internet continues growing at a tremendous high rate. The address space of the current IPv4 addresses is already too small to address every single node in the Internet and the growth of BGP routing tables sizes in the Default Free Zone (DFZ) becomes critical for the Internet's scalability [1]–[4]. Beyond that, mechanisms like traffic engineering (TE) or multi homing are further increasing the BGP tables sizes, as IP address blocks are artificially de-aggregated or advertised multiple times. While IPv6 is a promising solution for the shortage of addresses, it will probably increase the Border Gateway Protocol (BGP) routing table problem. Therefore, at least for IPv6, a new and scalable routing architecture is necessary. Besides that, more and more devices connected to the Internet are mobile, such as smart phones or netbooks. Even more and more cars, which are mobile by definition, have access to the Internet. However, the current Internet architecture has only very weak support for mobility, as the IP-address changes whenever a device roams between different access points.

All problems mentioned above occur because the IP-address, no matter if IPv4 or IPv6, is semantically overloaded with two completely independent values. It is used to *identify* a specific host in the Internet, while on the other hand, it is also used to *locate* this node. Separating the current IP address into two independent parts for reachability and identification is a promising solution to many problematic issues with today's Internet [5]. With this approach a known identifier (ID) can always be used to reach a specific host, no matter where it is currently attached to the network. Thereby, a logical communication session is always bounded to the ID and not to the locator. A locator change due to mobility is handled by the network stack and is completely transparent to the communication session and the overlying application. It does not disconnect an ongoing communication anymore. Furthermore, it is possible to assign more locators to a specific ID, e.g., if a host is multihomed.

However, not only the number of hosts has developed differently than initially expected, but also the way people use the Internet. While the current Internet architecture is designed for accessing a specific machine, todays focus has shifted on accessing a specific piece of information. The host storing the information is thereby of minor interest. The idea is a new network architecture named Content Centric Network (CCN) [6]–[8] where content and information can be directly addressed independent of its storage location, which usually points to a host. Furthermore, the emergence of social networks, Web 2.0 applications, Voice over IP (VoIP) and instant messaging applications additionally put the person in the focus of interest. People want to communicate with each other, regardless of the device each of the person is using.

The split of locator and ID thereby offers ideal prerequisites for the support of addressing schemes for content, information and persons. Using this paradigm, an ID is assigned for every host, content object and person. A highly scalable mapping system, which is a mandatory part of every

[1]Dr. Kunzmann is now working for DOCOMO Communications Laboratories Europe GmbH, Landsberger Strasse 312, Munich, Germany.

locator/ID separated Internet architecture, translates IDs into the corresponding locators. Note that the mapping system can not only return a set of locators, which are necessary to access a specific host, but it can also return a more complex description of a content object or a person.

A crucial question in conjunction with a locator/ID separated Internet architecture is how to name, assign and register IDs. As IDs are used as control information in communication protocols and packet headers, they are mostly fixed-length bit strings that can be hardly memorized by humans. However, in order to avoid a second resolution system like DNS that translates easy memorizable names to IDs, we need a way how to uniformly name IDs. In this work, we present a flexible and adaptable naming scheme for IDs that can be used to identify hosts, content, persons and is open for future extensions. We furthermore present techniques how to register IDs and how assign them accordingly. Although our approach can be adapted to basically any locator/ID separated Internet architecture, it is currently based on the HiiMap Internet architecture [9], as HiiMap provides a highly scalable and customizable mapping system. It does not rely on the Domain Name System and allows each entity to calculate the requested ID on its own.

The paper is structured as follows. In Section 2, we discuss related work and different concepts of locator/ID split architectures. Section 3 describes our approach of a new naming scheme for IDs while Section 4 deals with registration and assignment issues. Section 5 demonstrates necessary modifications in the network stack and Section 6 discusses a possible lookup algorithm that tolerates spelling mistakes and allows unsharp queries to a certain extent. Section 7 summarizes the results and concludes.

II. Related Work

Many proposals dealing with the split of locator and ID have been published so far, but only a few of them discuss how to name IDs. However, almost all of them use a bit-representation of constant length as ID.

A. Host-based approaches

The majority of these proposals are solely host-based approaches. Among them Hair [11], FIRMS [13], LISP [14], HIP [15], HIMALIS [16] and many others. In the following, we will describe three of them more detailed, as their way how to implement the locator/ID-split is representative.
LISP: In contrast to other architectures that are examined in this work, LISP does not separate the identifier from routing purposes. Within an edge network or autonomous system (AS), the normal IP-address still serves as so called *Endpoint Identifier (EID)* and routing address at the same time. While the EID is only routable inside a LISP-domain, an additional set of addresses is used for the routing between different LISP-domains, which are called *Routing Locators (RLOC)*. RLOCs are the public IP-addresses of the border routers of a LISP-domain, globally routable, and independent of the nodes' IP addresses inside the domain. Whenever a packet is sent between different LISP-domains, the packet is first routed to the *Ingress Tunnel Router (ITR)*, encapsulated in a new IP packet, and routed to the *Egress Tunnel Router (ETR)* according to the RLOCs. The ETR unpacks the original packet and forwards it to the final destination. A mapping system is necessary to resolve foreign EIDs (EIDs that are not in the same domain) to the corresponding RLOCs. However, as in normal IP networks, the EID changes whenever a node changes its access to the network. Furthermore, DNS is still necessary to resolve human readable hostnames to EIDs.

HIP: The Host Identity Protocol implements the locator/ID split by introducing an additional layer between the network and the transport layer. For applications from higher layers the IP address is replaced by the *Host Identity Tag (HIT)*, which serves as identifier. HIP leaves the IP-layer untouched and the locator is a simple IPv4 or IPv6 address. Therefore, HIP does not rely on special gateways, which is very migration-friendly. However, as it does not influence the routing system, it does not take any countermeasures against the growth of the BGP routing tables. Instead, the main focuses of HIP are security features. The *Host Identifier (HI)* is a public key of an asymmetric key pair and used for identification and cryptographic purposes at the same time. The HI is not used in packet headers, as the key length can vary over time. Instead, the HIT, which is a hash value from the HI, is used for addressing purposes. Encryption, authenticity and integrity can be achieved due to the presence of an asymmetric key pair. However, the coupling of identifier and public key is a major drawback, as the ID changes whenever the key pair changes. While no permanent validity may be a desirable feature for cryptographic key pairs, it is not for an identifier.

HIMALIS: Like HIP, the HIMALIS (Heterogeneity Inclusion and Mobility Adaption through Locator ID Separation in New Generation Network) approach realizes the locator/ID split by introducing an extra layer between network and transport layer, the so-called Identity Sublayer. Upper layers solely use the host IDs for session identification, while the lower layer use the locator for packet forwarding and routing. HIMALIS can use any kind of addressing scheme for locators and supports security features based on asymmetric keys. Despite that, it does not burden the ID with the semantic of a public key. HIMALIS uses domain names as well as host IDs to identify hosts. In contrast to other approaches, a scheme how to generate host IDs out of the domain name using a hash function is shown. However, HIMALIS uses three different databases for resolving domain names and hostnames to IDs and locators (Domain Name Registry, Host Name Registry and ID Registry), which results in increased maintenance effort to achieve data consistency.

B. Content-based approaches

Contrary to host based approaches, several proposals for a future Internet architecture take into account the changing demands regarding the intended use of the Internet, namely the focus on retrieving information [6].

The **NetInf** (Network of Information) architecture shows how locator/ID separation can be used for content-centric networking [8]. By introducing an information model for any kind of content, NetInf allows fast retrieval of information in the desired representation. Thereby, each information object (IO) includes a detailed description of the content and its representations, with locators pointing to the machine that stores the information. The ID is assigned to the IO and is composed out of different hash values of the content creator's public key and a label created by the owner. In order to find a specific IO, the creator's public key and label must be known exactly. A first approach of NetInf is realized as an overlay architecture that uses the current routing and forwarding system, while a second alternative plans to deal with name-based routing mechanisms that provide an integrated name resolution and routing architecture.

Another Future Internet Architecture focusing on content is **TRIAD** [7]. One key aspect of TRIAD is the explicit introduction of a content layer that supports content routing, caching and transformation. It uses character strings of variable length as content IDs and uses the packet address solely as locator. The TRIAD content layer consists of *content routers* that redirect requests to *content servers* and *content caches* that actually store the content. DNS is used to resolve locators for content objects and the content delivery is purely based on IP in order to achieve maximum compatibility with the current Internet architecture.

C. Hybrid approaches

A proposal for a Next Generation Internet architecture that supports basically any kind of addressing scheme is the **HiiMap** architecture [9]. Due to the locator/ID separation and a highly flexible mapping system, HiiMap allows for addressing hosts as well as content and is still open for future extensions and requirements. In the following, we use the term *entity* for any addressable item.

The HiiMap architecture uses never changing IDs, so called UIDs (unique ID) and two-tier locators. One part of the locator is the LTA (local temporary address) that is assigned by a provider and routable inside the provider's own network. The other part is the gUID (gateway UID). This is a global routable address of the provider's border gateway router and specifies an entrance point into the network. Thereby, each provider can choose its own local addressing scheme that can be adapted to specific needs. However, a common addressing scheme for all providers is necessary for the gUID in order to route packets [17].

HiiMap splits the mapping system into different regions, whereby each region is its own independent mapping system that is responsible for the UID/locator mappings of entities registered in this region. The mapping system in each region consists of a one-hop distributed hash table (DHT) to reduce lookup times. As DHTs can be easily extended by adding more hosts, the mapping system is highly scalable. In order to query for UIDs which regions are not known, a region prefix (RP) to any UID is introduced (compare Figure 1). This RP can be queried at the so-called Global Authority (GA), which resolves UIDs to RPs. The GA is a centralized instance and acts as root of a public key infrastructure, thus providing a complete security infrastructure. As RP-changes are expected to be rare, they can be cached locally.

Like other approaches, HiiMap uses fixed length bit strings of 128 bits as UID. As plaintext strings are not feasible as UIDs due to their variable length, a naming scheme is necessary to assign UIDs to all kinds of entities. Thereby, the existing Domain Name System is to be replaced by the more flexible HiiMap mapping system.

III. New naming scheme for Identifiers

In this section, we introduce a naming scheme for IDs that is suitable to address basically any entity and that can be generated out of human friendly information. Although we use the HiiMap architecture exemplarily for introducing this approach, it can also be adapted to other locator/ID split architectures.

A. General Requirements for Identifiers

When introducing a Future Internet Architecture based on locator/ID separation, the ID has to fulfill some mandatory requisites. In the following, we sum up general requirements for IDs proposed by the ITU [18]:

- The ID's namespace must completely decouple the network layer from the higher layers.
- The ID uniquely identifies the endpoint of a communication session from anything above the transport layer.
- The ID can be associated with more than one locator and must not change whenever any locator changes.
- A communication session is linked to the ID and must not disconnect when the locator changes.

In addition to the ITU we add further requirements:

- An ID must be able to address any kind of entity, not only physical hosts.
- Every communication peer can generate the ID of its communication partner out of a human readable and memorable string.
- The ID is globally unique, but it must be possible to issue temporary IDs.
- The registration process for new IDs must be easy.
- IDs must be suitable for DHT storage.

While some of these aspects mainly affect the design of a Future Internet Architecture based on a locator/ID split, some issues are directly related with the naming of IDs.

T	Type	Input to Hash(name)	Ext 1	Ext 2
1	static host	plain text domain name	hash of local hostname	service
2	non-static host	global prefix assigned by provider	hash of local hostname	service
3, 4	content	plain text content name	child content	version number
5	person	first + last name	random	communication channel

Table I: Generic contents of UID fields corresponding to different types

B. Generalized Identifier

As IDs are used in the transport layer protocol to determine the endpoint of a communication, we cannot avoid using fixed-length bit strings to realize packet headers of constant size. In combination with DHTs, which also require fixed-length bit strings, the usage of a hashing function is obvious. In contrast to other approaches, which compose the ID of one hash value only, we split the ID in several predetermined fields whose purposes are known to all entities.

In the following, we introduce a generalized scheme how to compose global unique IDs (UID) for any entity and give concrete examples how to name hosts, content and persons. Our scheme allows storing all these IDs in the same mapping database and is yet flexible enough to support different databases for different types of IDs.

Figure 1 shows the generalized structure of an ID, which is composed of a region prefix (RP) according to HiiMap and an UID. The UID consists of a type field (T), the hash value of a human friendly name for the entity to be identified as well as two extension fields (Ext 1 and Ext 2). The UID is stored in the mapping system of a specific region, denoted by the RP.

The type field T denotes to which type of entity the UID belongs to. T allocates the most significant bits (MSB) in the UID, which allows to map different ID types to different databases in the mapping system. As some entities require more complex entries in the mapping system, it may be desirable to use different databases that are optimized for the needs of a specific entity. We suggest using 128 bits for the UID, whereby 4 bits are used to determine the type, 76 bits are assigned for the hash value, 32 bits for Ext 1 and 16 bits for Ext 2. In the following, we show realizations for applying UIDs to different types: host, content and persons. Table I gives an overview how the UID is composed according to the type of entity. Each part is described in detail in the following subsections. Note that our scheme is not limited to these types, but can easily be extended.

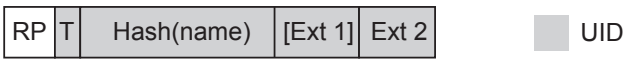

Figure 1: Identifier UID with regional prefix RP

C. Identifiers for Hosts

IDs for hosts are the most common use case today and DNS is used to resolve hostnames to IP addresses in order to access a specific machine. The hostname, or FQDN (full qualified domain name), which specifies the exact position in the tree hierarchy of the DNS, can be roughly compared to the ID in a locator/ID separated Internet architecture. However, the FQDN is solely used in the application layer and is not present in any lower layer network protocol.

Similar to today's hostnames, we introduce a hierarchy to our UIDs. However, contrary to FQDNs, our scheme is limited to two hierarchical levels: a global part and a local part. While the global part is used to identify a whole domain, e.g. a company or an institute at a university, the local part is used to identify single machines within this domain. Note that the term *domain* does not refer to a domain like in today's DNS hierarchy. A domain in our solution has a flat hierarchy and simply defines an authority for one or more hosts. We differentiate between two different types of host UIDs and give an overview in Table II:

1) Static Host Identifier: Static host UIDs are never changing IDs of type T=1 that can be generated by hashing a human readable hostname. Their main purpose is for companies or private persons that want to have a registered UID that is always assigned to their host or hosts.

Hash: The domain name part of the plain text hostname is used to generate the hash field of the UID. An example can be *mycompany.org* as domain name.

Ext 1: The hash value of the local host name is used to generate this field. A local hostname is unique inside a specific domain. An example for a local hostname could be *pc-work-003*. That way, together with the domain name, a host in the Internet is unambiguously identified.

Ext 2: The Ext 2 field is used to identify a specific service on the host. It can be compared to today's TCP or UDP ports. An application or service listening for connections must register its Ext 2 value at the network stack in order to receive data. Some values for Ext 2 are predetermined, e.g., for file transfer or remote administration, others can be chosen freely. However, specifying a value in Ext 2 is not necessary when requesting the locator for a specific host from the mapping system and can therefore be set to zero. As the host is precisely identified with the global and local UID part, it is not necessary to store identifiers for each service of

T	Type	Input to Hash(name)	Ext 1	Ext 2
1	static host	plain text domain name, e.g., *mycompany.org*	hash of local hostname, e.g., *pc-work-003*	0 used for requesting the locator from the mapping system $1..n-1$ standardized Ext 2 values for $n-1$ common applications and services $n..2^{16}$ further values for proprietary or open use
2	non-static host	global prefix assigned by provider, e.g., *provider12345*	as above	as above

Table II: Contents of UID fields for hosts

the host as they would all point to the same locator. Instead, Ext 2 is set to zero in the UID when querying the mapping system and filled with the specific service identifier when actually accessing the node.

For privacy reasons it is possible not to publish the UID for a private host in the global mapping system but only in a local database. For a single point of contact it is possible to use an UID with Ext 1 set to zero, which points to, e.g., a login server, router or load balancer that forwards incoming requests to designated internal hosts.

Note that the host has to update its mapping entry pointing to new locator(s) upon any locator change.

2) Non-static Host Identifier: Contrary to static host IDs and the basic idea of never changing UIDs there will always be the need for non-static host UIDs, i.e., IDs that do not have to be registered, that are assigned to a host for a specific time, and that are returned to the issuer if no longer needed. We assign the type value T=2 to these class of UIDs. An example can be a private household with a DSL, cable or dial-up Internet connection and a few hosts connected through a router. Each host needs its own, distinct UID to make connections with other hosts in the Internet. However, it does not need to have a registered, never changing UID if no permanent accessibility is needed.

Hash: The global part is assigned during the login process to the router or middlebox that provides Internet access to the other hosts. It can be compared to the advertisement of an IPv6 prefix. The global part, e.g., the hash of *provider12345*, is valid as long as the customer has a contract with its provider. A new global part is assigned if the customer changes its provider. Yet, the transfer of a global UID part between different providers should be possible. In order to assign non-static UIDs to customers, each provider holds a pool of global UID parts. The mapping entry for a specific non-static host UID is generated by the corresponding host immediately after assignment and whenever its locator changes. However, each host with no static UID assigned must proactively request a non-static host UID, either by its provider or router and middlebox, respectively. Note that the global part of a non-static host UID does not necessarily consist out of the hash value of a plaintext string. It depends on the provider if he uses hash values or just consecutive numbers out of a contiguous pool.

Ext 1: Like with static UIDs, the local part of a non-static UID is generated from the local hostname of a machine. Therefore, a name for each host is mandatory.

Ext 2: Identical to the Ext 2 field used for static host UIDs.

D. Identifiers for Content

As the focus of the users in the Internet is shifting from accessing specific nodes to accessing information and content, different approaches towards a content-centric network have been made as shown in Chapter II. By applying the idea of information models, like the NetInf approach, to our naming scheme, each content, which can be, e.g., a webpage or an audio or video file, gets its own distinct UID. Hereby, the UID does not point to the data object itself but to the information model of the content that has a further description and metadata stored. We use two different types for content UIDs: T=3 and T=4, whereas T=3 is used for content that is not subjected to public change, and T=4 which is used for content that is free to public changes.

Hash: For generating the UID of a content we have to use a meaningful name that can describe the corresponding content or information. While this is indeed quite a difficult task, possible solutions can be, e.g., the name of a well-known newspaper like *nytimes*, which refers to the front page of the New York Times online version. Similar, the name of an artist could refer to an information object where albums or movies are linked.

