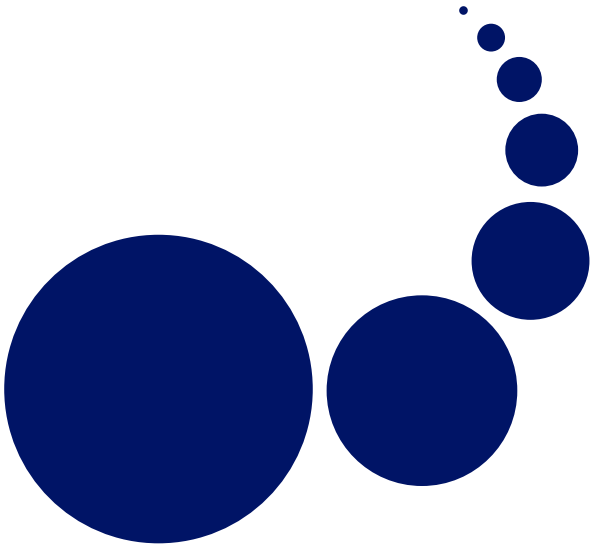
SCALABLE COMPUTING
Practice and Experience

Special Issue: Selected Papers From the 2nd Workshop on Software Services

Editors: Dana Petcu and Jose Luis Vazquez-Poletti



Volume 12, Number 3, September 2011

ISSN 1895-1767

SUBSCRIPTION INFORMATION: please visit http://www.scpe.org

Scalable Computing: Practice and Experience

Volume 12, Number 3, September 2011

TABLE OF CONTENTS

Selected Papers From the 2nd Workshop on Software Services:

Scalable Computing: Practice and Experience ISSN 1895-1767
Volume 12, Number 3, pp. iii–iv. http://www.scpe.org

INTRODUCTION TO THE SPECIAL ISSUE ON SELECTED PAPERS FROM 2ND WORKSHOP ON SOFTWARE SERVICES

Dear SCPE readers,

This September issue of Scalable Computing: Practice and Experience is the first of two issues devoted to Cloud Computing and Applications based on Software Services.

To ensure the high quality of this issue, a reduced number of contributions that were presented at 2nd WoSS (Timisoara, June 6-9 2011) were chosen and an invitation to provide an extended version was sent. The reviewers were the same as in the Workshop so the peer-review process was exhaustive, from the abstract status prior to the Workshop until the final version that you will be reading. Additionally, the priceless feedback gathered during 2nd WoSS definitely increased the contribution quality.

Many of the authors come from countries that joined the European Union recently. This demonstrates that the European Research Family is not only growing up but also reunites around the Cloud Computing table, a bleeding edge and promising area which is expected to bring much outstanding outcome.

Dana Petcu,
Computer Science Department
West University of Timisoara
and Institute e-Austria Timisoara, Romania

Jose Luis Vazquez-Poletti,
Distributed Systems Architecture Group
Faculty of Computer Science, Universidad Complutense de Madrid, Spain

Scalable Computing: Practice and Experience
Volume 12, Number 3, pp. 283–291. http://www.scpe.org
ISSN 1895-1767

VEHICLE ROUTING PROBLEMS WITH THE USE OF MULTI-AGENT SYSTEM

LUKASZ CHOMATEK* AND ANETA PONISZEWSKA-MARANDA†

Abstract. Increasing number of vehicles on the roads caused the increase of popularity of GPS devices that the drivers can install in their cars. Efficient vehicle routing is very significant task nowadays, as the number of vehicles on the roads is growing rapidly. As many drivers have an ability use a computer while planning their itinerary, they need to have an application to find the best route for them.

The paper describes a new approach for path finding problem. The proposition of solving the path finding problem with the use of multi-agent system is proposed. The idea of multi-agent system includes cooperation between autonomous software agents to complete a certain task.

Key words: vehicle routing, path finding problem, single source shortest path problem, multi-agent systems

AMS subject classifications. 68N02, 68T02

1. Introduction. Increasing number of vehicles on the roads caused the increase of popularity of GPS devices that the drivers can install in their cars. Internet sites, where it is possible to compute efficient route from one point to another are also popular, because they are not only maps but also can fulfill some needs of the user. The user's needs can be divided into some groups:

- saving of time - user of the system only needs to know the destination address and in some cases he needs to enter the start point as well because the path is computed automatically by the system,
- finding some information - system can show the way to the nearest restaurant, gas station or shop,
- informing about the situation on the road, e.g. traffic jams, speed cameras, road works, accidents.

Both mentioned websites and GPS devices have to execute large numbers of queries about the route between points on the map. The website is a service dedicated for the large number of users and GPS device has to reflect dynamically changing road conditions (i.e. driver was supposed to turn left on the crossroads but went straight and now the system must compute a detour). Large number of queries can only be handled when either users' requests can be processed in a longer time or by use of very efficient path finding algorithms.

The most efficient algorithms for solving Single Source Shortest Path (SSSP) are hierarchical approaches [1]. They are usually based on the fact that some road segments can be marked as having higher importance than others. What is more, road network can be preprocessed by removing some nodes and introduce some shortcuts instead (i.e. there is only one connection from one node to another, so all nodes between them can be substituted by a direct link for this nodes).

The paper describes the proposition of solving the path finding problem with the use of multi-agent system. The idea of multi-agent system includes cooperation between autonomous software agents to complete a certain task [9, 10]. For solution of SSSP problem based on road network hierarchy,the agents can be divided into some groups: graph constructing agents, agents interacting with the system user and miscellaneous agents. The second significant term in the domain of multi-agent systems is the environment, in which the agents are located [11]. In the case of road traffic it is very well defined and contains hardly any subjective factors. It includes vehicles, roads, road signs and signals and some important places which are usually named points of interest (POI). Road environment is obviously dynamic, due to the fact that hardly any part of it remains unchanged for a long time.

The paper is structured as follows: section 2 presents the approaches used to solve the problem of hierarchical single source shortest path. Section 3 describes the proposition of algorithm for road network division while section 4 deals with multi-agents system for road network hierarchization problem. Section 5 describes the results obtained during the use of system application.

2. Approaches of hierarchical single source shortest path. Typical algorithms designed for solving SSSP problems do not use any preprocessing of the graph. Preprocessing phase can take a long time, so that such algorithms can be easily applied, when there is a little number of queries about the shortest path. Most popular SSSP solving algorithms are Dijkstras algorithm, Bellman-Ford algorithm and A* algorithm.

*Institute of Information Technology, Technical University of Lodz, Poland (lukasz.chomatek@p.lodz.pl)
†Institute of Information Technology, Technical University of Lodz, Poland (anetap@ics.p.lodz.p)

Table 2.1: N_3^0 for all vertices of the sample graph

v	$N_3^0(v)$
0	$\{0, 2, 3\}$
1,2	$\{0, 1, 2\}$
3	$\{0, 3, 4\}$
4,5	$\{2, 4, 5\}$

Hierarchical algorithms include some kind of preprocessing of the graph in order to shorten the time required to process a single query. It is notably important when number of queries is very high and sometime can be expended before deployment of the system.

The algorithm of Hierarchical Path Views proposed in the literature [4, 5] was based on the following ideas:

- base road network was split into some fragments - places of split were chosen by its geographical coordinates,
- connections which are outside generated fragments belong to higher hierarchy level,
- division and level transfer is an iterative process.

The result of such a division are the matrices containing the shortest path lengths for each segment and each level. After the division phase, to perform a query, A* algorithm was used.

Base for other branch of hierarchical algorithms for solving SSSP problem was Highway Hierarchies algorithm proposed in [1, 2]. Dijkstras algorithm is used in the preprocessing phase to calculate the neighborhood for each vertex. Next, the vertices that fulfill some criteria are moved to the higher hierarchy level. When this phase is done, the higher hierarchy level is preprocessed that allows to generate shortcuts between certain vertices. Number of hierarchy levels and size of the neighborhood are parameters of the algorithm. Proper choose of them influences on the amount of time needed to process a single query.

2.1. Highway Hierarchies Algorithm. Highway Hierarchies algorithm requires two parameters: H that identifies the degree to which the requests for the shortest way are met without coming to a higher level in the hierarchy, and L, which represents the maximum permissible hierarchy level. The method used to iteratively generate a higher level with number $l + 1$ for a graph G^l is as follows:

1. For each vertex $v \in V$, build the neighborhood N_H^l for all vertices reached from v by using Dijkstras algorithm in graph G^l, respecting the H constraint. Set the state of the vertex V to "active".
2. For each vertex:
   - Build the partial tree $B(v)$ and assign to each vertex its state. The state of the vertex is inherited from the parent vertex every time when the vertex is reached or settled. Vertex becomes "passive" if on the shortest path $< v, u, ..., w >$, where $v \neq u \neq w$:

     $|N_H^l(u) \wedge N_H^l(w)| \leq 1$

     Partial tree is completed, when reached but unsettled vertices don't exist.
   - For each vertex t, which is a leaf node in the tree $B(v)$ move each edge (u, w), where $u(N_{\downarrow}H^{\uparrow}l(t), w(N_{\downarrow}H^{\uparrow}l(v)$ to the higher hierarchy level.

During the first stage, a highway hierarchy is constructed, where each hierarchy level G^l, for $l < L$, is a modified subgraph of the previous level graph G_{l-1}. Therefore, no canonical shortest path in G_{l-1} lies entirely outside the current level for all sufficiently distant path endpoints. This ensures that all queries between far endpoints on level $l - 1$ are mostly carried out on level l, which is smaller, thus speeding up the search.

2.2. Example of Highway Hierarchies Algorithm use. Let consider how the algorithm works for a simple graph. Let $L = 1$ and $H = 3$. First, N_3^0 has to be calculated for each vertex $v \in V$ using Dijkstras algorithm. The results are shown in table 2.1.

The construction of $B(v)$ for the example of vertex v_0 is shown above. This process is similar for other vertices:

1. Initial state of obtained v_0 is "active".

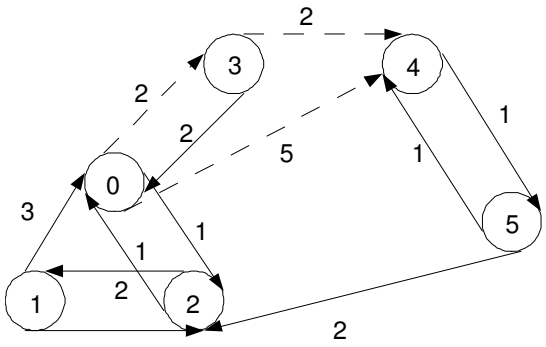


Fig. 2.1: Example of road network graph

2. Dijkstra's algorithm:
   (a) Vertex v_0 is settled, the path is empty.
   (b) Vertices v_2 and v_3 are reached from v_0 with, respectively 1 and 2. Their state is set to "active" (inherited from v_0).
   (c) Vertex v_2 with cost 1 is settled on the path $< v_0, v_2 >$. Passivity condition is not satisfied, because there are too few nodes on the path.
   (d) Vertex v_1 is reached from v_2 (cost 3) and its state is set to "active".
   (e) Vertex v_3 is settled with cost 2 on the path $< v_0, v_2 >$.
   (f) Vertex v_4 is reached from v_3 (cost 4).
   (g) Vertex v_1 is settled on the path $< v_0, v_2, v_1 >$ with cost 3. As $N_3^0(v_2) \wedge N_3^0(v_1) = \{v_0, v_2\}$, vertex v_1 stays "active".
   (h) Vertex v_4 becomes settled with cost 4 on the path $< v_0, v_3, v_4 >$. As $N_3^0(v_3) \wedge N_3^0(v_4) = \{v_3\}$, its state is set to "passive".
   (i) Vertex v_5 is reached from v_4 with cost 5 and its state is set to "passive" (inherited from v_4).
   (j) While there are no reached and active vertices, the algorithm terminates.
3. Leaf vertices are v_1 and v_5.
   (a) For vertex v_1 we iterate back on the path $< v_0, v_2, v_1 >$. For pair (v_1, v_2): $v_1 \in N_3^0(v_2)$ and $v_2 \in N_3^0(v_1)$. Therefore, that edge stays on level 0. The edge (v_2, v_0) also stays on level 0.
   (b) We perform the backward iteration process on the path $< v_0, v_3, v_4, v_5 >$. For example $v_\downarrow 3(N_\downarrow H^\uparrow l(v_\downarrow 5)$ and $v_\downarrow 4(N_\downarrow H^\uparrow l(v_\downarrow 0)$, so the $< v_3, v_4 >$ is moved to level 1.

The result of Highway Hierarchies algorithm is shown on figure 2.1. Dashed lines represent the edges on the level 1 and continuous lines represent edges on level 0.

3. Proposed road network division algorithm. Some parts of the construction phase of Highway Hierarchies algorithm can be performed concurrently:

- weight assignment for each road segment, in general using different rules,
- construction of N_h^l neighborhoods for each vertex in graph,
- construction of $B(v)$ trees.

We decided to try performing division of road network graph, so that Highway Hierarchies algorithm can be performed on a single part of this graph. After completion of the algorithm on each part, all subgraphs should be merged to obtain a final Highway Hierarchies graph.

To perform the devision of a graph, Breadth First Search (BFS) algorithm was applied for certain vertices as follows:

1. Get a list BFS_{start} of vertices mentioned to be start points for BFS.
2. For each vertex $v \in BFS_{start}$ create empty lists E_v and V_v to store the information about edges and vertices that belong to the subgraph.
3. For each vertex $v \in BFS_{start}$:
   (a) For the vertices from BFS queue, check if their children are allocated in any subgraph. If not, add them to BFS queue for current vertex and to V_v. Add corresponding edges to E_v.

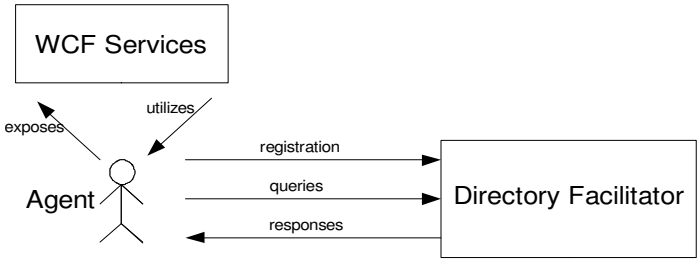


Fig. 4.1: System dependencies from the agent point of view

4. Check if all vertices of the base graph are not allocated in on of the subgraphs. If not, go to step 3.
5. Perform representation dependent postprocessing for each set V_v (i.e. reorder vertices if needed).

Such an algorithm can be applied to connected graph, if there are more than one connected components in the road network graph. Described division can be performed for each connected component treated as a base graph.

In the presented algorithm some edges can be not included in any graph. They can be denoted as E'. After performing construction phase of Highway Hierarchies algorithm on each subgraph, these edges must be included in the result graph. It is a two-step process:

1. For each vertex $v \in BFS_{start}$ add all of the vertices from V_v and all edges from E_v to the final graph.
2. For each edge $e \in E'$ that connects vertices V_s and V_d get the highest hierarchy level from all incoming edges of vertex V_s.

4. Multi-agent systems and its application for road network hierarchization problem. The standards of architecture for multi-agent systems were described by FIPA organization [7]. Due to this specification, multi-agent system consists of some number of Agent Platforms that were as a parts of the system and they can be used to host the agents. Each Agent Platform consists of three parts to handle the management of agents:

- Message Transport System (MTS) that is supposed to be used by the agents for communication,
- Agent Management System (AMS) that represents a catalog of existing agents,
- Directory Facilitator (DF) that stores the information about the services provided by the agents.

The analysis of modern programming techniques shows that some practices can be applied in newly designed multi-agents systems:

- use Service Oriented Architecture (SOA) to simplify and improve the possibilities of agent communication,
- make Directory Facilitator the mandatory part of multi-agent system,
- try to apply the enterprise design patterns such as dependency injection to coordinate the communication of agents on a single machine,
- simplify the architecture using the Windows Communication Foundation (WCF) [6, 7].

The introduction of web services allowed the developers to connect the applications based on different software and hardware platforms, for example Java and .NET Framework. The Web Services use a specific protocol to expose a schema of transferred data and allow the clients to make the synchronous calls of exposed methods [3].

Some generalization of such a system is described in [3]. In this approach the extended Directory Facilitator component plays the major role in the system because it keep all the information about services offered by the agents. All the services offered by agents are Web Services or some of their extensions such as WCF services (Fig. 4.1) [6].

4.1. Application of agents from building the road network hierarchies graph. Application of multi-agent system for building Highway Hierarchies graph was proposed in [8]. Two main assumptions were made for proposed application of multi-agent system for building Highway Hierarchies graph:

- system must be able to take into account the user's preferences (i.e. route should be the shortest, traveling time should be lowest) and environmental conditions (i.e. weather, time of a day),

- computations should be done concurrently, where it is possible to be done.

To complete the first of these assumptions, weights of the road segments must be assigned using different criteria, such as length, average traveling time, speed limits, etc. It was decided to introduce some number of reactive agents that collect the data from different road segments. This type of agents can work in two different ways, depending on the data structure which is used to store the road network technology. The first way is associated with the nodes as it is easy to get information about edges connected to the node. Second way is related to edges. If list of edges in the graph is directly provided, it can be divided into some parts and each part can be analyzed by a single agent.

However graph are usually represented in a hierarchical way, where nodes are on the top level and data for edges is usually kept as a list for each node. The complete list of edges is helpful for weight assignment criteria based only on some properties of a single edge (i.e. length, speed limit). On the other hand, some important information can be kept in nodes one can consider criterion of avoiding bigger crossroads so that all of the road segments connected to such node should have its weight properly adjusted.

In our system both graph nodes and edges are kept in the separate lists. However references are duplicated and it simplifies the way of access to the needed data and allows the simple and complex weight assignment rules.

Regardless of the chosen solution, this process can be performed in parallel, what means sharing work for several agents. Depending on the selection criteria by which individual weights are calculated, work on each road section may perform one or more agents (each can calculate the weight using different method). If the weight of the segment is calculated on the basis of several criteria, use of a coordinating agent for the weights assignment process can be considered. The coordinating agent can calculate weight in accordance with certain rules (e.g. use the weighted average of the values calculated by the agents). Coordinating agent may have some adaptive abilities, depending on the application of the system [8].

Concurrent computation can be also applied in the other parts of Highway Hierarchies graphs creation process. Obviously, calculation of neighborhood N_H^l for each vertex is independent of each other. The only nuisance is that for each vertex, different queue of vertices intended to be visited must be kept. Any number of agents can be used to calculate such a neighborhood. Depending on the developer choice, these agents cooperate directly with agents responsible for assigning weights to graph edges or with the coordinating agent.

The creation of trees $B(v)$ is another process that can be done in parallel by agents for the individual vertices of the graph. This process should to be implemented through the cooperation with agents that build the neighborhoods.

The responses to user's queries for the system should take into account his preferences regarding the itinerary and the current conditions on the road. It might be necessary to create several Highway Hierarchies graphs, which will be used to obtain a system response depending on certain factors. Different graphs can be prepared for example for the city center during peak hours and at night. To implement this assumption, the introduction of a special type of agent can be considered. Such an agent will redirect the user query to the appropriate Highway Hierarchies graph. Relay agent may assist in work of coordinating agent by suggesting the criteria by which the weight of the edge should be calculated [8].

Proposed architecture of multi-agent system described above is shown on figure 4.2. The tests revealed that for diverse criteria the calculated hierarchies differ very much. The results obtained for three proposed criteria: speed limits, traveling time and road length, shown that these hierarchies graphs have at each level only a few common edges with other hierarchies graphs. Moreover, expected convergence between dominating user's preference and number of common edges with the hierarchies graph for this criterion was observed.

4.2. Fastening Highway Hierarchies graph construction process. Application of multi-agent system described above shown that such architecture can be successfully used both for handling user's preferences and speeding up construction process of Highway Hierarchies graph. In order to apply improvements described in section 3, an architecture of multi-agent system must be significantly changed.

There are two new types of agents required to perform such process:

- *Graph splitting agents*, which are supposed to prepare proper split of the graph and pass the information to corresponding neighborhood calculators. Graph splitting agents can be considered as social agents as they have to cooperate with other graph splitting agents while doing their work, because some edges can be allocated in different subgraphs.
- *Graph merging agent*, which task is to merge subgraphs prepared by splitting agents according to certain

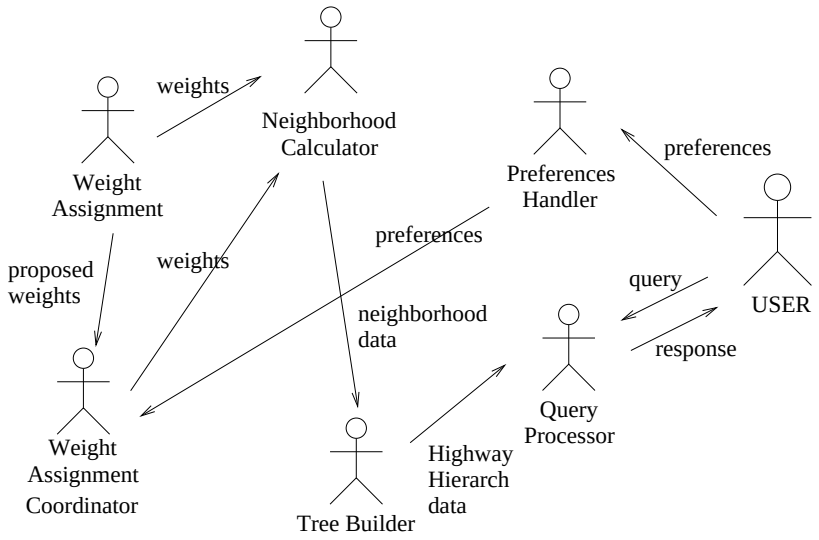


Fig. 4.2: Types of agents in Highway Hierarchies algorithm

rules. Although the work of this agent looks complicated, it is reactive agent it has to wait for the graph splitting agents to complete their work, so that he can perform the merging process.

Introduction of new types of agents to the system implies changes in the data flows in the system. *Preferences Handler agent* still cooperates with *Weight Assignment Coordinator agent* in order to pass the information about user's needs. *Graph Splitting agents* can divide graph into subgraphs and then cooperate with *Neighborhood Calculator agents* and *Tree Builder agents* to prepare partial Highway Hierarchies graphs. Next, *Graph Merging agent* can prepare final division of the road network into hierarchy levels. *Query processor agent* cooperates directly only with *Graph Merging agent*. Dependencies between other types of agents remain unchanged.

5. Results of system application. Algorithm described above was implemented using C# 4.0 language in Windows environment. Tests were run on different maps both for single and split road network graph. Exemplary result of full graph division is shown on figure 5.1. Figure 5.2 presents the results of algorithm performed on a subgraph.

In general, road segments on the top hierarchy levels in the full graph are on high hierarchy level in a part of the graph too, overall number on the highest hierarchy level is smaller for the subgraphs. It is caused by the fact that when the number of edges is smaller, promotion to the higher hierarchy level is harder.

The tests performed for both maps shown that the time needed to execute the calculations depends in slight degree on the number of edges in the subgraph. In case of real road map, both subgraphs contain after a division the exact number of vertices. However, the first part contains significantly more edges. The time needed to compute the hierarchy levels for both parts were almost identical. The second graph was an artificial road mesh, where after a division the number of edges was the same in the both parts. Performed tests shown that the time of computing for both subgraphs was also the same in this case.

Highway Hierarchies algorithm was run for the whole artificial and real road graph using such parameters:

- maximum hierarchy level: 3,
- Dijkstras neighborhood size: 5.

In the next step, Highway Hierarchies was run for the subgraphs with identical parameters. Obtained results for subgraphs show that when a division is made, number of edges on each level differs from number of these edges when the whole graph is taken into account in Highway Hierarchies. For real road network, such a difference was up to 70%. It was caused by the fact that large number of edges was included in the first subgraph. This difference for the artificial road network was smaller then in the real network.

It is harder to reach higher hierarchy level in smaller graphs (larger graphs usually contain longer paths). Therefore, we decided to decreases the size of Dijkstra neighborhood used in the division into the hierarchies. This was supposed to facilitate the promotion to higher hierarchy levels (step 2*b* of Highway Hierarchies al-

Fig. 5.1: Example of Hierarchical Division for neighborhood of Technical University of Lodz HH(3,5)

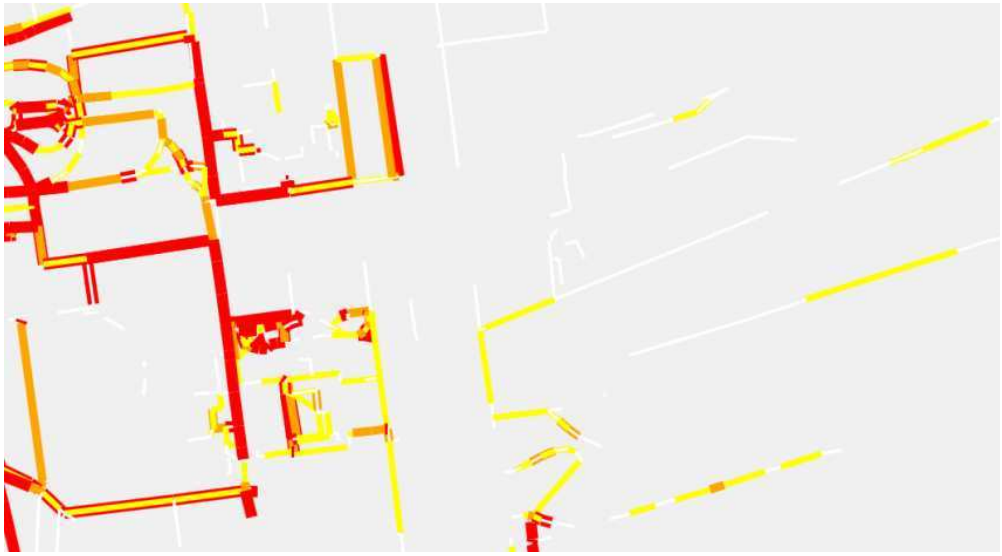


Fig. 5.2: HH algorithm performed for a subgraph HH(3,5)

gorithm). In the case of real road network, number of edges on each hierarchy level was closer to one in the reference division (about 37% better result). In the case of artificial road graph, the difference in number of edges on each level between result obtained for Highway Hierarchies computed for whole graph and for two subgraphs was about 7%. Generally, decreasing size of Dijkstra neighborhood resulted in lower differences between reference hierarchical division and a division made for subgraphs.

The second test was performed to check, whether HH algorithm for spitted graph is faster than the algorithm ran for the whole graph. Results are gathered in the table 5.2. Values are given in percents that represent the amount of time needed to perform each phase of the algorithm in addition to time needed to construct Highway Hierarchies for the whole graph. When the algorithm is supposed to build the hierarchies with greater number of levels, the gain is the highest. When number of maximum level is set to 1, the gain is not so high.

Table 5.1: Number of edges on each level for different HH parameters and maps

Road Network	Level	HH(3,5), Whole graph	HH (3,3), 2 subgraphs	% diff	HH (3,5), 2 subgraphs	% diff
Technical Univ. of Lodz	0	707	750	6,08203678	922	30,41018
	1	451	343	23,9467849	274	39,24612
	2	136	131	3,67647059	93	31,61765
	3	613	234	61,8270799	177	71,12561
Artificial mesh	0	242	215	11,1570248	251	3,719008
	1	264	239	9,46969697	261	1,136364
	2	184	170	7,60869565	184	0
	3	440	452	2,72727273	380	13,63636

Table 5.2: Time amount needed to complete each phase of the algorithm for split road graph in addition to time needed for HH construction for whole graph

HH parameters	First subgraph	Second subgraph	Merge	Split
Technical University of Lodz				
(3,5)	10,7%	11,4%	3,5%	2,1%
(3,3)	7,3%	7,5%	5,0%	3,0%
(1,3)	11,8%	11,9%	25,4%	15,2%
Artificial mesh				
(3,3)	18,0%	17,6%	18,5%	18,9%
(3,5)	19,6%	19,4%	9,5%	9,7%

6. Conclusions. The presented paper focuses on the problems of efficient vehicle routing. Nowadays these problems are very important because of increasing number of vehicles on the roads. More and more drivers have the abilities of use the devices to support the planning of their itinerary and it is important to find the new solutions, new algorithms to improve the applications for finding the best routs taking into consideration different criteria, static and dynamic. The use of agent concept and use of multi-agent approach seems to be the interesting solution for solving the problems with vehicle routing and finding the optimal itinerary.

Performing the hierarchical division of road network on the split of the road graph can improve the construction phase processing time due to the lower computational complexity. Number of edges on certain levels in subgraphs can differ very much from these from division of whole graph. Adjusting hierarchical algorithm parameters can improve results of divisions of subgraphs.

Multi-agent system can be utilized to solve this problem, what allows to compute most parts of the algorithm in parallel. Architecture of presented multi-agent system is extensible. There is a possibility to implement new types of agents for different graph split methods.

REFERENCES

[1] P. Sanders and D. Schulters, *Engineering highway hierarchies*, LNCS 4168, pages 804-816, 2006.
[2] P. Sanders and D. Schulters, *Highway hierarchies hasten exact shortest path queries*, LNCS 3669, pages 568-579, 2005.
[3] L. Chomatek and A. Poniszewska-Maranda, *Modern Approach for building of Multi-Agent Systems*, LNCS 5722, pages 351-360, 2009.
[4] G. Nannicini, P. Baptiste, G. Barbier, D. Krob and L. Liberti, *Fast paths in large-scale dynamic road networks*, Computational Optimization and Applications , 45 (1), pages 143-158, 2008.
[5] D. Eppstein, M. Goodrich and L. Trott, *Going off-road: transversal complexity in road networks*, Proceedings of 17th ACM SIGSPATIAL International Conference on Advances in Geographic Information Systems, pages 23-32, 2009.
[6] C. Vasters, *Introduction to Building WCF Services*, MSDN Library, 2005.
[7] FIPA, *Abstract Architecture Specification*, 2002.
[8] L. Chomatek, *Multi-agent approach for building highway hierarchies graph*, Proceedings of 31th International Conference Information Systems, Architecture and Technology, Szklarska Poreba, Poland, September 2010.
[9] M. Wooldridge, *An Introduction to MultiAgent Systems*, John Wiley & Sons, 2002.
[10] G. Weiss, *Multi-Agent Systems*, The MIT Press, 1999.
[11] M. Singh and M. Huhns, *Readings in Agents*, Morgan-Kaufmann Pub., 1997.

Edited by: Dana Petcu and Jose Luis Vazquez-Poletti
Received: August 1, 2011
Accepted: August 31, 2011

Scalable Computing: Practice and Experience
Volume 12, Number 3, pp. 293–305. http://www.scpe.org
ISSN 1895-1767

SUPPORT OF SEMANTIC INTEROPERABILITY IN A SERVICE-BASED BUSINESS COLLABORATION PLATFORM

KAROL FURDÍK[†‡] PETER BEDNÁR[†] GABRIEL LUKÁČ[†] AND CHRISTOPH FRITSCH[§]

Abstract. The paper describes a system prototype that is aiming at the provision of an environment for flexible project-oriented collaboration of networked enterprises, so-called virtual business alliances. The system, which was developed within the FP7 project SPIKE, employs the technology of semantically enhanced service bus, supported by underlying semantic structures such as ontologies and abstract business process models. The focus of this approach is to achieve effective interoperability of possibly heterogeneous services provided and consumed by members of the collaboration environment. The architecture of the main system modules is presented and the processing of semantically annotated services orchestrated in a collaboration workflow is explained in more details.

Key words: semantic annotation of services, business process ontologies, semantic interoperability, networked enterprises

AMS subject classifications. 94-04, 94B99

1. Introduction. Recent achievements in information and communication technologies (ICT) towards cross-network communication, cloud computing, service-based architectures and standardised interfaces bring new opportunities and challenges in many areas. When applied to the e-Business domain, these advanced technologies can improve flexibility and adaptability of business collaboration in so-called networked or cloud enterprises [26], which can be established as a temporal inter-organisational business alliance of enterprises and organisations co-operating on a well-defined project. Such a collaboration is usually characterised by rapidly varying business environments and requires proper technological background enabling interoperable provision and consumption of services between alliance members. Moreover, business processes in this type of networked enterprises need to be defined, structured and maintained on the alliance level as an inter-company composition of particular processes and services provided or consumed by the alliance participants in an interoperable manner.

The concept of interoperability in its technical, organisational, and semantic aspects was identified and emphasised in numerous initiatives and reports as a crucial, cross-cutting task in e-Business field [25] and other related areas. Semantic interoperability, which will be specifically addressed in next sections of this paper, refers to seamless service invocation, communication, and information exchange in an ICT environment of e-Business solutions based on principles of Service Oriented Architectures (SOA). According to this approach, services, which are formally described (i.e. annotated) by concepts of a standardised and shared knowledge base, can be provided, accessed, orchestrated, invoked, executed, and used in a flexible manner. Inputs, outputs, and other characteristics of possibly heterogeneous services can be semantically matched and integrated, enabling composition of services into customisable workflow structures. Moreover, the advanced technology of *Enterprise Service Bus* (ESB) [9] can be combined with the underlying semantic infrastructure and be employed to mediate potential incompatibilities of communicating services and applications, orchestrate their interactions, and make the integrated services available for broad access and re-use [20].

Various approaches for ICT solutions that enable networked enterprises by means of semantically enhanced ESB were designed and proposed, namely in several European research and development projects integrated in the FInES cluster [26]. Some of the most relevant projects, together with related approaches and technology frameworks, are discussed in the following subsection. In this context, we will present a specific approach for achieving semantically interoperable services, which was designed and adopted in the European project SPIKE (*Secure Process-oriented Integrative Service Infrastructure for Networked Enterprises*, FP7-217098, http://www.spike-project.eu). The project lasted from January 2008 till March 2011 and was co-founded by the European Commission within the 7th Framework Programme. The project consortium consisted of eight partners from five European countries and was coordinated by the University of Regensburg, Germany.

[†]Technical University of Košice, Faculty of Electrical Engineering and Informatics, Letná 9, 042 00 Košice, Slovakia. (Karol.Furdik@tuke.sk, Peter.Bednar@tuke.sk, Gabriel.Lukac@tuke.sk).

[‡]InterSoft, a.s., Floriánska 19, 040 01 Košice, Slovakia. (Karol.Furdik@intersoft.sk).

[§]University of Regensburg, Department of Information Systems, Universitätsstraße 31, 93053 Regensburg, Germany. (Christoph.Fritsch@wiwi.uni-regensburg.de).

SPIKE targets the interoperability of services and processes in networked enterprises, focusing on the design and development of a software platform that enables easy, fast, and secure start-up of virtual business alliances. Particular emphasis is given on semantically enhanced business processes that are capable to integrate heterogeneous services provided and consumed by alliance members, security aspects of service access, single-sign-on principles and identity federation, which is supported by the ESB-based semantic infrastructure of ontologies, services, and business processes [15]. The description of the SPIKE system, together with its principles and main outcomes, is presented in this paper. The remainder of this paper is organised as follows. Section 1.1 brings an overview of related research, including related projects and supportive technologies that were employed in SPIKE. The vision, objectives, and general approach of SPIKE are presented in Section 1.2. The following sections specifically focus on individual aspects of our designed and developed solution. Section 2 describes an overall system architecture, particular functional components and their interactions. The semantic infrastructure for service annotation and workflow specification is presented in Section 3. It includes a description of semantically enhanced ESB, means of dynamic service selection, composition, and execution in the environment of business alliances. Section 4 provides an overview of pilot applications and the results achieved during the testing of the system prototype. Finally, Section 5 summarises the project outcomes and identifies possible directions of future research and development.

1.1. Related Approaches and Technologies. The central concept of the SPIKE solution (cf. Section 1.2 and Section 3) is a semantically enhanced ESB that allows interoperability of services provided or consumed by participants of business alliance. It is based on general ESB technology, which can be seen as a messaging and communication middleware that defines standardised service interfaces and message routing for possibly heterogeneous and incompatible applications or services [9]. The implementation of the ESB technology in an organisation typically requires a specification of workflow structures that model particular processes, tasks, actions, and flow of information artefacts and messages between communicating applications. Construction of these workflow-based models is known as business process modelling and is nowadays mostly handled by the standardised BPMN (*Business Process Modelling Notation*, http://www.bpmn.org) and BPEL (*Business Process Execution Language*, [2]) notations. However, the ESB infrastructure itself and the notations for process modelling are both focusing on syntactic specification of interfaces, message exchange, and workflow structures. An effective integration of services and processes, which would be based on the meaning expressed in a machine-readable way, was identified as one of key challenges in ESB-related research [16].