In our proposal, this plaintext name is used as input to a known hash function to generate the hash part of the UID. As the spelling of the content description is not always exactly known, we suggest a lookup mechanism that can cope with minor spelling mistakes in Section VI.

Ext 1: This field is optional and can be used to access some more specific parts of the content or information that

T	Type	Input to Hash(name)	Ext 1 (optional)	Ext 2 (optional)	(Pointer to)
3, 4	content	meaningful content name or description, e.g., *nytimes*	**Child content or information:** • hash of child content name, e.g., *sports* • number/ID: e.g. *DOI* • value 0: request toplevel description of content object, i.e., list of possible values with short description	**Version of content:** 0 (when querying mapping) 0 (when requesting content) $1..2^{16}$ (when requesting content)	mapping entry of content object current or actual version some older versions

Table III: Contents of UID fields for content

is directly related with the main object. This can be, e.g., one specific article from a newspaper site or a sub-page that deals with a specific topic like sports or politics. Other examples are specific albums or pieces of music from an artist. Ext 1 can help to avoid downloading a maybe bigger object description of the main content to gather the desired information. Another benefit is that each child object has its own locator and therefore can be stored on different locations while still being accessible through its parent UID. This is not possible today as, e.g., the URL of a newspaper article is directly coupled with the host storing the information. Ext 1 is created by hashing the human readable name for the detailed description or just by using consecutive numbers. The latter is for example useful to assign a specific article of a newspaper. It can be compared to the Digital Object Identifier (DOI) in digital libraries [19]. In order to access a specific content object which Ext 1 value is not known, the user can access the top-level description of this content object by setting Ext 1 to zero. This description then includes lists with all possible values for Ext 1 with a short description.

Ext 2: This field can be used to access a specific version of the desired content or information. Like in a versioning system, the Ext 2 field allows the user to easily access any earlier version and the changes made to the information. The content description is obtained by setting Ext 2 to zero where different values for Ext 2 are listed. Of course, each new version needs an update of the mapping entry including potential new locators. Note that we do not create own mapping entries for each Ext 2 value. Instead, available values for Ext 2 are listed in the mapping entry that is identifier by the hash part together with Ext 1. Like with host addressing, Ext 2 is always set to zero when querying the mapping system and filled with the desired value when actually requesting the content.

Table III summarizes the purposes of the UID fields for content objects. Unlike with host addressing, we cannot simply connect to a locator returned by the mapping system. As the information object is a description of content or information, the requesting application or user has to evaluate the information object and select the desired representation according to the user's needs. Thus, the network stack will not evaluate the data received from the mapping system for a content UID query but forward it to the corresponding application. The detailed interaction of applications requesting content UIDs and the network stack will be discussed in Chapter V.

Note that in case a name, e.g., *nytimes*, refers to both a host (company) as well as content (webpage), the type field is used to differentiate whether a locator (for type host) or a content description (for type content) is returned.

E. Identifiers for Persons

With the emergence of social networks, Internet-capable devices, Voice-over-IP (VoIP), etc., the need for personal IDs arose, as the *person* itself is moving in the focus of interest. Whenever somebody wants to contact a specific person he is interested in the communication with that person and does not want to care about the device, e.g., which phone or computer the person is currently using for communication. However, the user must have the possibility to choose the desired communications channel. That can be an email, a phone call, a message on a mailbox, a chat with an instant messenger or a message in a social network and so on. Furthermore, the personal UID can be used to make digital signatures which is necessary for contracts whatsoever. It has the assigned type T=5.

Hash: The main part of a person's UID consists of a hash value calculated from the person's full name, i.e., first name plus last name. As many people have the same first and last name, the hash value is ambiguous and we need further information to distinguish between different persons.

Ext 1: For this purpose we use a pseudo-random number for Ext 1 when initially generating a person's UID [20]. This initial generation is not done by the person itself in order to avoid multiple usage of the same Ext 1 value, but is issued by a federal authority and valid for lifetime . It can be compared to the number of a passport or the social security number, which do not change over the persons lifetime.

Ext 2: This field is used to specify the communication channel to the corresponding person and has a set of predetermined values, e.g., for email, VoIP, or instant messaging.

T	Type	Input to Hash(name)	Ext 1 (optional)	Ext 2 (optional)	(Pointer to)
5	person	first + last name	(random) number issued by federal authority	0 request personal profile of person	mapping entry of personal profile
				1 request locator of machine	machine user is working on
				$2..n$ predetermined standart values	mail server, VoIP phone or VoIP provider, chat server, etc.
				$n..2^{16}$ open for future extensions	
			0	0	access directory

Table IV: Contents of UID fields for persons

Note, there are still enough unused values for future needs. According to each Ext 2 value, different locators can be stored in the mapping system. For example, the Ext 2 value referring to the VoIP account can point to the locator of a VoIP provider or directly to a VoIP phone, the value referring to the mailbox can point to a mail server. The mapping entry for Ext 2 set to zero includes the person's full name and, depending on the person's privacy settings, further details about the person like birth date or current residential address. We refer to this as a*personal profile* according to [21]. Ext 2 set to one is used to get the locator of the machine the person is currently working on if the corresponding person agreed to publish this information. Thereby, the communication channel can be signaled in a higher layer or by using the machines service identifier Ext 2 when contacting the corresponding host. Note that the exposure of the device's locator the person is currently using can be abused to generate a movement profile. We suggest to use a privacy service like showed in [22] to avoid these issues.

However, to contact a specific person, not only the person's name but also Ext 1 must be known. There are two possibilities: First, the initiating person knows the correct UID of its communication partner because they have exchanged it in any way, e.g., with business cards. Second, the holder of a personal UID can agree to be indexed in a directory that is accessible through a personal UID with Ext 1 and Ext 2 set to zero. It stores the personal profiles of all persons that have the same name including their random Ext 1 values plus additional information to distinguish persons. According to the persons privacy settings, further details about city or street can be published, like in a traditional printed or online phone book. Table IV summarizes the necessary fields for personal UIDs.

In order to avoid abuse of personal UIDs, the usage of PKI mechanisms that guarantee the presence of the corresponding private key is mandatory for every transaction.

IV. Identifier registration and assignment

As each UID is globally unique by definition, it must be ensured that only one entity at a time has a specific UID assigned. It must be further prevented that any entity is hijacking an UID for malicious purposes.

A. Static host UIDs

The registration process for a static host UID can be compared to today's domain names and is depicted in Figure 2. Whenever a user wants to register a new static host UID for his hosts, he has to register a UID for each host together with a public key of this host at his local registry or local NIC (Network Information Center) with a `REGISTER_UID`. If the UID is still available, the NIC creates the initial mapping entry in the mapping system with `CREATE_UID` and stores the hosts public key. Updates are only allowed if the `UPDATE_MAPPING` message is signed with the correct private key, thus avoiding the UID to be hijacked.

Whenever the host connects to the Internet, it first requests a valid locator at the designated router with `REQ_LOCATOR`. Together with the locator assignment, the locator for the mapping system is delivered (`ASSIGN_LOCATOR`) and the host can update its mapping entry. Whenever the host's access point changes, it will request a new locator and immediately update the mapping entry with the new valid locator.

The UID at the node is configured via a system file like */etc/hostname* and */etc/domainname*. The owner of a host must proclaim changing the key-pair of a node at the mapping region. Note that for static UIDs *every* new UID must be registered at the registry. It is not possible to register only the domain part and to freely choose values for Ext 1, as each static UID needs its own key pair.

B. Non-static host UIDs

The purpose of non-static UIDs is that they do not need a registration process, as their prefixes are dynamically assigned by a provider and therefore belong to that provider. However, it must be possible for hosts with non-static UIDs

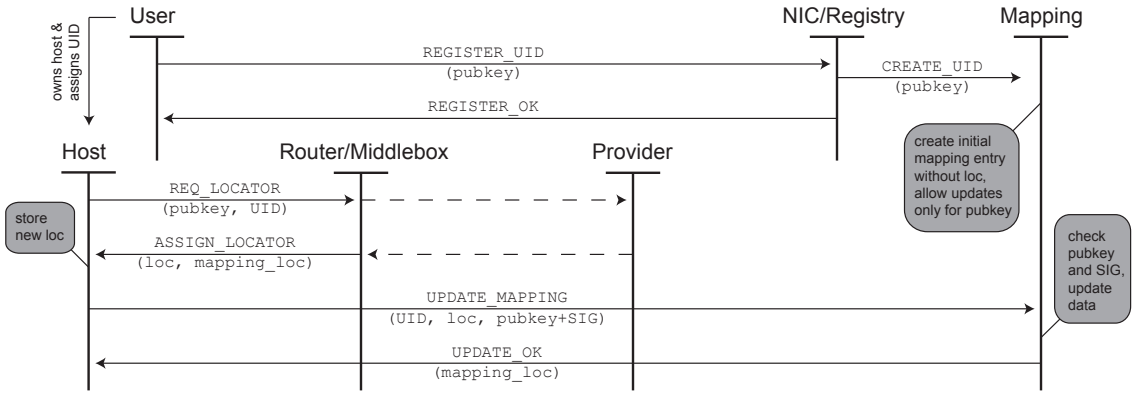

Figure 2: Signal flow chart for static UID registration and assignment

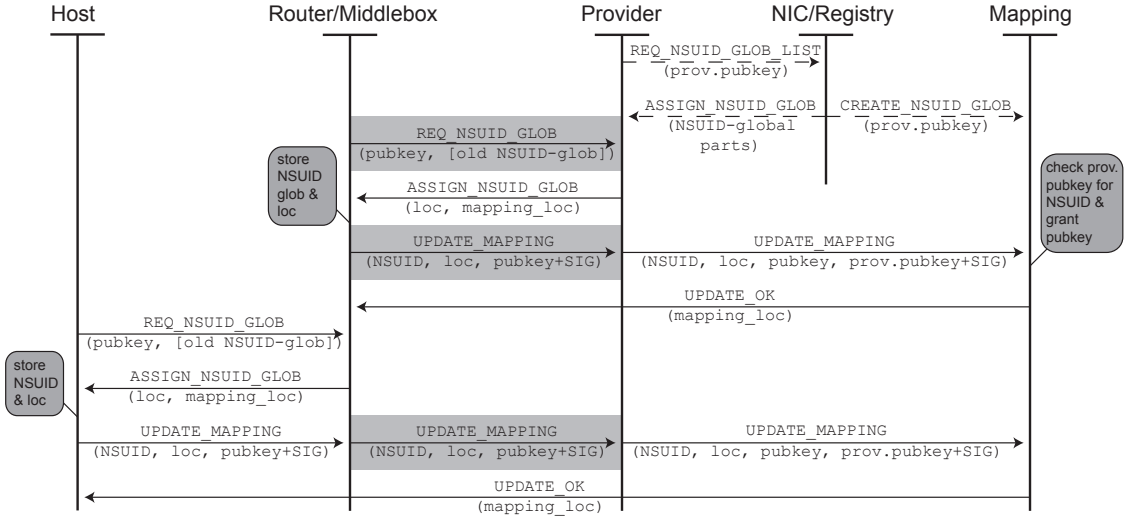

Figure 3: Signal flow chart for non-static UID registration and assignment

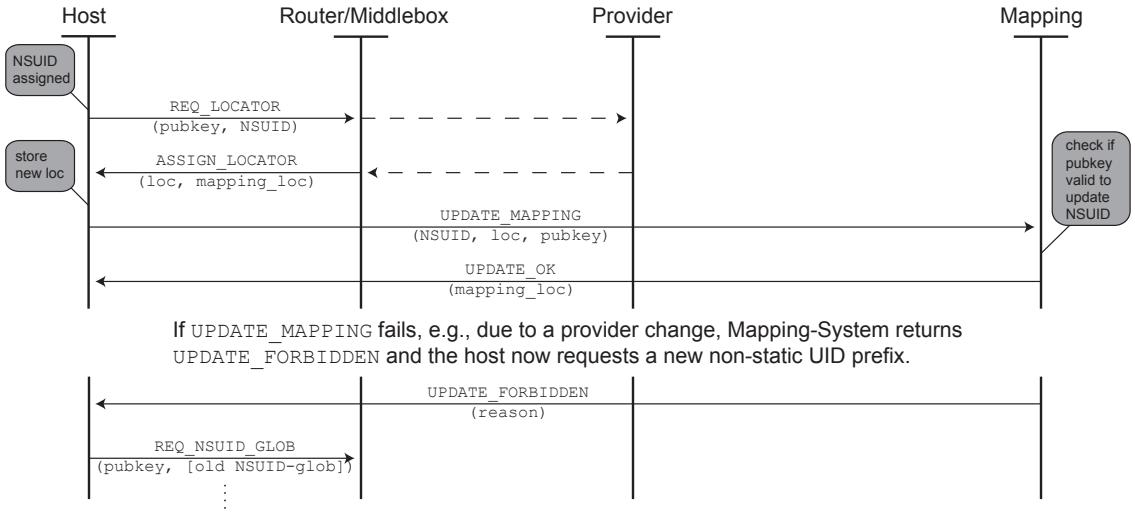

Figure 4: Signal flow chart for non-static UID roaming

to change their mapping entries due to roaming although they have not been individually registered.

In order to distribute global parts for non-static UIDs, the provider has to request its pool of non-static UID global parts at the local registry together with its public key. The NIC creates the initial mapping entries for these global non-static UID parts, whereby changes are only allowed by the provider. Figure 3 depicts the complete registration process.

Whenever a home router or middle-box without assigned global UID part tries to connect to the Internet, it requests a global non-static UID part by sending a `REQ_NSUID_GLOB` message to the provider. Note that this message must be sent over a providers line (messages on gray background), as the provider must verify if the device is allowed to request a global UID part. He can do this by, e.g., checking login credentials. Similar to a DHCP request, the middle-box can request a global UID part that has been assigned before (`old NSUID_GLOB`). After the assignment of a global UID part, a valid locator for the device, and the locator of the mapping system (`ASSIGN_NSUID_GLOB`), the middle-box updates its mapping entry. Again, this process must be done over the providers line, as only the provider can update the initial mapping entry, because it has the corresponding private key. Therefore, `mapping_loc` in the `ASSIGN_NSUID_GLOB` message points to a mapping relay at the provider, which signs `UPDATE_MAPPING` with the providers private key and instructs the mapping system to allow updates directly from the device in the future. The mapping system responds with `UPDATE_OK` and a new `mapping_loc`, which directly points to the mapping system. However, `mapping_loc` that points to the providers mapping relay is still stored in the middlebox, as it is needed for internal hosts to initially update their mapping entries.

Internal hosts are sending their request for a global UID part to the middle-box, which assigns the same global UID part received by the provider, together with a locator and the locator of the mapping relay. The update of the mapping entry is the same as for the middle-box.

In case of roaming (Figure 4), the host usually already has an assigned non-static UID. Therefore, it does not request a global UID part at its new access point, but just requests a new locator with `REQ_LOCATOR`. It receives a new locator with `ASSIGN_LOCATOR`, which also includes the locator of the mapping system to update its mapping entry. However, if the update at the mapping system fails, e.g., because the global UID part has become invalid, the mapping system returns `UPDATE_FORBIDDEN` and a reason for the reject. Depending on that reason the host will then start to request a new global UID prefix. Such an error can occur if, e.g., the contract between the provider and the customer that owns the host has expired.

C. Content UIDs

The procedure for content UIDs is basically the same like for static host UIDs. The content creator has to initially register the hash part of the UID at the mapping system. However, it does not need to register each single content object that is provided. Then the content provider can freely create new content that only differs in Ext 1 and Ext 2. By doing so, the content creator always uses the same public-key pair for all content objects, and the mapping system requires a valid signature upon changing any mapping entry.

While this procedure is sufficient for professional or commercial content that is managed by one authority, a different approach is necessary for content that is free to public changes, like articles in Wikipedia. Here, everybody is allowed to create a new version of the corresponding content that differs in Ext 2. However, no new mapping entry is generated for every new Ext 2 value, but the new Ext 2 value including the corresponding locator is added to the existing mapping entry. The changing person has to sign this new entry with its own private key and the mapping system must grant changes not only to the initial creator of the mapping entry. In order to differentiate between content that may be changed by anyone, we have introduced two different type values 3 and 4 that simplify the setting of the correct permissions for each mapping entry.

D. Personal UIDs

Unlike with UIDs for hosts or content, UIDs for persons are assigned by an authority of the state. As the personal UID can be used to make transactions and legal contracts, it has to be guaranteed that the UID cannot be abused. Furthermore, it has to be guaranteed that values for Ext 1 are unambiguous. That would not be the case if everybody would generate its own random value for Ext 1. Thus, during the registration process, an authority assigns a free Ext 1 value and creates the mapping entry for the person requesting a UID. The authority furthermore checks the presence of a valid key-pair and deposits the public key together with the mapping entry in the mapping system. Then, the person can update and create any entry for Ext 2 in its mapping entry on its own. Changing the key pair must always be accomplished through the issuing authority. Like before, Ext 2 is set to zero when querying for the mapping entry of a specific person and solely used when establishing a communication. The persons UID, together with the cryptographic key pair, maybe stored on a chip that is embedded in the ID card. This feature is already supported by many governments today.

V. Stack Interaction

As the current Internet architecture does not support any kind of locator/identifier separation, major changes in the network stack are necessary to enable this new network paradigm. Based on the HiiMap approach and in order to avoid a hierarchical and inflexible design like in the

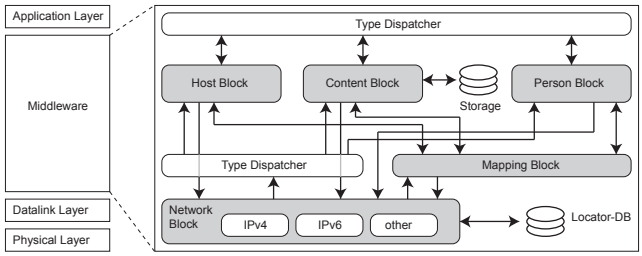

Figure 5: New network stack with middleware and interacting building blocks

Open Systems Interconnection (OSI) reference model, we suggest to use building blocks that can freely interact in the so-called middleware. Applications in the upper layer can choose to which types they want to connect and the middleware provides the mapping lookup, handles the connection setup and keeps track of changing locators. Note that this architecture requires no changes in the datalink and physical layer. Figure 5 shows the schematic model of the middleware. Building blocks are shown in gray and the *Type Dispatcher* forwards incoming and outgoing data packets to the responsible building block. Note this middleware is necessary on all devices connected to the Internet. However, during a migration phase, it is possible to gradually introduce the specific building blocks. Nevertheless, the *Network* and *Host Block* are mandatory from the beginning to allow at least host-to-host communication.

A. Network Block

The task of the network block is to select adequate protocols for inter-networking, according to the network configuration. These can be for example IPv4, IPv6 or future protocols with new addressing schemes and routing techniques. The network block is solely working on locator-level, and prepends the necessary routing and forwarding header to the data packets. Furthermore, the building block is responsible for the locator values of all connected network interfaces and also requests non-static global UID parts if necessary. It keeps track of all UIDs and the corresponding locator values that are being used in the locator database. It notifies all these communication partners upon a local locator change, thus ensuring a interruption-free data transfer. Moreover, it notifies the mapping module to update the entry in the mapping system.

B. Mapping Block

This building block is responsible for keeping the host's mapping entries up to date. Locator changes of the network interfaces are reported from the network block and updated in the mapping system. Despite that, receives requests for UID resolution from other building blocks and forwards these requests to the mapping system. It returns the corresponding locators, content descriptions or personal profiles to the requested building block. The mapping block uses the `mapping_loc` value that is returned at the locator assignment process for the communication with the mapping system.

C. Host Block

Every application that wants to be accessible through a specific service identifier (Ext 2 in host UIDs) must register this identifier at the host block. Incoming connections for UIDs with type "host" are dispatched at the type dispatcher and forwarded to the host block. If any application is listening at the corresponding service identifier, the data is forwarded. Furthermore, the host block handles outgoing connections to other hosts and accepts connection requests to host UIDs. It selects one locator value returned by the mapping block and forwards the data packet to the network block. The host block also keeps track of locator changes at its communicating peers.

D. Content Block

The tasks of the content block contain the reception of any kind of content, including the evaluation of the information object with the content description returned by the mapping system, as well as the provision of own content that is stored locally.

1) Content reception: The request of any application that wants to receive a specific content object is dispatched to the content block. According to the desired UID, the content block requests the information object from the mapping system. Furthermore the application has the possibility to set some filter rules regarding the requested content, e.g., minimal/maximal bit-rate for audio and video files, or resolution for pictures. Mobile devices can benefit from that filtering option, as the application has the possibility to request only the smallest available version of the content. If the filtered content description still has more than one source available, the final choice has to be taken by the application or user respectively.

2) Content provision: The other way round, the content block is also responsible for the provision of content objects that are stored on the local machine. Thereby, content that shall be publicly available is registered at the content block including the corresponding content UID. The content block notifies the mapping block about new available content, which in turn registers the content UIDs at the mapping system. Requests for content UIDs are dispatched to the content block, which delivers the desired content to the inquirer. Note that the content block must have access to the storage location of all registered content objects.

E. Person Block

The person block is responsible for any kind of personal communication between users. Every application dealing with chat capabilities, instant messaging, email, and voice-

or video communication registers itself at this building block. According to the connected applications and settings of the user, the person block, in conjunction with the mapping block, modifies the mapping entry of the persons UID in the mapping system. The personal UID is configured via a card reader and the persons ID card that also stores the persons private key. Incoming connections are immediately forwarded to the corresponding registered application. If the user agrees to continuously update the locator(s) of his machine, the person block takes care of this action.

VI. Lookup mechanism

The idea of our naming scheme for IDs is based on the fact that each UID can be generated out of a known plain text string with a known hash function and without an additional naming system like DNS. However, as the main part of any UID consists of a hash function, the desired entity can only be found if the plain text string that builds the UID is exactly known and no spelling mistake or typing error occurred. This is particularly a problem for querying UIDs, where phonetical identical search strings can be spelled in different ways, e.g, "color" and "colour". To overcome this drawback, we suggest a lookup mechanism that is based on n-grams in addition to the pure UID lookup. Harding [23] and Pitler [24] showed that the usage of n-gram based queries can significantly improve the detection of spelling errors in noun phrases and word tokens. Note that this feature must be supported and implemented in the mapping system as well as in the mapping block of each querying machine [25].

A. n-gram generation

Although DHTs only support exact-match lookups, it is possible to use n-grams to perform substring and similarity searches. Hereby, each plaintext string is split up into substrings of length *n*, which are called *n-grams*. The hash value of each n-gram together with the corresponding complete plaintext string is then stored as key/value pair in the DHT [25].

A typical value for n is two or three. As an example with $n = 3$, the content name P set to `nytimes` is split up into $I = 5$ trigrams h_i with $i = 1, ..., I$: `nyt`, `yti`, `tim`, `ime`, `mes`. In addition to the actual mapping entry indexed by the UID, the hash value $H(h_i)$ of each n-gram h_i is inserted in the mapping system together with the corresponding plain text name P. Thereby, the mapping entry for an n-gram consists of the tuple $\langle H(h_i); plaintext\ string \rangle$ [20]. Although these tuples are stored in the same mapping system like the UID, we suggest using a different database within the mapping system for performance reasons. Whenever the entity changes its location, no updates of the n-grams are necessary, as they do not contain any locator information but only the entities' plaintext name.

B. Querying UIDs

Whenever querying the mapping system for a specific UID, the first step in the lookup process is using the pre-calculated (or already known) UID as query parameter. Only if the mapping system is not able to find a mapping entry to the corresponding UID, e.g., because of a spelling mistake, the n-gram lookup is executed. It is up to the user or application if an n-gram based query request is initiated. An n-gram based query can also be initiated if the query does not return the desired result.

In doing so, the second step is to calculate the corresponding n-grams out of the plaintext string and query the mapping system for each n-gram. The mapping system sorts all matching n-grams according to the frequency of the plaintext string and returns the list to the user. With high probability, the desired plaintext has a high rank in the returned list. By further correlating the input string with each returned plaintext string, e.g., by using the Levenshtein distance, the result is even more precise [26].

As the results returned by an n-gram query must be further evaluated, the mapping block in the middleware will forward that data to the corresponding building block. This block can already select the best matching result, or forwards the possible choices directly to the application, which is responsible for correct representation to the user that takes the final decision. However, although this feature is similar to Google's "Did you mean...?", the mechanism is not suitable to handle complex queries with semantically coherent terms as Google can do. However, it can help to significantly improve the quality of returned search results and thus the quality of experience to the user, as phonetical similar words can be found despite different spelling.

VII. Conclusion

In this work, we presented a new naming scheme for IDs in locator/ID separated Future Internet Architectures based on the HiiMap proposal. The generalized ID scheme is suitable for basically addressing any kind of entity and we gave examples for hosts, content and persons. As each UID can be computed out of a human readable plaintext string, an additional naming system like DNS is not necessary any more. Due to the extendible type field, we have the possibility to assign ID-types also for, e.g., mobile phones, sensors, or even cars or abstract services that provide any functionality to a user. As IDs are independent from locators, a communication session is not interrupted upon an access point change. We showed instructions how new UIDs for each type are assigned and registered and which changes in the network stack are necessary in order to enable the addressing scheme proposed in this work. Furthermore, by introducing an n-gram based extended lookup mechanism we are able to cope with spelling errors and typing mistakes.

ACKNOWLEDGMENT

This work was funded by the Federal Ministry of Education and Research of the Federal Republic of Germany (Project ID 01BK0807). The authors alone are responsible for the content of the paper.

REFERENCES

[1] C. Spleiß and G. Kunzmann, "A Naming scheme for Identifiers in a Locator/Identifier-Split Internet Architecture," in *ICN 2011, Proceedings of 10th International Conference on Networks*, Sint Maarten, Netherland, January 2011, pp. 57–62.

[2] A. Afanasyev, N. Tilley, B. Longstaff, and L. Zhang, "BGP routing table: Trends and challenges," in *Proc. of the 12th Youth Technological Conference High Technologies and Intellectual Systems"*, Moscow, Russia, April 2010.

[3] X. Meng, Z. Xu, B. Zhang, G. Huston, S. Lu, and L. Zhang, "IPv4 address allocation and the BGP routing table evolution," *ACM SIGCOMM Computer Communication Review*, vol. 35, no. 1, p. 80, 2005.

[4] ISC, "The ISC Domain Survey," http://www.isc.org/solutions/survey, Internet System Consortium, 2010.

[5] B. Quoitin, L. Iannone, C. de Launois, and O. Bonaventure, "Evaluating the benefits of the locator/identifier separation," in *Proceedings of 2nd ACM/IEEE International Workshop on Mobility in the evolving Internet Architecture*, 2007, pp. 1–6.

[6] V. Jacobson, D. Smetters, J. Thornton, M. Plass, N. Briggs, and R. Braynard, "Networking named content," in *Proceedings of the 5th international conference on Emerging networking experiments and technologies*. ACM, 2009, pp. 1–12.

[7] D. Cheriton and M. Gritter, "TRIAD: A new next-generation Internet architecture," Tech. Rep., 2000. [Online]. Available: http://www.dsg.stanford.edu/triad

[8] C. Dannewitz, "NetInf: An Information-Centric Design for the Future Internet," *Proceedings of 3rd GI/ITG KuVS Workshop on The Future Internet*, May 2009.

[9] O. Hanka, G. Kunzmann, C. Spleiß, and J. Eberspächer, "HiiMap: Hierarchical Internet Mapping Architecture," in *1st International Conference on Future Information Networks*, October 2009.

[10] M. Menth, M. Hartmann, and D. Klein, "Global locator, local locator, and identifier split (GLI-split)," *University of Würzburg, Institute of Computer Science, Technical Report*, 2010.

[11] A. Feldmann, L. Cittadini, W. Mühlbauer, R. Bush, and O. Maennel, "HAIR: Hierarchical Architecture for Internet Routing," in *Proceedings of the 2009 Workshop on Re-architecting the Internet*. ACM, 2009, pp. 43–48.

[12] J. Pan, S. Paul, R. Jain, and M. Bowman, "MILSA: A Mobility and Multihoming Supporting Identifier Locator Split Architecture for Naming in the Next Generation Internet," in *IEEE GLOBECOM*, 2008, pp. 1–6.

[13] M. Menth, M. Hartmann, and M. Höfling, "FIRMS: a mapping system for future internet routing," *IEEE Journal on Selected Areas in Communications*, vol. 28, no. 8, pp. 1326–1331, 2010.

[14] D. Farinacci, V. Fuller, D. Oran, D. Meyer, and S. Brim, "Locator/ID separation protocol (LISP)," Draft, 2010. [Online]. Available: http://tools.ietf.org/html/draft-ietf-lisp-07

[15] R. Moskowitz and P. Nikander, "Host identity protocol (HIP) architecture," RFC 4423, Tech. Rep., May 2006.

[16] V. Kafle and M. Inoue, "HIMALIS: Heterogeneity Inclusion and Mobility Adaptation through Locator ID Separation in New Generation Network," *IEICE TRANSACTIONS on Communications*, vol. 93, no. 3, pp. 478–489, 2010.

[17] O. Hanka, C. Spleiß, G. Kunzmann, and J. Eberspächer, "A novel DHT-based Network Architecture for the Next Generation Internet," in *Proceedings of the 8th International Conference on Networks*, Cancun, Mexico, March 2009.

[18] ITU, "Draft Recommendation ITU-T Y.2015: General requirements for ID/locator separation in NGN," International Telecommunication Union, 2009.

[19] N. Paskin, "Digital Object Identifier (DOI®) System," *Encyclopedia of library and information sciences*, pp. 1586–1592, 2010.

[20] G. Kunzmann, "Performance Analysis and Optimized Operation of Structured Overlay Networks," Dissertation, Technische Universität München, 2009.

[21] C. Spleiß and G. Kunzmann, "Decentralized supplementary services for Voice-over-IP telephony," in *Proceedings of the 13th open European summer school and IFIP TC6. 6 conference on Dependable and adaptable networks and services*, 2007, pp. 62–69.

[22] O. Hanka, "A Privacy Service for Locator/Identifier-Split Architectures Based on Mobile IP Mechanisms," in *Proceedings of 2nd International Conference on Advances in Future Internet*, Venice, Italy, July 2010.

[23] S. Harding, W. Croft, and C. Weir, "Probabilistic retrieval of OCR degraded text using N-grams," *Research and advanced technology for digital libraries*, pp. 345–359, 1997.

[24] E. Pitler, S. Bergsma, D. Lin, and K. Church, "Using web-scale N-grams to improve base NP parsing performance," in *Proceedings of the 23rd International Conference on Computational Linguistics*. Association for Computational Linguistics, 2010, pp. 886–894.

[25] M. Harren, J. Hellerstein, R. Huebsch, B. Loo, S. Shenker, and I. Stoica, "Complex queries in DHT-based peer-to-peer networks," *Peer-to-Peer Systems*, pp. 242–250, 2002.

[26] L. Mangu, E. Brill, and A. Stolcke, "Finding consensus among words: Lattice-based word error minimization," in *6th European Conference on Speech Communication and Technology*, 1999.

Data Portability Using WebComposition/Data Grid Service

Olexiy Chudnovskyy, Stefan Wild, Hendrik Gebhardt and Martin Gaedke
Faculty of Computer Science
Chemnitz University of Technology
Chemnitz, Germany
olexiy.chudnovskyy@informatik.tu-chemnitz.de, stefan.wild@informatik.tu-chemnitz.de,
hendrik.gebhardt@informatik.tu-chemnitz.de, martin.gaedke@informatik.tu-chemnitz.de

***Abstract* - Web 2.0 has become the ubiquitous platform for publishing, sharing and linking of content. While users are empowered to create and manage their data, the latter is still scattered and controlled by distributed and heterogeneous Web applications. The data is usually stored in internal silos and is only partially exposed through platform APIs. As a result, the reuse possibilities, fine-grained access control and maintenance of distributed data pieces become a time-consuming and costly activity, if feasible at all. In this paper, we introduce our approach to develop Web applications based on the principles of data portability, decentralization and user-defined access control. Our main contributions are (1) a novel Web service-based storage solution for the read-write Web, combined with (2) a security framework for WebID authentication and authorization based on WebAccessControl lists (WAC), and (3) a corresponding systematic approach for Web applications development based on loosely-coupled and user-controlled storage solutions.**

Keywords - WebComposition; Data Engineering; REST; WebID; Web 2.0

I. INTRODUCTION

In the age of Web 2.0, it is the users, who produce a huge amount of data by contributing to blogs, wikis, discussion boards, feedback channels or social networks [1]. The numerous platforms on the Web facilitate this activity by providing sophisticated publishing tools, data sharing functionalities and environments for collaborative work. While users enjoy the simplicity and comfort given by these applications, they usually face the problem that "their" created data belongs to the service provider and is basically out of their control [2]. The data can be accessed, edited or linked only in the ways that were originally foreseen by platform developers. For example, user profile stored in Facebook cannot be synchronized with profiles in other social networks. An uploaded picture cannot be linked with others using new relationship types like "same event" or "same place". The accessibility and portability of data depends on APIs, usage terms and conditions of different platforms. In summary, users are not only hindered in their sharing, linking and publishing possibilities – they do not really control their data anymore. Application developers, on their side, are hindered in consuming the published content, either because existing platforms expose it in a restricted way or do not include enough metadata required for the particular domain.

There is a clear need for storage solutions, frameworks and tools, which would support both: users and developers in their corresponding activities. In this paper, we present our approach to decouple Web applications from storage solutions and analyze the resulting consequences regarding data access, security and application development process. In particular, the contributions of the paper are the following:

1. A novel Web-service-based storage solution enabling data publishing and linking based on the principles of Linked Data. The solution acts as a portable Web component, which provides repository functionality and also enables aggregation of access to distributed heterogeneous data sources.
2. A security framework for user-defined access control based on the social graph defined by the Friend-of-a-Friend ontology (FOAF). We apply the WebID concept [3] and WebAccessControl protocol [4] to design a reusable module for client authentication and authorization.
3. A systematic approach to develop storage-decoupled Web applications. We adapt modeling techniques and tools used to design application data domain and provide guidance in adoption of the presented architecture.

The rest of the paper is organized as follows: in Section II, we discuss the concept of application-independent storage solution and introduce our implementation based on WebComposition/Data Grid Service (DGS). In Section III, we describe the utilized authentication and authorization approach based on WebID and WebAccessControl protocol. Section IV discusses how traditional development process should be adapted to take the new architecture into account. In Section V, we illustrate our approach by implementing a simple photo management application. Finally, in Section VI, we summarize the paper and give an outlook into our further work.

II. WEBCOMPOSITION/DATA GRID SERVICE AND READ-WRITE WEB

WebComposition/Data Grid Service [5] is the core element of the fourth generation of the WebComposition approach [6]. It acts as an extensible storage and gateway

solution, which focuses on data integration and uniform access to distributed data sources [7][1]. Data Grid Service has several interfaces and applies the Linked Data principles to identify, describe and link internal resources.

Following, we describe the data model, interface and functional capabilities of Data Grid Service. Furthermore, we give an insight into its internal architecture and show extension possibilities.

A. *Data model*

The data space managed by WebComposition/Data Grid Service consists of a set of typed lists. Lists can have different nature and provide different operations on items inside. For example, the core modules of Data Grid Service implement operations on XML resources, which can be retrieved, updated, removed or linked with others. Extension modules implement handling of binary collections or structured access to external data sources, like relational databases, user tweets, blog entries, documents etc. In all cases, Data Grid Service provides a common view on distributed data spaces and exposes them to clients as lists of entries (Figure 1).

Beside typed lists, the so called virtual resources can be defined within Data Grid Service. While they do not offer any storage or gateway functionality, they are used to provide additional access and manipulation methods on the top of the existing lists. An example of such a virtual resource is the one enabling further representations of existing resources. With the help of transformation stylesheets like XSLT, the default XML representation of resources can be extended with RSS, Atom, JSON and other formats.

Collections and items within Data Grid Service can be annotated to provide additional information about their origin, author, creation time etc. Annotations also give information about resource behavior and defined access control. Furthermore, a repository-wide metadata is available, where specification of the lists and general service description are stored.

B. *Interface*

WebComposition/Data Grid Service follows the REST architectural style to identify and manipulate internal resources. All resources within Data Grid Service are identified using URIs. Some pre-defined URI-patterns are used to access metadata (`{resource URI}/meta`) or access control lists (`{service URI}/meta/acl`) etc. The standard HTTP methods GET, POST, PUT and DELETE are used to read, create, update and delete resources. Depending on the configuration, some of the resources may require authorization before executing the operations.

REST/HTTP interface provides several advantages for multi-tier architectures, where data storage is decoupled from services and third-party applications. First, it is simple, complies with well-proven Web standards and can be seamlessly integrated into the Web landscape [8]. Second, REST/HTTP enables loose coupling between services and clients. A REST-based storage solution can evolve independently and extend its functionality without breaking the contract. And finally, based on the HTTP protocol, third-party providers are empowered to provide additional services on the top of user-controlled storage solutions, e.g., caching, security etc.