The vision of semantic business process modelling, formulated in [17] and further elaborated, for example, in the European FP6 integrated project SUPER [4], aims at achieving a higher degree of automation in discovery and mediation of co-operating services [27]. The use of semantic technologies, namely Semantic Web services and underlying ontologies, for process modelling, service configuration, execution, monitoring, and analysis is envisioned as a method that can overcome the heterogeneity and incompatibility problems towards the semantically interoperable services. It may also help to reduce the human intervention throughout the life cycle of business process modelling [21]. The *Semantic Service Bus*, which can be seen as an enhancement of general ESB, makes use of semantic description of service capabilities, properties, and exchanged information artefacts, which then enables the service integration by means of automated service discovery, routing, composition and data mediation [20].

Semantically enhanced business process modelling, workflow management, and design of semantic ESB is in focus of research and standardisation organisations such as the Object Management Group (OMG, http://www.omg.org), W3C consortium (http://www.w3c.org), OASIS group (http://www.oasis-open.org), or Workflow Management Coalition (WfMC, http://www.wfmc.org). In the European context, particular solutions were provided as outcomes of several FP6 and FP7 research projects, mostly integrated in the FInES cluster initiative [26]. Some of the projects, identified as relevant and related to the SPIKE approach, are:

- STASIS (FP6-034980, http://www.stasis-project.net) provides an infrastructure for semantic mapping of services by means of a distributed peer-to-peer repository and shareable ontology structure [5];
- TrustCom (FP6-001945, http://www.eu-trustcom.com) has designed a framework for trust, security and contract management for service-based collaboration of networked enterprises;
- SUPER (FP6-026850, http://www.ip-super.org) provides a framework for semantic business process management, including generic formal languages, process models, and shareable ontology resources [4];
- COIN (FP7-216256, http://www.coin-ip.eu) targets the long-lasting enterprise collaboration, networking, and interoperability by integrating services and business processes in a generic service platform [19];

- NisB (FP7-256955, http://www.nisb-project.eu) is aiming at a provision of user-centric tools for hierarchical interoperability of enterprises, by means of designing and applying various business model archetypes and principles of dynamic business ecosystems.

The novelty of the SPIKE approach, in comparison to the above-mentioned projects, lays in the design and implementation of a light-weighted ESB framework, which is strongly supported by underlying semantic structures. The holistic and robust infrastructure, proposed in projects such as STASIS, SUPER, or COIN, was reduced in SPIKE to a rather simple and straightforward mechanism of semantic service bus, presented in more details in Section 3. From the technological perspective, the SPIKE solution is built on the WSMO framework (*Web Service Modelling Ontology*, http://www.wsmo.org), namely on its simplified variant WSMO-Lite [30]. The underlying semantic knowledge base was created in SPIKE following a generic and reusable methodology [15], which transforms user requirements, provided as textual descriptions of application cases, into a ready-to-use implementation of ontologies and business process models (cf. Section 3). In addition, a part of the knowledge base, namely the ontologies for modelling service properties and business process characteristics, was created by reusing ontologies of the SUPER project and other well-established semantic structures such as Dublin Core (http://dublincore.org), SKOS [24], etc.

Similarly as in the NisB project, a set of abstract models of sub-processes were developed in the BPMO notation [4]. WSMO Studio (http://www.wsmostudio.org) was used in SPIKE as a general toolkit for design and implementation of all the semantic representations, including ontologies, abstract business process models, and semantic annotations of services. Finally, the ESB runtime environment was designed to employ the semantic knowledge base for service mediation and orchestration in workflow sequences, directly focusing on information exchange in short-term business alliances. The results of TrustCom and STASIS projects were considered during the design of SPIKE architecture, namely in the specification of modules for business alliance maintenance and security [12].

1.2. SPIKE Objectives and General Approach. The SPIKE project primarily focused on the design, implementation, and testing of a software service platform that is capable to support a transparent, easy-to-use, and effective collaboration of organisations within an environment of virtual business alliances. At the organisational level, the project objectives were defined with respect to simplify the collaboration between members of a business alliance and to enable outsourcing of parts of the value chain to business partners (and vice versa, offering such parts in a form of services) in short-term business alliances [15]. Special focus was given on security and trust during all phases of alliance life cycle. The scientific and technology objectives of SPIKE targeted research, development, implementation and validation of software components for semantically enhanced business process management environment, which are capable to handle customised reference processes, ad-hoc defined workflow structures and distributed processes built from generic process fragments.

With respect to the defined objectives, three levels of collaboration were identified and designed as operational modes for the overall platform as follows:

- *Collaborative processes* that enable to produce physical or intangible artefacts and are modelled by means of complex workflow patterns;
- *Sharing services*, where alliance partners can offer their services in the scope of a given business process. Offered services can be retrieved, negotiated, contracted, and finally be used (i.e. invoked and consumed) by other alliance members according to the conditions specified in the service contract;
- *Identity federation*, enabling and mediating access of alliance members to the internal resources or services of other partners.

The approach adopted in the SPIKE project, aiming at a support of all defined collaboration levels, is schematically depicted in Figure 1.1. Three phases of the alliance life cycle, i.e., setting-up, running, and closing down, were addressed together with proper security settings and federation of identities enabling to access internal services or information resources of an alliance partner in a single-sign-on mode [7]. Business organisations, presented in the top bar of the Figure 1.1, may decide to form an alliance focused on a production of an artefact. A collaborative value chain, which determines particular steps and the main target of the short-time alliance, is defined as a first step. The value chain is then modelled and expressed in the Conceptual Layer in the standardised BPMN notation and can be further semantically enriched by means of the concepts from a shared knowledge model [17]. Resources and services of participating organisations can then be mediated and integrated according to known and formalised meaning. On the Service Layer, particular tasks in the process model are grounded to executable services provided by the alliance partners. It allows sharing and using the

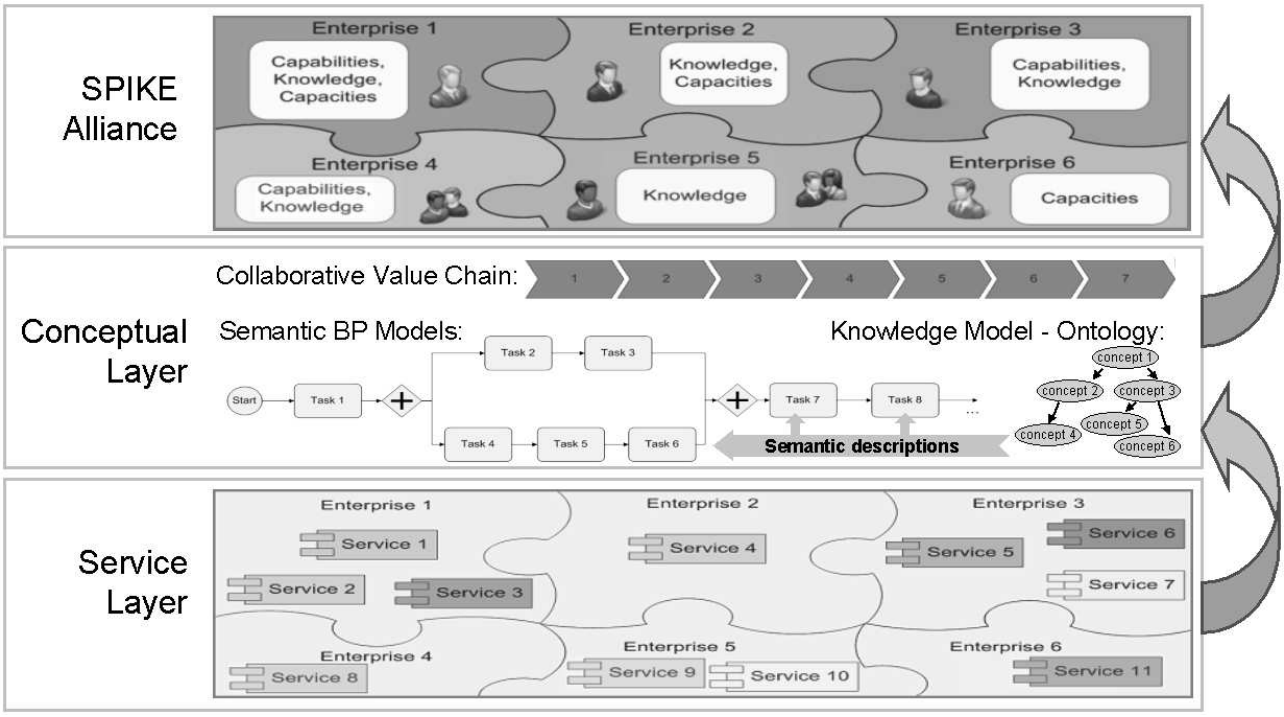


Fig. 1.1: Basic schema of the SPIKE approach for networked enterprises

services in the scope of defined processes by authorised organisations. Identities and credentials necessary for the service invocation are distributed to the authorised users in a secure way. The alliance can then operate according to a dynamic process model, which, if needed, may be modified and adapted during run time.

2. Architecture of the SPIKE Business Collaboration System. The described high-level functionality of a collaborative system for networked enterprises led to the design of the SPIKE system architecture. In accordance with the SOA principles, the architecture was proposed as highly modular and extensible [22]. The methodology of [28] was adopted to identify the viewpoints, perspectives, and stakeholders of the envisioned system. User partners of the SPIKE project, responsible for particular pilot applications (cf. Section 4), provided initial descriptions of required functionality from their perspective [31]. These descriptions have subsequently been used as a background for the specification of system views and perspectives as well as a platform for the validation of the system design.

2.1. Information View, Data Elements. The information view, as an initial step during architecture design, defines a structure of data elements and information resources that are stored and manipulated by the system. The design of data structures was accomplished by analysing the descriptions and requirements of user partners on information and data types that may be exchanged within the business alliance environment of the envisioned functionality [15]. The analysis resulted in a design of the main data elements, as presented in Figure 2.1.

The *Process*, *Workflow*, and *Task* elements are basic building blocks for modelling an alliance of collaborative business processes. The Task element, representing particular workflow actions, is further specified by the parameters such as inputs, transformations, and outputs. These parameters, consumed and produced by a task in a workflow, are represented by a set of sub-types of the generic and abstract *Resource* data element. Resource defines a common set of properties inherited by all its child data elements, in particular by the resource types such as *Document*, *Service*, *Report*, *Message*, etc. Properties of these information resources are provided as semantic metadata defined in an ontology schema (cf. Section 3). This solution enables to combine the standardised business process modelling with semantic descriptions created according to the Semantic Web principles [17]. The semantic knowledge base is represented by the *Ontology* and *Metadata* elements. These

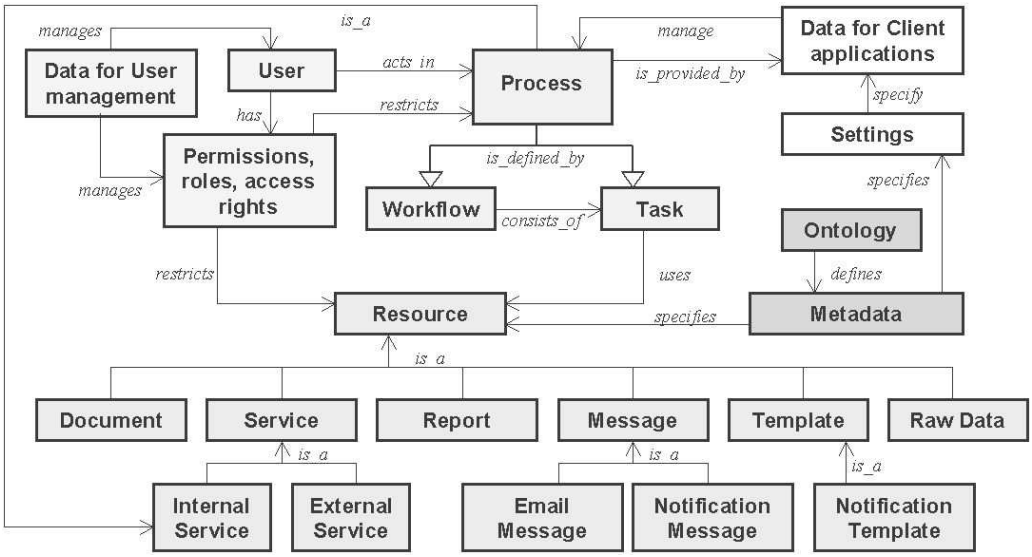


Fig. 2.1: Data elements and their structural relationships

elements store and provide both the metadata schema as well as instantiated data that specify and semantically describe the elements of other information resources. Finally, the information view contains the data elements for user management, security, authentication, and system settings needed for configuration of the client-side tools.

2.2. Functional View, Structure of System Components. The functional view of the architecture, which defines particular system modules, their inner components and interactions between them, was designed as a structure of user interfaces, data modules, and business logic of the semantic service bus [22]. Four main functional subsystems, schematically depicted in Figure 2.2, were specified as follows:

- The *SPIKE System Core* (SSC), a back-end providing functions for accessing and processing all the system data, namely the data storage, security, and maintenance of semantically enhanced business processes, workflow sequences, and services;
- The *SPIKE Portal Instance* (SPI), a web-based user interface that acts as a front-end to the SSC functionality;
- The *SPIKE Administration, Monitoring and Reporting* (SAMR), a subsystem that provides tools for overall system maintenance and day-to-day operation;
- The *SPIKE Service Bus* (SSB), an infrastructure that handles the communication between other SPIKE subsystems and external entities.

Each subsystem is further divided into a set of loosely coupled components, so-called managers, which provide autonomous and elementary functionality. The components of SSB and SSC subsystems are responsible for semantic workflow maintenance, including mediation, orchestration and execution of services in a pre-defined business process that corresponds to the alliance value chain.

Managers integrated into the SSC provide the core functionality of the overall system and are responsible for manipulations on the service level. The *Content Manager*, a very central component of the SSC, provides means for storage, retrieval and update of all data presented in and brokered through the SPIKE infrastructure, namely the ontologies, service registrations and descriptions, user sessions, and metadata of identity management. The *Service Manager*, which implements a built-in Web Service engine, provides discovery and execution capabilities for the services integrated into a process workflow. The *Semantic Manager* handles all functionality involved in dealing with semantic information, namely the semantic search, matching, mediation, mapping, and reasoning over semantically described data. Semantic metadata descriptions of services, sub-processes, and artefacts, which are specified as input and/or output parameters of services, are maintained and provided by the Semantic Manager by means of metadata mapping.

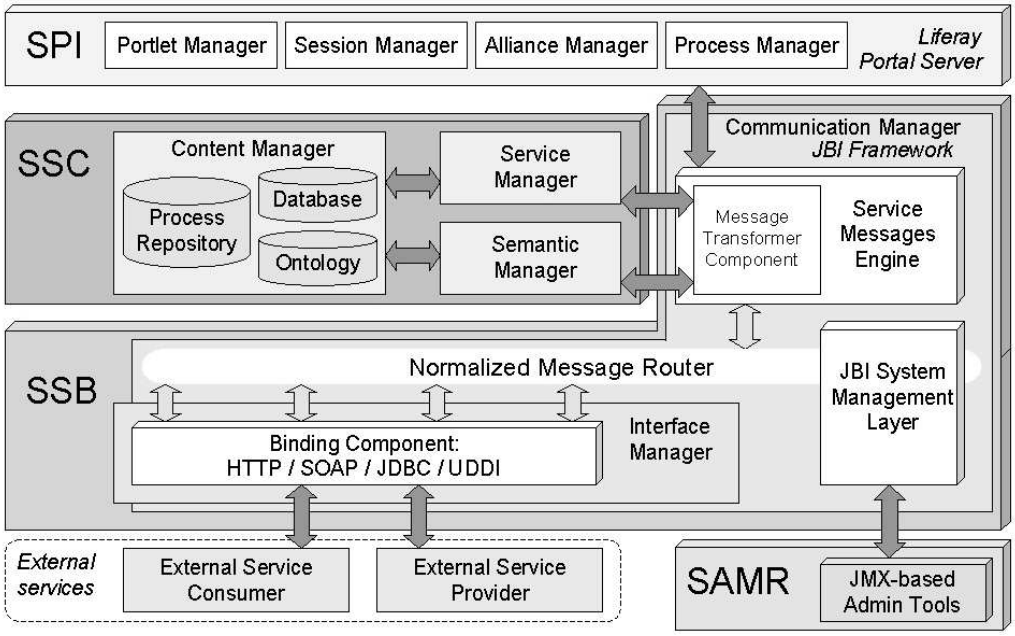


Fig. 2.2: Functional subsystems and components of the SPIKE platform

The business logic of the SSC managers is transmitted to the rest of the system by means of the SSB functionality. SSB serves as a central communication channel that handles messaging and data exchange between the system core, user interface, and administration parts of the SPIKE infrastructure. The *Interface Manager* is the only component employed by other managers to interact with external services. It provides basic capabilities for service usage, i.e. for connecting the services and transmitting service requests and responses to other components of the platform. The Binding Component acts as a proxy to remote services. It makes the services available to the service bus in a form of normalised messages, independently of the service's actual transport protocol and data format. The *Communication Manager* is an implementation of a semantically enhanced ESB, specifically designed for dynamic service selection and mediation (cf. Section 3). The service bus component is built on the Java Business Integration (JBI) specification JSR 208, using the Normalised Message Router as a central messaging backbone. The Service Messages Engine provides the lifting-lowering semantic transformations and the related business logic during the processing of external services (see also Figure 3.3 in Section 3.2). The grounded services are then orchestrated into a process workflow, exposed to the portal interface of SPI subsystem, and provided to the authorised alliance members.

3. Semantic Structures and Enhancements for Services and Business Processes. To support semantic message routing, service annotation and mediation in the environment of semantically enhanced ESB, the underlying semantic structures for the SPIKE system were built on the WSMO framework [23]. The value chain of a business alliance is semantically represented by an abstract business process model, which is implemented by the *Business Process Modelling Ontology* (BPMO) representation of WSMO [4]. The advantage of this approach is that BPMO is compatible with the standard notation of BPMN and, in addition, the underlying ontology format allows seamless and straightforward integration with other semantic elements. BPMO, as well as sBPMN, and sBPEL ontologies [18], published as outcomes of the project SUPER [4], were adopted as basic semantic structures for representing the elements of business processes in SPIKE.

The resource ontologies, which are capable to serve as a semantic base for annotating service inputs, outputs, non-functional properties, and various information artefacts exchanged between services in a workflow, were created in the format of *Web Service Modelling Language* (WSML, http://www.wsml.org). The developed resource ontologies can be divided into three logical groups [15]:

- *Process-related ontologies* provide conceptual models for semantic description of business processes and their elements such as *Process*, *Task*, *Service*, etc. Existing ontologies have been reused or new ones

derived from the above-mentioned BPMO, sBPMN, and sBPEL ontologies where necessary. Moreover, this ontology group contains the concepts enabling semantic annotation of services included in a SPIKE collaboration process and referenced by the Task data elements. It covers the WSMO-Lite ontology for semantic description of Web Services and a specific SPIKE service ontology that interconnects the service-related concepts with collaboration processes by means of human tasks and available types of online services.

- *System-related ontologies* semantically describe the platform environment. It includes concepts describing non-functional properties of WSMO services, SKOS classification schemes [24], concepts representing user profiles, and general concepts such as *CollaborationObject*, *Alliance*, *Contract*, *Organisation*, *Address*, *Person*, *Actor*, etc.
- *Domain ontologies* extend the conceptual models towards particular pilot applications of the SPIKE project (cf. Section 4). The domain ontologies, created from background materials and textual descriptions provided by the SPIKE user partners [15], cover areas such as identity federation, service contracting, authorisation and secure access to distributed legacy applications, and collaboration environment of documentation services.

Both the developed abstract business process models and resource ontologies are publicly available at http://www.spike-project.eu/BPmodels/ and http://www.spike-project.eu/ontologies/, respectively.

3.1. Semantic Annotation of Services and Processes. Developed semantic structures of ontologies and abstract process models can be seen as initial steps towards an orchestrated workflow of interoperable services. To anchor an abstract process model into real services and artefacts, its activity elements such as WSMO Goal tasks, Web Service tasks, and manual tasks need to be grounded to a concrete WSDL representation of executable services [10]. The semantic interoperability can then be achieved by associating the WSDL elements with proper ontology concepts that express the meaning of inputs, outputs, and characteristics of a service in a machine-readable way.

SPIKE adopts the specification of *Semantic Annotations for WSDL and XML Schema* (SA-WSDL) [11], which is probably the best known mechanism for semantic annotation of Web Services. It defines XML attributes for linking WSDL elements to the respective ontology concepts that may semantically specify service inputs, outputs, and types. The advantage of the SPIKE approach is that the WSMO framework directly supports the SA-WSDL annotation mechanism and the respective user interface is included as part of the WSMO Studio toolkit. It thereby enables seamless integration of WSMO ontologies, BPMO models of processes, and semantically described services. Additionally, SA-WSDL attributes can specify transformations between XML messages and related ontology instances, enabling semantic data mediation by lifting and lowering procedures from XML descriptions to ontologies and vice versa. This feature was employed in SPIKE for dynamic service selection and routing as described in Section 3.2.

Web Services, and online services in general, can obtain the WSDL descriptions inherently. However, in the case of SPIKE pilot applications, most services were of off-line type, where a human interaction was required. For this type of services, referenced in SPIKE as "human tasks", the description of properties can be modelled by means of standardised *XForms* format [6], while the *BPEL4people* extension [1] can be used to model these tasks in the executable process. In accordance with this technology background, inputs and outputs of all services that represent human tasks were enhanced in SPIKE pilot applications by respective SA-WSDL references to the semantic representations of artefacts required by service inputs and/or provided on the service output. Figure 3.1 presents sample XForms representation of input entry fields, which are required for a human task that initiates a collaboration process. The entry fields of the form are associated with proper ontology concepts in advance. During the workflow run time, a human actor in the process is asked to fill in the form fields with proper values, which are then automatically linked to the respective ontology concepts.

Abstract business process models of BPMO format specify an alliance workflow, consisting of a sequence of semantically annotated tasks. Tasks can be grounded to particular services of various types, including Web Services, electronic web forms, or offline services represented as human tasks. In principle, such an abstract process model, properly grounded and semantically described, can be semi-automatically transformed into its corresponding executable BPEL form. For such a transformation, SPIKE combines the *BPMO-to-sBPEL* translation mechanism [8] with the *Eclipse BPEL Designer* toolkit (http://www.eclipse.org/bpel/).

To process the executable workflow, a specific JBI runtime environment is created in the service bus for each of the SPIKE collaboration processes. Orchestration of services into a complex workflow is handled by

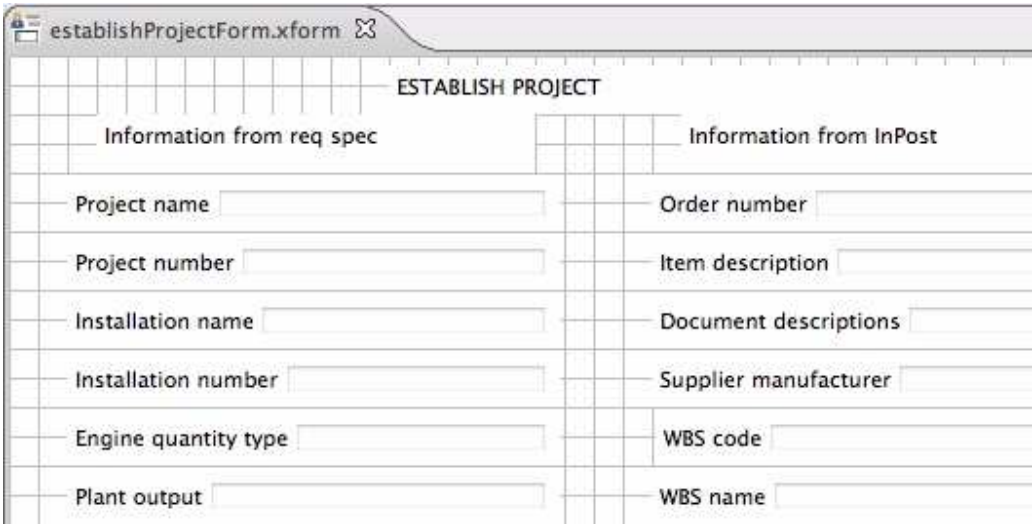


Fig. 3.1: XForms interface of input properties for a human task

the *Apache Orchestration Director Engine* (ODE, http://ode.apache.org), where a customised service resolving mechanism was implemented as an iterative selection of the best candidates for service execution, according to semantic Quality of Service properties.

3.2. Dynamic Selection and Mediation of Services in Semantic Service Bus. The presence of semantic annotations on all components of abstract business process models, i.e. on sub-processes, tasks, services, and exchanged artefacts, can facilitate the transformation to the executable workflow, as well as service interoperability, basically in two opposite modes. First, services may be linked to particular process tasks during the design time, using the correspondence between semantic descriptions of tasks and SA-WSDL attributes of services. This so-called "static service allocation", which is schematically presented on the left side of Figure 3.2, may be useful and advantageous to overcome heterogeneity problems between communicated services. The transformation of abstract models to executable BPEL processes is rather straightforward. The Invoke operation of BPEL is static; the service has to be furnished with a concrete WSDL of a service instance at the design time already. However, the role of semantic descriptions at run time is ignored or reduced to the matching of service inputs or outputs with provided or consumed artefacts. The drawback of this approach is that if such a statically allocated service is corrupted or not accessible for whatever reason at invocation time, then the running workflow is interrupted and may cause a failure of the whole process. Moreover, newly published services cannot be considered in the process model without altering and redeploying it.

The second option is so-called "dynamic service binding" [14, 13] and its schema is depicted on the right side of Figure 3.2. The *Process Layer*, which in SPIKE corresponds to the Process Manager component of SPI (see Figure 3 in Section 2.2), handles abstract process models and the respective semantic annotations of workflow tasks. Furthermore, this layer conducts the deployment of executable process representations to a workflow engine, as well as the execution and monitoring of running workflow. The *Mediation Layer*, i.e. an implementation of SSB, provides virtual interfaces (IF1-4 in Figure 3.2) for semantic descriptions of workflow tasks, which can be mapped to ontology instances and used for lifting and lowering transformations of SA-WSDL service descriptions. On the opposite side of the layer, there are binding components (BC1-4) to all available service instances that form the pool of service candidates. As a result, the Mediation Layer mediates between virtualised interfaces of workflow tasks and concrete instances of executable and available services. The *Service Implementation Layer* consists of the executable service instances, which are registered in the SPIKE service repository and properly contracted for usage within an alliance.

Dynamic service binding is implemented in SPIKE by means of the semantically enhanced ESB, as it is depicted in Figure 3.3. The process of service mediation is initiated by a service requester, which can be any stand-alone client or workflow engine. The requester sends a SOAP message containing a semantic description of required service to the message router. SSB then acts as a communication and messaging infrastructure

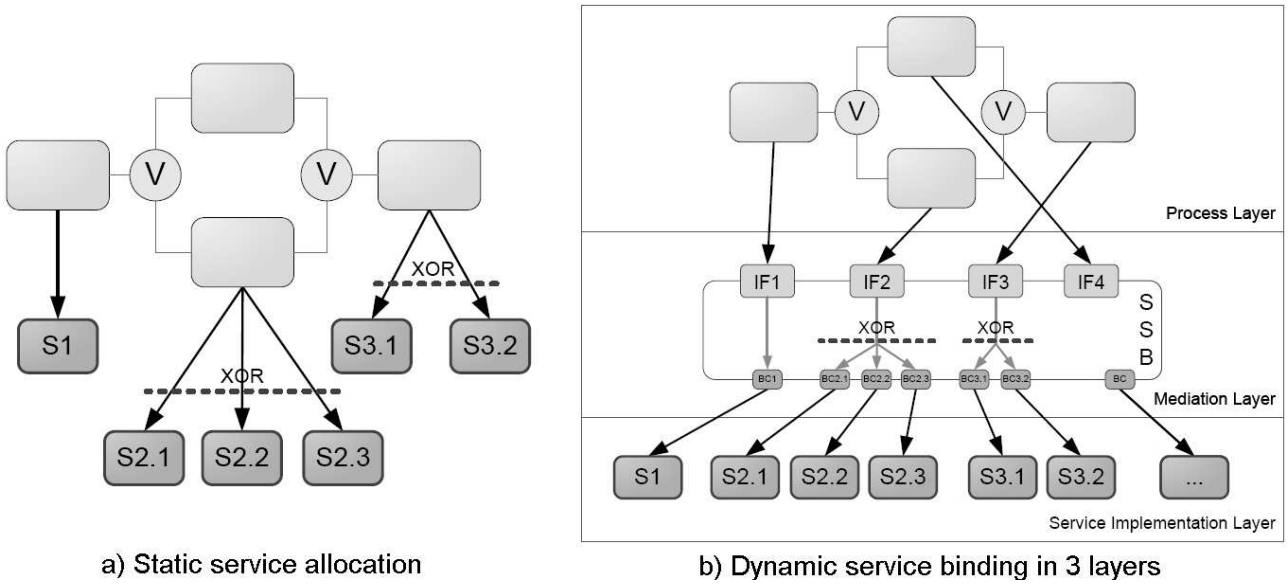


Fig. 3.2: Distinction between the static and dynamic service binding

and provides the JBI binding components for message sinks. The message router delivers the SA-WSDL virtual interface of the requested service to the Message Transformer, which forwards the interface to the lifting procedure. The virtual SA-WSDL description of the requested service contains definitions of required service properties, which are annotated with the *sawsdl:liftingSchemaMapping* attributes pointing to the instances of ontology concepts. These semantic instances are retrieved from the ontology and are used for semantic mediation and resolving of candidate services.

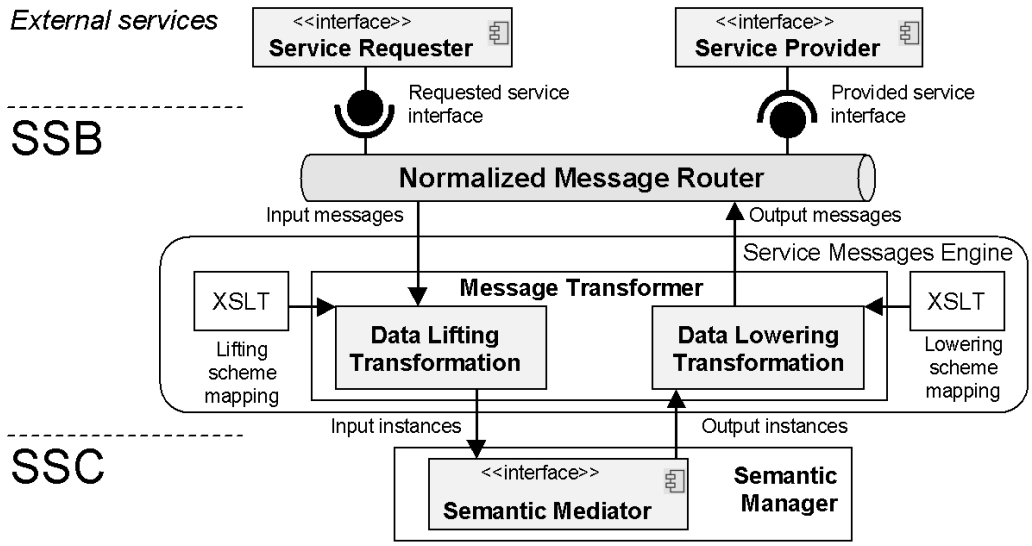


Fig. 3.3: Interactions during semantic transformation of service messages

The Semantic Manager component of SSC contains run-time implementations for ontology storage, maintenance, semantic mediation, validation and querying/reasoning over semantically described data on services and messages. In combination, it provides all required capabilities of the Mediation Layer for dynamic service binding, as presented in Figure 3.3. The service matching and resolving capability is based on semantic annotations

that may be assigned to virtual service interface using the *sawsdl:modelReference* SA-WSDL attribute. The annotations can be specified by concepts of the SKOS or domain ontologies. The semantic matching procedure enables an arbitrary combination of both types of annotation. During the subsequent reasoning, the hierarchical structure of categories and subclass/superclass relations of input/output types can be recursively expanded.

The process of semantic mediation is employed to overcome possible differences between the domain ontology that annotates the virtual service interface and the ontology of resolved service candidates. As a first step of the mediation, the lifting schema is used to transform the input semantic data from the virtual interface into a set of semantic instances. If there is a difference between the ontologies annotating input and resolved services, then the instances can be transformed from the source ontology to the target ontology by means of pre-defined semantic axioms and transformation rules (instance-to-instance transformation). Instances are transformed back to normalised messages using the lowering schema specified for the resolved service.

The developed solution of semantic ESB is based on the WSMO Lite framework [30]. The *Goal* and *Web Service* objects of general WSMO, as well as its choreography and orchestration mechanisms are not used. In-memory representation of ontology elements, together with facilities for ontology validation, parsing, and serialisation, is handled by the *wsmo4j* library (http://wsmo4j.sourceforge.net). Due to performance reasons, the WSML ontologies of SPIKE knowledge base were converted to the RDF format. Physical storage of RDF ontologies, implemented as a part of the Content Manager component, uses the *Sesame repository* (http://www.openrdf.org). Mapping of top WSMO Lite ontology elements into the RDF model is based on the *ORDI framework* (http://ordi.sourceforge.net), which allows integrating various data sources and provides a common RDF API for data access. Infrastructure components such as external service interfaces, runtime environment for service selection and mediation, were built on the JBI-compliant ESB *Apache ServiceMix* (http://servicemix.apache.org).

4. Prototype Evaluation on Pilot Applications. The SPIKE system was implemented as a prototype, which is now available under the LGPL / Apache 2.0 / X11 open source license. The prototype system was tested on pilot applications run by corporate project members in Finland and Austria. The individual application cases (AC) of the pilots focused on particular aspects of collaboration in a business alliance and can be summarized as follows:

- AC1: *Information Hotel* was dealing with scenarios in collaborative manufacturing and documentation development. This AC aimed at demonstrating the intra-enterprise collaboration on a process of creating a complete set of documentation materials for an industrial system. SPIKE provided an environment for maintenance of distributed documentation services by means of secure knowledge sharing and content management. Beyond that, the overall documentation development process and its involved actors have been represented and controlled by the SPIKE system.
- AC2: *Legacy Applications*, focused on the creation and management of business alliances, including maintenance of service providers (namely, the *Siemens DAMEX*©toolkit for maintenance of business contracts), location and configuration of services, integration into a workflow, as well as tracking, contracting, and ordering of services. A particular focus of this AC was the seamless and flexible integration of legacy applications and services into the virtual business alliance.
- AC3: *Identity Federation* demonstrated the management of user identities in a networked enterprise, namely the maintenance of access rights, credentials, roles, and resources within a collaborative environment. It particularly considered aspects of cross-organisational identity federation and the involved technical and organisational procedures.

The architecture and functionality of the SPIKE platform was designed as a generic solution, which is capable to support business alliances of any domain. However, the defined ACs were taken as a framework determining the scope and focus of the sample data for the system prototype. Namely, the semantic structures of ontologies and abstract process models, presented in Section 3, were created specifically for the mentioned ACs. Nevertheless, the solution of a semantically enhanced ESB, together with related mechanisms for semantic annotation and dynamic service binding, is rather general and can be adapted to a SPIKE application in any domain.

Design, implementation, and testing of the SPIKE system were performed in two rounds. The first phase of pilots testing mostly focused on proof-of-concepts implementations of research ideas and resulting functionality of particular components. The first trial phase was then accomplished in autumn 2009. This version was built upon the *Intalio BPMS Community Edition solution* (http://community.intalio.com), which served as

an environment for maintenance of executable processes. However, compatibility and licensing problems were encountered during the evaluation of this approach in the first round of pilots. For that reason we decided to substitute the Intalio environment in SPIKE by our own implementation of the Human Task Environment (cf. Section 3.1). The resulting solution enables an integrated management of processes, services, actors, and resources of virtual business alliances and was developed by employing the Eclipse BPEL Designer tool and Liferay portlet environment (http://www.liferay.com). This user side tool for maintenance of business alliances, referenced as Alliance Manager of SPI (see Figure 2.2 in Section 2.2), was functionally connected with the semantic ESB and other inner SSB/SSC components. In addition, the security environment, developed upon the Shibboleth framework (http://shibboleth.internet2.edu), was integrated to the SPIKE system to enable cross-company single sign-on and federation of identities between various external online services [12]. The second phase of SPIKE implementation was completed in winter 2010. The resulting system prototype was tested in January and February 2011 in the second trial of pilot applications AC1-AC3. Testing results, which are layed out in detail in [3], indicate the suitability of the overall SPIKE approach and prove the prototype functionality for service interoperability, secure and flexible collaboration in an environment of business alliances.