Though REST/HTTP is the main and most suitable interface for decoupled data storages, also SOAP and XML/RPC endpoints are foreseen to support business

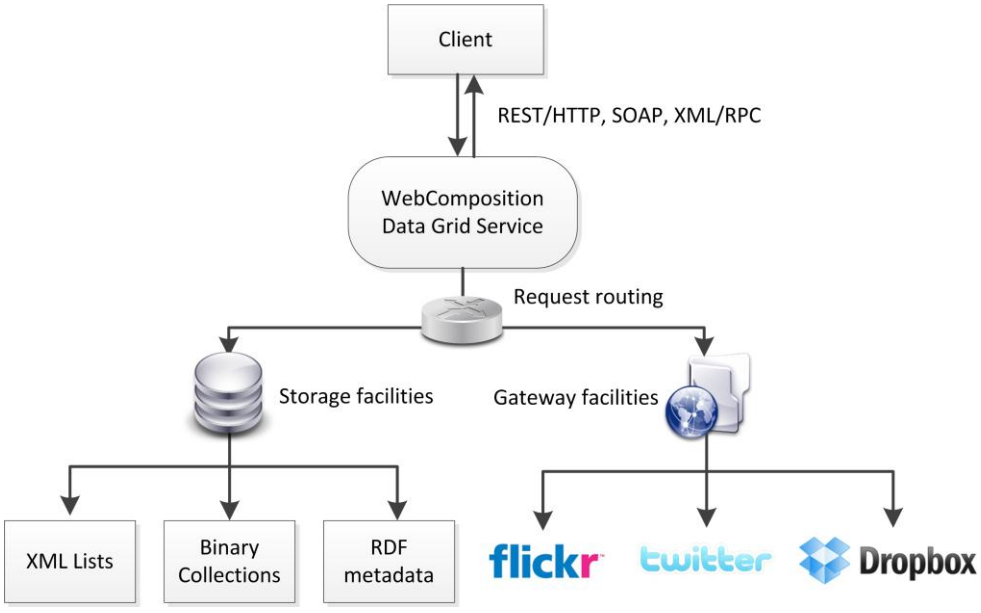

Figure 1 WebComposition/Data Grid Service

scenarios and proprietary clients.

C. Core functionality

Data Grid Service provides several core modules, which enable management of both structured and unstructured content. In particular, users have the possibility to upload their profile information, multimedia content and define fine-grained access control on the resources. Applications can utilize the published content and create new collections within Data Grid Service to store their internal data. In both cases, user is the only owner of the data and can extend, update or revoke access at any time.

Following example creates a new XML list within Data Grid Service:

```
<collection xmlns="http://www.w3.org/2007/app"
xmlns:atom="http://www.w3.org/2005/Atom"
xmlns:dgs="http://www.webcomposition.net/2008/02/dgs/">
<atom:title>profiles</atom:title>

<dgs:dataspaceengines>

  <dgs:dataspaceengine
dgs:type="http://.../CacheDataSpaceEngine" />

  <dgs:dataspaceengine
dgs:type="http://.../XmlDataSpaceEngine"
xmlns:dx="http://.../XmlDataSpaceEngine/" >
    <dx:primarykey>
          childnode1/childnode2/id
    </dx:primarykey>
  </dgs:dataspaceengine>

</dgs:dataspaceengines>
</collection>
```

Listing 1. Definition of new list

The request contains title and type of the list as well as list-specific metadata. The type of the list defines the module(s) (or so called Data Space Engines), which would be responsible for the HTTP requests on corresponding URI namespace. The metadata may describe the behavior of the module more precisely, e.g., security policies or configuration settings. After creation it is accessible under `{list URI}/meta` URI and can be retrieved in Resource Description Framework (RDF) format.

A single request can be processed by several modules, called within a pipeline, where output of one module is passed to another one as input. This enables pre- and post-processing of incoming data - for example, for caching or transformation purposes.

Following, we present the core modules of Data Grid Service, their capabilities and usage examples.

1) Data Space Engine for XML lists

XML is a well-understood and easy to use format, which is commonly used in distributed Web applications. Data Grid Service uses XML as the main data representation format, mainly because of many existing tools and standards to validate, to transform and to navigate through XML-based documents. The module "Data Space Engine for XML lists" provides a broad range of functionality to deal with XML resources. Though basic Create/Read/Update/Delete (CRUD) functionality is supported for all kinds of XML resources, the main purpose of this Data Space Engine is to manage so called XML lists, i.e., XML resources, which have a flat tree structure and contain a list of semantically and structurally related XML items. The list model can be applied for many kinds of Web applications, e.g., blogs, content-management-systems, e-commerce applications etc.

By restricting the view from general XML resources to XML lists, the Data Space Engine can provide additional functionality specific to lists. In particular, the module provides operations to identify and retrieve single items, append new or delete existing ones. Furthermore, XML Schema and XSLT stylesheets can be applied to perform data validation or create alternative representation formats of list items.

XML lists are useful to represent user- or application-produced content and make it available to others. For example, a simple address book service can model user contacts as XML items and publish them as a collection in the user's storage solution:

```
$curl https://dgs.example.org/contacts

<contacts>
 <contact id="1001">
    <firstname>John</firstname>
    <secondname>Smith</secondname>
    <address>
        <street>2nd Avenue</street>
        <number>54</number>
        <zip>11124</zip>
        <city>New York</city>
    </address>
</contact>
 <contact id="1002">
   …
 <contact id="1003">
   …
</contacts>
```

Listing 2. Example of XML list

Following the RESTful architecture style the contact items can be retrieved, created, updated or deleted using the corresponding HTTP methods. However, as soon as user enables write access to further service providers, she might want to restrict the structure of the list items or check them for valid content. For this purpose, an XML schema is defined for the list, causing validation of document after all incoming write requests.

The XML representation of list items empowers application developers to use complex data structures while describing their data. An XML item is a tree, whose depth can vary depending on the application needs. In order to update nested items, like the contact address in the example above (so called partial update operation), the module provides the concept of URI templates, which enables dynamic assignment of URIs to item pieces. URI templates are configuration settings for the module and can be defined at run-time by adding dedicated metadata to the XML list:

```
POST /contacts/meta HTTP/1.1
Host: user1.datagridservice.example.org
Content-Type: text/n3
…

@prefix meta:
<http://www.webcomposition.net/2008/02/dgs/meta/>.
<http://dgs.example.org/contacts>
meta:urlTemplate
[
meta:url "contacts/{value}/address";
meta:xPath "/contacts[@id='{value}']/address"
].
```

Listing 3. Definition of URI template

An URI template consists of 2 patterns - the one to be applied on URIs of incoming requests and the one to be applied on XML list to select the desired subnodes. As a result, the nested XML nodes get associated with the URI `https://dgs.example.org/contacts/{contact-id}/address` and can be manipulated the same way as the parent ones.

Furthermore, arbitrary views on the XML lists can be defined in the same way. The expressiveness of view definitions, however, is limited to the expressiveness of XPath query language. As an example, one could define a view on all persons living in a particular city and retrieve them using a dedicated URI pattern.

Many data-driven applications rely on entity-relationship models while designing and managing their data domain. To model the "JOIN" operations on resources, i.e., to retrieve all the related items for some given one, XML Data Space Engine introduces the concept of relationships. Relationships define the connection between two items in terms of some given ontology. The relationships are described using RDF and belong to the list metadata. The definitions can be consumed by service clients in order to discover and apply additional retrieval functions. A relationship is configured through 3 obligatory and 3 optional attributes:

- *Parent*: A URI of the list to act as a primary list, e.g., `http://dgs.example.org/contacts`
- *Child*: A URI of the list to act as a subordinate list, e.g., `http://dgs.example.org/pictures`
- *Predicate*: A URI of RDF predicate to act as a foreign key, defining a connection between primary and secondary list items, e.g., `http://xmlns.com/foaf/0.1/img`.

A relationship configured using the above attributes enables processing of the following URI pattern:

```
http://{service_host}/{parent_list_name}/
{parent_item_id}/{child_list_name}
```

As a result, only those items from child list are retrieved, which are linked to the parent list item using the relationship-specific predicate. For example, a GET request on `http://dgs.example.org/contacts/1001/pictures` would return picture descriptions associated with the contact 1001.

A POST HTTP request on the same URI is used to add new items to the child list linking them simultaneously with the specified parent list item.

To create an inverse link from child item to the parent one, each time a direct connection is established, the optional *Inverse Predicate* attribute is used, containing a URI of RDF predicate for the inverse relationship. A corresponding RDF statement is then automatically added to the child list metadata, acting as a foreign key to the parent list item. The same URI patterns can be applied to retrieve, create or delete parent items linked to some given child item.

If many relationships between the same parent and child list should be modeled (1:n, n:m), optional *Parent* and *Child Aliases* can be defined to match the incoming request with corresponding relationship definition. Listing 4 gives an insight into the list metadata and shows the internal relationship representation.

In summary, Data Space Engine for XML lists enables users and third-party applications to store and manage their data using simple and flexible data model. It follows principles of RESTful architecture style and can be applied to implement a broad range of data-centric Web applications.

2) Data Space Engine for binary resources

Current Web 2.0 applications require from storage solutions efficient support of both structured data and arbitrary binary content. Some of the data is supposed to be public, while access to other has to be limited. This fact elicits corresponding requirements on resource publishing and access control functions.

In case of public resources, users should be assisted by annotating and linking resources among each other's. For example, metadata available in the uploaded media files has to be exposed in machine-processable format, so that Web crawlers and service clients can utilize it to implement more intelligent search and discovery functions. Providing links to related content is also essential to enable third-party applications to explore the user space.

```
$curl https://dgs.example.org/contacts/meta

<rdf:Description rdf:about="http://dgs.example.org/contacts/">

<dc:creator rdf:resource="http://dgs.example.org/profiles/27" />

<dc:title rdf:resource="Contacts" />

...
</rdf:Description>

...
<rdf:Description
   rdf:about="http://dgs.example.org/contacts/meta/relationships/68">
   <dm:source rdf:resource="http://dgs.example.org/contacts" />
   <dm:target rdf:resource="http://dgs.example.org/pictures" />
   <dm:predicate rdf:resource="http://xmlns.com/foaf/0.1/img" />
   <dm:inverse-predicate rdf:resource="http://purl.org/dc/elements/1.1/creator" />
</rdf:Description>
```

Listing 4. Example of the list metadata

The Data Space Engine for binary resources implements basic CRUD functionality for the arbitrary content. Binary resources are grouped within collections. The GET request on the collection returns an Atom feed with basic descriptions of collection items. To create and update new resources, corresponding HTTP requests with MIME type specification are used. The annotations and metadata of the items is accessible using the `{resource URI}/meta` URI pattern. Third-party applications can consume this metadata according to their needs or update it using corresponding HTTP methods.

We have implemented automatic metadata extraction for some common MIME types (based on pre-defined mapping between file attributes and RDF properties). For example, just after uploading an MP3 file to Data Grid Service, the artist and album information are immediately published using RDF and terms coming from Music Ontology [9]:

```
<rdf:Description
rdf:about="http://dgs.example.org/music/31"
xmlns:ns0="http://purl.org/ontology/mo/">
    <dc:date>2005</dc:date>
    <dc:description>
    Amazon.com Song ID: 20206547
    </dc:description>
    <dc:title>Von Hier An Blind</dc:title>
    <ns0:album>Von Hier An Blind</ns0:album>
    <ns1:genre>Pop</ns1:genre>
    <ns2:singer>Wir Sind Helden</ns2:singer>
<rdf:Description>
```

Listing 5. Example of metadata extraction

Similarly, Data Space Engine analyzes PDF, JPEG and MPEG file encodings in order to extract the metadata and expose it using the RDF and common vocabularies.

3) Data Space Engine for XSLT transformations

Data Space Engine for XSLT transformations enables definition of further representations of XML resources. For example, contact details from the example above can be exposed as JSON list, CSV table, Atom Feed etc. For this purpose, new resources are added to Data Grid Service and configured to be processed by the XSLT module. The configuration contains the specification of the XML resource to be transformed, the MIME type of the resulting resource and the XSLT stylesheet with the transformation algorithm (Listing 6).

The resource configuration and the stylesheet are considered as resource metadata and can be updated later to adapt the module behavior.

4) Data Space Engines for external services

The design of the architecture foresees extensibility possibilities by implementing the pre-defined module interface. As such, further data spaces, e.g., user blogs, tweets, activity streams or multimedia content scattered across different platforms can be embedded into Data Grid Service. Third-party applications may access this content according to policies defined by the user. We implemented gateways to Twitter, Dropbox and Flickr as examples to let users and applications discover the data in one place and consume it using one single unified REST/HTTP interface.

5) SPARQL Endpoint

All of the resources within Data Grid Service can be annotated using RDF metadata. To let service clients find resources they are interested in, we implemented a dedicated SPARQL endpoint, accessible at the dedicated `{service URI}/meta/sparql` URI. By sending compliant queries, clients can search for resources within Data Grid Service's data spaces.

III. SECURITY FRAMEWORK FOR PORTABLE DATA

While consolidating the user data in one place, storage solution has to guarantee its safety, security and privacy. As such, only authorized clients are allowed to read and modify the data. The choice of the authorization mechanism is crucial in order to address different types of clients and the peculiarities of the Web domain. Our goal is to enable easy-to-understand but still fine-grained access control, where rules are expressive enough to take users' social graph and relationships in account.

```
<collection xmlns="http://www.w3.org/2007/app" xmlns:atom="http://www.w3.org/2005/Atom"
xmlns:dgs="http://www.webcomposition.net/2008/02/dgs/">
<atom:title>contacts-atom</atom:title>
  <dgs:dataspaceengines>
       <dgs:dataspaceengine dgs:type="http://.../XsltDataSpaceEngine"
                        xmlns:dsexslt="http://.../XsltDataSpaceEngine/" >
         <dsexslt:source>contacts</dsexslt:soruce>
         <dsexslt:mimetype>application/atom+xml</dsexslt:mimetype>
         <dsexslt:stylesheet>
           <xsl:stylesheet version="1.0" xmlns:xsl="http://www.w3.org/1999/XSL/Transform">
                <xsl:output method="xml" indent="yes"/>
                <xsl:template match="contacts">
                     <feed xmlns="http://www.w3.org/2005/Atom">
                        ...
                       <xsl:apply-templates select="contact"/>
                     </feed>
                </xsl:template>
              ...
           </xsl:stylesheet>
         </dsexslt:stylesheet>
       </dgs:dataspaceengine>
  </dgs:dataspaceengines>
</collection>
```

Listing 6. Data Space Engine for XSLT resources

The emerging WebID standard is a promising technology, which enables single sign-on, flexible identity management and complex authorization rules. Every agent (person, organization or application) is identified using a URI, which refers to the user's corresponding public profile.

To prove that the WebID belongs to the user, he creates a self-signed certificate, which points to his WebID. The X.509 v3 certificate standard allows the binding of additional identities to the subject of the certificate through a subject alternative names extension [10]. Furthermore, the user extends the profile document with certificate's public key. When the user agent tries to access a protected resource, the server asks for the client certificate, before connection is established. The possession of the certificate is verified during the SSL/TLS handshake and is the first step in the authentication process. As the second step, server compares the certificates' public key with the one stored in the WebID profile. The comparison is usually performed with a dedicated SPARQL query on user profile URI. The successful authentication process proves that the FOAF resource belongs to the user and provides additional information to the server about the user profile, social relationships, etc.

After the user has been authenticated, the storage solution determines if he has enough rights to access the requested resource. We utilize WebAccessControl mechanism, which complements WebID with access control lists based on Semantic Web technologies [11]. WebAccessControl is inspired by authorization mechanisms implemented in many file systems. It allows placing a set of permissions on a specific resource for a user or a group, identified by an URI:

```
@prefix acl: <http://www.w3.org/ns/auth/acl>.

[acl:accessTo <http://example.org/img.png>;
acl:mode acl:Read, acl:agentClass foaf:Agent].

[acl:accessTo <http://example.org/img.png>;
acl:mode acl:Read, acl:Write; acl:agent
<http://example.org/foaf#joe>]
```

Listing 7. Example of WebAccessControl list

The access control lists are RDF resources with precisely defined semantics. The example above is an N3 serialization of the access control list and protects the resource with URI `http://example.org/img.png`. The first rule makes the resource readable for every user agent with a valid WebID, while the second one grants write permissions to the identity `http://example.org/foaf#joe`.

Currently, WebAccessControl foresees four different access modes. In addition to the mentioned *Read* and *Write* modes, one is able to grant *Append* and *Control* permissions. *Append* is a restricted *Write* permission, where one is only allowed to attach new data to the resource (e.g., in case of log files). If the agent should be capable of modifying the access control list, the mode *Control* has to be set.

The WAC list is stored centralized within Data Grid Service. The owner of the storage solution has write permissions to the access control list and can define the access rules for other agents.

The presented authentication and authorization mechanism is implemented as independent and reusable module. It is invoked before the request reaches the Data Space Engines-pipeline and checks if user has a valid WebID and was assigned required permissions to access the resource. The check of the permissions is done by mapping the HTTP methods GET, POST, PUT and DELETE to its

equivalents in the WebAccessControl ontology. We mapped HTTP GET to *acl:Read*, HTTP PUT as well as HTTP DELETE to *acl:Write* and HTTP POST to *acl:Append*.

IV. Development of Web Applications using WebComposition/Data Grid Service

We envision that users will be the only owners of their data independently from its usage by third-party Web applications. Applications have to provide an added value on the top of the data and not limiting its reuse, sharing and linking possibilities. To deal with the fact, that the storage is decoupled from the application and is shared on the Web between many applications, the classical development processes, models and supporting tools for data-driven Web applications should be adapted (Figure 2).

To illustrate our approach, we consider a simple Web application for management of photo albums. The platform should enable users to manage their pictures, tag them, and assign them to photo albums. Users should be able to browse albums of others and search for pictures using different criteria. Though there are plenty of platforms on the Web providing similar functionality, all of them require users to put the data inside one single platform.

Following, we show how to engineer Web applications, which do not host the user data in internal data silos, but utilize user-controlled storage solutions.

The development of the Web application starts with a requirements engineering step. We analyze user needs and capture their requirements regarding functional and non-functional aspects of the Web application. For our example scenario, we refine and write down the functionality described above. Apart other possible non-functional requirements, we focus on the fact that the data should be stored decentralized in user-controlled storage solutions.

After the requirements are captured, we analyze the structure of business domain in order to produce the conceptual model of the application. The result of the analysis is usually an Entity-Relationship (ER) model, which captures different types of objects from the business world, their attributes and relationships. To meet the peculiarities of storage-decoupled Web applications, we extend the model and distinguish between local and global entities, which should indicate that entity belongs either to user or to the application data space. For example, pictures and albums are entities, which belong to user and should be maintained within his data space, while platform-wide categories and tags can be managed centralized in the application data space.

Entity-Relationship model is an important artifact, which is used, among others, for database schema specification and automated code generation. In our approach, we apply distributed and Web-based storage solutions instead of monolithic databases, so that the data can be shared between different application and services. To enable independent (and possibly serendipitous) data consumption, not only structure, but also semantics of the data should be unified and captured within dedicated models. To meet this need, we extend the Entity-Relationship model with semantic-specific aspects. In particular, we capture the semantics of entities, attributes and relationships using common ontologies and vocabularies. For example, entity *Picture* can be annotated with `http://xmlns.com/foaf/0.1/Image` concept, and its attributes with `http://purl.org/dc/elements/1.1/description` and `http://purl.org/dc/terms/created` coming from FOAF and Dublin Core profiles respectively (Figure 3).

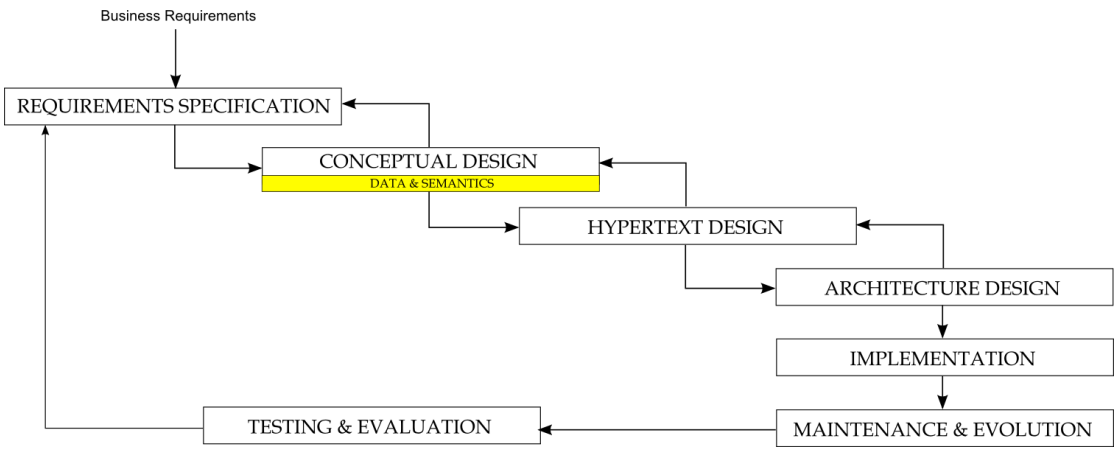

Figure 2 Development cycle of storage-decoupled Web applications

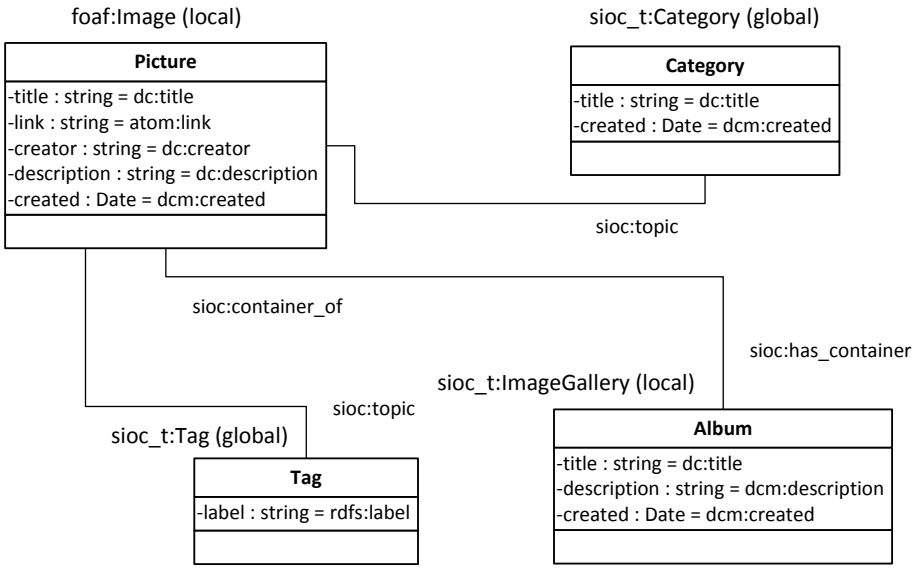

Figure 3 Extended Entity-Relationship model of the example application

By putting more semantics into the ER-model, the storage solutions are empowered to deliver data after the Linked Data principles. Due to the common model and explicit semantics, data stored within user storage can be discovered and reused more efficiently. In our approach, we transform the ER-model into a set of Data Grid Service XML lists, corresponding XML Schemas, relationship objects and transformation stylesheets to produce RDF/XML representation of application content.

The rest of the development process can be executed as in traditional Web applications. In the hypertext design phase we specify data display, input and navigation functions. We apply faceted navigation pattern for category browsing, thumbnails for album and set-based navigation for picture lists [12]. It is desirable that the storage solution has built-in support for these operations, so they perform more efficiently.

Our application has common three-tier architecture [13], where data server is a distributed layer of user-controlled storage solutions. We use WebComposition/Data Grid Service due to its broad support for both binary resources and structured XML content. Annotation and data transformation enables publishing of data after Linked Data principles, so that data created by one application can be seamlessly consumed by another. Furthermore we utilize ASP.NET Model-View-Controller framework [14] as "glue" between user interface and data storage.

To implement application authentication and authorization mechanisms, we apply the same approach as with securing user storages. The security modules described in Section III can be reused and integrated into the application. As a result, users authenticate themselves by presenting a certificate with WebID field, so that application can reuse the profile information stored within the FOAF file. To consume application services, user has to prepare a data space within his storage solution to be used by application and store it in his configuration settings. Corresponding access control rules have to be defined within the storage solution, so that application, identified by WebID as well, can access the required data.

The resulting application is tested and installed in the target environment (Figure 4).

Figure 4 The photo album management application based on Data Grid Service [15]

We notice that the application is loosely coupled with its data storage, which means, that their evolution can take place independently and hence be performed more efficiently.

V. RELATED WORK

Many distributed data storage solutions focusing on scalability, availability and simple data modeling appeared during the last couple of years [16]. In this section, the important contributions in this field of research are analyzed and discussed. The presented approaches can be roughly separated into the three areas: structured, distributed data storages, classical NoSQL databases and publishing tools.

DataWiki [17] is a platform to manage structured data using basic CRUDS operations enabled via a RESTful Web service API. To access or modify data sets stored in the DataWiki, mashups and forms can be created and hosted independently from the platform. All documents available in the DataWiki can be exported as Atom feeds. Information sharing with other systems is supported through built-in federation. Although the DataWiki approach separates data and representation from each other, it lacks of coping with large unstructured data sets, e.g., binary large objects.

data.fm [18] is an open source Web service implementing the REST architectural style, which supports common request methods and various response and media types, e.g., JSON, RSS or Atom, to perform access and retrieval operations on structured data sets available in the internal cloud storage. A graphical Web interface offers a convenient way to create new storage clouds with optionally restricting their access via ACLs. Data within the internal storage is organized in files and directories, which can be adapted from privileged persons through the API or GUI. Like the DataWiki platform, data.fm is well-suited for managing structured data, but compared to our approach, data.fm cannot apply post-transformations, e.g., via XSLT, to responses.

Another representative in this context is OpenLink Virtuoso [19], a structured data cluster providing certain virtualization layers to handle heterogeneous data sources and processing components. Principally, the software consists of an object-relational database accessible through specific database engines, integrated web and application servers. OpenLink Virtuoso provides a rich set of interfaces, e.g., SOAP, REST, XML-RPC, and SPARQL, to query for the uniquely identifiable elements stored in the database. In addition, the software supports protocols, e.g., Blogger and Atom 1.0, to publish data in a suitable form as well as components to interact with many types of application, e.g., wikis and address books. Although Virtuoso is a powerful tool to manage different types of data, it is complicated in installation and administration, which may become a crucial factor in success of user-controlled storage solutions. In contrast, Data Grid Service doesn't require a database or triple store in the backend and is installed using a simple installation wizard.

Similar to our approach, NoSQL solutions can be used as Web components, which support essential CRUD functionality for structured and unstructured data. For example, Apache CouchDB [20] stores schema-free data as name-value pairs, which are accessible through a RESTful Web interface. Like CouchDB, Amazon S3 [21] can be used to store unstructured data and access it through a REST/HTTP or a SOAP interface. In addition, Amazon S3 is often accompanied with Amazon SimpleDB [22], which allows saving structured, but schema-free data sets. NoSQL databases are designed to provide a scalable, fault-tolerant and flexible storage solution for schema-free data. Though NoSQL solutions can handle frequently changing document structures and new file types, the qualified data validation via document schemas is missing. Furthermore, they do not provide any support to centralize user data and enable fine-grained access control.

In conjunction with classic relational databases, the so called publishing tools can be applied to expose user data as Linked Data. The publishing tools automate the tasks of interpreting incoming HTTP requests, dereferencing URIs and converting the data in a proper form. One representative of this kind of tool is D2R server [23], which selectively transforms data from a legacy data source into RDF. The transformation is performed based on the parameters specified in the request. Currently, D2R server supports HTML and RDF browsers and provides a SPARQL endpoint to query the database content. Similar to D2R server, Triplify [24] converts the results of queries transmitted to a relational database into RDF, JSON or Linked Data. However, Triplify just executes a transformation of the output, i.e., queries have to be written in SQL. In contrast to our approach, the publishing tools only perform non-modifying operations on the legacy databases and do not aim to integrate other Web-based data sources.

Though the presented tools facilitate tasks of storing, publishing and linking data on the Web, they do not provide an integrated solution. Data wikis are flexible tools enabling collaborative data acquisition but cannot deal with unstructured data and distributed data spaces. NoSQL databases are scalable Web-based storage solutions, but are not so extensible in the sense of integrating additional data spaces. Finally, publishing tools support users gathering Linked Data out of legacy database. However, they cannot be used to propagate modifications back to the same storage.

VI. Conclusion and Outlook

In this paper, we have presented our approach to engineer Web applications based on user-controlled storage solutions. The separation of applications and data brings many advantages both for the end-user but also for application developers. Users have the full control of their data - they can specify what data should be public or private, what parts the third-party applications are allowed to access and how this data is linked to other resources. Application developers profit from the accessibility of user data and can deliver novel services more easily. Finally, the evolution of storage solutions and applications can take place independently and therefore less coordination and synchronization effort is needed.

We introduced WebComposition/Data Grid Service, a loosely coupled persistence and gateway layer for Web applications. We have shown how Data Grid Service can be used as a Web-based storage solution and how users can define access control using WebAccessControl lists. We presented reusable authentication and authorization modules based on the emerging WebID standard. Finally, we described a systematic approach to develop Web applications for decoupled storage solutions and illustrated it using a simple photo album management example.

In future, we expect the growth of the flexibility and functionality of user-controlled storages. As more and more applications access and change user data, the need to keep provenance information emerges. To meet this demand we

are going to add management and monitoring functionality for Data Grid Service. Especially quota assignment and event logging are important issues to protect the users' data space from malicious use and attacks. To link and synchronize data between storage solutions of different users, we are building dedicated publish/subscribe infrastructures. Finally, we are planning to extend the vocabulary for specification of access control lists and implement additional authorization rules based on the social graph of the user.

VII. REFERENCES

[1] O. Chudnovskyy and M. Gaedke, "Development of Web 2.0 Applications using WebComposition / Data Grid Service," in *The Second International Conferences on Advanced Service Computing (Service Computation 2010)*, 2010, pp. 55-61.

[2] T. Berners-Lee, "Socially aware cloud storage - Design Issues," 2009. [Online]. Available: http://www.w3.org/DesignIssues/CloudStorage.html. [Accessed: 23-Jan-2012].

[3] Manu Sporny, Toby Inkster, Henry Story, Bruno Harbulot, and Reto Bachmann-Gmür, "WebID 1.0 - Web Identification and Discovery," *W3C Editor's Draft*, 2011. [Online]. Available: http://www.w3.org/2005/Incubator/webid/spec/. [Accessed: 23-Jan-2012].

[4] W3C, "WebAccessControl - W3C Wiki," 2011. [Online]. Available: http://www.w3.org/wiki/WebAccessControl. [Accessed: 23-Jan-2012].

[5] O. Chudnovskyy and M. Gaedke, "WebComposition/Data Grid Service v1.0: Demo." [Online]. Available: https://vsr.informatik.tu-chemnitz.de/demo/datagridservice/. [Accessed: 23-Jan-2012].

[6] H.-W. Gellersen, R. Wicke, and M. Gaedke, "WebComposition: An Object-Oriented Support System for the Web Engineering Lifecycle," in *Electronic Proc. of The 6th International WWW Conference*, 1997.

[7] M. Gaedke, D. Härtzer, and A. Heil, "WebComposition / DGS : Dynamic Service Components for Web 2.0 Development," in *Proceedings of the 6th International Conference on Advances in Mobile Computing and Multimedia*, 2008, no. c, pp. 2-5.

[8] E. Wilde and M. Gaedke, "Web Engineering Revisited.," in *BCS Int. Acad. Conf.*, 2008, pp. 41-50.

[9] Y. Raimond, F. Giasson, K. Jacobson, G. Fazekas, T. Gängler, and S. Reinhardt, "Music Ontology Specification," 2010. [Online]. Available: http://musicontology.com/. [Accessed: 23-Jan-2012].

[10] D. Solo, R. Housley, and W. Ford, "RFC 2459: Internet X.509 Public Key Infrastructure Certificate and CRL Profile," 1999. [Online]. Available: http://tools.ietf.org/html/rfc2459. [Accessed: 23-Jan-2012].

[11] J. Hollenbach, J. Presbrey, and T. Berners-lee, "Using RDF Metadata To Enable Access Control on the Social Semantic Web," in *Proceedings of the Workshop on Collaborative Construction, Management and Linking of Structured Knowledge (CK2009)*, 2009.

[12] G. Rossi, D. Schwabe, and F. Lyardet, "Improving Web information systems with navigational patterns," *Computer Networks*, vol. 31, no. 11-16, pp. 1667-1678, May 1999.

[13] M. Fowler, *Patterns of Enterprise Application Architecture*. Addison-Wesley Professional, 2002, p. 560.

[14] E. Gamma, R. Helm, R. Johnson, and J. Vlissides, *Design Patterns: Elements of Reusable Object-Oriented Software le*. Addison-Wesley, 1994.

[15] O. Chudnovskyy and M. Gaedke, "DGS Photogallery: Demo." [Online]. Available: https://vsr.informatik.tu-chemnitz.de/demo/photogallery. [Accessed: 23-Jan-2012].

[16] R. Cattell, "Scalable SQL and NoSQL data stores," *ACM SIGMOD Record*, vol. 39, no. 4, p. 12, May 2011.

[17] Google, "DataWiki," 2011. [Online]. Available: http://code.google.com/p/datawiki/. [Accessed: 23-Jan-2012].

[18] Data.fm, "Data Cloud," 2011. [Online]. Available: http://data.fm/. [Accessed: 23-Jan-2012].

[19] OpenLink Software, "Virtuoso Universal Server." [Online]. Available: http://virtuoso.openlinksw.com/. [Accessed: 23-Jan-2012].

[20] The Apache Software Foundation, "Apache CouchDB: The CouchDB Project," 2008. [Online]. Available: http://couchdb.apache.org/. [Accessed: 23-Jan-2012].

[21] Amazon, "Simple Storage Service (Amazon S3)." [Online]. Available: http://aws.amazon.com/de/s3/. [Accessed: 23-Jan-2012].

[22] Amazon, "SimpleDB." [Online]. Available: http://aws.amazon.com/de/simpledb/. [Accessed: 23-Jan-2012].

[23] C. Bizer and R. Cyganiak, "D2R server - publishing relational databases on the Semantic Web," in *Poster at the 5th International Semantic Web Conference (ISWC2006)*, 2006.

[24] S. Auer, S. Dietzold, J. Lehmann, S. Hellmann, and D. Aumueller, "Triplify – Light-Weight Linked Data Publication from Relational Databases," *Proceedings of the 18th international conference on World Wide Web*, pp. 621-630, 2009.

Towards Normalized Connection Elements in Industrial Automation

Dirk van der Linden[1], Herwig Mannaert[2], Wolfgang Kastner[3], Herbert Peremans[2]

[1]Artesis University College of Antwerp, Electromechanics Research Group, Belgium
dirk.vanderlinden@artesis.be
[2]University of Antwerp, Belgium
{herwig.mannaert, herbert.peremans}@ua.ac.be
[3]Vienna University of Technology, Automation Systems Group, Austria
k@auto.tuwien.ac.at

***Abstract*—There is a tendency to Web-enable automation control systems, with consequently the challenge to propagate and aggregate data and control over the Internet. While classical industrial controller systems are limited to a local network, Web-enabled systems can be coupled in a new dimension. However, this also introduces larger impacts of changes and combinatorial effects. The Normalized Systems theory was recently proposed with the explicit goal of keeping these impacts bounded. It can be applied from the production control level up to the Web-enabled interface. One of the key principles of the Normalized Systems theory is to enforce Separation of Concerns in a multi-technology environment. To this end, this paper introduces Normalized Connection Elements as a stable interface between PLC software and field devices. As a case in point, the IEC 61131-3 code design of an ISA88 Control Module following these principles is discussed.**

***Keywords*-Normalized Systems; Automation control software; IEC 61131-3; ISA88; OPC UA.**

I. INTRODUCTION

Meeting the requirements of a software project has always been one of the top priorities of software engineering. However, not rarely, after taking in service, or even during the development, customers come up with new requirements. Project managers try to satisfy these additional requirements accompanied with an extra cost to the customer. The estimation of these additional efforts, depending on the development progress of the project, is often not straightforward. Managers tend to focus on functional requirements, while experienced engineers know that non-functional requirements can sometimes cause more efforts and costs. Evolvability became one of the most desirable non-functional requirements in software development, but is hard to control. One of the most annoying problems automation engineers are confronted with is the fear to cause side-effects with an intervention [1]. They have often no clear view in how many places they have to adapt code to be consistent with the consequences of a change. Some development environments provide tools like cross references to address this, but the behavior of a development environment is vendor-dependent, although the programming languages are typically based on IEC 61131-3 [2].