In more detail, we have learned during the trial runs that technological answers alone, even if they prove to be very well applicable, only slightly advance virtual business alliances. Additional aspects such as "ease of setup" and, even more important, "ease of use" will be among the non-technical key factors for the success of any kind of software that aims at supporting virtual business alliances. Similarly important is the consideration of organisational and social aspects (our insights can be looked up in [3] and several other SPIKE publications). Nevertheless, the trial results regarding the technology produced some meaningful findings. One major goal was to investigate the maturity and possibilities of semantic technology combined with portal and service bus architecture to resolve issues in managing technical information and technical information flow in networked enterprise. Our results show that there are plenty of possibilities in software for collaboration platform, in underlying technology and especially in semantic modelling of business processes. However, commercial phase and maturity of this technology is still quite far away from the level that it could be taken into use without extensive support and error-situation and reliability improvements. This is especially due to large number of involved users and their low IT competence. According to our experiences, the SPIKE platform has satisfactory potential to improve information management efficiency in networked enterprises even if final estimations are difficult. A tentative return on investment (ROI) calculation for AC1 indicates that in the current business process 20% cost savings could be made with 45% deployment rate in the customer base of the industry partner for AC1.

Beyond that, the trial results show that grasping the concept and benefits resulting from semantic annotations is still quite difficult even for inexperienced developers, let alone regular fellow employees. It still requires major efforts and knowledge engineers. However, if a 'semantic basis' is created, the resulting gains are considerable. Therefore, our approach of designing the system in a 'semantic agnostic' manner, meaning that in case services and processes are not semantically annotated, components can be still used (with some limitations regarding flexibility) proved to be very convenient. This allows that services can be just gradually annotated in the cases when needed. The same proved right for the consideration of legacy services and the integration of human tasks. While 'real' web services are currently still emerging in many companies, a huge amount of legacy services and manual tasks prove their capability in day-to-day business and are therefore worth integrating in collaboration platforms as the one presented.

5. Conclusions. The presented system prototype, designed and implemented within the European FP7 project SPIKE, provides an environment for service-based and process-oriented collaboration in virtual business alliances. The solution aims at effective support of service interoperability, which was achieved by employing semantically enhanced service bus that enables both static and dynamic binding of concrete executable services to the representations of tasks in a pre-defined abstract process model. Semantic structures employed for the solution were built upon the WSMO Lite framework [30], enhanced by customised service resolving, mediation, and orchestration mechanism. This light-weighted approach brings more flexibility to the composition and maintenance of applications employing ESB for intra- or inter-enterprise service-based collaboration.

The prototype of the SPIKE system was tested in application cases covering the main aspects of collaborative processes in business alliances, namely the collaborative manufacturing, inclusion of external legacy applications, and identity federation. Achieved testing results proved the usability of the developed solution [3], namely the capability to integrate various heterogeneous services in an interoperable manner. However, further

enhancements could be considered, for example, towards a more advanced support of service contracts between the alliance partners. It can be achieved by providing a support for formal operational level and service level agreements, as well as by adopting other standardised processes of service management and operation [29]. More information about the SPIKE system, including methodology materials, deliverables, and other project outcomes, is available at the project web site, http://www.spike-project.eu.

Acknowledgments. The SPIKE project was co-funded by the European Commission within the contract No. 217098. The work presented in the paper was also supported by the Slovak Grant Agency of the Ministry of Education and Academy of Science of the Slovak Republic within the 1/0042/10 Project "Methods for identification, annotation, search, access and composition of services using semantic metadata in support of selected process types".

REFERENCES

[1] A. Agraval et al, *WS-BPEL Extension for People (BPEL4People)*, Version 1.0, Active Endpoints Inc., Adobe Systems Inc., BEA Systems Inc., IBM Corporation, Oracle Inc., and SAP AG., 2007.

[2] T. Andrews, F. Curbera, H. Dholakia, Y. Goland, J. Klein, F. Leymann, K. Liu, D. Roller, D. Smith, S. Thatte, I. Trickovic, and S. Weerawarana, *Business Process Execution Language for Web Services. Ver. 1.1*, IBM developers' library, 2003.

[3] P. Bednar et al, *Deliverable 9.4: Evaluation of Trial 2 and specification for revision of the SPIKE system*, Project Deliverable, Consortium of the FP7 project SPIKE, ICT 217098, 2011.

[4] R. Belecheanu, L. Cabral, J. Domingue, W. Gaaloul, M. Hepp, A. Filipowska, M. Kaczmarek, T. Kaczmarek, J. Nitzsche, B. Norton, C. Pedrinaci, D. Roman, M. Stollberg, and S. Stein, *Deliverable 1.1: Business Process Ontology Framework*, Project Deliverable, Consortium of the FP6 project SUPER, IST 026850, 2007.

[5] D. Beneventano, N. Dahlem, S. El Haoum, A. Hahn, D. Montanari, and M. Reinelt, *Ontology-driven Semantic Mapping*, in Enterprise Interoperability III, Springer London, 2008, pp. 329–341.

[6] J. M. Boyer, *XForms 1.1*, W3C Recommendation, 20 October 2009, [on line], [cit. July 28, 2011], available at http://www.w3.org/TR/2009/REC-xforms-20091020/.

[7] C. Broser, C. Fritsch, O. Gmelch, *Towards Information Security Management in Collaborative Networks*, in Proceedings of 2010 Workshops on Database and Expert Systems Applications, DEXA conference, Bilbao, Spain, IEEE Press, 2010, pp. 359–363.

[8] L. Cabral, B. Norton, J. Nitzsche, T. van Lessen, *Deliverable 4.6: BPMO to sBPEL Two Way Translation*, Project Deliverable, Consortium of the FP6 project SUPER, IST 026850, 2008.

[9] D. A. Chappell, *Enterprise Service Bus*, O'Reilly Media, Inc., Sebastopol, CA, 2004.

[10] E. Christensen, F. Curbera, G. Meredith, S. Weerawarana, *Web Services Description Language (WSDL) 1.1*, W3C Note, 15 March 2001, [on line], [cit. July 28, 2011], available at http://www.w3.org/TR/wsdl/.

[11] J. Farrell, H. Lausen, *Semantic Annotations for WSDL and XML Schema*, W3C recommendation, 28 August 2007, [on line], [cit. July 28, 2011], available at http://www.w3.org/TR/ sawsdl/.

[12] C. Fernandez, G. Fernandez, M. A. Ramirez, J. M. Troya, *Deliverable 7.3: Implementation of Security Components for Service Bus Sub-System*, Project Deliverable, Consortium of the FP7 project SPIKE, ICT 217098, 2009.

[13] C. Fritsch, P. Bednar, G. Pernul, *DS^{3}I – A Dynamic Semantically Enhanced Service Selection Infrastructure*, in Proceedings of the 12th International Conference on E-Commerce and Web Technologies, EC-Web 2011, Toulouse, France, August/September 2011, Lecture Notes in Business Information Processing, Volume 85, Springer-Verlag Berlin Heidelberg, 2011, pp. 13–24.

[14] C. Fritsch, G. Pernul, *Security for Dynamic Service-Oriented eCollaboration - Architectural Alternatives and Proposed Solution*, in Proceedings of the 7th International Conference on Trust, Privacy and Security in Digital Business (TrustBus 2010), Springer-Verlag Berlin Heidelberg, 2010, pp. 214–226.

[15] K. Furdik, M. Mach, T. Sabol, *Towards semantic modelling of business processes for networked enterprises*, in Proceedings of the 10th International Conference on E-Commerce and Web Technologies, EC-Web 2009, Linz, Austria, September 2009, Lecture Notes in Computer Science, Volume 5692/2009, Springer-Verlag Berlin Heidelberg, 2009, pp. 96–107.

[16] P. Giangarra, J. DeMeester, *Enabling Network-Centric Operations with Semantic Web Technologies*, W3C submission, April 2005, [on line], [cit. July 28, 2011], available at http://www.w3.org/2005/04/FSWS/Submissions/14/Paper.pdf.

[17] M. Hepp, F. Leymann, J. Domingue, A. Wahler, and D. Fensel, *Semantic Business Process Management: A Vision Towards Using Semantic Web Services for Business Process Management*, in Proceedings of the IEEE International Conference on e-Business Engineering (ICEBE 2005), Beijing, China, IEEE Press, 2005, pp. 535–540.

[18] M. Hepp, D. Roman, *An Ontology Framework for Semantic Business Process Management*, in Proceedings of the conference Wirtschaftsinformatik 2007, Karlsruhe, Germany, 2007, pp. 423–440.

[19] S. Huber, C. Carrez, and H. Suttner, *Development of Innovative Services Enhancing Interoperability in Cross-organizational Business Processes*, in Enterprise Interoperability, Lecture Notes in Business Information Processing, Volume 76, Part 2, Springer-Verlag Berlin Heidelberg, 2011, pp. 75–88.

[20] D. Karastoyanova, B. Wetzstein, T. van Lessen, D. Wutke, J. Nitzsche, and F. Leymann, *Semantic Service Bus: Architecture and Implementation of a Next Generation Middleware*, in Proceedings of the 2nd International Workshop on Services Engineering 2007 (SEIW), ICDE Workshops, IEEE Press, 2007, pp. 347–354.

[21] D. Karastoyanova, T. van Lessen, F. Leymann, Z. Ma, J. Nitzsche, B. Wetzstein, S. Bhiri, M. Hauswirth, and M. Zaremba, *A Reference Architecture for Semantic Business Process Management Systems*, in Multikonferenz

Wirtschaftsinformatik 2008, Berlin, GITO-Verlag, 2008, pp. 1727–1738.

[22] M. Mach, P. Bednar, K. Furdik, *Support for Forming Temporal Business Alliances as Networked Enterprises*, in Proceedings of the Central European Conference on Information and Intelligent Systems (CECIIS 2009), Varazdin, Croatia, University of Zagreb, Faculty of Organisation and Informatics, 2009, pp. 179–186.

[23] M. Mach, J. Hreno, K. Furdik, *Prototype of a Platform for Business Collaboration*, in Proceedings of the Central European Conference on Information and Intelligent Systems (CECIIS 2010), Varazdin, Croatia, University of Zagreb, Faculty of Organisation and Informatics, 2010, pp. 347–354.

[24] A. Miles, S. Bechhofer, *SKOS Simple Knowledge Organization System Reference*, W3C Recommendation 18, August 2009, [on line], [cit. July 28, 2011], available at http://www.w3.org/TR/skos-reference/.

[25] Lios Geal Consultants Ltd., *E-Business Interoperability and Standards: A Cross-Sector Perspective and Outlook. Special e-Business W@tch report*, Enterprise and Industry Directorate General, European Commission, 2005.

[26] M. Missikoff, S. Drissi. R. Giesecke, A. Grilo, M. Li, N. Mehandjiev, and D. Werth, *Future Internet Enterprise Systems (FInES). Research Roadmap. Final report, ver. 3.0, June 2010*, European Communities, 2010.

[27] C. Pedrinaci, C. Brelage, T. van Lessen, J. Domingue, D. Karastoyanova, and F. Leymann, *Semantic Business Process Management: Scaling up the Management of Business Processes*, in Proceedings of the 2nd IEEE International Conference on Semantic Computing (ICSC 2008), Santa Clara, CA, IEEE Press, 2008, pp. 546–553.

[28] N. Rozanski, E. Woods, *Software Systems Architecture: Working with Stakeholders Using Viewpoints and Perspectives*, Addison Wesley, 2005.

[29] M. Sarnovsky, K. Furdik, *IT service management supported by semantic technologies*, in Proceedings of the 6th IEEE International Symposium on Applied Computational Intelligence and Informatics (SACI 2011), Timisoara, Romania, IEEE Press, 2011, pp. 205–208.

[30] T. Vitvar, J. Kopecky, D. Fensel, *WSMO-Lite: Lightweight Semantic Descriptions for Services on the Web*, WSMO Deliverable D11, Ver.0.2, DERI, 2008.

[31] S. Wiesbeck et al, *Deliverable 2.2: User requirements analysis and development/test recommendations*, Project Deliverable, Consortium of the FP7 project SPIKE, ICT 217098, 2008.

Edited by: Dana Petcu and Jose Luis Vazquez-Poletti
Received: August 1, 2011
Accepted: August 31, 2011

Scalable Computing: Practice and Experience
Volume 12, Number 3, pp. 307–315. http://www.scpe.org
ISSN 1895-1767

THE SEMANTIC MIDDLEWARE FOR NETWORKED EMBEDDED SYSTEMS APPLIED IN THE INTERNET OF THINGS AND SERVICES DOMAIN

PETER KOSTELNÍK† MARTIN SARNOVSKÝ† AND KAROL FURDÍK†‡

Abstract. The paper presents the LinkSmart middleware platform that addresses the Internet of Things and Services approach. The platform was designed to support the interoperability and seamless integration of various external devices, sensors, and services into the mainstream enterprise systems. The design and development of LinkSmart goes across two integrated European research projects, namely the FP6 IST project Hydra and the FP7 ICT project EBBITS. Modular architecture and functionality of LinkSmart prototype, developed by combining the service-oriented architecture, peer-to-peer networking, and semantic web services technologies, is described with focus on semantic binding of networked devices by means of underlying ontologies and knowledge-based inference mechanisms. Extensions of the solution towards the service orchestration, complex event handling, business process modelling and workflow processing are discussed and described on a mechanism of context-aware processing of sensor data.

Key words: semantic web services, Internet of Things and Services, ontology for services, devices and events, networked embedded systems

AMS subject classifications. 94-04, 94B99

1. Introduction. The innovative and rapidly evolving research area of *Internet of Things and Services* (IoTS) [16], [20] addresses an investigation of ways and means for seamless functional interconnection and effective, so-called intelligent, communication of various devices, services, information systems and resources towards operational scenarios. The aim of efforts in this area is focused on a development of platforms and solutions providing pervasive computing environment for networked embedded systems [6], which may be employed in real-world applications in industrial domains such as manufacturing, e-Business, e-Health, etc. In the European context, the IoTS research is supported, for example, by the CERP-IoT cluster [20], which helps to co-ordinate the research efforts in tens of involved FP6/FP7 projects. The projects such as ASPIRE, BRIDGE, CoBIs, CuteLoop, Hydra, and EBBITS can be explicitly mentioned as mostly related to the topic of this paper, i.e. the IoTS-enabling middleware. A brief survey of research activities, approaches, and technology solutions in this field will be presented in Section 1.1. The approach towards the IoTS middleware, which will be specifically discussed and presented later in this paper, was proposed and elaborated within the Hydra and EBBITS projects, where the authors of the paper were involved as development partners.

A wide scale of technologies is employed in IoTS frameworks or applications, ranking from Radio Frequency Identification (RFID) and sensor signals processing to computer network technologies, web services, or Service Oriented Architectures (SOA) of information systems. However, here we will focus on the middleware layer that enables, by means of semantic web services and related knowledge representation structures, a networking of physical devices, sensors, or components in order to provide higher value-added (i.e., more advanced, sophisticated, or intelligent) solutions to the end users. This was the objective of the FP6 integrated project Hydra (*Networked Embedded System Middleware for Heterogeneous Physical Devices in a Distributed Architecture*, http://www.hydramiddleware.eu), which started in July 2007 and finished in December 2010. Hydra aimed at the development of a middleware for intelligent networked embedded system, which is based on service-oriented architecture and is deployable on both new and existing networks of distributed wireless and wired devices [3]. The resulting system, which was named as the *LinkSmart* middleware, is described in Section 2.

Furthermore, the development continued by adapting the middleware for a broader exploitation. Directions of extensions and enhancements were identified namely in the underlying semantic structures, where several significant improvements were proposed - for example, a new ontological model of generated events, more advanced reasoning, inclusion of more types of devices, etc. To test these extensions, two application cases were specified in the areas of automotive manufacturing and food traceability. A new FP7 integrated project EBBITS (*Enabling Business-Based Internet of Things and Services*, http://www.ebbits-project.eu) [21] was launched in September 2010, where the major part of the Hydra consortium decided to participate. EBBITS is coordinated by the Fraunhofer Institute FIT (Institute for Applied Information Technology, http://www.fit.fraunhofer.de), which formerly coordinated the Hydra project as well. EBBITS consortium includes two industrial partners that

†Technical University of Košice, Faculty of Electrical Engineering and Informatics, Letná 9, 042 00 Košice, Slovakia. (Peter.Kostelnik@tuke.sk, Martin.Sarnovsky@tuke.sk, Karol.Furdik@tuke.sk).

‡InterSoft, a.s., Floriánska 19, 040 01 Košice, Slovakia. (Karol.Furdik@intersoft.sk).

are responsible for setting up and running pilot applications in automotive manufacturing and food production domains. Furthermore, seven research institutes and universities are the project partners that provide research, system design, development, and implementation work. Home institutions of authors of the paper, i.e. Technical University of Košice and InterSoft, a.s., are mostly involved in technology-related investigations of IoTS, design of the EBBITS system architecture, as well as in the design and implementation of semantic structures and supportive software components required for extending the LinkSmart middleware towards the advanced service/sensor data fusion and integration of these low-level data into business process workflow sequences.

The rest of the paper is organised as follows. The next subsection provides an overview of related technologies and research projects, which were investigated as background knowledge resources for both Hydra and EBBITS projects. Similarities, distinctions, and advancements of the Hydra / EBBITS approach in a comparison to the listed projects are briefly discussed. Section 2 presents the functionality and architecture of the LinkSmart middleware. General description of the system [3, 17] is accompanied by a description of using semantic infrastructure of LinkSmart for binding physical devices into a broader IoT network, which is demonstrated on a simple example. The semantic model of services, which can be considered as one of key conceptual structures in the LinkSmart infrastructure, is described in Subsection 2.2. The OWL ontology of services is presented together with the updates that we were already designed for applying the LinkSmart middleware in the IoTS platform of EBBITS. An overview of LinkSmart enhancements that will be accomplished in EBBITS is included in Section 3. Designed system architecture is presented together with the ontology of events, specifically developed for the purposes of complex event processing and inclusion of semantically enriched sensor data to the respective business processes. Section 4 summarises the adopted approach of IoTS, outlines planned pilot applications of EBBITS, and presents planned steps of further research.

1.1. Related Technologies and Research Streams. The area of IoTS is related to several research and application domains, which include semantic software architectures and ontologies, semantic web services, knowledge systems, middleware platforms, smart sensor networks, as well as a variety of systems providing features of distributed intelligence and/or content awareness for devices. A survey of enabling technologies, SOA-based middleware solutions, and IoTS applications can be found, for example, in [1]. Authors of this article emphasise the interoperability of interconnected devices as a central issue of IoTS, which includes "*an always higher degree of device smartness by enabling their adaptation and autonomous behaviour, while guaranteeing trust, privacy, and security*". The interoperability in its technical, semantic, and organisational aspects can be achieved by proper communication and networking infrastructure, adoption of semantic technologies, and integration of semantically annotated device events to complex workflow structures of high-level business processes, respectively.

In the IoT/IoTS middleware systems, the communication infrastructure is typically (but not exclusively [5]) based on SOA principles and peer-to-peer (P2P) networking [13], which is enabled by technologies such as JXTA (http://jxta.kenai.com), Java Message Service, event processing network [4], etc. To integrate the wireless sensor networks in the P2P platform by means of device proxies, the Contiki platform (http://www.contiki-os.org) can be employed to run the messages in 6LoWPAN standard (http://datatracker.ietf.org/wg/6lowpan/), which is based on IEEE 802.15.4-2003 standard. These technologies were proposed for EBBITS to enable the opportunistic sensor and device networking, which can be considered as a step towards the technical interoperability.

The semantic interoperability is achieved by employing technologies such as ontologies and knowledge management systems that facilitate logical reasoning, clustering of sensor data to more complex events, mediation of semantically heterogeneous data or interfaces, decision making and context awareness for networked sensors, devices, and services. Context-aware middleware characteristics and application types were analysed in [9], together with a benchmark of nine middleware systems that were produced mostly as outcomes of various research projects such as Aura, CARISMA, CORTEX, SOCAM, etc. [18]. This analysis served as a starting point for Hydra, which was focused on the development of SOA-based middleware providing a transparent communication layer for embedded devices. Special emphasis was given on features such as the interoperable access to data, information and knowledge across heterogeneous platforms, including web services, and support true ambient intelligence for ubiquitous networked devices [11, 3].

In the area of semantic web and knowledge systems, EBBITS will use knowledge engineering methods and tools to extend the Hydra ontologies and to develop advanced reusable domain models in the scope of pilot enterprise applications. From a scale of relevant approaches, we can mention two FP6 integrated projects:

- *SEKT* (http://www.sekt-project.com), which developed and exploited semantic knowledge technologies

of ontologies, metadata, and knowledge discovery. In EBBITS, we plan to re-use and extend these technologies towards the detection of useful complex features that can help in decision making process.
- *DIP* (http://dip.semanticweb.org), providing an infrastructure for semantic web services to e-Work and e-Commerce, is relevant to the further development of specific middleware services in EBBITS.

Since the EBBITS platform focuses on an effective applicability in a real-word industrial environment, the criteria such as robustness, usability, and scalability are of high importance. Based on a selection of the well-proven RDF/OWL knowledge representation formalisms and related reasoning mechanisms [7], the BigOWLIM[1] and AllegroGraph[2] storage platforms for RDF triples were identified as the best candidates to provide effective storage and access facilities for the middleware semantic data [2, 10].

Real word enterprise applications of IoTS middleware require a strong support of process workflows and related semantic web services, which addresses the organisational aspect of interoperability. On the side of services, it requires capabilities for discovery, composition (i.e. orchestration or choreography), deployment, and execution of services in a pre-defined or ad-hoc created workflow [22]. The workflow sequences can be modelled as abstract business processes, represented by the BPMN 2.0 notation (http://www.bpmn.org), which can be transformed to the respective executable form. From several available solutions we have selected the Drools platform (http://www.jboss.org/drools), which employs jBPM 5 toolkit for maintenance of BPMN process models combined with business rules (cf. Section 3).

Projects and approaches in the area of IoTS middleware, which were identified as the most relevant for the design and development of the EBBITS platform, are as follows:
- *ASPIRE* (http://www.fp7-aspire.eu), an FP7 project that has designed and developed a lightweight, royalty-free, and integrated middleware platform that could be used to implement the RFID identification part of the EBBITS system [8]. In an opposite way, EBBITS solution on distributed intelligence and semantic knowledge infrastructure could help ASPIRE to extend the architecture.
- *BRIDGE* (http://www.bridge-project.eu), an FP6 project that provides a suite of RFID tools and business cases that could be employed namely in the food traceability scenario of EBBITS pilot application. In addition, BRIDGE tools enable handling of the Electronic Product Code standard (EPC, http://www.epcglobalinc.org), that will be taken as one of resources to extend the Hydra device ontologies towards the EBBITS pilots.
- *SENSEI* (http://www.sensei-project.eu), an FP7 project that was aiming to create a business driven, scalable, pluggable and open framework for heterogeneous wireless sensor and actuator networks [19]. However, the project mainly addresses the scalability issue and the definition of services interfaces, which need to be extended in EBBITS by semantic-oriented integration capabilities and distributed intelligence features.

Other related approaches and solutions, as well as visions and challenges for the IoTS domain, are investigated, developed and provided as outcomes of projects co-operating within the CERP-IoT cluster [20], where both Hydra and EBBITS projects are included.

2. Architecture and Functionality of the LinkSmart Middleware. The Hydra project, briefly introduced in Section 1, was aimed at research, development, and validation of a middleware for networked embedded systems that would allow a development of cost-effective, high-performance ambient intelligence applications for heterogeneous physical devices [3]. To test the solution in a real-world environment, three pilot applications were prepared and accomplished in domains of facility management (smart homes), healthcare, and agriculture.

2.1. Semantic Binding of Devices in LinkSmart. The LinkSmart middleware, produced as the main outcome of the Hydra project, combines the semantic web services technology with SOA-based principles applied on the solution. The SOA and its related standards provide interoperability at a syntactic level. However, Hydra also aims to provide interoperability at the semantic level. One of the objectives is to extend the syntactic interoperability to the application level in terms of semantic interoperability. This was accomplished by combining the use of ontologies with semantic web services. In this context, Hydra introduces the *Semantic Model Driven Architecture* (SeMDA) [15], which was designed to facilitate an application development and to promote semantic interoperability for on-line services and devices of wireless or wired type [3]. The SeMDA of Hydra includes a set of ontologies, and provides the set of tools, which can be used both in application design

[1] http://www.ontotext.com/owlim/
[2] http://www.franz.com/agraph/allegrograph/

time and runtime [12]. The SeMDA concept, implemented in the LinkSmart middleware, makes all devices in a LinkSmart-based IoTS application accessible in an uniform way - as the semantic web services.

Basically, SeMDA in LinkSmart provides a mechanism for wrapping standard API interfaces of services, sensors, and various physical devices with a defined web service extension, which is enhanced by a semantic description of provided or generated WSDL files [6]. This process is called the Hydra-enabling of the device. Developer can Hydra-enable a new device using so-called Device Development Kit (DDK), included into the LinkSmart infrastructure [17]. The new device is annotated to the suitable class in the device taxonomy (e.g. mobile device) and the basic description, such as device model name and number, manufacturer information, energy consumption profile or device discovery information, is added. Since particular devices may have different connection and communication capabilities, the service calls have to be transformed into web service calls. For each service, the developer has to add a custom implementation, which includes common services as *StartWS*, *StopWS* or *GetWSEndpoint*, and services related to the energy consumption such as *CurrentPowerConsumption* or *RemainingBattery*. Each service is also annotated to the suitable service taxonomy class of LinkSmart Device ontology. This way, the devices and their local networks are both accessible by LinkSmart and connected to the outside world through broadband and/or wireless networks. For example, the binding of a thermometer device to the respective semantic description in Device.owl ontology of LinkSmart is as follows:

```
<linksmart:binding device="http://linksmart.eu.com/
  ontology/Device.owl#thermometer"/>
```

The LinkSmart can generate a stub of the related client and/or server code, which can be based, for example, on an available ontology instance that semantically describes the states of the device or sensor. The proxy stub is created according to the devices capabilities as either directly embedded on the device or using the *OSGi framework*[3]. The device can then be accessed and controlled in the application code of a networked embedded system using the following Java statements:

```
AppDeviceManager myMgr = new AppDeviceManager();
ThermoMeter.LSDeviceWS myThermometer = new ThermoMeter.LSDeviceWS();
myThermoMeter.SetLSID(myMgr.GetLSID("Off1Thermometer"));
Light.LSDeviceWS myLight = new Light.LSDeviceWS();
myLight.SetLSID(myMgr.GetHID("Off1Light"));
...
if (myThermometer.GetTemperatureC() > 25)
{
  myLight.Flash(2);
  myPhone.SendSMS("Too hot in the office 1,
    temperature:" + myThermometer.GetTemperatureC()+ "+421329264552");
}
```

The meaning of the presented code, which is rather simplified for demonstration purposes, is as follows. After initiating the Application Device Manager object (which is included in the Service Layer of Application Elements, as it is presented in Figure 2.1), a web service client is created for the thermometer device. It is required that the device is Hydra-enabled, so that it was properly wrapped by obligatory web service interfaces and annotated by LinkSmart device ontology. Then the LinkSmart identifier, abbreviated as LSID, is retrieved from a concrete physical device in our case, from the thermometer located in an office 1 (i.e. from Off1Thermometer). The identifier is used to create an endpoint URL for the device, forwarded as input parameter to the web service client that corresponds to the device. The same way, another web service client is created and initialised for the light device. Once the web service client was initialised and the URL for a device was properly established, the customised LinkSmart services of the device can be invoked and consumed. The rest of the code is obvious if the temperature in office 1, measured continuously by the thermometer, exceeds 25 degrees, then the light will start flashing and the cell phone will send a message.

The creation of a new Hydra-enabled device in design time introduces only basic device semantic representation, which can be later further extended. Each device ontology instance represents the specific device model and serves as the static information template [11]. In runtime, when new device enters in the LinkSmart

[3] http://www.osgi.org

application network, the best matching template is identified by the semantic discovery process, cloned and tied to the physical device using the LSID persistent identifier. The property values of the runtime instance can change as the device changes its state variables (e.g. measured values of thermometer or sensor). When physical device leaves the LinkSmart network, assigned device runtime instance is removed from the ontology.

2.2. LinkSmart Architecture and Components. LinkSmart middleware is typically installed as a node in the peer-to-peer network, which encapsulates interfaces of internally referenced devices and provides them as semantic web services to other network nodes - LinkSmart instances. The architecture of the main functional modules of LinkSmart is depicted in Figure 2.1.

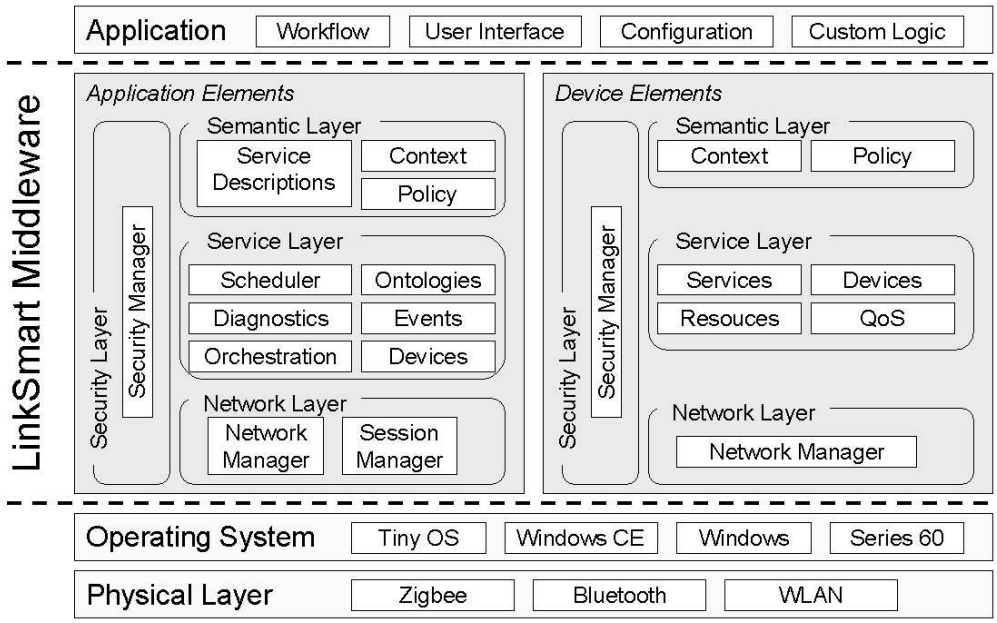


Fig. 2.1: Structural overview of the LinkSmart middleware layers

The inner middleware elements are enclosed by the physical, operating system and the application layers shown at the bottom and at the top of the diagram, respectively. The physical layer provides several network connections like ZigBee, Bluetooth or WLAN. The operating system layer enables managing the physical layer objects and provides methods for accessing the resources of network connections. The operating system layer provides means to access and manage the physical layer objects. The application layer contains customisable user applications that may include modules for workflow management, user interface, custom logic and configuration details. These three layers are not a part of the LinkSmart middleware.

The middleware itself is divided into the *Application Elements* and *Device Elements* parts, representing the close (i.e. running in a performance-wise mode, e.g. on the same machine as the resources used) and distant (i.e. remote, with a slow access or performance) components, respectively. Each of the parts consists of the network, service, semantic, and security layers, which contain the LinkSmart business logic, i.e. the functions for context sensing, service requests handling, network management and synchronisation of peer nodes, access control, etc.

The middleware functionality is supported by a structure of OWL ontologies, which provide a semantic basis for particular business logic elements. For example, the ontology structure of the LinkSmart service model is presented in Figure 2.2. Services, which are tied to devices, are described by the respective capabilities, input and output parameters such as name, data type, and unit. Similar ontologies were produced for modelling devices, network connections, and security issues [12].

3. EBBITS Extensions towards the IoTS Domain. The EBBITS project is aiming to shift the IoTS paradigm of Hydra more towards the services that are orchestrated in complex workflow sequences, i.e. in business processes that correspond to the real-world scenarios in industry or other application domain [21].

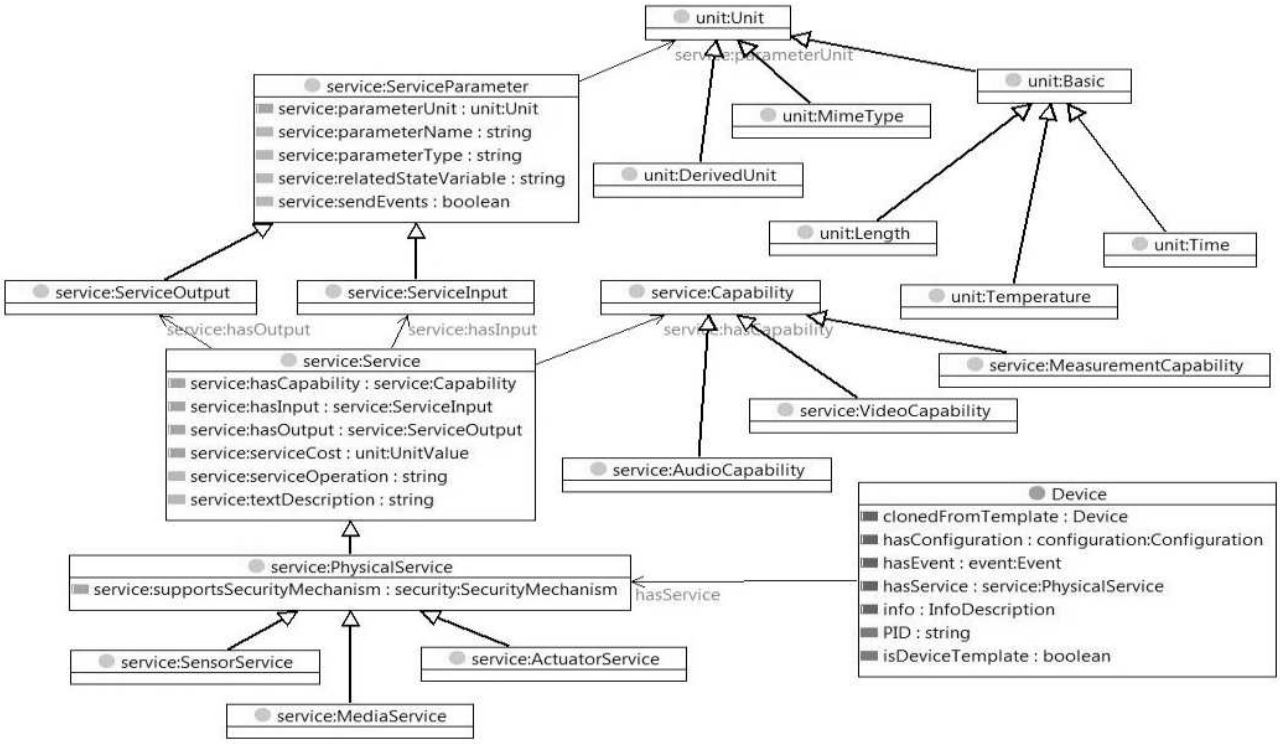


Fig. 2.2: Conceptual model of services in the LinkSmart ontology

This way, the EBBITS platform should provide a bridge between enterprise and public information systems, as well as between human users and things in the physical world.

The EBBITS platform is targeting to support interoperable business applications with context-aware processing of data separated in time and space, information and real-world events (addressing tags, sensors and actuators as services), people and workflow chains (operator and maintenance crews). Optimisation of service compositions will be supported by means of high level business rules that can be driven, for example, by energy or cost performance criteria. The key requirement for the business rules execution is that the EBBITS platform needs to be able to recognise and respond to physical world events. The information acquired from events, which may come from physical world and are generated by various devices, create the basis for decision making at the several levels of the EBBITS architecture (cf. Figure 3.1), including data fusion, situation patterns recognition, complex event processing, analysis of historical acquired data, etc. All these requirements need to work with a large amount of information related to the devices generating events or providing services for further processing by event/service orchestration, decision making, or business rules. In some cases it must be possible to use this information to analyse the historical data generated by particular events. All parts of decision making process will be supported by enriched semantic model that will enable a flexible knowledge representation of all included events, roles, services and processes.

Obviously, EBBITS builds on the outcomes of the Hydra project. The LinkSmart middleware system is taken as the implementation basis, which will be extended on the functions and capabilities of semantic business process modelling, workflow management, service choreography and orchestration, event handling and processing of complex events generated by devices. The development will also address the service interoperability issues and various enhancements that can be required on the security and networking maintenance.