The Normalized Systems theory has recently been proposed for engineering evolvable information systems [3]. This theory also has the potential to improve control software for the automation of production systems. In production control systems, the end user always has the right to a copy of the source code. However, it is seldom manageable to incrementally add changes to these systems, due to the same problems as we see in business information systems, such as undesired couplings, side-effects, combinatorial effects. Finding solutions for these problems includes several aspects. Some standards like ISA88 suggest the use of building blocks on the macro level. The Normalized Systems theory suggests how these building blocks should be coded on the micro level. Interfacing between modules can be supported with the OPC UA standard.

Just like transaction support software and decision support software systems, production automation systems also have a tendency to evolve to integrated systems. Tracking and tracing production data is not only improving the business, in some cases it is also required by law (in particular in the food and pharmacy sectors). Due to the scope of totally integrated systems (combination of information systems and production systems), the amount of suitable single vendor systems is low or even non-existing. Large vendor companies may offer totally integrated solutions, but mostly these solutions are assembled from products with different history. For the engineer, this situation is very similar to a multi-vendor environment. Also, assembling software instead of programming is a challenge Doug McIlroy called for already decades ago [4]. Several guidelines, approaches, tools and techniques have been proposed that aim at assisting in achieving this goal. Unfortunately, none of these approaches have proven to be truly able to meet this challenge.

Globalization is bringing opportunities for companies who are focusing their target market on small niches, which are part of a totally integrated system. These products can expand single-vendor systems, or can become part of a multi-

vendor system. Moreover, strictly single-vendor systems are rather rare in modern industry. Sometimes they are built from scratch, but once improvements or expansions are needed, products of multiple vendors might bring solutions. Hence, over time single-vendor systems often evolve to multi-vendor systems. Each of these systems can be considered as a different technology. Isolating these technologies to prevent them exporting the impact of their internal changes into other technologies is the key contribution of this paper.

Minor changes, often optimizations or improvements of the original concept, occur shortly after taking-in-service. Major changes occur when new economical or technological requirements are introduced over time. As a consequence, software projects should not only satisfy the current requirements, but should also support future requirements [5].

The scope of changes in production control systems, or the impact of changes to related modules in a multi-vendor environment is typically smaller than in ERP (Enterprise Resource Planning) systems and large supply chain systems. However, there is a similarity in the problem of evolvability [3]. Since the possibilities of industrial communication increase, we anticipate to encounter similar problems to the ones in business information systems. The more the tendency of vertical integration (field devices up to ERP systems) increases, the more the impact of changes on the production level can increase. Since OPC UA (Open Product Connectivity - Unified Architecture) [6] enables Web-based communication between field controllers and all types of software platforms, over local networks or the Internet, the amount of combinatorial effects after a change can rise significantly (change propagation).

This paper introduces a proof of principle on how the software of an ISA88 Control Module [7] can be developed following the Normalized Systems theory. Some developers could recognize parts of this approach, because (as should be emphasized) each of the Normalized Systems theorems is not completely new, and some even relate to the heuristic knowledge of developers. However, formulating this knowledge as theorems that prevent combinatorial effects supports systematic identification of these combinatorial effects so that systems can be built to exhibit a minimum of these combinatorial effects [3]. The Normalized Systems theory allows the handling of a business flow of entities like orders, parts or products. For these process-oriented solutions, five patterns for evolvable software elements are defined [8]. In this paper, however, we focus on the control of a piece of physical equipment in an automated production system. The code of an ISA88 based Control Module is not process-oriented but equipment-oriented [1]. The focus of this code is not about how a product has to be made, but about how the equipment has to be controlled. Consequently, we need another type of programming language because of the nature of industrial controllers. Since the patterns for evolvable software elements are fundamental, we can use them as a base for IEC 61131-3 code. For this code, we concentrate on 3 patterns: Data Elements, Action Elements and Connection Elements. The Connection Elements can connect software entities in two directions: first, towards the physical process hardware, and second, towards higher level, non-IEC 61131-3 software modules. The first concerns IEC 61131-3 code, the second typically Web-enabled platform independent systems via an OPC UA interface. In this paper, we focus on the connection with process physical hardware. The possibility of OPC UA-based Connection Elements is crucial to enable upcoming larger automation systems, whose parts are connected via the Internet, and will be worked out in detail in future work. Such automation software entities should be able to evolve over time. This is a key requirement in the beginning age of decentralized energy generators and consumers prominently known as smart grid [9].

The remainder of this paper is structured as follows. In Section II, we discuss the Normalized Systems theory. In Section III, we give an overview of industrial standards on which industrial production Control Modules can be based. These standards include software modeling and design patterns, communication capabilities, and programming languages. In Section IV, an evolvable Control Module is introduced. In Section V, we discuss some changes and their evaluation. We tested the robustness of the Control Module against these changes in our industrial automation laboratory. In Section VI, we conclude and introduce suggestions for future research.

II. Normalized Systems

Adding small changes or extending an existing software system with new functionality often leads to an increase in architectural complexity and a decrease in software quality [10]. This is regarded as more expensive than developing the same functionality from scratch. This phenomenon is known as Lehman's law of increasing complexity [11], expressing the degradation of information systems' structure over time:

> *"As an evolving program is continually changed, its complexity, reflecting deteriorating structure, increases unless work is done to maintain or reduce it."*

To challenge this law, the Normalized Systems theory has recently been established [12]. Since the Normalized Systems theory takes *modularity* as basis, the principles are independent of any specific programming language, development environment or framework. Modularity implies that every module hides its internal details from its environment: another module does not need a white box view (i.e., analysis of the internal data and code) of the first module in order to be able to call and use this module. Hiding internal details is referred to as the *black box* principle. The user or caller of a black box module only needs to know the

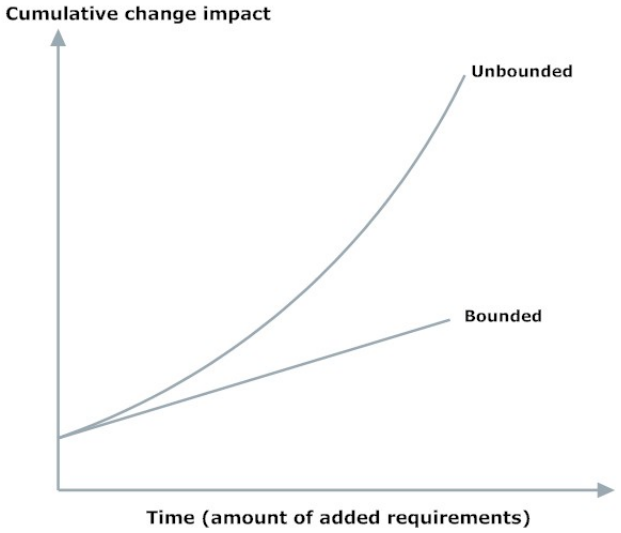

Figure 1: Cumulative impact over time [13]

interface of the module, i.e., the name of the module, the input and output parameters [3].

A. Stability

The starting point of the theory is system stability. In systems theory, one of the most fundamental properties of a system is its stability: in a stable system, a bounded input function results in bounded output values, even for $t \rightarrow \infty$ (with t representing time) [5]. This means that a limited set of changes, needed for maintenance or extension of the system, results in a limited amount of code changes or impacts to the system, even for $t \rightarrow \infty$. This includes the absence of side-effects in modules which are not changed, independent of the size of the system.

Stability demands that the impact of a change only depends on the nature of the change itself. Conversely, changes causing impacts that increase with the size of the system can be termed *combinatorial* effects and should be eliminated from the system in order to attain stability. Stability can be seen as the requirement of a linear relation between the cumulative changes and the growing size of the system over time. Combinatorial effects or instabilities cause this relation to become exponential (Figure 1). Systems that exhibit stability are defined as *Normalized Systems* [3].

> *"Normalized systems are systems that are stable with respect to a defined set of anticipated changes, which requires that a bounded set of those changes results in a bounded amount of impacts to system primitives."*

The challenge to control the impact of changes starts with identifying the changes systematically. Here, it is interesting to know the source or cause of a change. In terms of modularity, it is useful to know which parts of a module are changing independently. We should limit the size of a module to a cohesive part of content, which is changing independently of every other part. A cause or source of a change, which can be considered independently, to a software primitive can be called a 'change driver'.

B. Design theorems for Normalized Systems

Derived from the postulate that *a system needs to be stable with respect to a defined set of anticipated changes*, four design theorems or principles for the development of Normalized Systems are defined. They are briefly summarized in the following. A more detailed discussion can be found in the paper by Mannaert et al. [12].

1) Separation of Concerns: *An Action Entity shall only contain a single task.*

   The identification of a task is to some extent arbitrary. The concept of change drivers brings clarity here, because every Action Entity should only evolve because of a single change driver. Every task can evolve independently. If two or more aspects of a functionality are considered to evolve independently, they should be separated. It is proven that if one action contains more than one task, an update of one of the tasks requires updating all the others, too.

2) Data Version Transparency: *Data Entities that are received as input or produced as output by Action Entities shall exhibit Version Transparency.*

   We now concentrate on the interaction between Data Entities and Action Entities, more precisely, whether the passing of parameters or arguments affects the functionality of a module. Data Version Transparency implies that Data Entities can have multiple versions without affecting the actions that consume or produce them. In more practical terms, merely adding a field to a set of parameters that is not used in a specific Action Entity should not affect that Action Entity.

3) Action Version Transparency: *Action Entities that are called by other Action Entities shall exhibit Version Transparency.*

   In this theorem we concentrate on the interaction of Actions Entities with other Action Entities. Action Version Transparency implies that an Action Entity can have multiple versions, without affecting the actions that call the Action Entity. In other words, the mere introduction of a new version of an Action Entity or task should not affect the Action Entities calling the Action Entity containing the task.

4) Separation of States: *The calling of an Action Entity by another Action Entity shall exhibit State Keeping.*

We continue to concentrate on the interaction of Action Entities with other Action Entities, more specifically on the aggregation or propagation of Action Entities. Every Action Entity itself is responsible for remembering the calling of other Action Entities, and consequently the corresponding state. To comply with this theorem, a chain of actions calling other actions should always be asynchronous. Besides, asynchronous processing is usually associated with high reliability, and even performance. The latter is a result of avoiding locking resources related to one task during the execution of (an)other task(s).

C. Encapsulation of Software Entities

On the level of individual Action Entities, Theorems 2 and 3 (Data and Action Version Transparency) avoid instabilities caused by different versions of data and tasks. On the level of aggregations and propagations, Theorems 1 and 4 (Separation of Concerns and States) avoid unstable interactions between software constructs. Since software entities complying with Theorem 1 are very small, their application results in a highly granular structure. On the application oriented level, there is a need for larger building blocks, which are not focused on actions with only one small task, but on higher-level elements. The Normalized Systems Elements are manifestations of encapsulations, which represent each a typical building component in a software system:

- Data Encapsulation: This is a composition of software constructs to encapsulate Data Entities into a Data Element. Such a Data Element can also contain methods to access the data in a Version Transparent way, or can contain cross-cutting concerns – in separate constructs.
- Action Encapsulation: This is a composition of software constructs to encapsulate Action Entities into an Action Element. There can be only one construct for the core task (core Action Entity), which is typically surrounded by supporting tasks (supporting Action Entities). Arguments or parameters of the individual Action Entities need to be encapsulated as a Data Element for use in the entire Action Element.
- Connection Encapsulation: This is a composition of software constructs to encapsulate Connection Entities into a Connection Element. Connection Elements can ensure that external systems can interact with Data Elements, but can never call an Action Element in a stateless way. The concept of Connection Encapsulation allows the representation of external systems in several co-existing versions, or even alternative technologies, without affecting the Normalized System.
- Flow Encapsulation: This is a composition of software constructs to create an encapsulated Flow Element. Flow Elements cannot contain other functional tasks but the flow control itself, and they have to be stateful.
- Trigger Encapsulation: This is a composition of software constructs to create an encapsulated Trigger Element. Trigger Elements control the separated – both error and non-error – states, and decide whether an Action Element has to be triggered.

III. Industrial Standards

A. PLC coding with IEC 61131-3

Since their introduction in the late 1960s, PLCs (Programmable Logic Controllers) have found broad acceptance across the industry. Because they are programmable, they provided a higher flexibility than the previous control equipment based on hardwired relay circuits. PLCs were produced and sold all over the world with a large diversity of vendors. The programming languages used to program PLCs of various brands were more or less similar, but due to a lot of implementation details, intensive trainings were needed if an engineer wanted to move from one vendor's system to another.

To unify the way PLCs are programmed, the IEC (International Electrotechnical Commission) introduced the IEC 61131 standard, which is a general framework that establishes rules all PLCs should adhere to, encompassing mechanical, electrical, and logical aspects, and consist of several parts. The third part (IEC 61131-3) deals with programming of industrial controllers and defines the programming model. It defines data types, variables, POUs (Program Organization Units), and programming languages. A POU contains code; it can be a Function, Function Block, or a Program.

Functions have similar semantics to those in traditional procedural languages and directly return a single output value. However, besides one or more input values, the Function may also have parameters used as outputs, or as input and output simultaneously. They cannot contain internal state information. Consequently, they can call other Functions, but no Function Blocks.

Function Blocks are similar to classes in object oriented languages, with the limitation of having a single, public, member function. Function Blocks are instantiated as variables, each with their own copy of the Function Block state. The unique member function of a Function Block does not directly return any value, but has parameters to pass data as input, output or bidirectionally. Since Function Blocks have internal memory, they can call both Functions and other Function Blocks.

Programs can contain all the programming language elements and constructs and are instantiated by the PLC system. They are cyclically triggered by the PLC system based on

a configurable cycle time, or triggered by a system event. Typically they organize the progress of the PLC functionality during runtime, by calling Functions and Function Blocks.

All three types of POUs may be programmed in one of two textual languages (IL: Instruction Language; ST: Structured Text), or two graphical languages (LD: Ladder Diagram; FBD: Function Block Diagram). The standard also defines a graphical language for specifying state machines (SFC: Sequential Function Chart), that may also be used in Function Blocks or Programs. It should be noted that typically one of the other languages are used to code the SFC transition conditions and steps.

Programming in LD is similar to designing a relay based electrical circuit. It can be said that LD is a historical artifact. The very first PLCs were competing with existing control equipment based on hardwired relay circuits and therefore adopted a language similar to the design schematics of these electrical circuits in order to ease platform acceptance by the existing technicians.

The FBD language may be considered as a graphical incarnation of boolean algebra, where boolean OR, AND and more complex boxes are simply placed in the GUI. The inputs and outputs of the boxes are connected by drawing lines between them.

The IL language is similar to assembly. It is definitively a low level programming language, because it contains a *jump instruction*, which should be abolished from all "higher level" programming languages [14].

The ST language has a syntax similar to Pascal, and can be considered as a higher level language. Indeed, as proven by Dijkstra [14], there is no need for a *jump instruction* in ST because all processing algorithms can be implemented through three primitive types of control: *selection, sequencing, and iteration*.

B. Modeling with ISA88 (IEC 61512)

Manufacturing operations can be generally classified as one of three different processes: discrete, continuous, or batch. In October 1995, the SP88 committee released the ANSI/ISA-S88.01-1995 standard [7] (its international equivalent is IEC 61512) to guideline the design, control and operation of batch manufacturing plants.

The demand for production systems with a high flexibility, with regard to setting up the system for making product variants, became important. Process engineers focus on how to handle the material flow to meet the specifications of the end-product. Control system experts focus on how to control equipment. To optimize the cooperation of both groups, the SP88 committee wanted to separate product definition information from production equipment capabilities. Product definition information is contained in recipes, and the production equipment capability is described using a hierarchical equipment model. This provides the possibility for process engineers to make process changes directly,

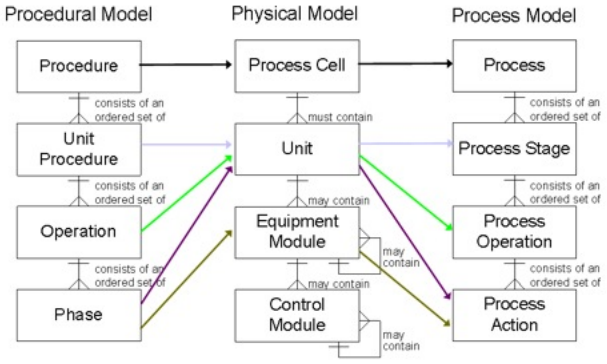

Figure 2: The relation between procedural, physical and process models [16]

without the help of a control system expert (reducing the setup costs). Moreover, the ability of producing many product variants with the same installation is achieved, increasing the target market. Expensive equipment can be shared by different production units (reducing the production costs). The utilization of ISA88 data models simplifies the design process considerably [15].

1) Challenges:

Despite the usefulness of ISA88 terminology and models to structure flexible manufacturing, different interpretations are possible. The standard does not specify how the abstract models should be applied in real applications. Implementers sometimes develop recipes and procedures which are far more complex than necessary. Since 1995 there have been many applications and a commonly accepted method for implementing the standard has emerged. The S88 design patterns [15] of Dennis Brandl (2007) address this. These patterns can reduce the tendency of implementers to make their recipes and procedures more complex than necessary.

When automated control was introduced to manufacturing, it was accompanied by the problem that control system programming became a critical activity in both initial startup and upgrades. Often the physical equipment can be reconfigured in days, if not minutes, if manually controlled and maintained. In contrast, the automatic control system, unless it was designed for reconfiguration, may take weeks or months to reconfigure and reprogram [15].

Turning the ISA88 models into well structured code is not straightforward [1]. Again, different interpretations are possible. On the macro level they provide a clear structure, but there is a need for prescriptive specifications to convert these models into code, or even to divide them into smaller sub-modules at the micro level.

2) Important ISA88 models for automation control:

During the development of ISA88, the SP88 committee was focusing on batch control, but made the models univer-

sal enough to make them suitable for other process types [16]. However, to implement these models, different design patterns are recommended for the different process types [15].

The most important key-point of ISA88 is the separation of (end) product definition information from production equipment capability. This separation allows the same equipment to be used in different ways to make multiple products, or different equipment to be used to produce the same product. Recipes are used to describe the product definition information, and a hierarchical equipment model is introduced to describe the production equipment capability. A consequence of this approach is a separation of expertise. Experts in different domains, who have to cooperate to achieve the production goals, are educated differently and think in different ways. Process engineers focus on how to handle the material flow to meet the specifications of the end-product. Control system experts focus on how to control equipment. Recipes are to be developed by process engineers, and control system experts will have to make the equipment run, based on information contained in the parameters and procedures of the recipes. ISA88 provides a physical model hierarchy to deal with equipment oriented control, and a procedural model hierarchy to deal with process oriented control. For researchers, the process model is provided (Figure 2). In this paper, we focus on the lowest level equipment oriented element: the ISA88 Control Module.