A high-level architecture of the main functional modules proposed for EBBITS is presented in Figure 3.1. Physical level of devices, sensors, external services or applications is constituted on the same principles as in LinkSmart solution. It means that the devices are included into the LinkSmart application network by the semantic binding mechanism and supported web service interfaces. Devices may generate events, which are collected on the *Physical World Adaptation Layer*. After a normalisation and resolving of initial semantic

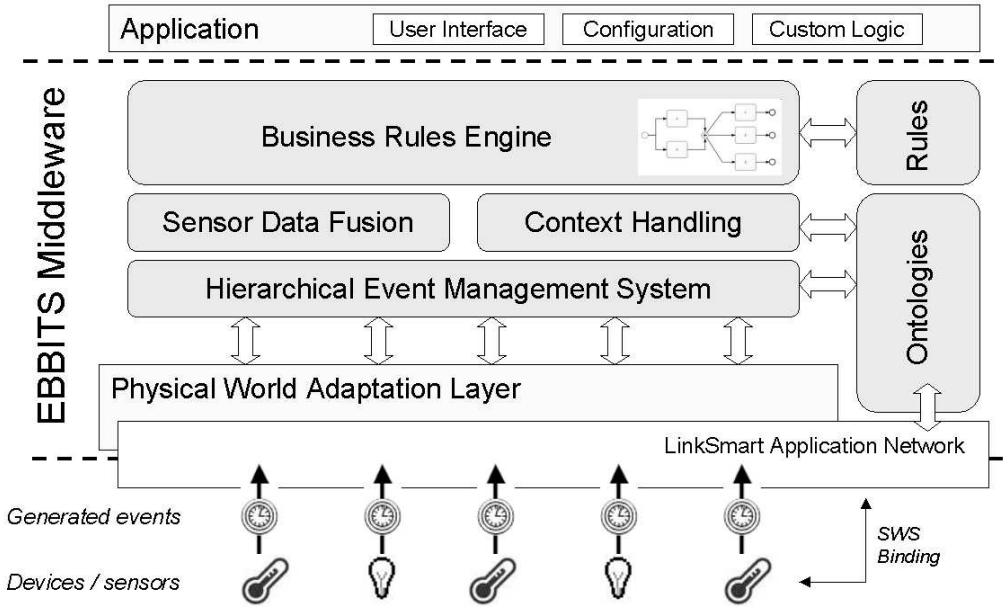


Fig. 3.1: Architecture of functional modules for the EBBITS platform

characteristics (using, for example, classification, pattern recognition, or clustering methods), the events are propagated to the *Hierarchical Event Management System* (HEMS). The *Sensor Data Fusion* module, invoked by HEMS, will combine low-level sensor events into more complex information structures. Semantic search, mediation, and reasoning mechanisms, supported by applied RDF triple store framework and related reasoning engines [10], will be employed for sensor data merging and context handling. Consequently, the data fusion should provide information chunks that are suitable for further processing by workflow elements and business rules of IF-THEN format, provided by the jBPM5 toolkit included into the Drools platform (see Section 1.1).

Communication and data/information transformation is expected in both directions. If the data flow goes from events to business rules, then the execution of a pre-defined business process is driven, or at least influenced, by generated events. In case of opposite data flow, a network of devices can be controlled and driven by business rules specified on the upper level of EBBITS middleware.

As one of initial steps towards the extensions required on LinkSmart system for providing the functionality proposed for EBBITS, it was necessary to update the semantic model for events, which has originally been developed in LinkSmart on an insufficient level of details. We have designed the events ontology that should cover all the information related to the hierarchical event handling and sensor data fusion in EBBITS, as it is depicted in Figure 3.2. The ontology was created with respect to the *SSN ontology* [14]. It includes a generic Event concept, models of event results and event stimulus, as well as the connection to the service that triggers the event. These features should support the data fusion by, for example, merging of events generated by devices of the same type, location, etc. The core taxonomy of events, required for hierarchical event management, consists of two main sub-classes, which are distinguished according to the event stimulus type, i.e., triggered by a real-world situation or continually generated in some frequency.

Further EBBITS enhancements of LinkSmart will include, among others, a semantic model for service composition, which will cover the service execution preconditions and post-conditions, the models for orchestration of services into processes or grounding the services to a concrete implementation. The design of these extensions will be most likely driven by OWL-S, WSMO, or similar semantic service ontologies.

4. Conclusions. To summarise, the Hydra and EBBITS projects present the concept of Semantic Devices [11]. The motivation behind the concept is the fact that the services offered by physical devices are generally designed independently of the particular applications in which the devices might be used. A semantic device, on the other hand, represents what a particular application would like to have. The basic idea of this approach is to hide all the underlying complexity of the mapping to, discovery of, and access to physical devices. The

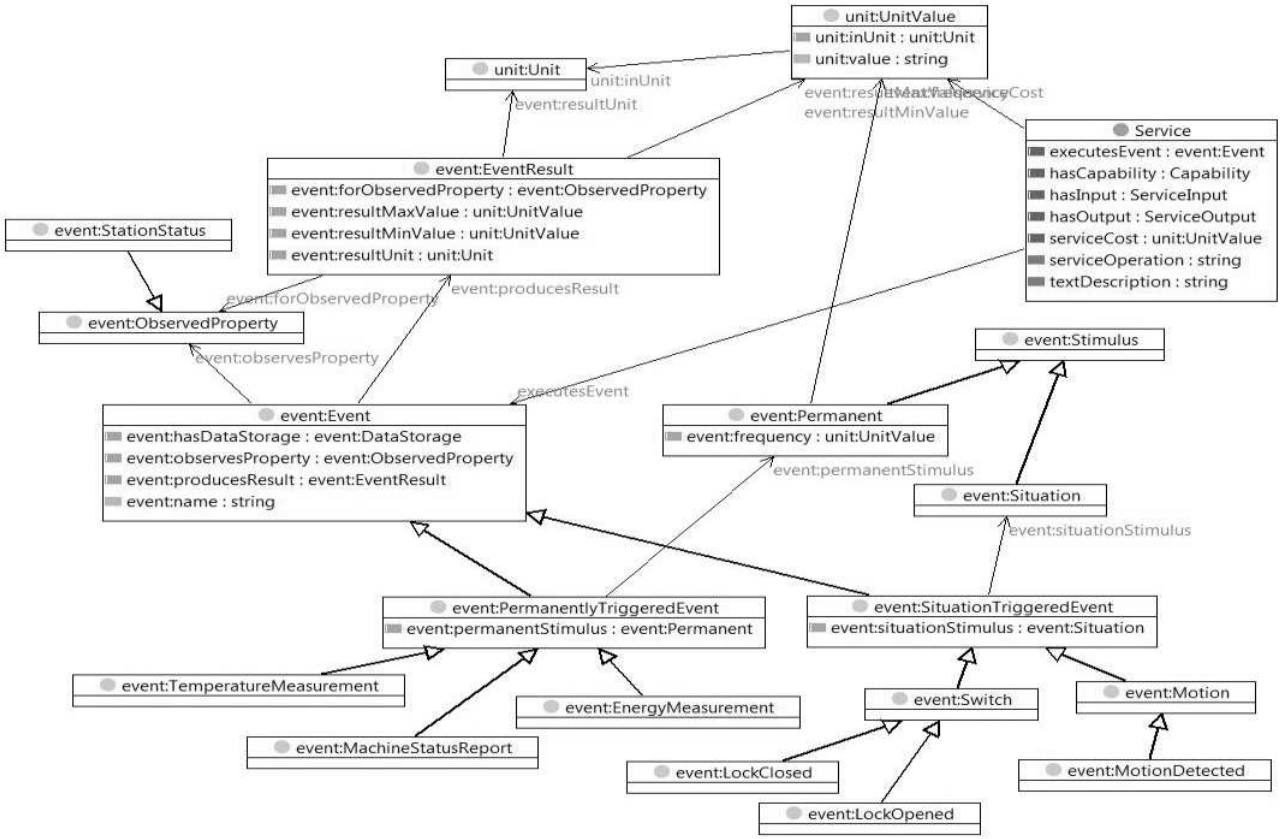


Fig. 3.2: Conceptual model of events proposed for the EBBITS platform

programmer just uses it as a normal object in his application focusing on solving the applications problems rather than the intrinsic of the physical devices.

The foundations of this concept were investigated in the Hydra project and resulted in the LinkSmart middleware system, currently available under the LGPL 3.0 open source licence. The EBBITS project builds on the Hydra outcomes and tries to extend the LinkSmart software on a fully featured support of business processes, event handling, and workflow processing. This approach will be tested in two pilot applications, namely:

- *Automotive manufacturing*, where enhanced LinkSmart middleware should help to manage the production optimisation with special focus on reducing energy consumption;
- *Food producing industry*, supporting the service and resource traceability through the whole process cycle, from the production to consumption stage.

Currently, in July 2011, the design and analysis work is completed and the specification of LinkSmart system updates is ongoing. Technologies and related approaches were identified for components and semantic structures that will be redesigned or newly developed within EBBITS. Specification of envisioned updates is available and the implementation is ongoing. The first prototype of the enhanced LinkSmart system should be available in autumn 2011.

Acknowledgments. The project Hydra was co-founded by the European Commission within the 6th Framework Programme, contract No. IST-2005-034891. The presented work is partially funded by the European research project EBBITS (7th Framework Programme, contract No. ICT-2009-5-257852), and by the Slovak Grant Agency of the Ministry of Education and Academy of Science of the Slovak Republic within the 1/0042/10 Project "Methods for identification, annotation, search, access and composition of services using semantic metadata in support of selected process types". We like to thank all partners in the EBBITS project for their support.

REFERENCES

[1] L. Atzori, A. Iera, G. Morabito, The Internet of Things: A survey, in Computer Networks, The International Journal of Computer and Telecommunications Networking, Volume 54 Issue 15, October, 2010, pp. 2787–2805.
[2] C. Bizer, A. Schultz, *Berlin SPARQL Benchmark Results*, Freie Universität Berlin, 2009, [on line], [cit. August 3, 2011], available at http://www4.wiwiss.fu-berlin.de/bizer/BerlinSPARQLBenchmark/results/index.html.
[3] M. Eisenhauer, P. Rosengren, P. Antolin, *HYDRA: A Development Platform for Integrating Wireless Devices and Sensors into Ambient Intelligence Systems*, in The Internet of Things, Springer, New York, 2010, pp. 367–373.
[4] O. Etzion, P. Niblett, *Event Processing in Action*, Manning Publications Co., Greenwich, CT, USA, 2010.
[5] C. Floerkemeier, C. Roduner, M. Lampe, *RFID application development with the Accada middleware platform*, in IEEE System Journal 1 (2), 2007, pp. 82-94.
[6] K. M. Hansen, W. Zhang, G. Soares, *, G.: Ontology-Enabled Generation of Embedded Web Services*, in Proceedings of the 20th International Conference on Software Engineering and Knowledge Engineering, Redwood City, San Francisco Bay, USA, 2008, pp. 345–350.
[7] J. Hreno, P. Nutakki, M. Knechtel, K. Furdik, T. Sabol, *Deliverable 4.2: Knowledge representation formalism analysis*, Project Deliverable, Consortium of the FP7 project EBBITS, ICT-2009-5-257852, 2011.
[8] N. Kefalakis, N. Leontiadis, J. Soldatos, D. Donsez, *Middleware Building Blocks for Architecting RFID Systems*, in Social Informatics and Telecommunications Engineering, Volume 13, Lecture Notes of the Institute for Computer Sciences, Springer-Verlag Berlin Heidelberg, 2009, pp. 325–336.
[9] K. E. Kjaer, A Survey of Context-aware Middleware, in Proceedings of the 25th conference on IASTED International Multi-Conference: Software Engineering, Anaheim, CA, USA, ACTA Press, 2007, pp. 148–155.
[10] M. Knechtel, J. Hreno, F. Pramudianto, M. Ahlsen, K. Furdik, *Deliverable 4.1: Analysis of Semantic Stores and Specific ebbits Use Cases*, Project Deliverable, Consortium of the FP7 project EBBITS, ICT-2009-5-257852, 2011.
[11] P. Kostelnik et al, *Semantic Devices for Ambient Environment Middleware*, in Proceedings of EURO TrustAmi 2008, Internet of Things and Services, Sophia-Antipolis, France, 2008.
[12] P. Kostelnik, M. Sarnovsky, J. Hreno, *Ontologies in HYDRA - middleware for ambient intelligent devices*, in Ambient Intelligence Perspectives II., Vol. 5, 2009, pp. 43–46.
[13] F. M. Lardies, P. A. Rafael, J. Fernandes, W. Zhang, K. M. Hansen, P. Kool, *Deploying Pervasive Web Services over a P2P Overlay*, in WETICE '09, Proceedings of the 18th IEEE International Workshops on Enabling Technologies: Infrastructures for Collaborative Enterprises, Washington, DC, USA, IEEE Computer Society, 2009, pp. 240–245.
[14] L. Lefort, C. Henson, K. Taylor, *Incubator Report*, W3C Semantic Sensor Network Incubator Group, 29 June 2011, [on line], [cit. August 3, 2011], available at http://www.w3.org/2005/Incubator/ssn/wiki/Incubator_Report.
[15] C. Pahl, *Semantic model-driven architecting of service-based software systems*, in Information and Software Technology, Volume 49 Issue 8, August, 2007, pp. 838–850.
[16] G. Santucci, *From Internet of Data to Internet of Things*, Information Society and Media, Directoriate General, European Commission, Brussels, 2009.
[17] M. Sarnovsky, P. Kostelnik, J. Hreno, P. Butka, *Device Description in HYDRA Middleware*, in Proceedings of the 2nd Workshop on Intelligent and Knowledge oriented Technologies (WIKT 2007), Kosice, Slovakia, 2007, pp. 71–74.
[18] C. F. Sorensen, M. Wu, T. Sivaharan, G. S. Blair, P. Okanda, A. Friday, H. Duran-Limon, *A context-aware middleware for applications in mobile ad hoc environments*, in Proceedings of the 2nd workshop on Middleware for pervasive and ad-hoc computing (MPAC 2004), New York, USA, ACM Press, 2004, pp. 107–110.
[19] M. Strohbach, M. Martin, *Towards a Platform for Pervasive Display Applications in Retail Environments*, in Pervasive Computing, Volume 10, Issue 2, IEEE Computer Society, 2011, pp. 19–27.
[20] H. Sundmaeker, P. Guillemin, P. Friess, S. Woelffl, *Vision and Challenges for Realising the Internet of Things*, CERP-IoT cluster, Information Society and Media, Directoriate General, European Commission, Brussels, 2010.
[21] V. Vajda, K. Furdik, J. Glova, T. Sabol, *The EBBITS Project: An Interoperability platform for a Real-world populated Internet of Things domain*, in Proceedings of the International Conference Znalosti (Knowledge), Technical University of Ostrava, Czech Republic, 2011, pp. 317–320.
[22] W. Zhang, K. M. Hansen, *Towards Self-managed Pervasive Middleware using OWL/SWRL ontologies*, in Proceedings of the Fifth International Workshop on Modeling and Reasoning in Context (MRC 2008), Delft, The Netherlands, 2008, pp. 1-12.

Edited by: Dana Petcu and Jose Luis Vazquez-Poletti
Received: August 1, 2011
Accepted: August 31, 2011

Scalable Computing: Practice and Experience ISSN 1895-1767
Volume 12, Number 3, pp. 317–336. http://www.scpe.org

SECURE ACCESS MECHANISM FOR CLOUD STORAGE

DANNY HARNIK, ELLIOT K. KOLODNER, SHAHAR RONEN, JULIAN SATRAN, ALEXANDRA SHULMAN-PELEG, AND SIVAN TAL *

Abstract. Emerging storage cloud systems provide continuously available and highly scalable storage services to millions of geographically distributed clients. A secure access control mechanism is a crucial prerequisite for allowing clients to entrust their data to such cloud services. The seamlessly unlimited scale of the cloud and the new usage scenarios that accompany it pose new challenges in the design of such access control systems.

In this paper we present a capability-based access control model and architecture appropriate for cloud storage systems that is secure, flexible, and scalable. We introduce new functionalities such as a flexible and dynamic description of resources; an advanced delegation mechanism and support for auditability, accountability and access confinement. The paper details the secure access model, shows how it fits in a scalable storage cloud architecture, and analyzes its security and performance.

1. Introduction. The rapid growth in the amount of personal and organizational digital data is a major characteristic of information systems in this decade. This growth is accompanied by increased demand for availability, as users wish to run their software and access their data, regardless of their type and location, from any web-enabled device at any time. Cloud computing addresses these challenges by providing virtualized resources as an online service and by allowing them to scale dynamically. Services offered over the cloud include applications (Software as a Service), virtual machines (Infrastructure as a Service), and data storage (Storage as a Service). The latter service, along with the storage cloud upon which it resides, is the topic of this paper.

Storage cloud systems are required to provide continuous availability and elastic scalability [4] while serving *multiple tenants* accessing their data *through the Internet*. The storage cloud infrastructures are comprised of tens of geographically dispersed data centers (DCs), where each DC is comprised of many thousands of compute and storage nodes, and should be able to efficiently serve millions of clients sharing billions of objects. Furthermore, to achieve availability and scalability, each data object is replicated at multiple data centers across the cloud. Each object replica should be accessible for reads and writes, and to optimize access latency and data throughput, it is preferable that clients be directed to the replica closest to them.

1.1. Security and access control in the storage cloud. Data security risks are a key barrier to the acceptance of cloud storage [25, 30, 43]. Many potential users are rightfully worried about moving their data to a storage cloud, where it will be stored by an external provider on an infrastructure also used for storing other customer's data. Storage cloud providers must therefore implement a secure access control system in order to reduce the risk of unauthorized access to a reasonably low level. *Access control* refers to managing identities, defining access policies, and controlling access to resources in a secure manner.

Security has its costs, as the structure of very large scale storage systems incurs a trade-off between performance, availability and security [23]. Balancing this trade-off is particularly challenging in the storage cloud environment, as providers must implement a secure access control system that scales elastically and meets the high availability requirements of the cloud as described above. Moreover, even though the consistency of the data itself can be reduced to improve availability [41], the system must always consistently enforce access control at all access points for all data objects.

In addition to the scale and availability requirements, today's new web applications such as mashups, introduce new data access practices. Namely, data is not necessarily accessed directly by its original owner but rather through various applications, in flexible sharing scenarios and with various billing methods. These applications put forth new functional requirements for the access control mechanisms. These include, for example, the ability of a client to delegate a flexible subset of his access rights to anyone he or she wishes and support for the related notion of *discretionary access control (DAC)* [1]. Also required is support for hierarchical management of rights, assigning administrators privileged access to a data domain and allowing them to delegate partial access to other principals under their control.

Traditional file-based storage systems employ a monolithic access control system. In such a system, a client is authenticated upon logging in to a session, and is then authorized by the data server for each file access, according to the access policies (e.g., *ACLs*) associated with the accessed file. This approach is insufficient for a cloud storage system for several reasons. First, HTTP is a stateless protocol so session-based security is

*IBM Haifa Research Lab, Haifa, Israel 31905, Email: {dannyh,kolodner,shaharr,julian_satran,shulmana,sivant}@il.ibm.com

not always applicable; even with secure HTTP sessions, it is inefficient and sometimes impractical to associate a session with each client. Second, performing identification, authentication, and authorization for each data access request creates a performance overhead especially in very large scale systems with high multi-tenancy. Furthermore, some of the new requirements, such as replication in an eventually-consistent environment, make the update of access rights more difficult. We advocate, instead, a capability-based approach, already used in distributed file systems (DFSs) such as Ceph [42], which seems more suitable for cloud-scale storage.

1.2. Our contributions. In this paper we propose a capability-based access control model designed to address the emerging challenges of storage clouds. Extending previous capability-based access control models [39, 11, 31, 23, 12] we introduce the following innovations:

- We present a model supporting fine grain and dynamic scope for access rights. We introduce a dynamic definition of the resources via a flexible description with pattern matching on object attributes (e.g. object name, type, or metadata). Thus, the accessible subset of resources can change as objects are added or removed from the system.
- We enable user-to-user and user-to-application delegation, where one user can directly delegate a subset of his access rights to a subset of resources without requiring the intervention of a system administrator or the storage cloud. We introduce mechanisms for auditing and access confinement that can serve as a basis for determining accountability, compliance and billing.
- We present an architecture of a data center that supports capability-based access control. It separates authentication and authorization from the data access path, allowing non-mediated and efficient access to data without the need to specify the accessed storage server (i.e. the same credential is valid for all the replicas regardless of their physical location).
- We enable integration with external identity or access management components used by enterprises and applications. This allows combining the advantages of capabilities with those of other access control models (e.g. ACLs). This enables fulfilling the following requirements that no single access control mechanism can satisfy alone (1) Compatibility with existing systems and APIs; (2) Centralized monitoring of the access rights granted to all users over specific resources; (3) Achieving the advantages of capability-based models (see Section 6).

We show that our protocol has the same security properties as other capability-based models and does not introduce new threats. We believe that the benefits of our access control mechanism, both in scalability and in functionality, remove obstacles preventing potential clients from adopting cloud storage.

1.3. Paper organization. Section 2 details the requirements from a storage cloud access control system, and provides an overview of state-of-the-art storage cloud systems, highlighting their current limitations. Section 3 describes our protocol, elaborating on the new features. Section 4 evaluates the security of our proposed method. Section 5 describes an overall architecture for a storage cloud system incorporating our protocol. Section 6 includes a discussion and evaluation of our system in light of the requirements set forth in Section 2.1. We conclude and discuss future work in Section 7.

2. Background. In this paper we introduce a solution for new requirements regarding secure access for cloud storage. We begin by describing the access control requirements for a secure cloud-scale storage system and the motivation for them. We then analyze how existing systems satisfy these requirements.

2.1. Access control requirements. The challenge of access control is to selectively control who can access and manipulate information and allow only users with the proper privilege to carry out operations. Here we focus on functional properties of access control specific to storage in a cloud environment.

Secure chaining of services. When a client uses an application or service on the cloud, this application often relies on another application to perform an action, which in turn may invoke yet another application and so on. When chaining services in this way, it is important to keep track of the access rights of the client at the end of the chain; otherwise, the system could become vulnerable to attacks like clickjacking and cross-site request forgery that can occur when accessing the web [7]. Such attacks can be prevented by giving each client the least authority [40] required to perform its request, and correctly maintaining the access permissions of the client along the chain, i.e., taking care that they are neither expanded nor restricted.

User-to-user access delegation. Cloud systems aim to provide their users with the ability to easily share resources on a very large scale. To allow a cloud system to scale arbitrarily, the role of system administrators needs to be minimal. In particular, a user should be able to delegate access rights to other users without

requiring the intervention of a system administrator. In addition, a user should be able to delegate access rights (or a subset of them) to his resources (or a subset of them) to whomever they choose, including users from other administrative domains, such as users registered to other cloud service provider, or not registered to cloud services at all. Cross-provider delegation makes it easier to spread one's data or services across multiple cloud providers. While this is a recommended practice for increasing availability and resistance to DDOS attacks, it is hard to implement using the current access control mechanisms [4]. Furthermore, a user who is granted access rights from the administrator or from another user should be able to further delegate the rights to other users; this is known as "transitive delegation" [29]. Additionally, to allow flexibility in delegation, the set of delegated resources should be defined using predefined criteria (e.g., regular expressions), in order to allow the set of delegated resources to change dynamically as the contents change (see Section 3.1.1 on page 321). Finally, in order to provide a secure environment that allows auditing and compliance to regulations, it should always be possible to determine who had access to a specific resource and who is responsible for the delegation.

Scalability and high availability. Cloud services are expected to always be available, even during failures and disasters, and to adjust themselves to significant load fluctuations within short periods of time. Distribution of resources and data is a key instrument in meeting these goals, and should be coupled with a corresponding distribution of the access control system. For example, a central authentication point or authorization point would not scale well: it could create bottlenecks, be a single point of failure and become an attractive attack target (e.g. [34]). Furthermore, any point in the cloud possessing a resource should be able to serve the request while enforcing access control. We also add the requirement that the access control mechanism must not affect the data access latency substantially.

2.2. Review of the state of the art. We now proceed to examine how the existing solutions satisfy these requirements. Distributed File Systems (DFS) have been used for file sharing within organizations for many years. However, as production DFS protocols (e.g., NFS, AFS, CIFS) were not designed for file sharing across organizational boundaries, they fail to meet the user-to-user delegation requirements we set above [29, pp. 25-26]. While some experimental DFSs (e.g., Fileteller [19] and DisCFS [28]) and file-sharing services (WebDAVA [24]) meet some of the delegation requirements, they are based on the Public Key Infrastructure (PKI), which has been shown to have multiple limitations for adoption in cloud-scale systems [34]. On the other hand, efforts like OAuth [16] which do not assume PKI, require that access delegation grants pass through the service provider, thus not meeting our user-to-user delegation requirement, and also imposing a communication overhead.

Some existing capability-based models, for example, OSD [11, 31, 23], SCARED [39], and CbCS [12], satisfy the requirements of secure chaining of services and user-to-user delegation. However, these protocols are not geared toward the scalability and high availability required by the cloud environment, as they were designed for storage systems with data models (e.g. device/object) that are not adequate for the cloud.

Production storage solutions designed for cloud environments have difficulties in achieving the described requirements. The major players in the market [3, 10, 26, 33, 37, 35] use *Access Control Lists (ACLs)* for authorization and do the authorization stage in the data access path. This approach has limitations both in its ability to easily scale and in its ability to chain services without losing track of the privileges of the principals, thus failing to enforce the principle of least privilege (as described in [7]). In addition, none of the services we examined enable a secure user-to-user delegation, with the ability to delegate a subset of access rights using predefined criteria.

Following is a brief survey of the most significant cloud storage solutions with respect to their secure access approach.

Amazon S3 [3, 2]. Amazon S3 has a simple method for granting access to other parties using ACLs, but the other parties must be registered S3 users or groups. Objects can be made public by granting access to the *Anonymous Group* but there is no way to selectively grant access to principals outside of the S3 domain. An additional limitation is that the S3 ACLs are limited to listing 100 principals. Authentication is based on user identifiers (unique within S3) and secret keys shared between the users and the S3 system. The requester includes his identifier in the request and signs selected parts of the request message with a keyed hash (HMAC) using his secret key. Since the authentication is embedded in the HTTP request, an authorized user can create an authenticated request and pass it to another user to get an access to a specific object. Furthermore, since the authentication information can be passed in a query string as part of the URL of the HTTP request, this method, known as *Query String Authentication (QSA)*, can be used by web applications interacting with web browsers to provide seamless access delegation.

EMC Atmos Online [9]. Atmos has an authentication method similar to Amazon S3, and the same limitations apply. The identity and authorization system is a little different. Atmos has a hierarchy of organizations (called *subtenants*) and users within them. ACLs are maintained within subtenants with no option to grant access permission to any principal outside the subtenant to which the object's owner belong. User groups are not supported, and objects cannot be made public for anonymous access.

Microsoft Windows Azure Storage [26, 27]. Azure has an authentication method similar to Amazon S3 and EMC Atmos. Like S3, Azure allows defining objects as public to allow anonymous read access. Like Atmos, the ACL support is minimal - there is no group support; moreover, the access policy is either public (anyone) or private (only the owner). On top of that, Azure supports limited capability-based access. This is done using a *Shared Access Signature*, which encodes a capability to perform specified operations on a specified object (container or blob) for a specified period of time. The Shared Access Signature is signed with the owner's secret key. It can then be passed to any user and used in a URL's query string to gain access.

Following is comparison of the cloud storage solutions with the requirements outlined earlier.

Secure chaining. One thing in common to all the systems described above, is that every data access request is authenticated using an authentication string based on a secret key that the user shares with the storage system. Existing systems do not provide a built-in mechanism that allows one service to chain the services of another, and therefore the solution should be implemented in one of the following ways: (1) Obtaining the secret keys or tokens that will identify their clients to the storage servers; or (2) Moving the control of the data from the end user to the service provider, who will access the data on his behalf. Both options are limited and inconvenient.

Access delegation. The basic delegation mechanism of most systems requires changing the ACL configuration at the storage servers and can not allow controlled access to users who are unknown to the system. Among registered users, transitive delegation is possible only by granting write permissions to the delegated resource's ACL. This has multiple security risks and delegates more rights than actually needed, thus violating the principle of least privilege. Furthermore, there is no mechanism for controlled transitive delegation among users that are unknown to the system. In addition, existing services limit the definition of resources for delegation to fixed criteria (e.g., names of specific objects or buckets/containers). They do not support dynamic criteria (e.g., all objects created after a certain date), which are important when handling billions of objects and million of users, as is common in cloud environments.

Scalability and availability. With the exception of Azure's Shared Access Signature, the authentication string authenticates the identity of the user, and authorization has to be done by the storage system when serving each request. That means that authorization is done on the data access path, which impacts the scalability and availability of the system. Furthermore, all these systems (including Azure) authenticate the client using its shared secret key for each access request, which adds more limitations on the scale and availability. The design and distribution of the authentication and the authorization services need to be tightly coupled to the distribution of the data objects.

In summary, today's cloud solutions fall short of satisfying the requirements we define above. We proceed to present a capability-based model that addresses these requirements without compromising the system's security, scalability, or availability.

3. Secure Access Protocol. In this section we present a capability-based access control mechanism for secure access to objects in a storage cloud. Extending the OSD security model [11] and enhancing the delegation technique of Reed et al. [39], our protocol satisfies the cloud storage requirements introducing the following key functionalities: a *dynamic capability scope*, *advanced delegation*, *auditing and access confinement mechanisms.*

We assume the simple and most common cloud storage data model [3, 10], which is comprised of the following types of resources: **Namespaces**, which are abstract containers providing context (such as ownership) for the data objects; and **Data objects** of arbitrary size[1], containing arbitrary content type. Every object belongs to a single namespace and has a unique identifier within that namespace. Replication degree and replica placement are determined at the namespace level. Nevertheless, the access control protocol described here is independent of the specific data model and does not depend on these specific architectural decisions.

Capability-based access protocols involve three entities that are illustrated in Figure 3.1: *clients*; a *security/policy manager*, which authenticates and authorizes the clients and grants the credentials; and *storage servers* that enforce access control by validating the credentials. The access flow in such systems consists of

[1]Typically an implementation has a limit on object size, but this is not our concern here.

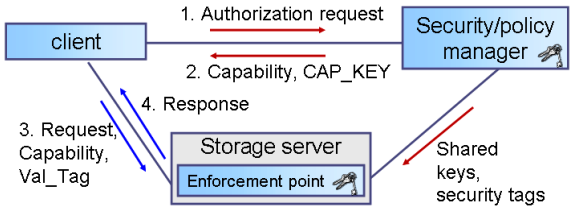


Fig. 3.1: **The OSD [31] security protocol.** The client authenticates with the security manager and receives a *credential* comprised of a *capability* and a *capability-key (CAP_KEY)*. It then sends its request together with the received capability and the validation tag (Val_Tag) to the storage server, that validates the request and then performs it.

two stages: (1) the client requests a credential from the security manager and then (2) uses it for accessing the data on the storage servers. The *enforcement point* is the component in a storage server that enforces access control.

Below we describe the capability structure and basic protocol, present the cloud-related enhancements in detail, and then illustrate them with specific examples.

3.1. Capabilities. A *capability*, in the context of this work, is a data structure that encodes certain access rights on one or more identified resources. The full details of the capability fields used in our system are provided in the Appendix. Table 3.1 presents the capability arguments that enable the new functionalities that we describe in the sections below.

3.1.1. Flexible and dynamic capability scope. In our model, the capabilities can provide a flexible description of resources based on attributes such as object name, type and metadata. To allow this, the *ResourceDescriptor* component in the capability can contain regular expressions and other selection criteria. This enables efficient data access management by allowing creation of capabilities with dynamic scope describing resources that can change over time (e.g. as new objects matching the selection criteria are added). An interesting example is for Information Lifecycle Management (ILM): a "housekeeping" application may receive a capability to move, delete or compress objects that are older than a specified age, or have not been accessed for a specified amount of time.

3.1.2. Flexible access rights. The *Permissions* field in the capability can specify any type of operation which is properly defined to the security manager and the enforcement point. For example, in addition to standard operations such as *Create*, *Read*, *Update*, *Delete*, or even *UpdateMetadata*, it can specify operations exclusive to specific content types such as *Compress*, *Resize*, *RotateFlip* or *ReduceResolution*. Such operations are introduced in some storage cloud offerings, for example, Nirvanix [32] already added methods like *RotateFlip* to their API. Using our proposed protocol, the granularity of the access policy can match the functionality supported by the storage service.

3.2. Credentials and access control flow. Below we describe the two stages of access: (1) obtaining credentials and (2) accessing the resource.

1. *Obtaining the credentials.* A client sends a request for a capability to a security manager, which in turn validates the identity of the requestor and performs an authorization process. If the request is approved, the security manager responds with the credential comprised of the *capability* and the *capability-key*. The capability (CAP) denotes the public part of the credential specifying one or more resources and the granted access rights. The capability-key (CAP_KEY) is the cryptographic hardening of a capability with a secret key and pseudorandom function (PRF), which can be a keyed cryptographic hash such as HMAC-SHA1 or HMAC-SHA256. The key used for this operation, $KeyNS$, is shared by the security manager and the storage server, for the addressed namespace. Thus, $PRF_{KeyNS}(CAP)$ denotes the pseudo-random function calculated with the key $KeyNS$ over the fields defined by the capability CAP. Since the capability-key is the secret part of the credential it should be sent to the client in encrypted form, either as part of the protocol or by using an encrypted communication channel.
   $Credential = [CAP, CAP\_KEY]$
   $CAP\_KEY = PRF_{KeyNS}(CAP)$

Field	Description
Resource – Descriptor	Resource(s) to which the capability applies (see Section 3.1.1).
Permissions	Operations allowed on the specified resource (see Section 3.1.2).
Discriminator	A nonce ensuring the uniqueness of capabilities (see Appendix).
Identity	Optional field (see Section 3.5).
Audit	Optional field (see Section 3.6).
SecurityInfo	The security method and protocol control information (see Section 3.2).
Delegatability	A boolean value allowing/disallowing delegation.
PolicyAccessTag	Used for revocation purposes (see [11]).

Table 3.1: **Capability Structure.**

Security Method	Val_Tag calculation	Setting	Benefits
CHID	Channel identifier	Secure channel (not necessarily encrypted), e.g. IPSEC, HTTPS.	Protects against replaying the same message outside the specific channel.
MSGH	Message fields (HTTP method, URI, Date, Content-Type and Content-MD5)	HTTP requests over an open channel.	Authenticates the message. Prevents modification and replay attacks.

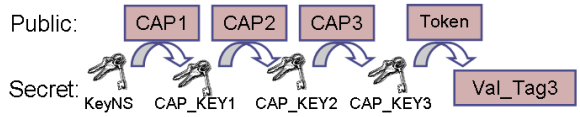
Table 3.2: **Security methods.** Summary of the security methods intended for secure and insecure channels.

2. *Accessing the resource.* To access the object of interest the client attaches to the *Request* (typically a combination of a method and data, e.g., *write* and the contents of the object to be written), a credential consisting of two parts: the capability and the validation tag (Val_Tag). The validation tag is a keyed hash that is computed with the client's capability-key; it is used to authenticate the capability and additional information as defined by the security method described below. The parameter used for the *token* depends on the security method that is encoded in the capability's $SecurityInfo$ field.
$Message = [Request, CAP, Val\_Tag]$
$Val\_Tag = PRF_{CAP\_KEY}(token)$
The choice of the security method depends on the guarantees of the underlying communication channel. We consider the following two security methods to be the most suitable for a storage cloud environment– one for a secure channel and the other for an insecure channel:
Channel identifier (CHID) This method is similar to the CAPKEY method of the OSD protocol [31]. It is suitable for use over a protected communication channel such as IPSEC or HTTPS. In this case the $Token$ is taken to be the unique channel identifier of the protected communication channel. The channel identifier should be unique to the combination of client, server, and the particular link on which they communicate, and should be known by both end points.
Message headers (MSGH). Suitable for access via an open HTTP protocol. In this case the *token* contains some of the HTTP message fields that are significant in terms of access rights. These include the standard HTTP headers such as the HTTP method and resource URI, as well as Host, Date, Content-Type and Content-MD5. As explained in more detail in Section 4, this method provides additional security in an insecure network environment at the cost of having to re-compute the Val_Tag for every command. In conjunction with the Content-MD5 field, it authenticates the data in addition to the request and capability.

Table 3.2 summarizes the two security methods and their typical use and guarantees, which are discussed further in Section 4.

When a request message is received at the enforcement point in the storage server, it is validated by performing the following steps:

1. *Calculating the capability-key.* The capability-key is computed based on the namespace key shared with



Capability	Capability Key	Validation Tag (Val_Tag)
CAP1	CAP_KEY1 = PRF_{KeyNS} (CAP1)	PRF_{CAP_KEY1} (Token1)
CAP1 CAP2	CAP_KEY2 = PRF_{CAP_KEY1} (CAP2)	PRF_{CAP_KEY2} (Token2)
CAP1 CAP2 CAP3	CAP_KEY3 = PRF_{CAP_KEY2} (CAP3)	PRF_{CAP_KEY3} (Token3)

Fig. 3.2: **Delegation chaining**. Access delegation among 3 different users. The security manager uses the namespace key, $KeyNS$, to generate a capability, $CAP1$, and a capability key CAP_KEY_1 for the first user. Using this key the user can create validation tags for his requests as well as generate the credentials for the second user, who in turn can generate the credentials for the third.

the security manager and the received capability.

2. *Calculating the expected validation tag.* The capability-key computed in the previous step is used to compute the expected validation tag, according to the security method specified in the capability.
3. *Comparing the validation tags.* The received validation tag is compared to the expected validation tag.
4. *Validating the capability and request.* The received capability is validated to be formatted correctly and to match the requested operation. After the credential is validated, other required conditions and request parameters may be checked as done normally (e.g., timestamp validation, Content-MD5 validation and so on).

3.3. User-to-user delegation. In this section we describe an advanced delegation mechanism that enables a user to create (and pass to another user) a new capability that encodes a subset of the rights on a subset of the resources encoded in an existing capability. It should be noted that delegation is an inherent feature of capability-based systems. Since the credential encodes a capability rather than an identity, it can be delegated to any principal. However, it is often desirable for a principal to delegate a subset of his rights to another principal. For example, a user that has full control over a namespace may want to allow another user to access a subset of the namespace's objects, perhaps for *read only* access. We follow a technique presented by Reed et al. [39] and adapt it to the storage cloud environment. Below we detail how the delegated credential is built and validated. Section 3.4 further illustrates these mechanisms with specific examples.

3.3.1. Delegation mechanism. During delegation a user creates and passes to another user a new capability, that is derived from the original capability he received from the security manager. Since an important goal of generating this new capability is the reduction of access rights, we term it a *reduced capability*. Having a credential $[CAP, CAP\_KEY]$, a *reduced capability credential* can be created by appending a reduced capability to the original capability, and creating a new capability-key by hashing the appended part using the original capability-key as the key.

Formally: A delegation chain is created such that $Credential_i = [CAP_i, CAP\_KEY_i]$ is defined as follows:

$$
\begin{aligned}
&Credential_1 = [CAP_1, PRF_{KeyNS}(CAP_1)] \\
&For\ n > 1: \\
&\quad CAP\_KEY_n = PRF_{CAP\_KEY_{n-1}}(CAP_n) \\
&\quad Credential_n = [(CAP_1, ..., CAP_n), CAP\_KEY_n]
\end{aligned}
$$

Notes:

- A Client cannot generate $Credential_1$ because that requires knowledge of $KeyNS$.

- Generating CAP_KEY_n requires knowledge of CAP_KEY_{n-1}. Yet, CAP_KEY_n does not reveal any useful information regarding CAP_KEY_{n-1}.
- It is crucial that all capability keys be sent over authenticated and encrypted channels.
- Our key generation technique computes CAP_KEY_n based only on CAP_n as opposed to $(CAP_1, ... , CAP_n)$ used by Reed et al. [39]. This reduces the cryptographic overhead, while preserving the same level of security.

This mechanism of creating new capabilities and capability-keys based on existing ones is illustrated in Figure 3.2, which depicts the structure of the credentials in the delegation chain.

3.3.2. Reduction of access rights. When allowing user-to-user delegation it is important to ensure that each user can only reduce his own access rights and cannot delegate access beyond his own permission. We define one capability CAP_{i+1} to be a subset of another capability CAP_i if and only if the value of each field in CAP_{i+1} does not describe a higher privilege than the corresponding field in CAP_i. For example, we require that during delegation the new *ResourceDescriptor* describes a subset of the resources, and *Permissions* describes a subset of the operations, and the *ExpiryTime* field contains a lower or equal expiration time. Two fields are an exception to this rule: (1) We require that the $SecurityInfo$ of the two capabilities contain exactly the same security method; (2) we do not compare the *Discriminator* fields.

3.3.3. Delegation validation. When a chained capability credential is processed at the enforcement point it is validated by the following mechanism, which is an extension of the one described above.

1. *Calculation of the chain of capability keys.* First, CAP_KEY_1 is calculated based on CAP_1 and $KeyNS$. Then, each subsequent CAP_KEY_i is calculated from CAP_KEY_{i-1} and CAP_i as described above (see Figure 3.2).
2. *Calculation of the validation tag.* The last capability-key in the chain is used to calculate the expected validation tag.
3. *Compare the validation tags.* Compare the received vs. the expected tags.
4. *Validation of the capability delegation chain and the request.* Validate that each capability in the chain is a proper subset of the preceding capability as defined above and that the request conforms with the last capability. This ensures one cannot increase the access granted by a given credential.

3.4. Examples. Using the scenario illustrated in Figure 3.3, in this sub-section we describe some specific examples that demonstrate the use of the delegation mechanism.

Let *SP* be a service provider using the storage cloud as its back-end to provide data services to its clients. *SP* creates a namespace *SP1* and receives a capability credential from a security manager that grants it full control over *SP1*, as shown below:

$$CAP_{SP} = \begin{bmatrix} ResourceDescriptor\ of namespace\ SP1, \\ Permissions = full\ control\ , ... \end{bmatrix}$$
$$CAP\_KEY_{SP} = PRF_{KeySP1}(CAP_{SP})$$
$$Credential_{SP} = [CAP_{SP}, CAP\_KEY_{SP}]$$

Delegating access to a subset of resources. Alice is a client of *SP* wishing to have full control over her resource, object *A*, which is stored in the aforementioned namespace *SP1*. *SP* is able to act as a manager of all access rights to *SP1* without any intervention of the storage cloud security manager. In our example, *SP* generates and sends Alice the following credential granting her the desired access rights[2]:

$$CAP_{Alice} = \begin{bmatrix} ResourceDescriptor\ of\ Object\ A \\ in\ namespace\ SP1, \\ Permissions = full\ control\ , ... \end{bmatrix}$$
$$CAP\_KEY_{Alice} = PRF_{CAP\_KEY_{SP}}(CAP_{Alice})$$
$$Credential_{Alice} = \left[(CAP_{SP}, CAP_{Alice}), CAP\_KEY_{Alice}\right]$$

When Alice uses this credential to access her resource, the enforcement point validates it in the following way: it calculates the expected validation tag (via computing the capability keys CAP_KEY_{SP} and CAP_KEY_{Alice}), compares it to the tag received, and finally it checks that CAP_{Alice} is a subset CAP_{SP} and that CAP_{Alice} is a valid capability that authorizes the requested operation.

[2]Essentially, the service provider acts as the security manager for his managed namespace

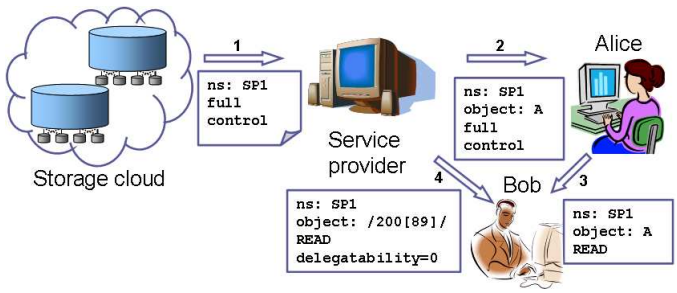


Fig. 3.3: **Delegation chaining**. Examples of user-to-user delegation: (1) The service provider (SP) uses a storage cloud as its back-end. SP receives credentials allowing it full control over its namespace $SP1$. (2) SP grants Alice a credential allowing full control over her object A stored in the storage cloud through SP. (3) Alice grants Bob a credential allowing read-only access to her object A. (4) SP grants Bob full control over objects having the strings "2008" or "2009" in their title.

Delegating a subset of the access rights. Let Bob be a user, unknown to the storage cloud or to the service provider, to whom Alice wishes to grant permission to read her object. Using the same mechanism as above, Alice can delegate a subset of her rights to Bob, creating a new credential with read-only access to her object.

$$CAP_{Bob} = \begin{bmatrix} ...\ ObjDescriptor\ of\ A, \\ Operations = READ\ ... \end{bmatrix}$$
$$CAP\_KEY_{Bob} = PRF_{CAP\_KEY_{Alice}}(CAP_{Bob})$$
$$Credential_{Bob} = \left[(CAP_{SP1}, CAP_{Alice}, CAP_{Bob}), CAP\_KEY_{Bob}\right]$$

When Bob uses this credential to access the resource, the enforcement point validates it as described above, with the added steps due to the additional capability in the chain.

Dynamic capability scope and delegation control. In another scenario, let Bob be an external financial accountant, preparing the reports for years 2008-2009. The service provider should be able to grant Bob access to the subset of objects relevant for these years. For example, these can be described as all objects with object name matching the regular expressions /200[89]/ or $/\hat{}report.+200[89]\$/$ in their name. When new objects matching the pattern are added to the system, this credential automatically provides access to them. In addition, by setting the value of the field *Delegatability* to zero, the service provider may prevent Bob from further delegating the access and generating new credentials. Figure 5.2 presents a sample access request and a credential that can be used in this example.

3.5. Identity Testing and Access Confinement. A key design point in our capability-based access control system is that it relieves the enforcement point from the duty of authenticating the identity of the request sender. Removing this requirement (as well as authorization tasks) from the data access path is instrumental in allowing high scalability and performance of the system (see further discussion in Section 6). However, at times it may be desirable to limit the use of a capability granted to a specific user or group as an extra measure of security, typically for more sensitive data. One consideration is that one may grant a user access to an object, but not trust him to keep his capability credential securely without leaking it. For this purpose, we add an optional *Identity* field in the capability that can be used to confine the credential and make it usable only by the specified identity. It may contain the identity of a principal or a group. The *Identity* is filled in by the delegating person and is verified by the enforcement point. Using this field allows enforcing the *access confinement* property, defined by Lampson [22] (see more in Section 4).

For credentials in which the *Identity* field is used, the enforcement point needs to authenticate the requesting client's identity, and potentially to validate his membership in a group. How the client is authenticated is out of the scope of this protocol. Two points worth mentioning: (1) the authorization is still separated from the enforcement point; and (2) caching of relevant identity information at the enforcement point may help improve the performance in the cases where the *Identity* field is used (as pointed out by Karger [20]).

3.6. Auditability and accountability. In most cases, an access control system is required to provide mechanisms for auditing and accountability of all accesses to a resource. The technical aspects of auditability

and accountability are similar, as both are achieved by generating information that later provides proof of access control decisions and operations. Auditing is of special concern in capability-based access control systems, since a capability-based credential, once obtained, can be used anonymously without having to authenticate the requester's identity (unless the *Identity* field is used, see 3.5). To address this concern, an *Audit* field is added to the capability. For instance, the security manager can store in this field the identity of the client to which the capability was issued. Since the *Audit* field is part of the capability, it is authenticated by the capability-key and cannot be modified. The enforcement point can log this information so it can later be used to learn which capabilities were used and when. This can serve as a tool for *billing* in the cloud, as it may record who is accountable for each access, and is especially useful in implementing today's new pay-per-access policies.

Note that the *Audit* field does not provide identity information about the entity actually accessing the resource. Rather, it provides the identity of the entity that received the credential. The entity that was granted the credential is responsible for securing it and in most cases will be accountable for all usage of the credential. During user-to-user delegation, the delegating client can use the *Audit* field to add further information on the accountability of the delegated credentials. For example, an audit log entry might record that a resource was accessed using a credential that was given to Alice at time T1 and was delegated from Alice to Bob at time T2. In this case, Bob can be held accountable for the access, and Alice can be held accountable for granting the access right to Bob. This method allows for high flexibility in billing schemes as it records the full path of delegation and access. For example, a bill for accessing data can be split between an application providing a service on the data and the end client providing credentials for accessing this data.

4. Security assessment. The access control scheme essentially inherits the security properties of the OSD protocol when translated to the cloud setting. In this section we briefly overview these properties and emphasize the new features in our protocol and their effect on security.

The basic property of an access control mechanism is naturally the assurance that an operation will only take place when it is invoked by a user that is privileged to execute it. In our system privileges are embodied in credentials at varying granularity (such as object, group of objects, read, write, etc.). Security includes denying inadvertent access attempts by users as well as attempts by malicious eavesdroppers monitoring the communication channels. It also includes preventing attacks on the network ranging from simple network errors to attempts at modifying the content of messages, and malicious replaying of messages (replay attacks).

The security of the protocols presented in this paper hinges on two basic properties, both are guarantees of the underlying pseudorandom function (PRF) [14]. The first is the inability of an adversary without knowledge of a namespace key $KeyNS$ to gain any information on a capability key CAP_KEY for a capability CAP of his choice (even when knowing keys for other capabilities). The second is the inability of an adversary that does not possess the capability key CAP_KEY for a capability CAP, to generate a request with a validation tag that corresponds to CAP.

Therefore, a crucial assumption for our setting is that all communication of keys in the protocol is done on a secure encrypted channel. This includes communicating keys to clients of the cloud, and communicating keys within the cloud–between the security manger, the key manager, and the various enforcement points. This also includes client to client communication, such as in the case of delegating capabilities.

Confinement of Corruption. We note that the architecture described in Section 5 attempts to limit the distribution of keys to the necessary minimum. For example, an enforcement point only holds the keys that pertain to the namespaces that it holds. This property ensures that in the case that a server or component are compromised, the damage is confined only to the namespaces that were directly related to this component.

4.1. Security Methods. In the next paragraphs we consider all communication, other than obtaining capabilities and keys from the security manager. We discuss the security for the two possible methods of computing validation tags.

The CHID Method. This method, called CAPKEY in the OSD standard, assumes that clients contact the storage cloud by initiating an anonymous IPSEC channel, an HTTPS connection or an equivalent secure channel. The guarantee of such a channel is that its originators are the only ones able to send messages on it. Namely, an eavesdropper cannot modify messages on the channel, replay messages on it or initiate new messages. The channel may further encrypt all communications, but this is not mandatory for access security. When working in such an environment, basic access control is achieved since an eavesdropper cannot use the messages passed on a different channel because the validation tag binds the messages to the original channel. As pointed out, generating a validation tag for a different channel, message or capability is impossible without

knowledge of the capability key. In addition, the properties of the channel protect against the various network attacks. Altogether, the protocol's use of the channel ID guarantees that no useful information is gained toward manipulation of data in other channels, even though the channel does not have to be encrypted.

The MSGH Method. This setting is very relevant for public clouds, and uses parts of the HTTP message in order to generate the validation tag. In particular, this includes the specifics of the operation, and therefore forms a binding between the credential and a specific command. During write operations, including a content-MD5 field in the generation of the validation tag also guarantees that data being written to an object cannot be modified. Essentially, this offers a guarantee that an eavesdropper intercepting the message can only use it to repeat the exact same command, in the same time frame as specified. While this does constitute a replay attack, it is very limited. There are two key techniques in further limiting replay attacks:

Timestamping The timestamp included in the computation of the validation tag creation, is used at the enforcement point to reject both outdated and future operation requests. This renders messages useless for replay outside of a small time window. However, since clock synchronization is imperfect, the policy employed by the enforcement point is to allow operations that are reasonably close to the timestamp (this clock skew parameter can be modified according to the quality of clock synchronization in the cloud). Note that there is a tradeoff between more lenient and harsher timestamping policies. A harsh policy will suffer more from occasional rejections of legal operations due to message delays and clock skews; resolving this requires regeneration of a new command. A lenient policy increases the time window in which a replay of messages can happen.

Versioning and Nonces While timestamping reduces the possibility of a replay attack significantly, there are techniques for further eliminating such risks. This is particularly relevant in the case of write operation as they may allow some harmful adversarial manipulations. For example, if a delete request for an object is followed closely by a write request, then replaying the delete request will overwrite the last operation. Such mishaps (of out of order write operations) are not restricted only to replay attacks and are also an inherent risk in systems with *eventual consistency*.[3] We suggest to piggyback methods aimed at improving the consistency model (and specifically for ensuring monotonic session write consistency) in order to further eliminate the mentioned replay attacks. For example, by the inclusion of an object version field, write commands can be accepted only for new versions (and deletes for old ones), thus guaranteeing monotonic write consistency. Including this information in the computation of the validation tag ensures that replay attacks will be ruled out altogether since no write or delete command can be repeated twice with the same version number. A similar technique is used in the CMDRSP and ALLDATA modes in the OSD protocol. These methods are similar in spirit to the MSGH method, only MSGH is tailored towards usage in a cloud environment. Specifically, the OSD protocol used a sophisticated combination of nonces and timestamping in order to eliminate replay attacks. However, this technique is less suitable for the setting of a cloud. Note that some operations, e.g., read, are less adapt to such nonce or versioning mechanisms as they do not include inherent replication: one replica is read and the others need not be notified about this.

Altogether, the MSGH method essentially provides authentication on an open and insecure channel, such as a standard HTTP request, and prevents replaying the same request by a malicious party.

4.2. Delegation Security. A central feature in our protocol is the extensive delegation capability. The main concern is the fact that in delegation, capabilities can actually be created by an external principal (external to the cloud) rather than the security manager component. Nevertheless, this additional feature does not compromise the overall security of the protocol. An attack on the system consists of a malicious party performing an operation that it is not allowed to do. For this to happen it must produce a series of capabilities along with a corresponding validation tag for its request. Moreover, this series must deviate from all such other series that it may have obtained legally. There are basically two options for an adversary to present such a series. The first option is to generate a validation tag without knowledge of the end capability key, which as above it is incapable of doing. The other option is that somewhere along the chain of delegation, the adversary obtained a capability key CAP_KEY_i for a capability of its choice without knowledge of the previous key in the chain CAP_KEY_{i-1}. This too is ruled out by the properties of the underlying pseudorandom function [14].

4.3. Limiting Delegation and Access Confinement. Lampson [22] defined the problem of *confinement* (we term this *access confinement*). In essence, it means that not only must no principal be allowed to access

[3]Eventual consistency is a consistency model that seems inevitable in a storage cloud where data is replicated over different geographical regions (see survey by Voegels [41]).

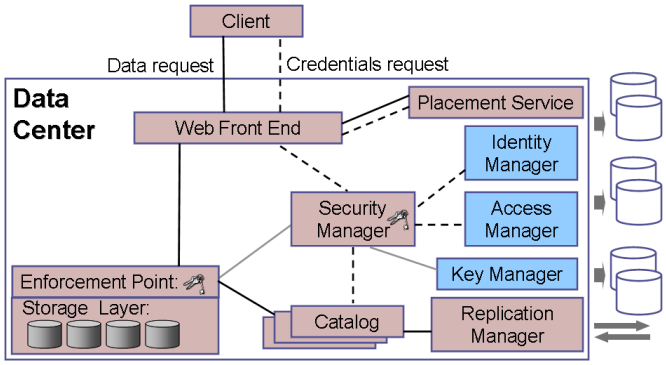


Fig. 5.1: **Secure access architecture.** Dashed lines show the interaction between the components involved in requests for credentials. Solid lines indicate the flow during the data operation request. There are several possible deployment models of these components and some of them may be distributed to several physical machines. The three access management components that are colored blue (Identity, Access and Key Managers) can be deployed at customer premises externally to the data center.

data it was not privileged to, but also that no series of supposedly legal operations by privileged users can end in leakage of information. Karger and Herbert [21] and Boebert [5] showed that the basic capability-based systems are inherently incapable of solving the access confinement problem. The problem stems from the inherent attacks allowed by delegation of credentials. Halevi et al. [15] showed how the OSD protocol can be modified to achieve access confinement, proving this statement by rigorously defining confinement and proving it under the universal composition framework [6]. Due to the similarity of our capability model to that of OSD [11], our protocol achieves the same type of security when invoked with the Identity field in the capability. Recall that when this field is used, the enforcement point is required to validate the identity of the requester and verify that it matches the one in the field. Performing identity checking at the enforcement point achieves the desired access confinement according to the definitions of Halevi et al. [15] (using essentially the same proof).

5. Architecture. We describe an architecture of a storage cloud consisting of multiple data centers (DCs), supporting the presented capability model and satisfying the requirements described Section 2.1. We consider a cloud comprised of a set of geographically dispersed DCs, collectively storing billions of objects and peta-bytes of storage capacity[4]. At this scale, no component that takes part in the data access service can be realized with a single physical entity. Every component has to be distributed and should be able to easily scale. Furthermore, every DC stores and manages only subsets of cloud resources and there is no single component contains all the information about all the objects in the storage cloud.

Figure 5.1 illustrates the main components of our architecture. We start this section with a description of the components required for the general functionality of the DC performing its basic functions such as data serving, replication and placement. Then, we describe the components of the secure access architecture and describe their integration with the rest. Finally, we discuss the scalability issues of our secure access architecture and how we address them.

5.1. General architecture. *The Web Front End* receives client requests and propagates them to the appropriate component inside the data center, or redirects them to the right target DC as needed. It integrates with the web server (httpd) and handles the HTTP protocol related functions. It is designed as a stateless component. It can dynamically scale with load balancing capabilities. According to the type of the received message, which can be either a credential or a data request, the Web Front End communicates with the Security Manager or the Enforcement Point respectively.

The Catalog is a key-value store containing object metadata, both external (such as user metadata) and internal (such as mapping to a storage location). It is accessed by other components and allows efficient lookup of data attributes. For example, supporting the dynamic capability scope, it allows recognizing the objects

[4]Amazon S3 is reported to store more than 100 billion objects as of March 2010 [http://www.datacenterknowledge.com/archives/2010/03/09/amazon-s3-now-hosts-100-billion-objects/]

with the attributes matching the specified selection criteria. It is designed to be distributed across the nodes of the same DC and dynamically scale to handle the large number of objects managed by each DC. It plays an important role in data management and replication across DCs.

The Storage Layer provides persistent storage for objects. It is responsible for allocating space and storing object data. Our prototype implementation uses a clustered file system (within each DC but not spanning DCs) but it can also use underlying block storage directly.

The Placement Service maintains the information regarding the placement of resources in data centers. In accordance to our data model, it maintains the list of data centers that hold a replica for each namespace in the storage cloud. We choose to replicate this information across all the DCs. Each data center has an always consistent list of the replicas it is responsible for; the list of other replicas that are not in its responsibility is eventually consistent.

The Replication Manager is responsible for the replication of object updates across the DCs. The replication mechanism is designed to work asynchronously, providing high availability for write with eventual consistency for read.

5.2. Secure access architecture. Here we describe the main components of the architecture related to access control. Some of the components handle the capability-based access control model described in Figure 3.1, while the others are supporting components that handle the security functions required by the model: authentication, authorization, and key management.

The Enforcement Point processes all data access requests. It enforces access control by validating the client requests and credentials. It is responsible for denying a request that does not include valid credentials authorizing it. When the CHID security method is used, the validated credentials can be cached at the Enforcement Point to improve performance by eliminating the recalculation of capability-keys and validation tags for every request validation.

*The Security Manager*is responsible for handling authorization requests and generating credentials. It orchestrates the authentication and authorization procedures as well as credential revocation, key management and key exchange with the Enforcement Point. Each Security Manager is responsible for a certain subset of resources, which usually reside in the same DC. However, to allow federation of resources, a Security Manager in one data center may serve an authorization request for the data stored in another DC. This is especially useful when two enterprises federate their storage resources without federating the identity or the access management components.

The Identity Manager authentication component responsible for verifying the identity of a principal requesting authorization, as well as group membership inquiries. Under the orchestration of the Security Manager, multiple different Identity Managers can be deployed in the cloud. This allows enterprise customers to continue using their own identity management servers, while other customers, like those providing Web 2.0 and Mashup applications, may rely on the identity management offered by the cloud provider.

The Access Manager responsible for authorization and access policy management, which serves as the basis for the decisions of the security manager that either grants or denies the requests for capabilities. Similarly to identity managers, the cloud architecture allows deployment of multiple external access managers. Each server may use a different model to manage the access control policies (e.g. ACLs, RBAC, etc) allowing compatibility with existing customer infrastructures. This allows the customers to maintain their own access policies, while relying on the cloud security manager as the credential generation authority. However, this is not mandatory and applications may select to maintain internal access managers that will be responsible for the delegation of credentials generated by the cloud provider.

The Key Manager is the key management component responsible for the secure generation, serving and storage of the cryptographic keys shared between the Security Manager and the Enforcement Point. Every namespace has a set of keys associated with it, and efficient key management is provided by storing a copy of the namespace keys in each DC that contains a replica of that namespace.

5.3. Secure access flow. To illustrate the interactions between the components we detail the flow of the two-stage access control model described in Section 3.2.

Obtaining the credentials. Upon receiving a credential request, the Web Front End checks with the Placement Service whether the local DC has a replica of the pertinent namespace (and thus has a replica of the

authorization information for it). If it has not, the message is forwarded to another DC[5]. However, if such a replica does exist in the local DC, the request is passed to the Security Manager, which in turn, communicates with the Identity Manager and the Access Manager components to authenticate the user and authorize his request. Using the key generated by the Key Manager, the Security Manager generates the capability and capability-key, which are returned to the client.

Accessing the resource. When the Web Front End receives a data access request, it also calls the Placement Service to find the replica locations for the pertinent namespace. If it resides in the local DC, the request is passed to the Enforcement Point, that validates the credential and handles the request. It may be required to replicate updates to other data canters, and the Replication Manager is called to perform this task. Since the replication mechanism is out of the scope of this paper, we do not provide details for it.

5.4. Scalability of Access Control. In this sub-section we describe how we address the scalability challenges described in Section 2.1. To ensure availability we distribute the Security Manager across all the DCs. Each Security Manager instance serves only authorization requests pertaining to the namespaces that reside in the respective DC.

Similarly, the Access Manager is distributed across the cloud, with an instance in each DC holding access policy information and serving authorization requests only for namespaces that have a replica in the local DC. This is aligned with the design of the Security Manager. It should be noted, that this distribution of access control is mainly suitable when the Access Manager components are owned by the Cloud provider. When integrating external access managers that are owned and managed by the customers, this distribution may be more difficult due to the trust establishment problems.

The Identity Manager is also distributed across the cloud. However, in contrast to the access policy information, the identity-related information cannot be partitioned as the namespaces are, because users and groups have a global scope and do not pertain to certain namespaces. Therefore, the Identity Manager component includes the ability to retrieve information from (or forwarding requests to) a remote Identity Manager, as well as the ability to cache retrieved information locally for enhanced performance. The ability to integrate with external Identity Providers is important for supporting a federated environment, although this was not part of our prototype implementation.

The Key Manager component generates and stores highly sensitive information, which enables accessing any resource in the cloud. It is divided into two parts: (1) A central Key Manager that generates keys and maintains their backup and lifecycle management. This is implemented as a set of synchronized instances set up in master-slave hierarchy with failover capability; (2) A local key management component in each DC holds encrypted copies of the keys required to serve the namespaces residing in the DC. This local "key vault" component can be reused by both the Security Manager and the Enforcement Point.

It should be noted that a credential generated and provided by the Security Manager in one DC is valid across the storage cloud and can be used to access the pertinent objects regardless of where they are replicated and accessed. The distribution of security and access control functionality is totally transparent, similar to the distribution of the data.

5.5. RESTful Implementation. We developed a prototype of a storage cloud comprised of data centers with the architecture described above. One of our design principles was to support the philosophy of REST (REpresentational State Transfer) [13], which became widely adopted due to its simplicity, scalability and easy deployment [36, 17]. In order to implement the capability-based access control model as part of a RESTful web service, we followed all the fundamental principles of REST, which in our set up can be formulated as follows: (1) All resources, including objects, namespaces and credentials are referenced using a uniform resource identifier (URI); (2) All resources are manipulated by the basic HTTP methods (GET, PUT, DELETE and POST); and (3) All resource requests are stateless.

To support this, our system provides a RESTful API for the communication with the Security Manager, addressing the requested credential as a resource. In essence, an authorization request is sent using an HTTP GET request to a resource of type *credential*. To allow the easy integration of credentials in data access requests, we embed them in the HTTP headers of namespace and data object requests. They are represented as JavaScript Object Notation (JSON) strings, as illustrated in Figure 5.2, which presents an example of a user request to access (GET) an object. It shows the capability and the validation tag that are sent in an HTTP request of a

[5]It may also be redirected to another DC with HTTP redirect.

```
GET SP1/report-March-2009.doc HTTP/1.1
host: www.cloud.acme.com
x-acme-credential:
  [{ capability: [
    { ResourceDescriptor :
      [{ nsid :SP1},{ security tag: 1}]
      [{oid=~/^report.+200[89]$/}]
    { Operations : read, add},
    { Discriminator : 0xFDCED0016EE87953472},
    { Identity :  }, { Audit : Bob},
    { ExpiryTime : "Thu, 31 Jan 2011 17:15:03 GMT},
    { Delegatability: 0 }]
  { Val_Tag: 0xAABF099916EE87953472}]} ]
```

Fig. 5.2: HTTP request to GET an object from the storage cloud. A credential containing a capability and a validation tag are added to the HTTP header. The credential value is represented as a JSON string.

client to the server. Our prototype shows that it is possible to implement the capability-based access control model as a RESTful protocol in which all the resources (including the credentials) are controlled by components that communicate via a standardized HTTP interface and exchange representations of these resources.

6. Discussion. In this section we begin with a general discussion of the architecture of access control systems. Then we revisit the requirements set forth in Section 2.1, and compare the capability-based access control architecture described in Section 5 to alternative access control mechanisms showing how it satisfies the requirements.

6.1. Architectural Overview of Access Control. Every secure access control system has three components:

1. *Authentication*, which authenticates the client entities and reliably assigns them identities.
2. *Authorization*, which makes authorization decisions allowing or denying access by clients to resources.
3. *Enforcement* component, which enforces access control on resources based on information coming from the client, authentication and authorization.

Clearly, enforcement must be done in the data access path. However, authentication and authorization can be done outside the data access path. Access control systems differ on where and when authentication and authorization are done. We identify three main access control architectures and name them accordingly.

Monolithic access control. In this type of access control architecture authentication, authorization and enforcement are all done on the data access path. The client passes authentication information, which the data server uses to identify and authenticate the client, and authorize it to access the requested resource. Although the authentication or the authorization components may execute separately from the enforcement point, they are used by the data server as part of the enforcement in the data access path.

Separation of authentication. In this type of access control architecture, authentication is taken out of the data access path. The authentication component authenticates a client and provides it with a token that vouches for its identity. The token is validated on the data access path without the client needing to provide its authentication information (other than the token). Authorization decisions are still made for each data access request.

Separation of authorization. In this type of access control architecture, both authentication and authorization are separated from the data access path. A client submits a request to access a resource (or a set of resources). After it is authenticated by the authentication component, the authorization component makes the access decision based on its authenticated identity and the access policy for the requested resources. Then the client receives an access token that encodes a *capability* to access the resources. This access token is validated by the data server as part of the enforcement in the data access path.

The capability-based access control model proposed in this paper separates both authentication and authorization from the enforcement point, streamlining the data access path and allowing maximum flexibility and scalability of access control as required in the cloud environment. We explain the benefits of capability-based access control in the following subsection.

6.2. Advantages of capability-based access control.

Chains of services. Only the capability-based access model allows propagating the end user access rights securely through a chain of services all the way down to the accessed resource. When resources are accessed by services on behalf of clients, possibly through a chain of services, capability-based access control enables propagating the end-client access token and enforcing the access control based on the authorization of the end-client. This allows preventing attacks like clickjacking and cross-site request forgery that can occur due to the violation of the principle of least privilege possible in the ACL-based models [7].

When authentication is not separated from the data access path, authentication information has to be propagated through the chain of services all the way down to the enforcement point that calls the authentication component. When authorization is not separated from the data access path, identity information has to be propagated all the way down to the enforcement point that calls the authorization component. Propagating authentication and identity information through the chain of services is cumbersome and reduces the overall security. Furthermore, this information is variable in size and format, since there are numerous standard authentication and authorization methods, each with its own definitions and models. Propagating a capability-based access token allows a unified service protocol, while enabling different authentication and authorization components to be plugged into the system.

User-to-user access delegation. Capability-based access control facilitates access delegation and empowers each client entity to play the authorization role for resources to which it has access. This facilitates several use-cases that are essential to cloud environments. Discretionary access control (DAC) is facilitated by providing every user the ability to delegate his/her access rights to other end users, even when they do not even have a recognized identity in the storage system. (The delegation of access can be limited and controlled as explained in Section 3). Hierarchical authorization structure is facilitated by providing access credentials to domain and sub-domain administrators, allowing them to delegate their access to other entities (subordinate administrator and end-client entities) using authentication and authorization components of their choice and control.

Performance. Capability-based access control systems pay the price of authentication and authorization up-front, and then streamline the data access path by using a capability-based access credential that is propagated from the client entity to the enforcement point. If every access to a resource requires the client to get a credential for the resource, the capability-based system suffers from an overhead that stems from the fact that the client has to perform two operations serially - first get a credential and then access the resource. However, when an access credential is reused in many resource requests, capability-based access control is beneficial and improves the performance. The improvement is particularly significant in large scale environments in which the enforcement point is relieved from accessing remote centralized authentication and authorization servers in the data access path.

When a client sends a request to a data server, a credential is embedded in the request, and the validation of the credential is done locally at the enforcement point without interactions with remote entities. Thus, the overhead is primarily due to the cryptographic calculation required for validation. In particular, the expected Val_Tag is calculated and compared to the one sent by the client (see Section 3.2). The time overhead depends on the pseudorandom function used, the size of the capability, c, the length of the delegation chain d, and the size of data over which the Val_Tag is computed e. Let us denote the time it takes to compute the pseudorandom function over data of length n as $PRF(n)$. Then the time to compute the validation tag at the server side can be described by the following formula:

$Val\_Time(c, d, e) = d \cdot PRF(c) + PRF(e)$

The time increases with the number of delegations. We considered two implementations with HMAC-SHA1 and HMAC-SHA256 used to calculate both the CAP_KEY and Val_Tag. On an average capability size of 400 bytes, the average CAP_KEY calculation time with these functions was 14μ and 23μ respectively[6]. The calculation of the validation tag depends on the security method and when using the $MSGH$ method the size of the input message headers may be about the same size of a capability. Thus, the overall validation time of a message with delegation chain length d can be approximated by $(d + 1) \cdot PRF(c)$. For example, when using HMAC-SHA256 the validation of delegations of length 4 and 5 will take 115μ and 138μ respectively. That is the time it takes on the server side. On the client side it only takes $PRF(e)$ because the CAP_KEY is already computed when it receives the credential. Since the data access latencies in scalable and distributed storage cloud is usually measured in tens of milliseconds [8], we consider the overhead of the security mechanism to be insignificant.

[6]The measurements were taken on a system with a quad core 2.83 GHz processor

High availability and scalability. Taking authentication and authorization services out of the data access path improves the availability and scalability of the data by eliminating potential points of failure. Intuitively, it may seem that tying authentication and authorization to the enforcement point (as in monolithic access control) improves the availability of the system, as the client interfaces with a single service rather than two different services in order to access the resource. However, this is not really the case when the storage system scales horizontally. In this case the authentication and authorization components need to be (highly) available to the enforcement point at all times. When separating the authentication and authorization from the data access path, these components can more easily be designed for high availability independent of the data service itself. In addition, it should be noted that once a client receives a capability-based credential, it can use it even when the authentication and authorization components are unavailable. The credential can be validated by the enforcement point without requiring the availability of authentication or authorization services.

Our architecture for the data service is distributed across multiple data servers, where each namespace can be served by a set of data centers (typically a small number). From an availability perspective, we designed the components in a way that does not reduce the availability of the data itself. Therefore, we have a security manager component at each DC. But, as with the data itself, not all the authorization information is replicated across all the DCs. Access policy information (i.e., governing authorization), is replicated at granularity of a namespace and aligned with the data placement and replication. Thus, wherever a resource is stored, so is the policy information to authorize its access. In terms of authentication, since the scale is much lower, we replicate all the identity management information across all the DCs. Unlike the access policy data, which changes rapidly as objects are created and deleted, the client information is much more stable and doesn't change as frequently. Therefore, the instance of the security manager available in each DC is capable of authenticating any client and authorizing any access to the namespaces with a replica in that DC.

Revocation. Systems with ACLs allow manual revocation which is achieved by changing the access control configuration. With the recent progress of ACL-based models which allow richer and more complicated description of access rights [29] this mechanism becomes more exposed to human errors. For example, consider a situation where a user changes the ACLs configuration for delegation purposes to grant a one time access to a resource. The granting principle can easily forget to delete this configuration to revoke the unnecessary access rights. Unlike ACL-based systems, the proposed capability-based model has the advantage of supporting three mechanisms for automatic and manual revocation with different level of granularity: (1) Automatic revocation through the "ExpiryTime" field, which determines the time period during which the capability is valid; (2) Manual per resource revocation which is achieved by changing the field "PolicyAccessTag" as described in [11]; (3) Per namespace revocation which is achieved by replacing the keys shared between the security manager and enforcement point.

6.3. Overcoming the limitations of capability-based models. While capability-based access control has multiple advantages it is also important to discuss its limitations. Below, we point out the main barriers in the adoption of capability-based models and describe the solutions that allow to overcome the shortcomings of capabilities while benefiting from their advantages.