3) ISA88 Control Modules:

The lowest level of the ISA88 physical model is the Control Module, but not all parts are necessarily physical. In an automated system, Control Modules are partly (PLC) software. In their simplest form, Control Modules are device drivers, but they can provide robust methods of device control too, including functions such as automatic and manual modes, simulation mode, Interlocks and Permissives (ISA88 terminology), alarming. Control Modules execute basic control and minimal coordination control. They perform no procedural control functions. The most common method of programming basic control are any of the IEC 61131-3 programming languages, such as LD, FBD, IL, ST. Control Modules usually make up the majority of control system code, but they are also the mechanism for defining significant amounts of reusable code [15].

Typically, at least two state machines are introduced for a Control Module, one for the state of the device itself (e.g., on, off, fail), and a second for the mode (e.g., manual and automatic). Normal operation of the Control Module should be commanded from an equipment module (the equipment oriented element right above Control Module in the ISA88 physical model); however, a Control Module may also be controlled manually. Thus, the Control Module may be in one of two modes – automatic or manual.

The ISA88 standards provide the models, but do not prescribe how these models should be coded. After the standard was released, many engineers applied the standards in ways that the original authors had not considered. To address this problem, the ISA88 models have been extended with the so-called S88 design patterns [15]. These patterns are not normative, but they are effectively applied in multiple industries. The design patterns have been applied in almost every kind of batch, discrete, and continuous manufacturing applications.

C. OPC Unified Architecture (IEC 62541)

Reusable software components made their entry in automation technology and replaced monolithic, customized software applications. These components are preferably connected by standardized interfaces. In the mid-1990s, the OPC Foundation was established with the goal to develop a standard for accessing real-time data under Windows operating systems, based on Microsoft's DCOM technology.

OPC has become the de facto standard for industrial integration and process information sharing [17]. By now, over 20,000 products are offered by more than 3,500 vendors. Millions of OPC based products are used in production and process industry, in building automation, and many other industries around the world [18]. However, in the period when Internet based systems were introduced, the DCOM technology resulted in limitations. To challenge these limitations, and truly support Web-enabled automation systems, a new standard family has recently been released: The OPC Unified Architecture.

Web-based technology is the key to taking interoperability to a new level. Web Services (WS), are totally platform independent – they can be implemented using any programming language and run on any hardware platform or operating system. Components can be flexibly arranged into applications and collaborate over the Internet as well as corporate intranets. OPC UA is considered one of the most promising incarnations of WS technology for automation. Its design takes into account that industrial communication differs from regular IT communication: embedded automation devices such as PLCs provide another environment for Web-based communication than standard PCs.

The concepts of OPC UA include enabling Version Transparency in a system with a high diversity of components. OPC UA complements the existing OPC industrial standard by adding two fundamental components: different transport mechanisms and unified data modeling [19]. Scalability, high availability, and Internet capability open up many possibilities for new cost-saving automation concepts. Alternative platforms, including typical embedded (systems) operating systems, can be accessed directly, eliminating the need for an intermediate Windows PC to run the OPC Server.

For this paper, the most important aspects of OPC UA are the parts Address Space and Information Modeling

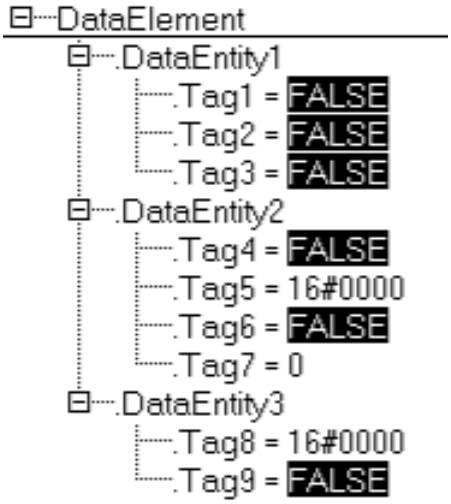

Figure 3: Data Encapsulation

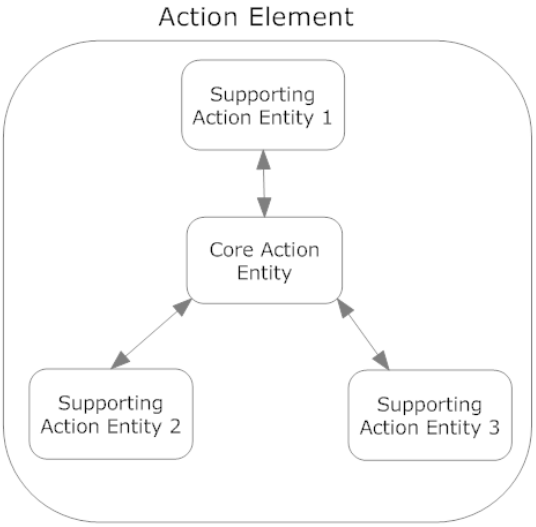

Figure 4: Action Encapsulation

of the OPC UA specification [6]. Indeed, these models include interesting concepts on converting our IEC 61131-3 based ISA88 Control Module to an evolvable black box module, accessible by an OPC UA-based interface [20], complying with the principles of Normalized Systems. This enables the use of automation control building blocks over a standardized network, even the Internet. As a result, PLC projects, which are typically intra-process, can become inter-process with a higher potential of production system integration.

IV. Evolvable Control Modules

A. Design concept

The Normalized Systems theory defines five encapsulations of software entities. These encapsulations are defined in a fundamental way, and further worked out in the form of design patterns. These design patterns are exemplified in the background technology of the Java programming language [3]. We propose an interpretation of the fundamental encapsulations for the IEC 61131-3 programming environment. When we base the design of a Control Module on the Normalized Systems theory, we propose 3 building components of a Control Module. In this paper, we focus on Data Encapsulation, Action Encapsulation, and Connection Encapsulation. The latter is a special case of Action Encapsulation:

- Data Encapsulation: The composition of software entities, i.e., encapsulating all tags of a larger building block into a single data element, implies that only one (complex) parameter shall be passed to and returned from this building block. Note that in an IEC 61131-3 program, the use of *structs* can tackle the problem of adding extra parameters. By extending the struct, all parts of the struct, old and new fields, remain visible and accessible by every entity. We apply this concept to the core module of this paper, an ISA88 Control Module. Inside the borders of an IEC 61131-3 project, this leads to a "struct" for the whole production control device, containing smaller "substructs" for every action or task in that device. There is no straightforward concept of data hiding available in IEC 61131-3, so we cannot hide the new fields in such a struct for older entities. However, this is not causing data type conflicts, because in IEC 61131-3 no runtime construction of instances is supported, so all data instances of this complex parameter have the same type structure. On the inter-process level, different type instances are possible, but we can use data hiding based on OPC UA, where an interface is made for, e.g., SCADA or MES software, which can run on several technologies or platforms (inter-process). We define the entire parameter, main struct and substructs together, as a *Data Element* (Figure 3).

- Action Encapsulation: the composition of software entities to encapsulate all Action Entities into a single ISA88 Control Module implies that the core action (state machine of the Control Module) shall only contain a single functional task, not multiple tasks. In concept, we consider one core task, surrounded by supporting tasks (Figure 4). Arguments and parameters of the larger building block (ISA88 Control Module) should be an aggregation of all encapsulated Data Entities: the single complex datastruct or Data Element (Figure 3). We define the larger building block,

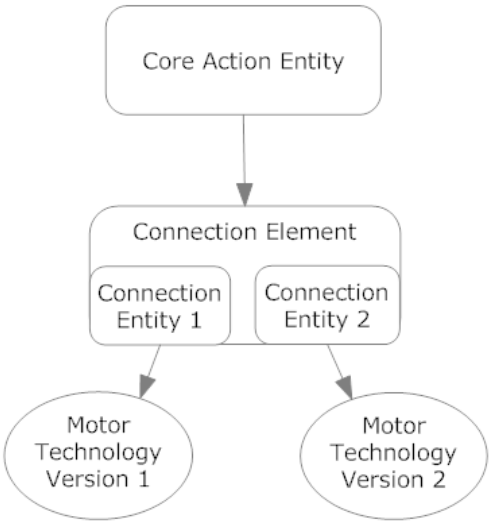

Figure 5: Connection Encapsulation

encapsulating the core Action Entity together with supporting Action Entities as an *Action Element*.

- Connection Encapsulation: Typically, three actors are interacting with this Action Element as an implementation of an ISA88 Control Module: An Equipment Module (automatic mode, recipe based control interface), the operator (manual mode, low level HMI: Human Machine Interface) and the process equipment hardware (e.g., a motor, valve or instrumentation device). Each of these actors is considered to represent an external technology, in possible new versions or even alternative technologies over time. The (supporting) Action Entities of the Control Modules, which are handling the connection of these actors with the core action, are defined as *Connection Entities*. In case of multiple versions of this special kind of Action Entities, every connection is an encapsulation of several versions or alternative technologies. We define such a Connection Encapsulation as a *Connection Element* (Figure 5). This encapsulation implies that the corresponding Data Element has a subsubstruct for each Connection Entity related to the substruct of the Connection Element.

The study of Flow Elements and Trigger Elements is outside the scope of this paper. Remember that IEC 61131-3 Programs are triggered by the PLC system. Consequently, Trigger Elements are integrated in the configuration part of the IEC 61131-3 environment. Following the ISA88 modeling rules, Control Modules should not contain procedures, so Flow Elements are not applicable in an ISA88 Control Module.

B. Anticipated Changes

In the design of evolvable Control Modules, we want to create a module which is stable with respect to a defined set of anticipated changes. We distinguish high-level changes and elementary changes. A lot of engineers know only the high-level changes, which are either real-life changes or changes with respect to implementation related aspects. These changes reflect additional functional requirements, which can typically be found in requirements documents or new customer requests. The elementary changes are related to software primitives, and formulated in terms of software constructs. One additional functional requirement corresponds to at least one elementary change, but, more probably, several elementary changes. First, we discuss the elementary anticipated changes of software primitives, and second, we discuss how real-life changes can be translated into these elementary changes. The elementary anticipated changes are:

- A new version of a data entity
- An additional data entity
- A new version of an action entity
- An additional action entity

Remember we make an aggregation of several data tags into an IEC 61131-3 struct, with a substruct for every Data Entity corresponding with an Action Entity. Extending a substruct with one or more tags corresponds with the change “A new version of a data entity”; adding a new substruct corresponds to the change “An additional data entity”. The core task of a Control Module is a hardware device driver. This core task is surrounded by supporting tasks, like manual/automatic control, simulation, Permissives, alarm. The introduction of a new surrounding task corresponds with the change “An additional Action Entity”. A change of the functionality of a task corresponds with the change “A new version of an Action Entity”.

An experienced programmer should be able to transform real-life changes into changes of software primitives. However, in a team where inexperienced engineers do the coding, a “change architect” should fulfill this task. The systematic translation of high-level requirements into the more elementary form is outside the scope of this paper, but we do provide some examples in Section V.

C. Managing action versions

To comply with the third theorem of Normalized Systems, Action Version Transparency, we distinguish three cases: Transparent Coding, Wrapping Functionality and Wrapping External Technologies. Each of them is another approach, but provides a similar result to ‘the outside’.

1) Transparent Coding: Since Normalized Systems require a high granularity, it is not unexpected that the individual (small and straightforward) modules or

subroutines end up to be a simple piece of code, on which the programmer has a clear overview. In such cases, the programmer can preview the effect of a functional change on the previous version(s). If the change is not contradictory with one of the previous versions, it might be possible to apply Transparent Coding. This means the new functionality can just remain in the module without affecting the original code, even if a calling entity is not aware of the new functionality. We provide some examples in Section V.

2) Wrapping Functionality: There will be lots of cases where Transparent Coding is not possible, because the code is too complex for the programmer to have a reliable overview, or if the new functionality is contradictory with one or more of the previous versions. To exhibit Action Version Transparency, the different versions can be wrapped. The calling action has to inform the called action which version should be used, by way of a version tag. In addition, following the 'Separation of States' principle, the called action has to inform the calling action whether the (type version of the) instance of the called action is recent enough to perform the requested action version.

An Action Entity which is designed according to the concept of Wrapping Functionality is aggregating all the versions as separate Action Entities, and is therefore called an Action Element. Each of the nested Action Entities contains a version of the core functionality. Following the 'Separation of Concerns' principle, the wrapping Action Entity (Action Element) should not contain any core functionality, but is limited to wrapping the versions as a kind of supporting task.

3) Wrapping External Technologies: It is very unlikely that an external technology complies with the Normalized Systems theorems. On the contrary, Lehman's Law of increasing complexity probably applies in this external technology. Consequently, we assume that lots of combinatorial effects and unbounded impacts are generated in case of any change in this external technology. We do not want to allow these effects and impacts to penetrate into our stable system. To isolate these effects, we use the concept of Connection Encapsulation. A Connection Entity is an Action Entity dedicated to doing nothing but mapping requests from an internal action to an external technology, and mapping the responses of the external technology to the calling internal action.

When there is a change in the external technology, this change might have effect on our Connection Entity. If it is a small update or hot fix, the Connection Entity could handle this change by way of Transparent Coding, but in general, the Connection Entity will remain dedicated to the version of the external technology that it was developed for. A new version of the external technology leads to the introduction of a new Connection Entity. An Action Entity, which is representing the core functionality of the external technology, has the task to wrap the different Connection Entities. This wrapping Action Entity is defined as a Connection Element. Again, following the 'Separation of Concerns' principle, the functionality of the wrapping entity should be limited to mapping requests from an internal action to a specific Connection Entity, and mapping the responses of the specific Connection Entity to the calling internal action. Every Connection Entity, which is part of the wrapped versions, should have a version tag of the instance of the external technology it is connected to. The calling internal action should inform the Connection Element (wrapping action) about which Connection Entity or external technology version is desired by way of a version tag. The Connection Element should inform the calling action whether the requested version is available.

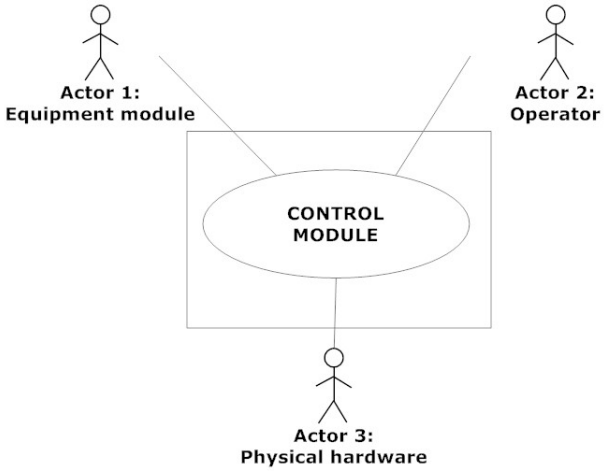

Figure 6: Actors on the Control Module [20]

With the concept of Wrapping External Technologies, the separation of versions is done by wrapping Connection Entities, each representing a version of the external technology. This concept can be easily extended with the introduction of Connection Entities, representing alternative external technologies. Every new version of an alternative external technology leads to the introduction of a new Connection Entity.

D. Evolvable Control Module

In this section we introduce a Control Module for a motor. We aim at making this motor control software module as generic as possible. Instead of introducing new formalisms, we based our proof of principle on existing standards. For the modeling, we used concepts of ISA88 (IEC 61512), for interfacing, we used OPC UA (IEC 62541), and for coding we used IEC 61131-3. More specifically, we rely on the S88 design patterns [15] (derived from ISA88) because

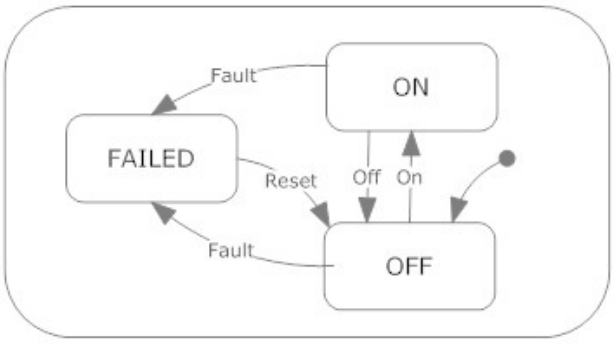

Figure 7: Example of a motor state model [1]

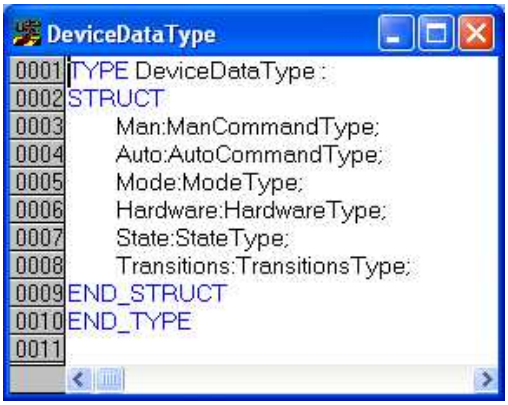

Figure 8: The Data Element

these patterns can be used not only in batch control, but also for discrete and continuous manufacturing. None of these standards contains prescriptive suggestions on how the internal code of a Control Module should be structured. We introduce a granular structure following the theorems of Normalized Systems. Every task (action), which must be executed by the Control Module, is coded in a separated Function Block.

In the most elementary form Control Modules are device drivers, but they provide extra functions. In our proof of principle we integrated the functionality manual/automatic mode and alarming. We kept the functionalities 'Interlocking' and 'simulation' as possible 'future changes', since it should be able to add them without causing combinatorial effects.

The proposed design of an evolvable Control Module contains *one* Data Element and *one* Action Element, which can include several Connection Elements. These Elements are implementations of Data, Action and Connection Encapsulation.

1) (One) Data Element: To make sure the interface of an action will not be affected in case of adding an additional tag or Data Entity, we work with one single struct and define this struct as a Data Element (Figure 3). The Data Element is a struct, which contains a substruct for every Data Entity.

2) (One) Action Element: The Action Element is a Function Block, which contains other Function Blocks, one for each Action Entity. The Action Element contains one core Action Entity, surrounded by supporting Action Entities. The tags controlled by each Action Entity belong to the corresponding substruct of the Data Element (Figure 4). An Action Entity can read all tags of the other substructs, but can only write in its own substruct (Data Entity).