Interoperability. Resource-oriented access control model, such as those based on ACLs, are the most common access control mechanism in use and pure capability based systems are implemented mostly in research systems [29, 18]. Thus, pure capability based systems have interoperability problems, which limit their potential adoption. Thus, we propose an architecture that allows the integration of external identity or access management components that are already used by enterprises and applications. These can support ACL-based, RBAC or any other access control model that fulfill the monitoring or compatibility requirements of the enterprise. The architecture that we propose in this paper allows to utilize the access control configurations and decisions made by these servers to grant capabilities that will provide some extra functionality such as the user-to-user delegation of a subset of access rights that is not provided by currently existing servers. We believe that such a hybrid system, which allows to combine the advantages of capabilities with other access control models, has higher chances to be adopted in commercial production systems.

Per-resource auditability. Unlike ACL based systems, the capability models allow an easy tracking of all the resources that a specific user can access. However, they do not allow an easy monitoring of all the users who have an access to a specific resource [18]. To support this type of monitoring a capability-based system should implement an additional layer of tracking the granted access rights. The data center architecture proposed in this paper addresses this requirement by allowing to incorporate external access management servers, which

commonly implement the support the required monitoring capabilities either with the help of policy lists or ACLs.

7. Conclusions and Future Work. We present an access control mechanism, which addresses the new challenges brought forth by the scale and applications introduced in the cloud setting. Observing that most ACL-based systems are limited in their ability to support such features, we developed a capability-based system that addresses them. We present a general architecture of a data center in a storage cloud with integrated security components that addresses the scalability requirements of storage cloud systems. Our architecture allows integration of existing access control solutions toward hybrid architectures that combine the benefits of capability-based models with other commonly used mechanisms such as ACLs or RBAC. We built a prototype implementing each of the components.

In the future we intend to work on the integration of this architecture with existing enterprise systems and commonly used access control standards such as SAML [38]. We hope to use our experimental system to evaluate the overall performance of the access control mechanism on real workloads. We are encouraged by our initial evaluation and are working to incorporate this work in a production storage cloud system. Lastly, we aim to address the challenges raised in federation scenarios, where several enterprises need to federate their resources across geographically distributed administrative domains, while ensuring comprehensive and transparent data interoperability.

8. Acknowledgments. We would like to thank Guy Laden and Eran Rom for their contribution to the development and implementation of the data center architecture presented in this paper. We would also like to thank Dalit Naor for her contribution to the ideas presented, and Aviad Zuck for the prototype implementation performance measurements. The research leading to these results has received funding from the European Community's Seventh FrameworkProgramme (FP7/2007-2013) under grant agreement n 257019.

REFERENCES

[1] Trusted computer system evaluation criteria. Technical Report DoD 5200.28-STD, Department of Defense, December 1985. http://csrc.nist.gov/publications/history/dod85.pdf, accessed Jan 20, 2010.

[2] *Amazon Simple Storage Service Developer Guide (API Version 2006-03-01).* Amazon, a. http://docs.amazonwebservices.com/AmazonS3/2006-03-01/, accessed Jan 12, 2010.

[3] *Amazon Simple Storage Service (Amazon S3).* Amazon, b. http://aws.amazon.com/s3/.

[4] M. Armbrust, A. Fox, R. Griffith, A. D. Joseph, R. Katz, A. Konwinski, G. Lee, D. A. Patterson, A. Rabkin, I. Stoica, and M. Zaharia. Above the clouds: A berkeley view of cloud computing. Technical Report UCB/EECS-2009-28, EECS Department, University of California, Berkeley, February 2009. http://www.eecs.berkeley.edu/Pubs/TechRpts/2009/EECS-2009-28.html.

[5] W. Boebert. On the inability of an unmodified capability machine to enforce the *-property. In *7th DOD/NBS Computer Security Conference*, pages 291–293, 1984.

[6] R. Canetti. Universally composable security: A new paradigm for cryptographic protocols. In *FOCS*, pages 136–145, 2001.

[7] T. Close. ACLs don't. http://www.hpl.hp.com/techreports/2009/HPL-2009-20.pdf.

[8] G. Decandia, D. Hastorun, M. Jampani, G. Kakulapati, A. Lakshman, A. Pilchin, S. Sivasubramanian, P. Vosshall, and W. Vogels. Dynamo: amazon's highly available key-value store. In *SOSP '07: Proceedings of twenty-first ACM SIGOPS symposium on Operating systems principles*, pages 205–220. ACM Press, 2007.

[9] *Atmos Online Programmer's Guide.* EMC, a. https://community.emc.com/docs/DOC-3481, accessed Jan 12, 2010.

[10] *EMC Atmos Online Services.* EMC, b. http://www.emccis.com/.

[11] M. Factor, D. Nagle, D. Naor, E. Riedel, and J. Satran. The OSD security protocol. In *IEEE Security in Storage Workshop*, pages 29–39, 2005.

[12] M. Factor, D. Naor, E. Rom, J. Satran, and S. Tal. Capability based secure access control to networked storage devices. In *MSST '07: Proceedings of the 24th IEEE Conference on Mass Storage Systems and Technologies*, pages 114–128, Washington, DC, USA, 2007. IEEE Computer Society. ISBN 0-7695-3025-7. doi: http://dx.doi.org/10.1109/MSST.2007.6.

[13] R. T. Fielding and R. N. Taylor. Principled design of the modern web architecture. *ACM Trans. Internet Technol.*, 2(2): 115–150, May 2002. URL `http://dx.doi.org/10.1145/514183.514185`.

[14] O. Goldreich. *Foundations of Cryptography: Basic Tools.* Cambridge University Press, 2000.

[15] S. Halevi, P. Karger, and D. Naor. Enforcing confinement in distributed storage and a cryptographic model for access control. Cryptology ePrint Archive, Report 2005/169, 2005. `http://eprint.iacr.org/`.

[16] E. Hammer-Lahav. *The OAuth 1.0 Protocol.* Internet Engineering Task Force, February 2010. http://tools.ietf.org/html/draft-hammer-oauth-10.

[17] H. Han, S. Kim, H. Jung, H. Y. Yeom, C. Yoon, J. Park, and Y. Lee. A RESTful approach to the management of cloud infrastructure. In *Proceedings of the 2009 IEEE International Conference on Cloud Computing*, CLOUD '09, pages 139–142, Washington, DC, USA, 2009. IEEE Computer Society. ISBN 978-0-7695-3840-2. doi: http://dx.doi.org/10.1109/CLOUD.2009.68. URL `http://dx.doi.org/10.1109/CLOUD.2009.68`.

[18] V. C. Hu, D. F. Ferraiolo, and D. R. Kuhn. *Assessment of Access Control Systems*. NIST IR 7316, September 2006. csrc.nist.gov/publications/nistir/7316/NISTIR-7316.pdf.

[19] J. Ioannidis, S. Ioannidis, A. D. Keromytis, and V. Prevelakis. Fileteller: Paying and getting paid for file storage. In M. Blaze, editor, *Financial Cryptography*, volume 2357 of *Lecture Notes in Computer Science*, pages 282–299. Springer, 2002. ISBN 3-540-00646-X.

[20] P. Karger. Improving security and performance for capability systems, ph.d. dissertation. Technical Report 149, Cambridge, England, 1988.

[21] P. Karger and A. Herbert. An augmented capability architecture to support lattice security and traceability of access. In *IEEE Symposium on Security and Privacy*, pages 2–12, 1984.

[22] B. Lampson. A note on the confinement problem. *Commun. ACM*, 16(10):613–615, 1973.

[23] A. W. Leung, E. L. Miller, and S. Jones. Scalable security for petascale parallel file systems. In *SC '07: Proceedings of the 2007 ACM/IEEE conference on Supercomputing*, pages 1–12, New York, NY, USA, 2007. ACM. ISBN 978-1-59593-764-3. doi: http://doi.acm.org/10.1145/1362622.1362644.

[24] A. Levine, V. Prevelakis, J. Ioannidis, S. Ioannidis, and A. D. Keromytis. WebDAVA: An administrator-free approach to web file-sharing. In *WETICE*, pages 59–64. IEEE Computer Society, 2003. ISBN 0-7695-1963-6.

[25] E. Messmer. Are security issues delaying adoption of cloud computing? *Network World*, April 2009. http://www.networkworld.com/news/2009/042709-burning-security-cloud-computing.html, accessed Jan 21, 2010.

[26] *Windows Azure Platform*. Microsoft, a. http://www.microsoft.com/windowsazure/windowsazure/.

[27] *Windows Azure Storage Services API Reference*. Microsoft Corp., b. http://msdn.microsoft.com/en-us/library/dd179355.aspx, accessed Jan 17, 2010.

[28] S. Miltchev, V. Prevelakis, S. Ioannidis, J. Ioannidis, A. D. Keromytis, and J. M. Smith. Secure and flexible global file sharing. In *USENIX Annual Technical Conference, FREENIX Track*, pages 165–178. USENIX, 2003. ISBN 1-931971-11-0.

[29] S. Miltchev, J. M. Smith, V. Prevelakis, A. Keromytis, and S. Ioannidis. Decentralized access control in distributed file systems. *ACM Comput. Surv.*, 40(3):1–30, 2008. ISSN 0360-0300. doi: http://doi.acm.org/10.1145/1380584.1380588.

[30] R. L. Mitchel. Cloud storage triggers security worries. *Computerworld*, July 2009. http://www.computerworld.com/s/article/340438/ Confidence_in_the_Cloud, accessed Jan 21, 2010.

[31] D. Nagle, M. Factor, S. Iren, D. Naor, E. Riedel, O. Rodeh, and J. Satran. The ANSI T10 object-based storage standard and current implementations. *IBM Journal of Research and Development*, 52(4-5):401–412, 2008.

[32] *Nirvanix Web Services API Developer's Guide*. Nirvanix, a. http://developer.nirvanix.com/sitefiles/1000/API.html, accessed Jan 13, 2010.

[33] *Nirvanix Storage Delivery Network*. Nirvanix, b. http://www.nirvanix.com/.

[34] Z. Niu, H. Jiang, K. Zhou, T. Yang, and W. Yan. Identification and authentication in large-scale storage systems. *Networking, Architecture, and Storage, International Conference on*, 0:421–427, 2009.

[35] *ParaScale Storage Cloud Supports Virtual File System*. ParaScale. http://www.parascale.com/index.php/products/data-access-security.

[36] C. Pautasso, O. Zimmermann, and F. Leymann. Restful web services vs. "big"' web services: making the right architectural decision. In *Proceeding of the 17th international conference on World Wide Web*, WWW '08, pages 805–814, New York, NY, USA, 2008. ACM. ISBN 978-1-60558-085-2. doi: http://doi.acm.org/10.1145/1367497.1367606. URL `http://doi.acm.org/10.1145/1367497.1367606`.

[37] *The Rackspace Cloud: Cloud Files*. Rackspace. http://www.rackspacecloud.com/cloud_hosting_products/files/.

[38] N. Ragouzis, J. Hughes, R. Philpott, E. Maler, P. Madsen, and T. Scavo. *Security Assertion Markup Language (SAML) V2.0 Technical Overview*. OASIS Committee Draft, March 2008. http://www.oasis-open.org/committees/download.php/27819/sstc-saml-tech-overview-2.0-cd-02.pdf.

[39] B. C. Reed, E. G. Chron, R. C. Burns, and D. D. Long. Authenticating network-attached storage. *IEEE Micro*, 20:49–57, 2000. ISSN 0272-1732. doi: http://doi.ieeecomputersociety.org/10.1109/40.820053.

[40] J. H. Saltzer and M. D. Schroeder. The protection of information in computer systems. *In Proc. IEEE*, 63(9):1278–1308, 1975. http://web.mit.edu/Saltzer/www/publications/protection.

[41] W. Vogels. Eventually consistent. *Queue*, 6(6):14–19, 2008. ISSN 1542-7730.

[42] S. A. Weil, S. A. Brandt, E. L. Miller, D. D. E. Long, and C. Maltzahn. Ceph: A scalable, high-performance distributed file system. In *OSDI*, pages 307–320. USENIX Association, 2006.

[43] T. Wilson. Security is chief obstacle to cloud computing adoption, study says. *DarkReading: Security*, November 2009. http://www.darkreading.com/securityservices/security/perimeter/ showArticle.jhtml?articleID=221901195, accessed Jan 21, 2010.

9. Appendix. Below are the full details of the capability arguments used in our system.

$$CAP = \left[\begin{array}{l} ResourceDescriptor,\ Operations, \\ Discriminator,\ Identity,\ Audit, \\ ExpiryTime,\ SecurityInfo,\ Delegatability, \\ DerivedFrom \end{array}\right]$$

$$ResourceDescriptor = \left[\begin{array}{l} NamespaceIdentifier,\ ObjectIdentifier \\ ResourceType,\ ResourceCreationTime, \\ SecurityTag \end{array}\right]$$

- *ResourceDescriptor* - defines the resources to which the capability applies. In our data model, this construct consists of the following fields:
  (1) $NamespaceIdentifier$ - a unique identifier that describes the namespace resource; (2) $ObjectIdentifier$ - an identifier, uniques within the namespace, that describes the object resource or resources; (3) $ResourceType$ - the type of the resource to which the capability applies. For example, this field can be 'namespace', in a capability allowing to add objects to a namespaces. (4) $ResourceCreationTime$ - specifies the resource creation time, preventing the situation where a resource is deleted and a another resource with the same name is created; (5) $SecurityTag$ - is a security tag, which is used for revocation. In order to revoke the capabilities of a certain resource, the security manager increases the security tag and notifies the enforcement point. The enforcement point checks the security tag and considers all the capabilities with an old tag as invalid.
- *Operations* - describes the operations allowed to the client possessing the credential on the described resource. It can describe any type of operation, which is properly defined to the security manager and the enforcement point. For example, in addition to standard operations such is *read*, *write* and *execute* it can describe new operations like *zip* or *reduce resolution*.
- *Discriminator* - is a unique nonce which allows distinguishing between different capabilities even when they allow similar access rights. To ensure uniqueness even when generated by different security managers, discriminators can consists of such parameters as timestamp and IP address.
- *Identity* - optional field containing the identity of a principle or a group allowed to use the capability. When this field is empty the capability can be used by anyone who posses it (see Section 3.5).
- *Audit* - optional field containing the client or application information required for auditing (see Section 3.6).
- *ExpiryTime* - the capability expiration time.
- *DerivedFrom* - prior capabilities used for delegation chaining (see the example below).
- *Delegatibility* - setting this field to zero or one, prevents or allows the user-to-user delegation respectively.
- $SecurityInfo$ - identifies the security method and the calculation of the message validation tag as detailed in the next section.

Edited by: Dana Petcu and Jose Luis Vazquez-Poletti
Received: August 1, 2011
Accepted: August 31, 2011

Scalable Computing: Practice and Experience
Volume 12, Number 3, pp. 337–350. http://www.scpe.org
ISSN 1895-1767

A VIRTUALIZATION-BASED APPROACH TO DEPENDABLE SERVICE COMPUTING

CIPRIAN DOBRE*, FLORIN POP†, VALENTIN CRISTEA‡, AND OVIDIU-MARIAN ACHIM§

Abstract. Dependability represents a critical requirement for modern distributed systems. Today new requirements emerged. Among them reliability, safety, availability, security and maintainability are needed by more and more modern distributed service computing architectures. In this paper we present an approach to ensuring dependability in distributed infrastructures using virtualization for fault-tolerance, augmented with advanced security models. The proposed solution is part of a hierarchical architectural model that allows a unitary and aggregate approach to dependability requirements while preserving scalability of large scale distributed systems. In this context we propose dependability solutions based on the use of virtualization and modern security models, combined together to automatically protect services and applications belonging to the distributed system. We also present evaluation results for several scenarios, involving applications and services with different characteristics.

Key words: virtualization, dependability, large scale distributed systems

AMS subject classifications. 15A15, 15A09, 15A23

1. Introduction. Dependability represents a critical requirement for modern distributed systems. Both in the academic and industrial environments there is an increasing interest in large scale distributed systems (LSDS), which currently represent the preferred instruments for developing a wide range of new applications. While until recently the research in the distributed systems domain has mainly targeted the development of functional infrastructures, today researchers understand that many applications, especially commercial ones, have complementary necessities that the "traditional" distributed systems do not satisfy. Together with the extension of the application domains, new requirements have emerged for LSDS. Among these requirements, reliability, safety, availability, security and maintainability are needed by more and more modern distributed applications.

In systems composed of many resources the probability of a fault occurring is higher than in traditional infrastructures. When failures do occur, the system should limit their effects and possible even initiate a recovery procedure as soon as possible. Dependability, therefore, depends on the possibility to detect failures, and successfully recover from them. By failures we mean both hardware and software, permanent or transient, but also failures to execute operations as well as security breaches in the LSDS. Thus dependability also includes the use of adequate security models, policies, and technologies to detect and limit the effect of possible entities not obeying well-established rules.

The main contributions of this paper are: (1) *a solution designed to detect application- or service- specific failures occurring in different parts of the system*, (2) *a virtualization-based solution designed to facilitate fault recovery by freezing services running in good states and which further uses these virtual images for future state recovery or automatic migration of entire file systems, or sets of processes, for increased LSDS redundancy*, and (3) *a Mandatory Access Control (MAC) security layer designed to encapsulate services in layers with highly well-protected security access control rules.* Used together, these contributions can increase dependability in case of traditional service-based LSDS. We propose a virtualization approach to ensuring dependability by using checkpoint strategies and protection domains using virtual hosts, coupled with a proactive replication strategy necessary to maintain consistent states of the system in case of failures. The solution is part of the DEPSYS dependability architecture ([1]). It assumes that on top of a typical operating system the services run in specialized virtual environments. Such virtual environments are saved and, in case of failures, moved and re-executed on another workstation in the distributed system. The re-execution also uses appropriate consistency algorithms.

The virtual environments running on top of the operating systems and hosting the services running inside

*Computer Science Department, Faculty of Automatic Controls and Computers, University POLITEHNICA of Bucharest, Romania(ciprian.dobre@cs.pub.ro principal author).

†Computer Science Department, Faculty of Automatic Controls and Computers, University POLITEHNICA of Bucharest, Romania(florin.pop@cs.pub.ro corresponding author).

‡Computer Science Department, Faculty of Automatic Controls and Computers, University POLITEHNICA of Bucharest, Romania(valentin.cristea@cs.pub.ro).

§Computer Science Department, Faculty of Automatic Controls and Computers, University POLITEHNICA of Bucharest, Romania(ovidiu.achim@cs.pub.ro).

the distributed system form a separate layer. It allows better fault tolerance, by separating faults in different containers, or replication of virtual sandboxes to multiple nodes. It also allows quick integration with advanced security policies, a second requirement for dependability. We present examples of complementing the virtualization approach with security policies directly at the level of the operating system.

The rest of this paper is structured as follows. Section 2 presents related work. Section 3 presents the architectural design on which the dependability layer is based. In Section 4 we present the virtualization-based approach. Section 5 presents solutions designed to secure services and virtual containers, using modern security models, in large scale distributed systems. Section 6 presents experimental results. In Section 7 we conclude and present future work.

2. Related work. *Fault tolerance* includes detection and recovery. *Fault detection* in LSDS was approached in [2] through an adaptive system. The detection of faults is achieved by monitoring the system and dynamically adapting the detection thresholds to the runtime environment behavior. The prediction of the next threshold uses a Gaussian distribution and the last known timeout samples. The solution has several limitations. It cannot differentiate between high response times that are due to the transient increase of the load in the communication layer and those due to service failures, so that both are interpreted as service failures. We solve these problems and propose a solution which adapts to both the failures in the infrastructures, but also to the different requirements imposed by applications (for example, real-time application require a higher response time, instead of failure detection accuracy).

For *failure recovery* virtualization has lately enjoyed a great surge of interest. Still, with few exceptions, current solutions in this space have been largely ad-hoc. Authors of [21] analyze methods of leveraging virtualization for addressing system dependability issues. Their analysis, based on combinatorial modeling, show that unless certain conditions (e.g., regarding the reliability of the hypervisor and the number of virtual machines) are met, virtualization could in fact decrease the reliability of a single physical node. In fact, motivated by this observation, the authors propose a reliable Xen virtual machine monitor (VMM) architecture, called R-Xen. It consists of a hypervisor core and a privileged virtual machine called Dom0. Dom0, being much bulkier than the hypervisor core, is the weak link for Xen reliability. We extend such results and present a solution which considers replication inside the virtual machines sets of virtual containers, each one hosting sets of services. Thus, we protect the services as replicated containers, and the containers as part of replicated virtual machines. The virtual machine monitoring architecture is completely shielded inside the VM, thus our solution solves the problem of weak links regarding reliability.

An alternative solution proposed in [3] groups several server nodes into a set that appears virtually to clients as a single node. Upon receiving a request from a client, a node forwards the request to the nearest neighbor that offers the service. The service discovery is performed using an amended version of anycast routing scheme by using the properties of the Mobile IPv6 protocol. The disadvantage of this solution is that it works only on nodes running the XtreemOS operating system. Also, the system assumes the clients support the Mobile IPv6 protocol. The solution concentrates on the mechanisms to detect a working service from a set of replicated services. However, it does not include mechanisms to recover a request when a service fails. We present a more generic failure recovery mechanism that masks the replication of virtual nodes (combined with a container-based solution we previously demonstrated in [1]), works with a wide range of transport protocols, detects failures with higher accuracy, and takes recovery decisions that are adequate to be used with various SOA middleware.

A solution to handle fault tolerance in Grid systems is presented in [4]. The paper describes a resource failure detector, together with a fault manager, that guarantees that tasks submitted are completely executed using the available resources. The solution uses the Intra-cluster and Intra-grid load balancing model [5]. It assumes a Grid architecture that is mapped on a tree structure, where several fault managers collect failure information from fault detectors running on lower level nodes. The idea is similar to the one used in DIGS [6], which aims to increase fault tolerance of web services using the model of a fault-tolerant container. A container is a logical set consisting of several service instances. All requests to these services are mediated by specialized entry points. This is used to enforce access policies, and to increase fault tolerance of service accesses. The *security* is therefore approached at the level of the container, much like in our case. For fault-tolerance each container manages a set of replicated service instances. The containers can be configured with various fault tolerance policies. For example, an equivalent service instance is invoked when another one fails, or multiple equivalent instances are invoked with the same request and a voting mechanism is applied. However, the proposed solution uses one proxy for each service. The client accesses the service through a proxy, not directly,

so that the use of the Proxy server is not transparent to the user. To access the service container customers must know the URI of the Proxy service. In addition, the replicated business services invoked by the proxy are not necessarily deployed in the same container as the proxy service, which claims for the use of the URIs of replicas for invocations. The solution proposed in our paper eliminates these disadvantages. Each container is also protected against unauthorized accesses through a dedicated MAC security layer, designed with well-protected security access control rules.

3. An architectural model for dependability in large scale distributed systems. The proposed dependability approach is part of the architectural model that we proposed in [1]. The general approach to ensuring fault tolerance in LSDS consists of an extensible architecture that integrates services designed to handle a wide-range of failures, both hardware and software. These services can detect, react, and confine problems in order to minimize damages. By learning and predicting algorithms they are able to increase the survivability of the distributed system. They include services designed to reschedule jobs when resources on which they execute fail, services capable to replicate their behavior in order to increase resilience, services designed to monitor and detect problems, etc. The proposed architecture is also based on a minimal set of functionalities, absolutely necessary to ensure the fault tolerance capability of distributed systems.

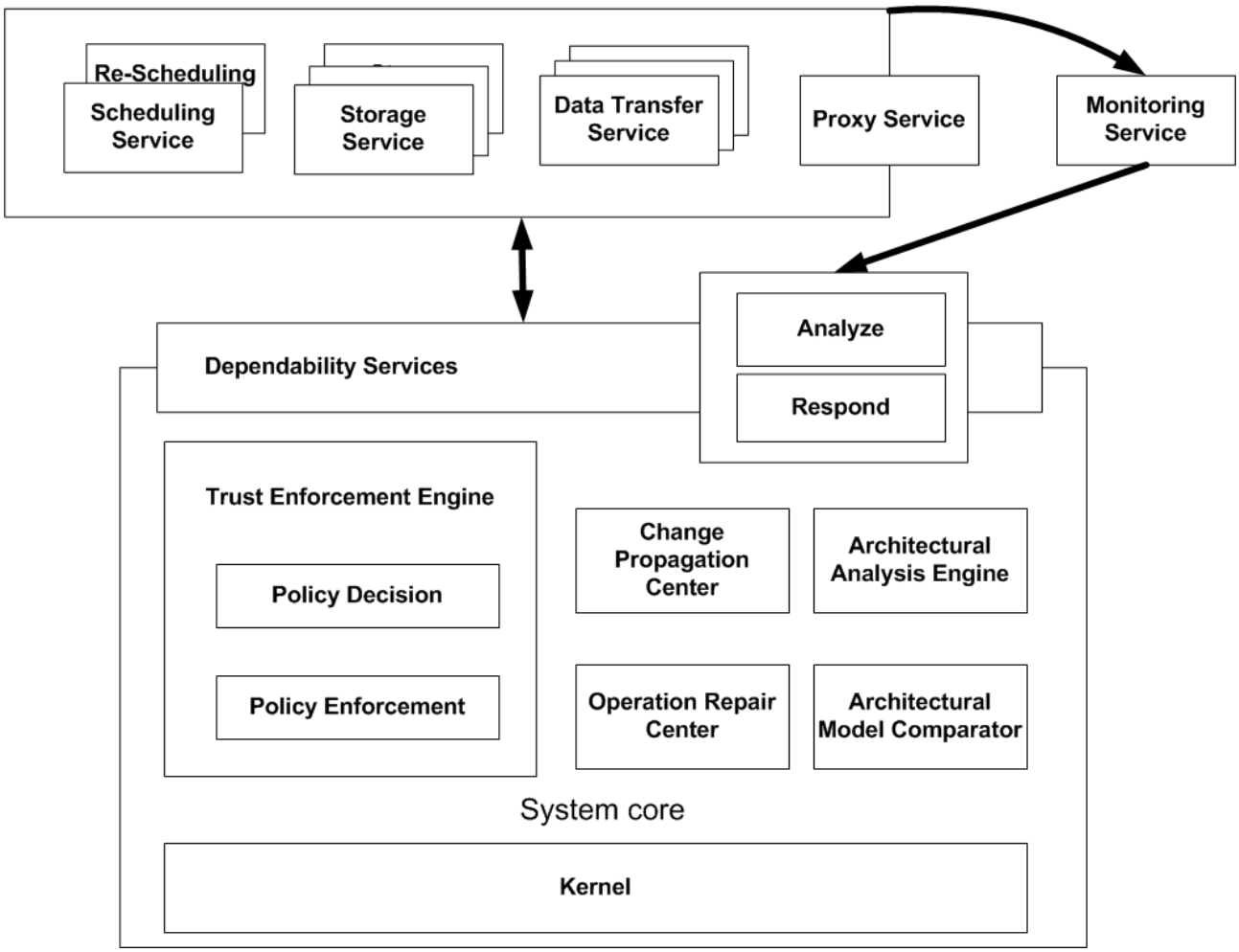


Fig. 3.1: The dependability architecture for LSDS.

An abstract model of the components making up the architecture at the middleware layer of the distributed system is presented in Figure 3.1. These components are designed to ensure fault tolerance between different hosts composing the system. At the bottom of this architecture is the core of the system, designed to orchestrate

the functionalities provided by the other components. Its role is to integrate and provide a fault tolerant execution environment for several other components.

The architecture also includes mechanisms for ensuring resilience based on replicating components of the system, such as the ones responsible for communication, storage and computation. It also considers combining the replication mechanisms with solutions to ensure survivability of the system in the presence of major faults. The solution to develop an architecture in which the system survive by adapting in the presence of fault arise naturally by explicitly acknowledge the impossibility to include a complete solution to ensure reliability considering the resulting resources and technologies. Because of this we adopted a strategy based on using replication only for the most basic core functionality of the system. We use replication in the form of fault-tolerant containers; the fault-tolerant containers can easily manage a set of replicated services, an approach presented in the next Sections.

4. An accrual failure detection service. Failure detection is an essential service of any fault tolerant distributed application that needs to react in the presence of failures. The detection of a failed process is difficult because of the unpredictability of the end-to-end communication delays. To solve the agreement problem of distinguishing between a failed process and a very slow one, various authors proposed the use of local failure detectors ([16], [17]). These are modules attached to the monitored processes and which gather data about their current status.

A failure detector (FD) combines two important functions: monitoring and interpretation. This distinction is most clear in case of accrual protocols. The family of accrual failure detectors consists in error detection processes that associate, to each of the monitored processes, a suspicion value. Previous proposed failure detectors are poorly adapted to very conservative failure detection because of their vulnerability to message loss. In practice message losses tend to be strongly correlated (i.e., losses tend to occur in bursts).

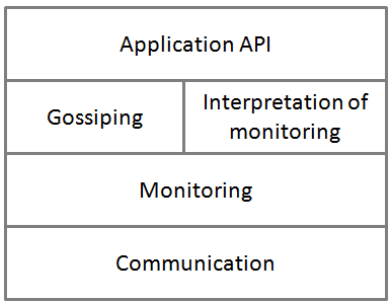


Fig. 4.1: The layers of the failure detection system.

We propose the use of an FD service that improves previously proposed solutions with several capabilities: (1) a scalable communication infrastructure used between remote failure detection processes, (2) the use of a proposed estimation function based on sampling previous responses and a formula for the computation of the suspicion level of a process being failed which more accurately reflect the dynamic nature of LSDS, (3) the addition of gossip-like protocols for failure detection which leads to the elimination of wrong suspicions because of the varying network traffic conditions, and (4) the capability of providing the failure detection function as a service to various distributed applications.

The proposed failure detector is based on the results previously presented in [18]. Several important results were further demonstrated in [19]. The service provides the detection capabilities of a distributed system composed of several monitoring and detection processes. The failure detection processes run inside the distributed environment, each being responsible for the monitoring of a subset of other processes or applications. The system is composed of four layers (see Figure 4.1).

The *Communication* layer handles a scalable, fault tolerant and dynamic communication infrastructure for the upper-layers. The failure detection processes are grouped in clusters. A cluster contains detectors that are geographically close, and also experience small communication delays. The failure detectors monitor only other processes inside the same cluster. The communication involves both heartbeat messages and gossip updates. The clustering minimizes the time needed to send messages between failure detection processes.

Each cluster is under the management of a cluster coordinator process. This process is responsible for the management of the local failure detection processes, as well as for intra-cluster and inter-cluster communication management. The coordinator handles the cluster management for example when processes enter or exit the system. Inside each cluster there are several failure detection processes, capable of exchanging messages between each others to identify possible failures. At cluster level the coordinator acts as a communication gateway. The clusters are interconnected through these gateways. The approach forms a hierarchical interconnecting communication infrastructure. Such an approach ensures scalability because the failure detectors communicate only with the processes inside the same cluster, and all intra-cluster communication is channeled through dedicated network links.

Within the second layer, *Monitoring* (see Figure 4.1), each detection process is responsible with the monitoring and logging of the data. Each FD process is responsible for monitoring several other FD processes from the same cluster. Periodically, each monitored process must issue a heartbeat message announcing it is alive. Periodically, every $T_{monitor}$ seconds, each FD process scans the list of monitored processes. It sends a heartbeat message to the each of the remote monitoring processes. Based on the receive heartbeat message, FD process updates the corresponding suspicion level.

It is difficult to determine the nature of a failure that affects a process. In an unstable network, with frequent message losses or high process failure rates, any detection algorithm is almost useless, because the processes cannot distinguish between permanent and transient failures. The third layer of the architecture attempts to solve this drawback by employing an approach based on *gossiping*. The role of gossiping is to reduce the number of false negative (wrong suspicions) and false positive (processes are considered to be running correctly even though they have failed) failure decisions. For this, each FD process periodically exchanges local failure detection information with other FD processes within the same cluster.

At this layer, we propose using a component responsible with the interpretation of monitored data (see Figure 4.1). The component analyses and further processes the monitored data. The data processing involves the use of a function for the estimation of the next heartbeat message (adaptive threshold), as well as a function for the computation of the probability that a remote process experienced failure (see Figure 4.2).

We call the probability value a *suspicion level*, as it represents the suspicion associated with the possible failure of a remote process. The suspicion level represents the degree of confidence in the failure of a certain process. While in case of accrual detection the suspicion level increased on a continuous scale without an upper bound, in this case the suspicion level takes values in the $[0, 1]$ interval so that a zero value corresponds to an operational process, and the probability of failure increases while the value approaches 1. Each failure detector maintains a local suspicion level value $\mathrm{sl}_{qp}(\mathrm{t})$ for every monitored remote process.

The computation of the suspicion level is based on the sampling of past heartbeat arrival times. The arrival times of heartbeat messages are continuously sampled and used to estimate the time when the next heartbeat is expected to arrive. The estimation function uses a modified version of the exponential moving average (EMA) function called KAMA (Kaufman's adaptive moving average) ([19]). This function ensures a more accurate prediction of the arrival time for the next heartbeat message using a trend of recent timestamps. The predicted value is further used to compute the suspicion level of the failure in case of each remote process. The suspicion value increases from 0 to 1. In case of a heartbeat H in the beginning, while H is not yet expected, the current suspicion level is 0. As time passes and H does not arrive, the suspicion increases to 1, this value being associated with the certainty in that H is lost. The suspicion value $\mathrm{sl}_{qp}(\mathrm{t})$ is computed as:

$$sl_{qp}(t) = \frac{t-1}{t+1}, \; where \; t = \frac{t_{now}}{t_{pred}} \tag{4.1}$$

The proposed function returns values in the $[0, 1]$ interval. It has a relatively quick evolution in the $[0, 0.8]$ interval and a slow one in the $[0.8, 1]$ interval. The function leads to a high probability of failure recognition in a reasonable amount of time, aspect previously demonstrated in [19].

The last layer of the FD system is represented by the interface with the applications. The failure detector is composed of several distributed processes. The failure detection capability is provided to application in the form of a *Web service*. This allows for standardized methods to access and communicate with the failure detectors, with advantages such as interoperability, flexible integration in various technologies, etc. The service provides operations such as the registration for specific failure detection events (a notification mechanism), and the interrogation for failure suspicion values.

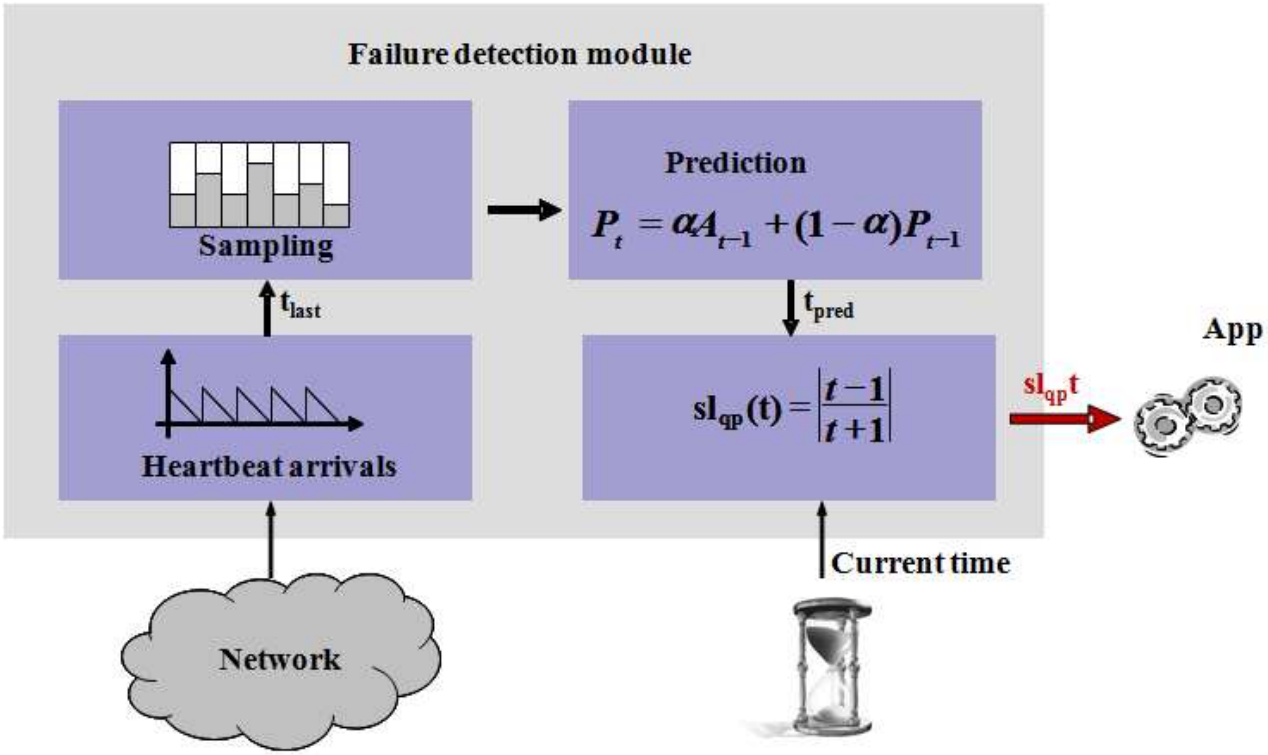


Fig. 4.2: Information flow within the failure detector.