3) Connection Elements: A Connection Element corresponds with a special kind of Action Entity in the sense that the change driver is an external technology, or, more generally, the change driver is coming from the outside of the Control Module. Typically, we have three actors on the Control Modules: the operator (low level HMI), the Equipment Module, which owns the Control Module, and a Process Hardware Device (Figure 6). Following the 'Separation of Concerns' principle, each connection has to be handled with a separate software module. If the device hardware has several versions, a Connection Entity is needed for every version (Figure 5).

In the following, we specify the software entities which represent the content of the Elements. We used the design pattern shown in Figure 7. This state machine is very simple. When the control system powers on, the motor enters in the 'OFF' state. It can be started and stopped via the 'On' and 'Off' commands. Hardware failures can cause the motor to go to the 'FAILED' state, from where a 'Reset' command is required to return to the 'OFF' state. The concept of this 'FAILED' state brings us a very important benefit: process safety. Besides, it forms the base for failure notification [15]. This functionality is implemented in a Function Block we call 'CoreStateAction'. This Function Block has only one parameter we call 'Device'. The datatype of this parameter is called 'DeviceDataType'.

4) Data Entities:

The 'DeviceDataType' is a struct containing 6 substructs (Figure 8). Four of them include information coming from 'the outside':

- Man: Manual commands. Contains the tags 'On' and 'Off', which allow the operator to start and stop the motor. Additionally, this substruct contains the tag 'Reset' to allow the operator to bring the status of the Control Module back to the initial state after a failure.
- Auto: Automatic commands. Contains the tags 'On' and 'Off', which allow an ISA88 Equipment Phase to start and stop the motor in automatic mode. In this

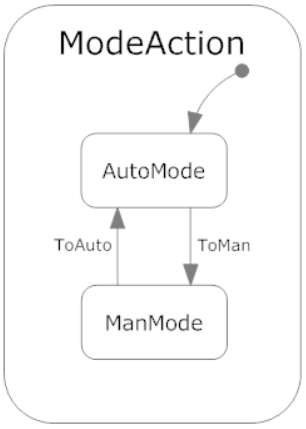

Figure 9: The Action Entity ”ModeAction”

version, we choose that in Automatic Mode the reset functionality can not be accessed.

- Mode: The mode of the Control Module. Contains the tags ‘ManMode’ and ‘AutoMode’, which indicate the mode of the Control Module. Additionally, this substruct contains the tags ‘ToMan’ and ‘ToAuto’ to allow an entity from ‘the outside’ to switch the Mode between Manual and Automatic.
- Hardware: The Hardware tags. First, this substruct contains the tag ‘Qout’, which can be linked with a PLC output address to electrically control starting and stopping of the motor. Second, this substruct contains the tag ‘FeedBack’, which can be linked with a PLC input address to check whether the motor is physically running or not. Third, this substruct contains a tag ‘Fault’, which assumes the value ‘TRUE’ if the output is not corresponding with the feedback of the motor.

A fifth substruct ‘State’ contains the state data of the core state machine: ‘On’, ‘Off’ and ‘Failed’. The transition tags ‘ToOn’, ‘ToOff’ and ‘Reset’ are placed in the sixth substruct ‘Transitions’. These tags contain the results (output) of an Action Entity, which decides whether the operator or the Equipment Module has control (based on the mode).

5) Action Entities:

Our evolvable Control Module contains four Action Entities:

- ModeAction: Mode action (Figure 9). This is a state machine, which maintains the mode. Mode commands switch between manual mode and automatic mode. The inputs of this action are the mode commands of the substruct ‘Mode’. The outputs of this action are both mode states of the same substruct.
- TransAction: Transition action (Figure 10). The inputs

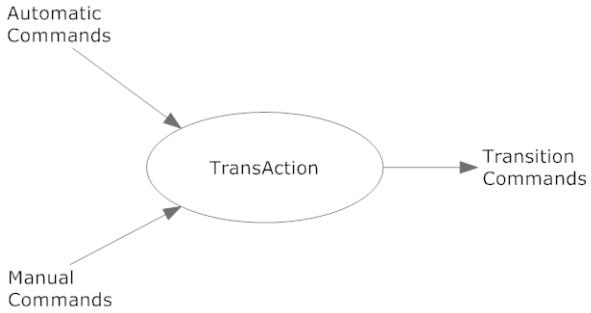

Figure 10: The Action Entity ”TransAction”

of this action are all requests of both automatic control and manual control entities, available in the substructs ‘Auto’ and ‘Man’, respectively. The outputs of this action are the tags in the substruct ‘Transitions’.

- StateAction: Core state machine action (Figure 7). This is the state machine, which maintains the state of the Control Module. The inputs of this action are the transitions tags of the substruct ‘Transitions’. The outputs of this action are the state tags of the substruct ‘State’.
- HardwareAction: The hardware action (Figure 11). The inputs of this action are the states of the substruct ‘State’, and the input ‘FeedBack’ of the substruct ‘Hardware’. The outputs are the tags of the substruct ‘Hardware’.

Please note that the entity which is performing manual commands (typically the low level HMI), must check the tags of the mode state machine to check whether the manual commands will be accepted. In automatic mode, the manual commands will be ignored. Similarly, the ISA88 Equipment Module (automatic mode entity) must check that automatic mode is active before sending requests.

6) Connection Entities:

In fact, the action ‘HardwareAction’ is a Connection Entity, which connects the control software (with the core state machine as its central task) of the Control Module to physical production process hardware. Remember that a Connection Entity is a special case of an Action Entity. Indeed, in the first version of our proof of principle it maps the ‘On’ state of the core state machine to the hardware output ‘Qout’. In addition, it checks if the value of ‘Qout’ corresponds with the input value ‘FeedBack’. If there is a discrepancy, it sets the tag ‘Fault’ to inform the core state machine action that the hardware is not responding as expected.

In a second version, we connected a bidirectional motor to the Control Module. Consequently, we added a tag

Figure 11: The Action Entity "HardwareAction"

'Qreverse' to the substruct 'Hardware'. The Connection Entity sets this 'Qreverse' tag in case the core state machine action requests to run the motor in reverse direction. As a result, when the core state action requests the motor to run in the original direction, only the tag 'Qout' is set. When the core state action requests the motor to run in the reverse direction, both the tags 'Qout' and 'Qreverse' are set. Since the way of setting the tag 'Qout' is not changed, we could apply Transparent Coding. Instances of the Control Module which are connected to a unidirectional motor will just start and stop the motor and neglect the tag 'Qreverse' (which is initialized to the value 'FALSE').

In a third version, we have a bidirectional motor, but it is controlled differently. Instead of having a tag which controls whether the motor should run or not and another tag indicating the direction, we have two tags controlling a direction each. If one of them is TRUE, the other must be FALSE to provide an unambiguous command to the device. Obviously, since the interface to the device is changed, Transparent Coding is not possible. So for this third version, we used the wrapping concept. We added a tag 'Version' to the substruct 'Hardware'. The code of the action 'HardwareAction' is moved to a new module called 'HardwareActionV0'. The newly introduced code in the action 'HardwareAction' is a selection, associated with the version tag. If the version tag has the value '0', the request to the action 'HardwareAction' is forwarded to the action 'HardwareActionV0'. Besides, the Action Entity 'HardwareAction' could more appropriately be called 'Connection Element' now, because it is only selecting versions and mapping. Another new Function Block 'HardwareActionV1' contains the newly introduced functionality of the new motor. HardwareActionV0 and HardwareActionV1 are called Connection Entities.

To connect the automatic procedure (typically an ISA88 Equipment Phase) to our Control Module, no code is needed. Indeed, such a Phase (Figure 2) is typically coded in an ISA88 Equipment Module by way of the IEC 61131-3 language SFC. Since the Equipment Module, which has control over our Control Module, is coded into the same PLC as the Control Modules it owns, it only needs access to the instance of the Data Element 'Device'.

Besides, even for manual control no IEC 61131-3 connection code is needed. Similarly, the low level HMI just needs access to the Data Element 'Device'. However, since the low level HMI is located in an external technology, it would lead to the coding of a Connection Entity or Element. Thanks to the OPC UA IEC 61131-3 companion specification [21], it is expected that we will not need to code, but only configure the Connection Entity. Based on this OPC UA companion specification, software constructs of IEC 61131-3 can be mapped to OPC UA. Unfortunately, this companion specification is rather recent, and we could not find any commercial products supporting this standard at the moment of submitting this paper.

V. Changes and Evaluation

A way to test evolvability is adding changes and evaluating the impact of these changes. In general, we start with a first version. Then we maintain one or more running instances of the Control Module with the initially expected behavior. Second, we consider the addition of a change, and consequently a possible update of the datatype, existing actions or introduction of a new action. Finally, we make a new instance, check the new functionality and the initial expected behavior of the older instances as well.

We provide some examples for the transformation of high-level changes to the anticipated changes of software primitives.

- We consider the situation that manual operations could harm automatic procedures. For instance, stopping a motor manually could confuse an algorithm if it is happening during a dosing procedure. To prevent this, we add the feature "manual lock". This means, we still support manual mode, but we disable manual mode during the period a software entity such as an equipment module requires the non-interruptible (exclusive) use of the Control Module.
  In terms of elementary changes, this requires an additional version of a Data Entity, more specifically the addition of a tag 'ManLock' in the substruct (Data Entity) dedicated to receiving automatic commands. Additionally, a new version of an Action Entity is introduced. The action dedicated to select manual or automatic mode adds the 'ManLock' tag as a constraint to switch over to manual mode.
- We consider the situation of a motor instance where the motor must be able to run in two directions (while the functionality of earlier unidirectional motor instances should remain).
  In terms of elementary changes, this requires an additional version of three Data Entities. First, the addition of a tag 'Reverse' in the substruct (Data Entity) dedicated to hardware control. Second, the addition

of a tag 'ManReverseCmd' in the substruct dedicated to receiving manual commands. Third, the addition of a tag 'AutoReverseCmd' in the substruct (Data Entity) dedicated to receiving automatic commands. Moreover, the action which is processing the result (aggregation) of the requests of both manual and automatic commands needs a new version to provide a command 'ReverseCmd' for the core state machine action. Finally, the core state machine Action Entity needs a new version to add the new state 'ReverseState', accompanied by transitions from and to this new state.

- We consider the situation where one wants to introduce a simulation mode (for testing purposes), to neglect the Fault transition if no hardware is connected.
  In terms of elementary changes we need three changes. First, an additional Data Entity or substruct which can be used to store the state of the simulation mode. Second, an additional Action Entity to process the newly introduced state machine, and third, a new version of the core state machine Action Entity to neglect the Fault transition when in simulation mode.

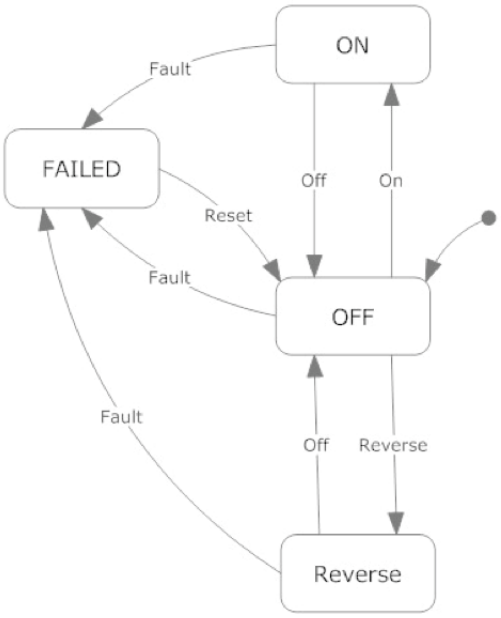

Figure 12: Core state machine bidirectional motor

Remember that manual operations could affect automatic procedures. In terms of elementary anticipated changes, this requires the addition of the tag 'ManLock' in the substruct 'Mode'. The action 'ModeAction' adds the 'ManLock' tag as a constraint to switch over to manual mode. We applied the concept of Transparent Coding. In the IEC 61131-3 data type declaration part of this additional tag in the substruct, we explicitly declared the initial value to be FALSE. We did not remove or change the calls of instances which do not need this feature. For older calls the behaviour did not change, and for the new instances we can indeed prevent the mode going to automatic.

We consider again the situation of a motor instance (as above) which must be able to run in two directions. In the previous section we discussed new versions of the Connection Entity 'HardwareAction', updated to the Connection Element 'HardwareAction', which is containing the two Connection Entities 'HardwareActionV0' and 'HardwareActionV1'. The elementary changes with regard to this Connection Element can be done without affecting the other actions or Data Entities. However, this does not mean that the related high-level changes or real-life requirements are met. Remember that *one* additional functional requirement corresponds typically to *more* elementary changes. For our two directions motor instance, a change of the core state machine was necessary, in addition to the elementary changes needed for the Connection Element. First, in the Data Entity (substruct) 'State', an additional state 'Reverse' was introduced. Second, in the data entity (substruct) 'Transitions', the tag 'Reverse' was added. Third, the functionality of the Action Entity 'StateAction' was extended by way of Transparent Coding. Transparent Coding for a state machine means that no states can be removed, no states can be changed, and no transitions can be changed or removed. In other words, the allowed changes are only additions of states and transitions (Version Transparency principles). We end up with the new version of the state machine depicted in Figure 12.

We provide two more examples of Transparent Coding. Remember the manual lock feature. We can code the initialization of the new tag 'ManLock' to FALSE. By assuming this default value, the code of the change can be made in a way that this default value guarantees the behavior of the previous version. More precisely, if the value is FALSE, the manual lock functionality will not apply until a newer version instance is setting it to TRUE, which is not going to happen if an older version is used on this specific instance.

Remember the motor instance, where a new version can let the motor run in the reverse direction. Similarly, if the default version of the tag 'Reverse' is initialized to FALSE, the motor will run in the original direction until a newer version instance is setting the reverse tag to TRUE, which again is not going to happen if an older version is used on this specific instance.

VI. Conclusion and Future Work

Evolvability of software systems is important for IT systems, but also a relevant quality for industrial automation systems. IEC 61131 Function Blocks of automation systems

are programmed close to the processor capabilities. For example, there is a similarity between the IEC 61131-3 language Instruction List (IL) and assembly. The key point of Normalized Systems is a high granularity of software modules with a structure which is strictly disciplined to the related theorems. As a consequence, making a proof of principle close to the processor is a very informative exercise to concretize the Normalized Systems theory. In addition, this approach can be of great value for improving the quality of industrial automation software projects.

It must be stated that implementing these concepts was highly facilitated by the use of existing industrial standards. They provide us with methods to develop the macro-design of software modules, while the Normalized Systems theory provides guidelines for the micro-design of the actions and data structures encapsulated in these modules. Adding functionality or even adding an action to a (macro) building block, in our case the Control Module, can be done with a limited impact (micro manageable) towards other (macro) entities (bounded impact). To define the most basic actions (tasks) and data structures, the identification of the change drivers of the concerned entity is essential. This confirms the first theorem for software stability, Separation of Concerns.

Our future work will be focused on other (macro) elements of ISA88, which contain different types of control. A Control Module contains mainly basic control, together with limited coordination control (the mode). We will extend this study to elements with more advanced coordination control code and procedural control, again designed to comply with the Normalized Systems theory.

Moreover, future work will also be focused on interfaces. Since OPC UA is very generic, we wonder if constraints must be added to the standard to let data communication be compliant with the second theorem of software stability, Data Version Transparency. It will be interesting to investigate whether both currently existing OPC UA transport types, UA binary and UA XML, can be handled in a Data Transparent way.

REFERENCES

[1] van der Linden D., Mannaert H., and de Laet J., "Towards evolvable Control Modules in an industrial production process", ICIW 2011, 6^{th} International Conference on Internet and Web Applications and Services, pp. 112-117, 2011.

[2] International Electrotechnical Commission, "Programmable controllers - part 3: Programming languages", IEC 61131-3, 2003.

[3] Mannaert H. and Verelst J., "Normalized Systems Re-creating Information Technology Based on Laws for Software Evolvability", Koppa, 2009.

[4] McIlroy M.D., "Mass produced software components", NATO Conference on Software Engineering, Scientific Affairs Division, 1968.

[5] van Nuffel D., Mannaert H., de Backer C., and Verelst J., "Towards a deterministic business process modelling method based on normalized theory", International Journal on Advances in Software, 3:1/2, pp. 54-69, 2010.

[6] OPC Foundation. "OPC Unified Architecture", www.opcfoundation.org.

[7] The Internation Society of Automation, "Batch Control Part 1: "Models and Terminology", ANSI/ISA-88.01, 1995.

[8] Mannaert H., Verelst J., and Ven K., "Towards evolvable software architectures based on systems theoretic stability", Software, Practice and Experience, vol. 41, 2011.

[9] Kuhl I. and Fay A., "A Middleware for Software Evolution of Automation Software", IEEE Conference on Emerging Technologies and Factory Automation, 2011.

[10] Eick S.G., Graves T.L., Karr A.F., Marron J., and Mockus A., "Does code decay? Assessing the evidence from change management data", IEEE Transactions on Software Engineering, vol 32(5), pp. 315-329, 2006.

[11] Lehman M.M., "Programs, life cycles, and laws of software evolution", Proceedings of the IEEE, vol 68, pp. 1060-1076, 1980.

[12] Mannaert H., Verelst J., and Ven K., "The transformation of requirements into software primitives: Studying evolvability based on systems theoretic stability", Science of Computer Programming, 2010.

[13] van der Linden D. and Mannaert H., "In Search of Rules for Evolvable and Stateful run-time Deployment of Controllers in Industrial Automation Systems", ICONS 2012, 7^{th} International Conference on Systems, accepted for publication, 2012.

[14] Dijkstra E., "Go to statement considered harmful", Communications of the ACM 11(3), pp 147-148, 1968.

[15] Brandl D., "Design patterns for flexible manufacturing", ISA, 2007.

[16] van der Linden D., "Implementing ISA S88 for a discrete process with the Bottom-Up approach", AGH - Automatyka 12/1, pp. 67-76, 2008.

[17] Hannelius T., Salmenpera M., and Kuikka S., "Roadmap to adopting OPC UA", 6^{th}IEEE International Conference on Industrial Informatics, pp. 756-761, 2008.

[18] Lange J., Iwanitz F., and Burke T.J., "OPC: von Data Access bis Unified Architecture", VDE-Verlag, 2010.

[19] Mahnke W., Leitner S., and Damm M., "OPC Unified Architecture", Springer, 2009.

[20] van der Linden D., Mannaert H., Kastner W., Vanderputten V., Peremans H., and Verelst J., "An OPC UA Interface for an Evolvable ISA88 Control Module", IEEE Conference on Emerging Technologies and Factory Automation, 2011.

[21] PLCopen & OPC Foundation, "OPC UA Information Model for IEC 61131-3 1.00 Companion Specification", 2010.

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