5. A virtualization-based approach to dependability. The failure detector previously presented is complemented with a recovery solution that uses virtualization. The approach consists in the use of several type I virtual images (hosted directly on the computer hardware) running the same services [23]. These are native virtual images that are smaller than an entire operating system snapshot (unlike VMWare images, our implementation uses OpenVZ images which host only subsets of processes - an aspect crucial for communications costs involved by migration and similar operations). Such a virtual server hosts a small set of processes. These processes are generally the web containers hosting the services of the upper-layer LSDS middleware. Therefore, at this layer, we are interested in protecting the processes rather than the services themselves. A solution to protect the services themselves from LSDS-specific failures was also previously demonstrated in [14].

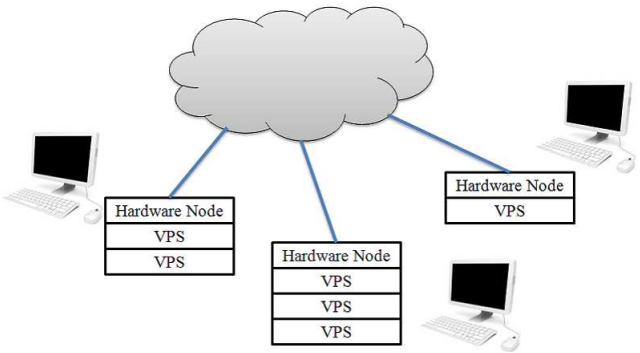


Fig. 5.1: The architecture of the system as composed by virtual environments.

At this layer the virtual environment is developed with support for both OpenVZ and LXC virtual en-

Fig. 5.2: The algorithm run by the coordinator.

```
procedure coordinator {
  send VOTE_REQUEST to all nodes
    log SEND VOTE REQUEST
  if timeout OR receive wrong answer {
    send GLOBAL_ABORT to all nodes
    call ESTABLISH_CONNECTION_STATUS
  }
  if all sent VOTE_COMMIT {
    send GLOBAL_COMMIT to all nodes
  }
}
```

Fig. 5.3: The algorithm run by the client.

```
procedure client {
  wait for VOTE_REQUEST
  receive VOTE_REQUEST
    log VOTE REQUEST
  send VOTE_COMMIT
    log VOTE COMMIT
  start timer
  wait GLOBAL_COMMIT or GLOBAL_ABORT
  if timer expired {
    /* Clientul lost GLOBAL_COMMIT or GLOBAL_ABORT */
    call ESTABLISH_NEIGHBOUR_STATE
  } else if received GLOBAL_COMMIT {
    initialize checkpointing
  } else if received GLOBAL_ABORT {
    continue
  }
}
```

vironments, and it is reinforced with security models assured by technologies such as SMACK or SELinux. The virtualization layer includes, therefore, several virtual servers (VPS) hosting services belonging to the distributed system (see Figure 5.1). OpenVZ ([7]) uses a type I hypervisor (running native on top of the physical machine). Multiple OpenVZ environments can share the same operating system's kernel. The overhead is lower than in alternative virtualization approaches such as Xen or VMware. LXC (Linux Containers) is a Linux native approach similar to OpenVZ. Except for creating the virtual environment in which services runs separately, fault tolerance is achieved using checkpointing.

There are two types of VPS nodes. A first type of node assumes the role of activity coordinator. So inside such nodes special processes (called coordinators) implement the detection algorithm (which monitors the processes running inside the virtual servers), as well as manage specific recovery actions. A coordinator is responsible with checkpointing, restoring and load balancing of virtual images. The coordinator can initiate a migration process of virtual images working with the hypervisor, and using the secured processes (such as ssh) on the remote workstation.

The other virtual nodes host the services belonging to the distributed system. They are under the control of the coordinator, which can detect and initiate repairing actions using the last known working snapshot of the virtual server. The virtual servers are also protected by various security policies.

For consistency, the implementation uses a modified coordinated checkpointing algorithm based on a Two-

Phase Commit protocol. The coordinator acts as mediator and initiator of the algorithm (see Listing 5.2). It starts by sending a VOTE_REQUEST to all nodes. If the coordinator receives from all clients a VOTE_COMMIT than a consistent global checkpointing is possible and, thus, it further sends a GLOBAL_COMMIT message. If any of the slave nodes is not responding, or it simply sends VOTE_COMMIT, then the coordinator responds with GLOBAL_ABORT. After sending a GLOBAL_ ABORT message, the coordinator tries to determine what happened to the nodes that did not respond. Depending on the answer, it can decide whether or to further keep the node in the list. The algorithm is presented in Listing 5.2. For fault tolerance, the coordinator itself is replicated (as a VPS node). When it fails, another coordinator can initiate a recovery action consisting of a retry to restore the failed node using a checkpoint image and synchronizing the differences.

When a slave node receives a VOTE_REQUEST message it responds with VOTE_ COMMIT and starts an internal timer. If it receives GLOBAL_COMMIT before the timer expires, the node simply starts its checkpointing action. If the timer expires and the node does not receive GLOBAL_COMMIT, then it sends to all other nodes a GLOBAL_ABORT. The algorithm is presented in Listing 5.3.

State restoring is similar to the distributed checkpointing approach. Again, a coordinator initiates the restore and if all nodes agree the system is restored to a consistent state.

For load balancing the coordinator monitors the load of the clients. When the load exceeds a predefined threshold the coordinator migrates VPS nodes on other station. The migration process is done without interrupting the connection with the node. The same principle is applied for fault tolerance. When the coordinator detects failures inside a VPS in uses the last saved checkpoint to initiate a recovery procedure and spawn new VPS nodes. These new nodes take over the faulty ones, and are initiated for failure tolerance on different adjacent nodes.

In our experiments we managed to automatically correct full-stop failures (i.e., complete shutdown of a node inside LSDS, or the lost of network connectivity) (see [14]). This is possible because the coordinator can recover a VPS, possibly on a completely different host inside LSDS, starting from a snapshot saved on the hard drive. The approach can also lead to further research towards the automatic corrections of transient failures. The virtualization can lead to diversity, which in turn means we can run the same VPS possible under different hosts. Or gossiping algorithms can be used to mediate between possible outcomes of operations.

The costs involved with the use of virtualization for fault tolerance was considered a problem by previous authors ([20]). For each service a XEN- or VMWare-based solution requires several virtual images (each one containing its own operating system, memory occupied by the processes, stacks, file systems, etc.) cloned inside a Cloud. This increases the expenses at the benefit of fault tolerance. Our solution does not rely on the cloning of an entire virtual machine for fault tolerance. In our case the virtual image can host several VPSs, so that if one fails another one can take its place. For full fault tolerance (to protect also the operating system for example) one needs to deploy two virtual machines inside the Cloud. But they are used for entire collections of services, each possibly running inside its own VPS. Thus, our solution decreases the expenses while still providing extensive fault tolerance.

6. The security layer. Many times reliability (and, hence, dependability) of a LSDS depends not only on fault tolerance, but also security. If a control network is compromised due to poor security policies (lack of patches, slow patch cycle, etc), the reliability of the network is decreased. If an attacker can perform attacks (e.g., man-in-the-middle) and send commands that disrupt the functionality of the LSDS, its reliability is decreased. Nearly all security threats can be seen as threats to the systems reliability.

For modern LSDS the security features provided by the operating system are simply not enough. Various authors argue that access control mechanisms for safe execution of untrustworthy services should be based on security models such as Discretionary Access Control (DAC) [8] or Mandatory Access Control (MAC) [9]. Such models are at the basis of our security layer, which is designed to augment the virtualization infrastructure previously presented with security features. For fault tolerance each VPS contains replicas of the same processes. We next assumed that each process represents a service container. We augmented these containers with access control and policy enforcement mechanisms (see Figure 6.1).

In the DAC model the access to information is determined by the identity of subjects or groups [8]. In addition, the model assumes that subjects can pass permission to other subjects. The model suffers from various limitations. Users authorized to access some information may not be the owners of that information. This leads to situations where a compromised application can control resources far beyond the needs of that application. In information security the principle of least privilege requires that in a particular abstraction layer of a computing

environment, every module must be able to access only the information and resources that are necessary for its legitimate purpose. The DAC model lacks enforcements of the least privilege principle. It also lacks domain separations for users logged into the system.

The MAC model was considered more adequate for our purpose. In this model the access is restricted based on the sensitivity (as represented by a label) of the information contained in the objects and the formal authorization of subjects [9]. This model is more adequate to be used for securing distributed services: it allows the domain separations of users and to enforce the least privilege principle. There are currently several research-level implementations of this model at OS level: SELinux, Smack or AppArmor are among the most advanced solutions for Linux-based systems [10].

Security Enhanced Linux (SELinux) implements for Linux a security model that combines Type Enforcement (TE) model, with Role-Based Access Control (RBAC) and Multi-Level Security (MLS) models. The Type Enforcement model provides fine-grained control over processes and objects in the system while the RBAC model provides a higher level of abstraction to simplify user management as stated in [11]. Similar to SELinux, Smack also implements the MAC security model [12]. It was intended to be a simple mandatory access control mechanism but it purposely leaves out the role based access control and type enforcement that are the major parts of SELinux. Smack is geared towards solving smaller security problems than SELinux, requiring much less configuration and very little application support. Smack permits creation of labels according to the security requirements of the system.

These solutions implement the MAC model. The security mechanisms in SELinux and Smack are based on inodes, while in AppArmor are based on file paths. If a program is restricted from accessing a particular le using AppArmor, a hardlink to that file would still provide with access, since the protection states only the original path of the file. From these three, SELinux has the most complex implementation, because it combines complex mandatory access controls such as those based on type enforcement (TE), roles and levels (RBAC) of security (MLS). Because SELinux provides more security mechanisms, because of its flexibility in design, we selected it for the case study of securing enforcement in case of several distributed services.

Security can be applied at various levels. Security mechanisms applied within the application layer have the advantage of high granularity, while protecting sensitive information, and permit construction of complex policies. The disadvantage of such mechanisms is the high overhead. The operating system (OS) is the one that mediates accesses initialized by applications to hardware components. Therefore, access control mechanisms applied at the OS layer can provide high level of granularity for protecting processes, files, sockets, etc. At this layer there are also various DAC mechanisms that are already used to protect services.

Our solution creates an orthogonal security policy which, together with the existing security mechanisms provided by the OS, can be used to enhance the security characteristic of a distributed system.

Services can be secured as other processes running inside a computer system. By reducing the services to a simple process we can better localize a security issue from the OS point of view. This is illustrated in Figure 6.1. The example considers three services. Each service runs in a separate service container (P1-3). The security policies and access rules are applied at the level of each container (in our example permission to access resources are specified as R1-3). Several such resources can be applied for each container, or several services can be protected by the same rule. In the example also we show the relation with the fault tolerance solution - the access control is performed at the level of each process, while the fault tolerance is applied at the level of sets of processes, running in virtual VPSs.

So each Pi process has permissions to access the resource Ri. The example illustrates the use of several security layers. The processes are first protected by the DAC mechanism provided at the OS mechanisms. If this layer is compromised, each service is further protected by an individual security policy. Furthermore, each service is enclosed inside a unique sandbox that does not include any other process. In this case, even if one service is compromised, the other services are still intact and further protected.

These security mechanisms were implemented using SELinux. Over the DAC security layer provided by many Linux flavors, we used the MAC security layer assured by SELinux. The SELinux solution already confines twelve daemons inside specific domains. Based on the provided security functions, we developed protection mechanisms for several services. The result is a system having an enhanced security characteristic because beside been protected from the damage made by the twelve daemons it further offers protection guarantees against damage made by the web container and the monitoring service. These services are confined in specific sandboxes according to their activities.

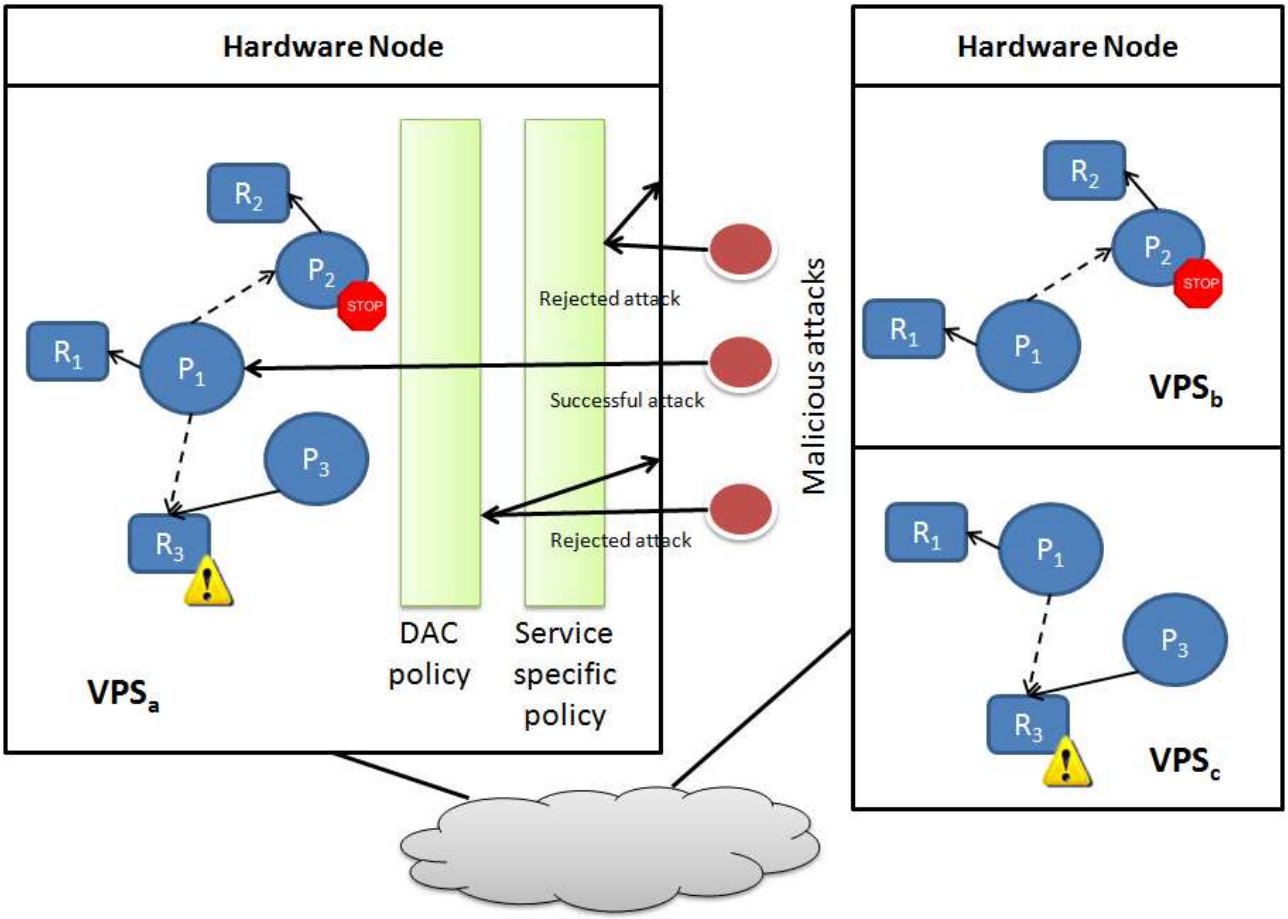


Fig. 6.1: Malicious attacks against services.

7. Evaluation results. In the sequel we present a series of results demonstrating the capabilities and performances of the proposed solutions. We conducted our tests on the NCIT testbed [22], an experimental Grid platform with advanced scheduling and control capabilities. The testbed is part of the national Grid collaboration and it includes several sites geographically distributed in various places in Romania. For the experimental setup, we considered the use of 10 nodes belonging to the NCIT Cluster at University Politehnica of Bucharest. The nodes are equipped with x86 64 CPUs running at 3 GHz, 2 GB RAM, interconnected via a 10 Gigabit Ethernet network. An additional set of 5 nodes situated at CERN, Geneva, were used to evaluate the overhead of the solution.

We evaluated the dependability mechanisms by combining virtualization with the use of the security mechanisms. The scenario involved an ApMon component, which is already used in real-world distributed infrastructures at the monitoring layer [13], and then the Proxy service previously used in evaluations, and that is designed to enhance the fault tolerance capability of distributed systems [14].

Monitoring services are encountered in many distributed systems, and, especially for fault-tolerance, one needs to have guarantees that the monitoring information is unaltered by malicious attackers. On the other hand, the Proxy service illustrates the mechanisms applied to protect a service container.

We first closed each service inside a SELinux sandbox, running on a virtual VPS that is its own domain. This domain confines inside any possible damage. For example, the ApMon is running within the *apmont* domain. To allow ApMon the access to monitor in this domain we specified how a process is allowed to run in the *apmont* domain, and what it is allowed to do. The entry point of the *apmont* domain is any executable file labeled with the type $apmon_exect$. Once a process executes this file, it transitions in the *apmont* domain and runs only under the allow rules of the domain. This example is illustrated in Figure 7.1. In the example

the executable file is */bin/apmon* and the allowed actions are reading the configuration file, accessing */proc* contents and sending monitoring information to a MonLISA service.

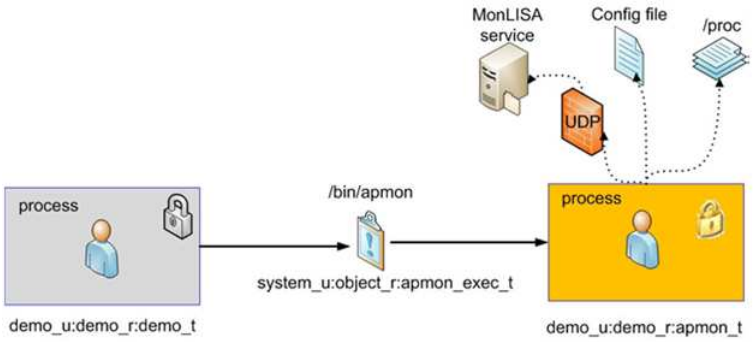


Fig. 7.1: Security mechanisms applied to an ApMon service.

A different situation is illustrated by the security mechanisms applied to the Proxy service. In this case we extended the enhancing security mechanisms to service containers, themselves running on different VPS nodes. What a service is allowed to do and what not sometimes differ greatly when treating security for services unitarily and independent from the container. A Proxy service may need access to some type of files, while a monitoring service wants for example access to other types. They can both run as applications requiring searching and loading dynamic libraries, memory execute permissions and network access. But one service might require some type of restrictions, while another might require completely different security limitations. In this case, we first developed the SELinux policies for specifying what the process represented by the service container (the current implementation is based on Tomcat as a web container for services) is allowed to do.

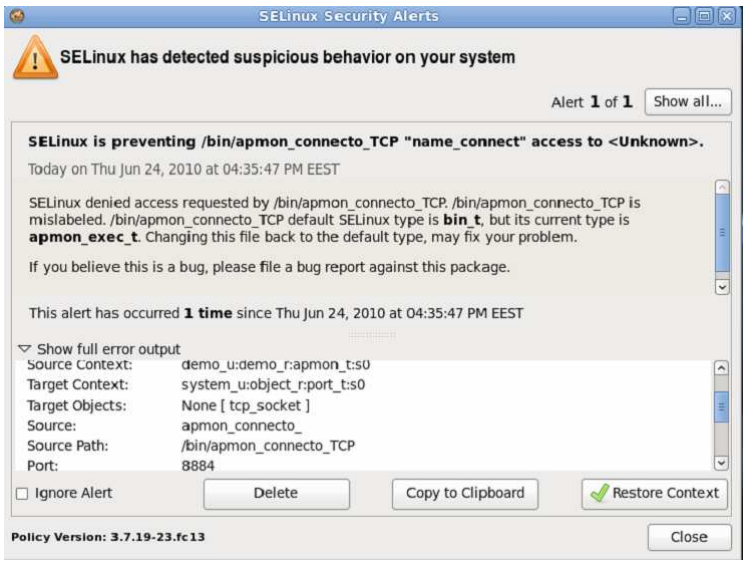


Fig. 7.2: The malicious ApMon application is denied connect access to port 8884.

We defined the domains and types of the policy that help to sandbox the restraint Tomcat container. For example, the Tomcat process only runs in the local system as daemon, started from the *initrct* domain, the SELinux domain for the *init* processes. This was further augmented with policies for the individual services running inside the container. The policies are enforced using two domain transitions before reaching the domain of a usual Java application. This idea was first introduced in [15].

We evaluated several situations in which failures and attacks from malicious code can be contained and avoid the damage of the whole system though validating imposed policies. We used experiments using from strict policies, where nothing is allowed to run and one has to define specific allow rules for each actions, to more relaxed ones, such as every subject and object can run uncontrolled except specific targeted daemons that are constrained by proposed rules.

A first experiment uses a verification that an ApMon application still works. In this scenario the ApMon application sends information about the system inspecting the */proc* directory. The monitored information is sent using datagrams to a MonaLISA farm. Next we tested what happens when ApMon is compromised and it tries to connect on another port. As expected, the system detects an attempt to break the imposed execution policy and interrupts the malicious activity (see Figure 7.2).

In another experiment we evaluated a Tomcat web service container augmented with the proposed dependability solutions. The results of these show experiments that the solution is able to preserve the defined policy. For example, in an experiment we were interested if the Tomcat web services container is able to contain the damage, in case of a situation where the monitoring service itself inside Tomcat suffers an attack and, as a consequence, it behaves maliciously and tries to send valuable information about requests received from the Proxy service somewhere else than it is supposed to. The reaction of the security policy is as expected thus the attempt of connecting on an incorrect port fails because the action was not specified as an allowed action (see Figure 7.3).

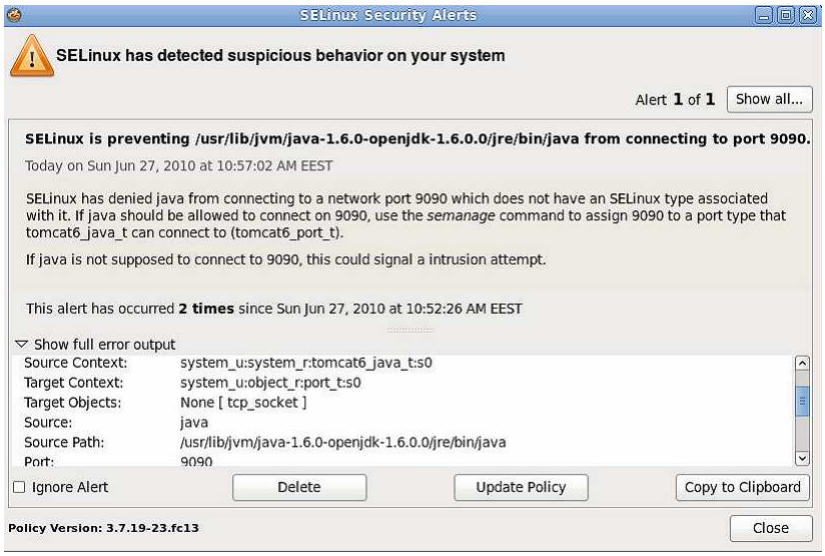


Fig. 7.3: Test service is denied to connect on port 9090.

We next continued with a series of experiments using LISA [24], a lightweight dynamic service that provides complete system and application monitoring. In this case the objective was to evaluate the overhead of the proposed solutions in terms of network traffic and the load of the systems on which services are running. The testbed involved three stations, two located in CERN, Switzerland and one in Romania. Each station hosts a VPS. We first assumed two services running on each of the nodes in Switzerland, and on the node in Romania we started the Jini service. At one point, a process fails in order to see how the system reacts to its failure.

During these experiments we measured a sustained additional traffic varying between 5 and 20 Kbps, while the machines load did not change significantly (see Figure 7.4). The measured values demonstrate that the overhead caused by running the processes is very low, both in terms of network traffic and CPU usage or system load.

These experiments reveal a good potential for integrating the services into various distributed environment for increasing dependability. The failure detection service is able to recognize various errors and, in the same time, assures good running performances. The detection is further augmented with the VPS solution that is able to provide transparent failure recovery, by resubmitting faulty requests to still-running service replicas. In

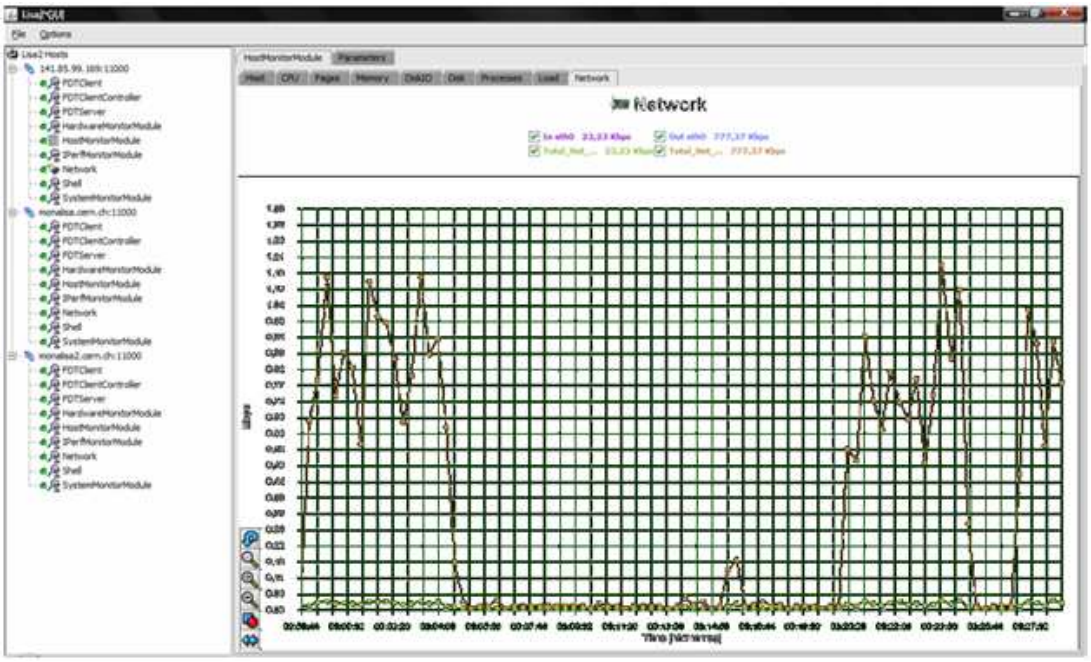


Fig. 7.4: The results for the evolution of the network traffic.

the end, security attacks are correctly recognized by the MAC mechanisms. Combined, these services provide increase reliability and availability properties.

8. Conclusions and future work. In this paper we presented an approach to ensuring dependability in LSDS using virtualization for fault-tolerance, augmented with advanced security models. Today dependability remains a key element in the context of application development and is by far one of the most important issues still not solved by recent research efforts.

Our research work is concerned with increasing reliability, availability, safety and security in LSDS. The characteristics of such systems pose problems to ensuring dependability, especially because of the heterogeneity and geographical distribution of resources and users, volatility of resources that are available only for limited amounts of time, and constraints imposed by the applications and resource owners. Therefore, we proposed the design of a hierarchical architectural model that allows a unitary and aggregate approach to dependability requirements while preserving scalability of LSDS.

We presented implementation details of such proposed methods and techniques to enable dependability in LSDS. Such solutions are based on the use of tools for virtualization and security, in order to provide increased levels of dependability. We proposed several solutions to increasing fault tolerance and enforcing security. The fault tolerance is based on the use of virtual containers, either in the form of virtual sandboxes running on top of the operating systems, but we are currently also working on proposing logic containers composed of various replicated services served by an intermediary Proxy service. We also presented solutions to introduce modern security models, such as MAC, to various distributed services. The security policies in this case are applied at various levels, by offering protection at operating system level, at service containers or further to the individual service.

Acknowledgments. The research presented in this paper is supported by national project "DEPSYS - Models and Techniques for ensuring reliability, safety, availability and security of Large Scale Distributes Systems", Project CNCSIS-IDEI ID: 1710. The work has been co-funded by national project "TRANSYS - Models and Techniques for Traffic Optimizing in Urban Environments", Contract No. 4/28.07.2010, Project CNCSIS-PN-II-RU-PD ID: 238, and by the Sectoral Operational Programme Human Resources Development 2007-2013 of the Romanian Ministry of Labour, Family and Social Protection through the Financial Agreement POSDRU/89/1.5/S/62557. The contributions from all authors to this paper are equal.

REFERENCES

[1] V. Cristea, C. Dobre, F. Pop, C. Stratan, A. Costan, and C. Leordeanu, *Models and Techniques for Ensuring Reliability, Safety, Availability and Security of Large Scale Distributed Systems*, in Proc. of the 3rd International Workshop on High Performance Grid Middleware, the 17th International Conference on Control Systems and Computer Science, Bucharest, Romania, May 2009, pp. 401–406.

[2] H. Jin, X. Shi, W. Qinag, and D. Zou, *DRIC: Dependable Grid Computing Framework*, IEICE - Transactions on Information and Systems. Volume E89-D, Issue 2, February 2006, pp. 126–137.

[3] P. Guillaume, *Design of an Infrastructure for Highly Available and Scalable Grid Services*, D3.2.1. Technical Report, Vrije Universiteit, Amsterdam, 2006.

[4] J. Jayabharathy, and A. Ayeshaa Parveen, *A Fault Tolerant Load Balancing Model for Grid Environment*, Pondicherry Engineering College, Pondicherry, India, International Journal of Recent Trends in Engineering, Vol 2, No. 2, November 2009.

[5] B. Yagoubi, and M. Medebber, *A Load balancing Model for Grid Environment*, in Proc. of the 22nd International Symposium on Computer and Information Sciences (ISCIS 2007), Ankara, Turkey, 2007, pp. 1–7.

[6] I. Sommerville, S. Hall, and G. Dobson, *Dependable Service Engineering: A Fault-tolerance based Approach*, Technical Report, Lancaster Univ., 2005.

[7] J.P. Walters, and V. Chaudhary, *A fault-tolerant strategy for virtualized HPC clusters*, J. Supercomput., 50(3), Dec. 2009, pp. 209–239.

[8] S. Dranger, R.H. Sloan, and J.A. Solworth, *The Complexity of Discretionary Access Control*, in Proc. of the International Workshop on Security (IWSEC 2006), Kyoto, Japan, October 2006, pp. 405–420.

[9] H. Lindqvist, *Mandatory Access Control*, Master's Thesis in Computing Science, Umea University, Department of Computing Science, SE-901 87, Umea, Sweden, 2006.

[10] C. Wright, C. Cowan, S. Smalley, J. Morris, and G. Kroah-Hartman, *Linux Security Modules: General Security Support for the Linux Kernel*, USENIX Security, Berkeley, CA, 2002, pp. 17–31.

[11] S. Smalley, *Configuring the SELinux policy*, NAI Labs Report #02-007, 2002.

[12] C. Shaufler, *The Simplified Mandatory Access Control Kernel*, Whitepaper, 2008.

[13] I. Legrand, H. Newman, R. Voicu, C. Cirstoiu, C. Grigoras, C. Dobre, A. Muraru, A. Costan, M. Dediu, and C. Stratan, *MonALISA: An agent based, dynamic service system to monitor, control and optimize distributed systems*, Computer Physics Communications, Volume 180, Issue 12, December 2009, pp. 2472–2498.

[14] M. Nastase, C. Dobre, F. Pop, and V. Cristea, *Fault Tolerance using a Front-End Service for Large Scale Distributed Systems*, in Proc. of 11th International Symposium on Symbolic and Numeric Algorithms for Scientific Computing, Timisoara, Romania, 2009, pp. 229–236.

[15] C. Hger, *Security Enhanced Linux - Implementierung und Einsatz*, Technical Report, Technical University Berlin, Complex and Distributed Systems, 2008.

[16] C. Fetzer, M. Raynal, and F. Tronel, *An adaptive failure detection protocol*, in Proc. of the 8th IEEE Pacific Rim Symp. on Dependable Computing, Lanzhou, Gansu, 2001, pp. 146–153.

[17] X. Defago, N. Hayashibara, and T. Katayama, *On the Design of a Failure Detection Service for Large-Scale Distributed Systems*, in Proc. of the Intl. Symp. Towards Peta-bit Ultra Networks (PBit 2003), Ishikawa, Japan, 2003, pp. 88–95.

[18] C. Dobre, F. Pop, A. Costan, M.I. Andreica, and V. Cristea, *Robust Failure Detection Architecture for Large Scale Distributed Systems*, in Proc. of the 17th International Conference on Control Systems and Computer Science (CSCS 17), Bucharest, Romania, 2009, pp. 133–141.

[19] L. Andrei, C. Dobre, F. Pop, and V. Cristea, *A Failure Detection System for Large Scale Distributed Systems*, in Proc. of 2010 International Conference on Complex, Intelligent and Software Intensive Systems (CISIS 2010), Krakow, Poland, 2010, pp. 482–489.

[20] L.M. Silva, J. Alonso, and J. Torres. *Using Virtualization to Improve Software Rejuvenation*, IEEE Trans. Comput. 58, 11 (November 2009), pp. 1525–1538.

[21] B. Jansen, H.V. Ramasamy, M. Schunter, and A. Tanner. *Architecting Dependable and Secure Systems Using Virtualization*. In Architecting Dependable Systems. Lecture Notes In Computer Science, Vol. 5135. Springer-Verlag, Berlin, Heidelberg (2008), pp. 124–149.

[22] The NCIT Cluster, *http://cluster.grid.pub.ro/*, Retrieved September 14, 2011.

[23] J.R. Douceur, and J. Howell. *Replicated Virtual Machines*. Technical Report MSR TR-2005-119, Microsoft Research, Sep 2005.

[24] I.C. Legrand, C. Dobre, R. Voicu, C. Cirstoiu. *LISA: Local Host Information Service Agent*. Proc. of the the 15th International Conference on Control Systems and Computer Science (CSCS-15), Bucharest, Romania, (2005).

Edited by: Dana Petcu and Jose Luis Vazquez-Poletti
Received: August 1, 2011
Accepted: August 31, 2011

Scalable Computing: Practice and Experience
Volume 12, Number 3, pp. 351–362. http://www.scpe.org
ISSN 1895-1767

AN ADAPTIVE AND SCALABLE REPLICATION PROTOCOL ON POWER SMART GRIDS*

JOAN NAVARRO† JOSÉ ENRIQUE ARMENDÁRIZ-IÑIGO‡ AND AUGUST CLIMENT §

Abstract. Cloud based storage systems are known to provide high scalability and reliability overcoming the traditional constraints of static distributed systems. The processing capacity over thousands of machines makes this approach especially suitable for many environments. In particular, we focus on power networks. These systems are currently decentralizing their architectures due to the growth of renewable sources and the increasing power demand which are obstacles to the traditional power network radial distribution. This new decentralized architecture, which demands computing abilities for network monitoring and improving customer services, is denoted as power smart grid. This paper proposes a new scalable dynamic storage architecture and its associated replication protocol, with its correctness proof, aimed to store data with different consistency levels. In addition it is able to perform some parallel data computations adapting itself to the physical and dynamic power smart grid layout.

Key words: Dynamic systems, cloud computing, eventual consistency, smart grids, replication.

1. Introduction. Power networks are demanded to be highly reliable and available because they have to supply all the infrastructures of a country at anytime and anywhere. This prevents power companies from updating and improving their systems because most of the changes may seriously affect critical services they are currently providing since novel devices might not be as tested as older ones. This leads to inefficient—due to their centralized nature—schemes which are expensive and even harder to maintain and scale.

With the growth of renewable energies the power network centralized model is not only able to scale but also cannot work properly; the aforementioned renewable energy sources behave different than traditional sources. Moreover, current power networks are not able to remotely monitor power consumptions on the low voltage (LV) network which prevents companies from building new business strategies fitted to the end user needs [8]. This situation urges a substantial change which consists of decentralizing the power network and building a distributed system able to fulfill the current society requirements and technologies.

Recently, this new paradigm has also been referred to as power smart grid (intelligent grid). The goal of a smart grid is to take advantage of the current digital technologies and build up an intelligent information system over all devices within the power network: from suppliers to consumers. This might allow companies to efficiently tune the power distribution and route energy where and when it is needed.

Smart grids have been a hot topic during the last few years and several approaches have been proposed: the gridSmart project [1] proposes an upgrade of the Ohio electric grid by using digital communications and automated functioning. This permits customers showing how smart grid technologies provide customers with greater energy control. It can also improve electricity delivery and cut energy consumption to delay the need to build more power plants. The Masdar Eco city [23] project proposes to build a energetically sustainable city in Abu Dhabi. IBM and Malta's government are powering a project [24] which consists in transforming the distribution network to improve operational efficiency and customer service levels by changing the current electricity meters to smart devices and connecting them to an information system enabling remote reading, management, and monitoring throughout the entire distribution network.

The decentralization of a power network, actually the smart grid design, covers several disciplines such as (1) electricity, there are multiple power sources using different technologies; (2) networking, there must exist a secure communication between all the nodes which generate data on the system; (3) computer engineering, in the sense that this data must be stored and computed.

The purpose of this paper is to focus on the computer engineering field and propose an architecture and its storage protocol, able to efficiently store and ease the computation of any data generated by the power network inspired by the flavors of cloud computing. This distributed storage architecture is slightly different than the ones used on web services [27] or in pure cloud computing based storage architectures [34, 25] since smart grids

*The research leading to these results has received funding from the European Union European Atomic Energy Community Seventh Framework Programme (FP7/2007-2013 FP7/2007-2011) under grant agreement nr. 247938 for Joan Navarro and August Climent and by the Spanish National Science Foundation (MEC) (grant TIN2009-14460-C03-02) for José Enrique Armendáriz-Iñigo.

†Distributed Systems Research Group, La Salle - Ramon Llull University, 08022 Barcelona, Spain (jnavarro@salle.url.edu).

‡Dpto. Ing. Matemática e Informática, Universidad Pública de Navarra, 31006 Pamplona, Spain (enrique.armendariz@unavarra.es).

§Distributed Systems Research Group, La Salle - Ramon Llull University, 08022 Barcelona, Spain (augc@salle.url.edu).

demand a set of requirements that have not been explored yet. Hence, we also provide a rough correctness verification of the distributed storage protocol.

The reminder of this paper is organized as follows: Section 2 describes the requirements that the distributed storage system must fulfill within the power smart grid framework. Section 3 describes the proposed architecture and explains how it has to be included within the smart grid. Section 4 reviews the system correctness in absence of failures. Section 5 discusses our proposal and suggests other domains of application. Finally, Section 6 concludes the paper.

2. Smart Grids Storage Requirements. Smart grids, as opposite to classical power networks, have become data driven applications since they own a management layer which takes decisions based on the current status of the network. This issue forces designers to redefine the whole power network architecture and its specifications, as now there is a need for storing and processing these datum besides supplying power. This section reviews such requirements and states the basics of the architecture proposed in Section 3.

The strongest requirements of any device inside a smart grid are availability and reliability since denials of service are not acceptable at any situation. Moreover, smart grids are demanded to perform many computations from data collected by smart meters and intelligent electronic devices (e.g. circuit breakers, voltage regulators, etc.). This, extends both requirements—availability and reliability—not only to the physical infrastructure, targeted at supplying energy, but also to datum and their storage.

To fulfill these constraints, we propose a distributed storage architecture built on top of the power smart network able to afford the dynamic behavior of smart grids (e.g. a solar panel may stop supplying energy or an end-user consumer may switch from its local power generation device to the supplier generation network). Actually, distributed storage systems are known to provide availability, reliability, and fault tolerance on many scenarios [15, 17].

Distributed storage systems can be either static or dynamic. Static systems [17, 26] require to know the identity of all nodes a priori in order to be able to distribute storage and computation. On the contrary, dynamic systems [2, 34, 22, 25, 13] do not make any assumption about the system composition, which allow processes joining and leaving at will. This kind of systems are designed to be fault tolerant, understood as the ability to tolerate erratic behaviors from random nodes, and used to improve the scalability of static systems by relaxing the data consistency restrictions [7].

Smart grids demand a trade-off between both scenarios because they behave as a dynamic system but they may need some strong consistency [33] requirements that typical cloud based techniques are currently unable to offer. Hence, our proposal is to build a hybrid system which takes advantage of both distributed storage system schemes, static and dynamic.

Far from just storing data, there are also several applications (also referred to as smart functions) that must run over the smart grid, such as power flow monitoring, under/over voltage monitoring, load shedding, or fault analysis. Each application has its own particular requirements so the proposed architecture must be flexible enough to support such variety of functions. Thus, the distributed storage architecture must provide the following:

1. Reliability. The proposed architecture must be fully tested since major changes on it may imply eventual denial of services.
2. Availability. The architecture must ensure that there always be available data despite its level of consistency.
3. Fault tolerance and recovery. The system must be able to reconfigure its internal characteristics in order to keep supplying and storing data in case of failure.
4. Dynamic consistency. Smart grids run several functions that might require different levels of consistency. For example, on one hand, data needed to perform a load shedding requires strong consistency [33] since it performs critical operations with the current values of the network. On the other hand, data needed to perform power monitoring might require a weaker level of consistency since this function tolerates some kind of delay.
5. Minimum message exchange. It is important to keep a low network overhead in order to guarantee that there were no bottlenecks, and data will flow over the network in an efficient way.
6. Simplicity. This is the key to build a system easy to maintain and adaptable to other domains of application.

The next section proposes the distributed storage architecture and places it inside the smart grid.

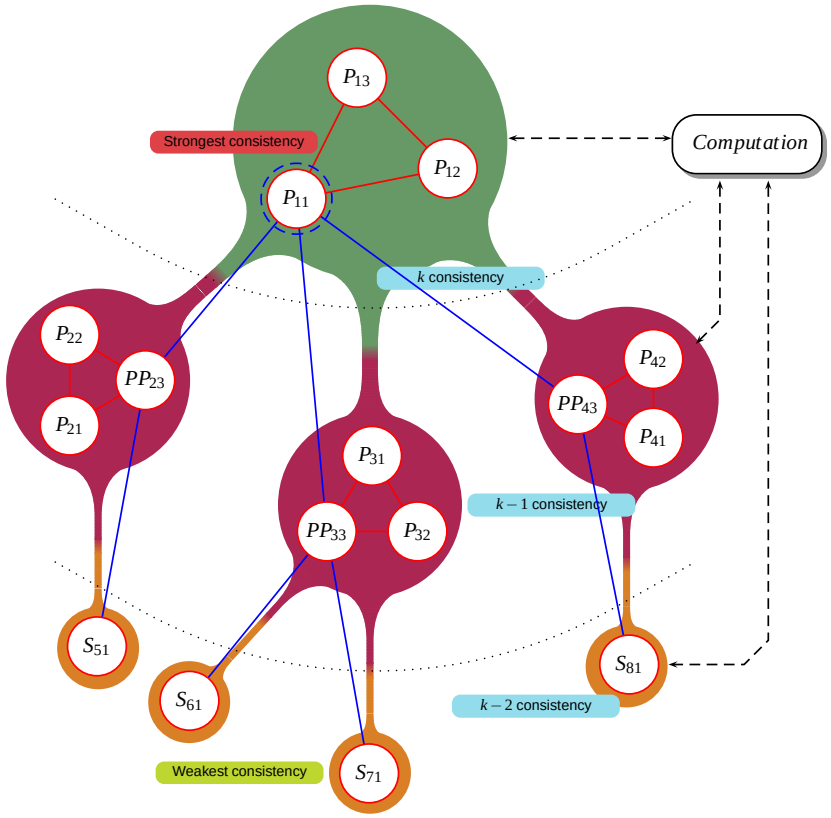


Fig. 3.1: Proposed distributed storage system.

3. System Architecture. Our proposal is inspired on the Primary Copy replication technique [35] used in transactional environments, where all updates are handled by a single node (also referred to as primary) and it propagates them to their replicas (also referred to as subscribers). However, our approach is considerably different from the classical primary copy protocol in terms of (1) primary updates, (2) subscriber updates, and (3) scalability.

Our architecture supports several primary nodes; actually each node can be considered as a primary which means that now updates' load is balanced between several devices. Each primary node treats the rest of devices in the smart grid as subscribers. In addition, an epidemic updates based protocol is proposed in order to replicate data across subscribers. As shown below, in order to overcome the classical scalability constraints of transactional systems our architecture breaks up with the relational scheme and deals with datum as key-value pairs.

This section details thoroughly these three key points and fits the replication protocol on top of the power smart grid infrastructure. Hence, it does not depend on any low layer hardware nor software specification.

As depicted in Fig. 3.1 a smart grid is seen as a set of intelligent electronic device clusters linked by a telecommunications network (i.e. power line communication, wireless network, wide area networks, etc.). A cluster is composed of up to ten devices (drawn as rounded circles in Fig. 3.1) placed on the same geographical area. Each device has limited storage and computing capabilities since it might not be able to solve the whole required smart functions on its own. We also consider that smart meters are attached to these devices and rely on them to report their measurements to the rest of the power smart grid through the computation unit.

Each device in the cluster is labeled as X_{ij} where X corresponds to the device role in the cluster (**P**rimary, **P**seudo-**P**rimary or **S**econdary); i is the cluster identifier, and j is the device identifier. In the same way, we find very useful to define the ancestor of a cluster$_m$ as the node X_{ij} (that obviously belongs to cluster$_i$ $(m \neq i)$) which is updating an arbitrary pseudo-primary k of this cluster (PP_{mk}). Fig. 3.1 shows an example where we have that region 2 is formed by devices $P_{2k} \mid k = [1, 3]$ where P_{21} is the primary of this region; P_{22} is a common

device; P_{23} is the pseudo-primary, (that's why it is named $\mathbf{PP}_{23}$); and, its ancestor is P_{11}. Respectively, S_{61} is the only device on region 6 and its ancestor is PP_{33}.

Regarding data consistency, we define the replication depth r as the amount of different clusters that data are allowed to cross when they are being replicated. This value might be dynamically tuned according to the computation latency or the system performance.

Next, we describe the proposed architecture and explain how we solve the replication, consistency, and fault tolerance issues.

3.1. Architecture Overview. Although the number of smart sensors may substantially increase as time goes by, the number of devices that control them should not grow in the same way. The proposed architecture focuses on the devices instead of the smart meters which is an attempt to avoid scalability issues from the latter ones by hiding their dynamism. However, the system must be robust against possible node failures which forces designers to implement some techniques commonly used in dynamic systems [15, 13, 2].

In order to provide a high available system able to ease the smart functions distributed computation, we propose an architecture inspired on the Primary Copy [17, 35, 30] (also referred to as Primary Backup) scheme.

We distinguish two different types of smart clusters: (1) storage clusters which do not generate data (regions 5, 6, 7, and 8 in Fig. 3.1) and (2) active clusters that generate data (regions 1, 2, 3, and 4 in Fig. 3.1).

Any device belonging to an active cluster may simultaneously adopt different roles according to the current situation: (1) primary master, (2) primary slave, (3) pseudo-primary, and (4) secondary. When a device is propagating data from their directly attached smart meters, it will act as a primary master and will treat the rest of devices in its cluster as their primary slaves (in Fig. 3.1, P_{11}, marked with a dashed blue circle, is the primary master and P_{12}, P_{13} are their primary slaves). When a device receives data from another cluster it will be acting as a repeater (pseudo-primary) (in Fig. 3.1 PP_{23}, PP_{33}, and PP_{43} are the pseudo-primaries of P_{11}). Moreover, if a device receives updates from other clusters but it does not propagate them, it will be acting as a secondary (in Fig. 3.1, $S_{51}, S_{61}, S_{71}, S_{81}$ behave as secondary devices).

Note that blue lines in Fig. 3.1 just illustrate the particular case of P_{11} broadcasting data. In fact, the proposed architecture must be understood as if all nodes where generating data. Hence all nodes may act as primary, pseudo-primary, and secondary devices at a time.

Smart grids need to compute many smart functions [10] indeed. Our architecture is flexible enough and able to adapt itself to the data freshness requirements [30, 4] imposed by each smart function. This way of propagating updates along the replication tree structure allows the system to find the most appropriate version for computing a function while circumventing all traditional problems of scalability and availability in these approaches [26, 4, 19, 35]; this will be described in a more detailed manner in Section 3.2. Hence, the primary master of a region (i.e., P_{11}) must decide when to passively replicate its data to the pseudo-primaries of the neighboring regions (P_{21}, P_{31}, P_{41}). At the same time, each pseudo-primary has to take the same decision with its data and their pseudo-primaries or secondaries. These decisions must be taken according to (1) the function periodicity (i.e., flow monitoring will require faster updates than asset management), (2) link status and congestion, and (3) cluster status (i.e., a general master might decide to asynchronously replicate its data when it detects that there are very few alive nodes on its cluster).

Actually, once the primary master has sent its data to a pseudo-primary node of another cluster, a recursive process starts where each pseudo-primary looks for another pseudo-primary in another neighboring cluster to propagate its data. This process finishes when there are no more clusters or there is a cluster which has no more neighbors (i.e., S_{51}, S_{61}, S_{71}, and S_{81}). The decision process must be aware of not falling in cluster loops and ensure that in each cluster there is only one pseudo-primary node that contains data from the primary master.

Each time the computation unit of the smart grid needs to calculate the result of a given smart function, it first attempts to use data from its nearest neighboring cluster. If data contained on that cluster has a consistency level k greater than l, where l is the freshness level required by the function, it will use that cluster to perform computation. Otherwise, it will get redirected to its ancestor node with a freshness index $k-1$ and repeat the operation.

The following subsections detail how our architecture deals with replication, consistency and fault tolerance issues.

3.2. Replication. Distributed systems replicate data to provide scalability, availability, and fault tolerance. However, replication increases the number of messages in the sense that all replicas have to be synchronized

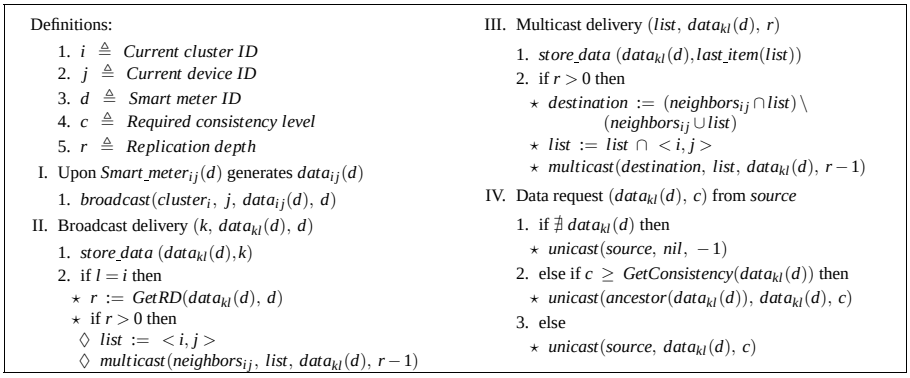


Fig. 3.2: Replication protocol at smart device$_{ij}$

which can potentially reduce the system throughput. To avoid this situation several techniques are proposed in the literature [31, 15, 11, 2].

Regarding the time when updates get propagated to the replicas there exist two major strategies: eager and lazy replication. On one hand, eager replication [5] consists of writing data to all replicas before finishing the write operation (similar to 2PC in databases). This solution provides strong consistency [33] but has limited scalability. On the other hand, lazy replication [21, 35] consists of writing to all replicas after exhausting the write operation. This technique achieves higher scalability but has more difficulties to maintain consistency—i.e. replicas may diverge.

Regarding the amount of replicas that update data, there exist two major strategies: active and passive replication. Active replication [3] broadcasts updates to all replicas at a time. Again, this technique provides strong consistency since all replicas are easily synchronized but has low scalability. Passive replication [29] processes updates on a single site called *primary* which propagates its updates to another site called *backup*. At the same time, this backup site can also propagate its state to another backup site until achieving the desired replication depth. This solution provides higher scalability but has some troubles on maintaining strong consistency since all replicas might be unsynchronized.

The final consideration regarding to scalability is the number of messages exchanged [36]; this is a critical factor too since the greater the number of messages exchanged per operation are the more network stalls we have. We can consider that there can be a linear interaction where the number of messages exchanged depends on the kind of operation; or, otherwise, constant interaction where a fixed number of messages is exchanged per operation. The former features poor scalability while the latter one can increase the scalability though we have to take into account that they have to be kept to a minimum (ideally one message and, at most, two).

In our system we have taken these previous considerations and propose a hybrid solution. We have considered to mix passive and active replication; however, we consider the propagation of changes to a small subset of replicas and the replicas belonging to this subset are responsible for propagating the changes to another disjoint small subset of replicas and so on up to the replication depth specified by the system for a certain variable. As depicted in Fig. 3.2, our proposal is a hybrid solution between all these techniques and benefits from the strengths of each solution:

1. When a device (primary master, P_{11} in Fig. 3.1) receives data from its smart meters it eagerly replicates them by *broadcast*ing (step I in Fig. 3.2) these data to all devices within its cluster (primary slaves, i.e., P_{12} and P_{13} in Fig. 3.1). Therefore, it is performing an active replication (step II.1 in Fig. 3.2).
2. If the replication depth r associated to these kind of data is greater than 0, the primary master has also to lazily replicate these data to other clusters (1) avoiding replication loops and (2) *multicast*ing relevant meta-data.
   In order to avoid replication loops (i.e., the same cluster has different versions of the same object), the device must build a *list* with the ancestors of data it is currently processing and remove all devices of its neighbor list that are in the ancestor list. Recall that each primary, or pseudo-primary, device has only one pseudo-primary per region.
   Regarding the neighbor discovery, we assume that given a neighboring cluster B from cluster A, all the

nodes in cluster A will choose the same pseudo-primary from cluster B. If this pseudo-primary fails, the next pseudo-primary chosen will be the one with the lowest identifier in cluster B. For example, in Fig. 3.1, if P_{33} fails, P_{32} and P_{33} will also chose S_{61} and S_{71} as their pseudo-primaries.
Upon the neighbor list has been pruned (and updated), the device *multicast*s (1) the ancestors *list*, (2) the stored $data_{kl}(d)$ and (3) a decremented value of replication depth to its neighbors (i.e., PP_{23}, PP_{33} and PP_{43} in Fig. 3.1) as shown in step II.2 in Fig. 3.2.

3. This is repeated while the replication depth is greater than 0 as shown in step III in Fig. 3.2. Note that we are actually performing active replication within devices of primaries' area and passive replication within devices of different clusters.
The first time data from $smart\ meter_d$ achieves the latest pseudo-primary (r has reached the 0 value), or secondary (i.e. $S_{51}, S_{61}, S_{71}, S_{81}$ in Fig. 3.1) this device will send to the computation unit its identifier. This ensures that the computation unit will eventually know where to find the furthest replica of data associated to $smart\ meter_d$. Recall that the nearest device that contains these data is its primary, which is also known by the computation unit in advance.
4. When a device receives a data query from the computation unit, it first checks if the consistency level of its stored data is enough to perform computation. If it is greater than the one required c, it will give back its stored data, otherwise it will forward this query to its ancestor as shown in step IV in Fig. 3.2. This solution is inspired in the passive replication technique and eases distributed computation as discussed in Section 5.

Therefore, this replication protocol performs both (1) active and eager replication in the primary-master's cluster and (2) passive lazily replication in other clusters inspired in the fundamentals of a primary-copy scheme. This improves the scalability issues of classical architectures [17, 19] since we have restricted an controlled the amount of replicas that receive the changes at a time and implement different regions (set of clusters) with different consistency levels. This last feature is very interesting; some computations do not need to be exclusively forwarded to the primary replica, it can be delegated to other replicas and, thus, the system achieves a better performance. This can be best seen with two examples, if we want to check the consumption of a subscriber for billing purposes then we will need to go to the primary (strong consistency). The other example can consist in checking the power consumption of a certain urban area to detect the variation and decide whether to derive or not more power to that area (this is specially useful in summer with air conditioning systems) and the consistency is not so critical.

Finally, we define the *replication chain* as the closed set of devices which exchange versions of the same data item. For example, in Fig. 3.1 there are four replication chains concerning data generated by smart meters attached to P_{11}: $\{P_{11}, PP_{23}, S_{51}\}$, $\{P_{11}, PP_{33}, S_{61}\}$, $\{P_{11}, PP_{33}, S_{71}\}$, and $\{P_{11}, PP_{43}, S_{81}\}$.

The following section describes how consistency is kept under this replication protocol.

3.3. Consistency. Research on consistency protocols has been conducted for many years and several approaches have been proposed by the community. First attempts [5] on keeping consistency in distributed systems consisted of building serializable plans by avoiding read and write operations performed concurrently over the same data item. This technique ensures that data will be always seen by anyone immediately after its update; which is also known as strong consistency [33]. Later, as the performance requirements increased, researchers relaxed the consistency guarantees by defining the weak consistency [33] with the aim of improving scalability and availability. These techniques [33] accept a limited period of time (also referred to as inconsistency window) where updates are available only to a subset of sites in the system. When the inconsistency window expires, they ensure that data is consistent.

Generally, this leads us to two major alternatives when defining the consistency properties of a system: (1) strong consistency and (2) weak consistency. Strong consistency provides us with high consistent systems but with poor scalability and availability since all replicas must be synchronized. In the other hand, weak consistency provides higher scalability and availability but limited consistency features.

Regarding our proposal, we take advantage of both strong and weak consistency strategies and propose a hybrid solution inspired by cloud-based storage and data stream warehouses [15, 11, 16].

Cloud computing-based storage techniques [34, 25], on its effort to achieve high scalability and availability, typically implement a specific form of weak consistency: eventual consistency [33]. This technique states that once data is updated, all replicas will eventually get that value if no more updates are applied.

Smart grids' behavior has many similarities with stream warehouses [16] since there is a continuous stream

of data (generated by smart meters) which has to be stored. In order to overcome the classical update and query consistency issues given on such scenarios [16], our architecture guarantees that there will never exist multiple attempts to write (or read) the same data item into the same place. Note that two or more replication chains of the same data item never converge to the same cluster. Nevertheless, we find very useful to use multi-version [6] techniques used in stream warehouses to maintain a notion of consistency between different data stored in each device and enhance the global system performance.

Although IEC 61850 standard [20] defines the data model that a smart grid should store and our architecture deals with this model by hiding data structures inside devices, we have to take care about data periodicity. Each smart meter generates data at different intervals, i.e., voltage measurements might be generated with a higher frequency than heat measurements. Hence, each data measurement is stored with the time stamp k it was originally acquired, similar to the version technique used in stream warehouses [16, 6].

Actually, there exists a close relationship between this time stamp k in which data are generated and the consistency of these data. Roughly speaking we can assure that the greater the k, the more recent data measurement is but its consistency is potentially weaker. Recall that here we understand consistency as the property which states that all the members in the replication chain own the same data item with time stamp k (also referred to as version).

In fact, when our architecture is required to store stream data measurements [16, 33], the most recent data versions will be always located closer to their sources; whereas oldest versions might have already reached the furthest devices in the replication chain. However, if a given measurement is not so frequently taken, then the most recent version will be found anywhere in its associated replication chain.

In some use cases, the computation unit can still work with older versions (weakly consistent) to perform the calculations required. Hence, when it wants to obtain a specific data item, it can include a k value to state that the computation should be done with values that equal to (or greater than) k. To this end, queries will be traveling along the replication chain associated to each data measurement to find the proper version up to its master (in the case that the required k version is not found).

In the master's cluster we implement strong consistency between all replicas (P_{11}, P_{12} and P_{13} in Fig. 3.1). This improves the fault tolerance of the system since another device (primary-slave) of the cluster could easily take over from a primary-master's fault. Moreover, this keeps us safe from the typical single point of failure problem. Actually, all data stored in any device belonging to the master's cluster has the highest k level of consistency, since no newest data have been generated.

Once data are strongly consistent in the master's cluster, devices start propagating them with a time stamp k to their pseudo-primaries (as shown in Fig. 3.2, P_{11} will broadcast to PP_{23}, PP_{33} and PP_{43}). Recall that decisions of when data must be propagated are taken according to the smart functions and system status. Therefore, we are currently implementing an eventually consistent system between the pseudo-primaries. It is more likely that data stored in these pseudo-primaries will provide weak consistency since new data measurements will come from smart meters with a time stamp strictly greater than k.

To sum up, from the consistency point of view, we have shown how our hybrid architecture uses both strong and weak (actually k-weak) consistency techniques. Next we describe how our architecture deals with fault tolerance.

3.4. Fault Tolerance. Fault tolerance is understood as the ability of the system to recover from a spontaneous site fault. Distributed systems are prone to different types of site failures [12]. Researchers have to adapt their techniques according to the domain of use and the intrinsic characteristics of the distributed system. This section explores (1) which kind of failures smart grids are prone and (2) how our proposal acts against them.

Since smart grids are hardly dependent on the communication network, we can assume that this channel will be reliable enough and focus our efforts on the distributed storage architecture. Therefore, our goal is to implement such a policy that in case of a node failure, the global system could still behave properly. In this case, we assume that any site may fail according to the crash model [12].

If a server started behaving in an arbitrary manner (also referred to as byzantine model), it would be either because it is returning or propagating an arbitrary or older (though valid) version of a variable. We control this by adding a digest to the value stored, similar to what it has been proposed in [9, 28]. Whenever we found a mismatch between them we would force the replica to shut down.

Regarding our proposal, there may exist two different failure cases. The first one corresponds to the failure of a primary (i.e., P_{11} in Fig. 3.1) and the second one to the failure of a pseudo-primary (i.e., PP_{23} in Fig.

3.1). In the former, we inherit the advantages of the active replication techniques and are able to recover easily: when the primary master fails (e.g., P_{11} in Fig. 3.1), any other primary-slave can immediately take over the situation. Recall that due to eager replication, they are completely sharing the state of the primary master.

In the latter failure scenario, any of the nodes of the cluster where the failure takes place can become the new pseudo-primary and continue with data transmission and replication. Unfortunately, due to the fact that pseudo-primaries perform passive and lazy replication, the takeover process is not as fast as in the primary's cluster. As soon as the ancestor, belonging to cluster A, of the failed node, belonging to cluster B, detects its unresponsiveness, it will select a new pseudo-primary from cluster B. If no more pseudo-primaries (or secondaries) are available (i.e., cluster 7 in Fig. 3.1) it will send a message to the computation unit informing that it is the last device of the replication chain. Otherwise, thanks to the neighbor discovery function, it can easily find its successor in the replication tree and continue with the replication of the data in the system.

The challenge here is that the takeover solution in a pseudo-primary implies that the previous versions of a given data item are lost. We have to define a state transfer protocol so that every cluster contains the most complete and up-to-date information. Otherwise, there might exist certain network clusters where we cannot achieve the desired degree of consistency due to the fact that these *new* pseudo-primaries do not completely store the state. In Section 5, we will deal with this data transfer that can range from a full state transfer to nothing sent. In all this range, we are going to see the advantages and disadvantages of each solution.

However, this is not enough since most of these processes do not completely stop forever after their failure. This system has certain dynamism, in the sense that components can be repaired. Hence, there can be a role re-assignment and the need of recovery tasks for previously crashed nodes.

The following subsection proposes an analytical study of the scale out factor of our protocol in order to provide with some arguments to warrant the viability of our approach.

3.5. Proof of concept. As one of the major goals of our proposal is achieving high scalability, we use the analytical model and notation from [31] which estimates the scale out factor in a replicated database system when there is an increase of the number of sites and replicas. Thus, we briefly describe the adaption to our system model used to proceed with the computation of the scale out factor.

The scale out factor determines how the performance of the global system is increased or decreased by using replication. As shown in Equation 3.1, this is computed as the sum of work executed at each replica divided by the processing capacity of a non-replicated database.

$$Scale\ \ Out = \frac{1}{C} \sum_{i=1}^{n} \sum_{j=1}^{m} C \cdot U_{ij} \cdot ACC_{ij} \quad [31]. \tag{3.1}$$

From the previous equation, we have that C is the processing capacity of a non-replicated database, n is the number of replicas in our smart grid, m is the number of stored objects, U_{ij} defines the location of object j at site i and ACC_{ij} defines the accessibility of object j at site i. Hence, if the object j is at site i, then $U_{ij} = 1$ which defines the replication schema. Actually, ACC_{ij} defines the access rate to object j at site i.

However, Equation 3.1 assumes that read and update operations launched against the replicated distributed system are uniformly spread across all replicas. Hence, this equation is not directly applicable to our proposal because our solution has an in-built load balancing mechanism which redirects operations to replicas according to the required consistency level k. Thus, we show how we have adapted the analytical description of the replicated system to fit in our system characteristics.

Recall that the term $(C \cdot U_{ij} ACC_{ij})$ is equivalent to $(C_r \cdot ACCR_{ij} + C_u \cdot ACCU_{ij})$ where C_r is the read processing capacity, C_u is the update processing capacity, $ACCR_{ij}$ is the read accessibility and $ACCU_{ij}$ is the update accessibility. So, replacing this expression in Equation 3.1 we obtain Equation 3.2 which is now useful in our replication protocol.

$$Scale\ \ Out = \frac{1}{C} \sum_{i=1}^{n} \sum_{j=1}^{m} C_r \cdot ACCR_{ij} + C_u \cdot ACCU_{ij}. \tag{3.2}$$

Actually, Equation 3.1 can be considered a generalization of 3.2. We now evaluate the different replication strategies for a system from 10 to 80 sites, 10 objects, 80% of read operations and 20% of write operations to

Table 3.1: Scale out evaluation

Sites	Full Replication	Partial Replication	**Smart Grid**
0	0	0	0
10	10	9,8	9,82
20	15,7	16,9	18,15
40	25	28,2	30,7
80	40	48,2	53

all objects. The distribution of objects and the operations on objects are evenly distributed. We also assume that all sites have the same processing capacity (i.e. all smart meters have similar specifications and storage capabilities).

We have compared the scale out factor obtained in our proposed protocol against the replication strategies proposed in [31]: (1) full replication—all sites contain the same data—and (2) partial replication —a reduced set of sites contains a given data. In the case of the latter, we have chosen to replicate each data item in $n/2$ sites in order to obtain comparative results. The scale out evaluation of these protocols is shown in Table 3.1.

Ideally, the scale out factor should be equal to the number of sites of the system which meant that all incoming updates are being processed without saturation. We can see that with a full replication scheme the scale out is quite poor: 40 with 80 sites. When the number of sites increases, the system cannot scale anymore as the full replication policy forces that all updates have to be sent to all replicas which takes a considerable amount of time.

In contrast, with the partial replication (limited to half of the replicas) the system scales slightly better because the cost of propagating the replicas is not so high (scale out of $48,2$ with 80 sites). Finally, our cloud based replication protocol scales out is even better (53 at 80 sites). In this case the replicas that have the most recent data version have a higher C_u and lower $ACCR_{ij}$ because most read operations are executed in replicas that do not have necessary the last version of the data. For these reasons, we show that the scale out is better than traditional full replication and partial database replication protocols. This proof of concept makes us believe that this is a good approach in power smart grids.

This subsection finishes the description of our proposal. In the following section, we roughly verify the correctness properties of this system if all nodes behave properly and faults never occur.

4. Correctness Guarantees. This section provides arguments for correctness of the global distributed system in a failure free environment. Distributed algorithms are said to be formally correct when their liveness and safety properties are satisfied and shown to be correct. Regarding the liveness property, it can be best seen as something good will eventually happen; while, the safety property can be stated as nothing bad will happen.

Our proposal needs to propagate the measures from a given smart meter from zone to zone up to its replication level; hence, changes need to get propagated and applied in the same order in all pseudo-primaries. Both asserts constitute the liveness and safety properties of our system.

Next, we point out some guarantees of the system extracted from the previous sections in order to have enough arguments to warrant the correctness properties.

Guarantee 1. Any primary (or pseudo-primary) will never send the same data item to more than one device per neighboring cluster.
This is guaranteed since data ancestors are excluded from the device's neighbor list as shown in step III.2 in Fig. 3.2.

Guarantee 2. The computation unit knows which is the last pseudo-primary (or secondary) of the replication chain.
As described in section 3.2, when a device notices that a new data item has gone through all its replication depth ($r = 0$), it will send a message to the computation unit identifying itself.

Guarantee 3. Any device belonging to a master cluster has always the latest version of data generated by any smart meter within that cluster.
This is satisfied since data is eagerly replicated to all devices of the master cluster as shown in steps I.1 and II.1 in Fig. 3.2.

From this guarantees, we can state the safety and liveness properties of our system. As the system behaves

different depending if it is replicating data (also referred to as write) or executing a query from the computation unit (also referred to as read), both properties (safety and liveness) must be analyzed in two facets: read and write.

4.1. Safety Properties. The safety properties of our architecture are stated by the following claims:

CLAIM 1. **Safety on write.** Safety on write operations is guaranteed since there will never occur a situation where the same data is being written from two or more different sources.
This is guaranteed since (1) there is only one $smart\_meter_d$ generating $data_d$, (2) point to point communication channels do not disorder messages, and (3) the *neighbor* function will never find more than one device per cluster as stated in Guarantee 1.

CLAIM 2. **Safety on read.** There will always exist a consistent version of the requested data item queried by the computation unit.
This is assured provided that the replication protocol (Fig. 3.2) guarantees consistent writes throughout the whole replication chain.

4.2. Liveness Properties. The liveness properties of our architecture are stated by the following claims:

CLAIM 3. **Liveness on write.** Liveness on data updates is trivially assured by Guarantee 1.
Data generated on the smart meter will follow the replication chain and being consistently written at each device until the replication depth reaches a value of 0 or there are no more neighbors.

CLAIM 4. **Liveness on read.** Liveness on data reads is guaranteed provided that Claim 2 is accomplished.
The computation unit will always send queries to the last device of the replication chain. If data contained in it have not the required consistency level k, the device will redirect the query to its ancestor. This will happen in a recursive way until the required level k is found. If any device can reach the required level k the primary will give back with its latest version which is strongly consistent as provided by Guarantee 3.

Finally, in the following section we discuss some limitations of our approach.

5. Discussion. As shown in the previous sections, our proposal takes benefit from many techniques used in distributed systems. However, to the best of our knowledge, these techniques have never been put together nor tested. Hence, there are several aspects that have been intentionally left out and need to be discussed:

Master cluster reduction. Not all the members of the master region have to participate on active replication. Along this work, we have assumed that all nodes of a given primary cluster participate in the active replication of all data. In fact, this is not necessary at all: although the number of nodes belonging to a zone can be in the range of tens, we think that we can speed up the replication process by selecting a set of representatives for each subset of smart meters. It is well known that active replication does not scale well [35] and with the proper selection of representatives the rest can become secondaries of each representative.

Enhancing the takeover process. A pseudo-primary (PP) could do active replication within its associated cluster. This role is not an exclusive one in the cluster, it can be also responsible for several smart meters and, thus, collaborate in the active replication protocol.

Failure detection. In Section 3.4 we have stated that the ancestor detects the failure of its successor in the replication hierarchy; this can be achieved by implementing a timeout plicy. However, the update propagation frequency may vary since data to be transmitted is very different (i.e., monthly power consumption is less frequent than voltage monitoring at a given point in the network).

Hence, it makes sense to think that inside the cluster of a given pseudo-primary (or secondary) the active replication among their nodes would detect the failure of the pseudo-primary (or another device). If so, they can agree with selecting a new pseudo primary in that cluster and notify the ancestor about this fact. Again, we have to reconstruct the new hierarchy tree in order to add the new pseudo-primary. However, this would overload the pseudo-primaries' clusters and might worsen the global performance of the system.

State transfer in presence of faults. It is said, that when a pseudo-primary fails we need to transfer its data (missing data) to the new pseudo-primary as its copy is lost once it fails. We can have several alternatives to mitigate this effect as this system is derived from a data driven application.

On one hand, we can perform an active replication of each pseudo-primary in its cluster zone. As mentioned in the previous point, this approach might not be feasible since we increase the load of the system as we go deep in the replication structure of the system, which potentially may flood the whole network; leading with a not desirable situation. Note that this solution does not need to transfer any state information from anywhere in case of fault.

On the other hand, we have the alternative to transfer the full state to the new successor from the ancestor (recall that it belongs to another cluster and this could be costly); however, this alternative has several drawbacks. The first one is that it might affect the availability of the system since transferring the whole data may affect the transmission of new data to the successors. The second one is that the volume to transfer might be so big that it could not catch up with the current state of the system [32].

There may exist an hybrid solution: we could perform a partial state transfer of data. This implies that the replication algorithm has to ensure that each pseudo primary has a set of secondaries in its associated cluster where the updates get also propagated asynchronously. Therefore, when a given pseudo-primary fails, it is only needed to transfer a much less amount of data to the new pseudo-primary than in the case of full state transfer. Nevertheless, this has to be tested and checked in order to find out the proper number of pseudo-primary slaves and the amount of data transferred per round.

Distributed computing. Our proposed architecture allows to perform distributed computation on the read steps. Thanks to the fact that required data travel across the replication chain (depending on the required consistency level k) each node might be able perform a piece of the computation required.

Actually, as suggested in [14] for cloud computing environments, with our architecture it would be possible to implement something similar to MapReduce; where the node owning the required data version run the *map* tasks and the rest of nodes in the replication chain continuously run the *reduce* tasks. Such distributed computation not only might reduce the size of the traveling data but also improve the computation throughput of the smart grid.

However, we have not considered this feature in this work but it can be seen as a chance to improve the intelligence and power of the system.

Dynamic replication depth tuning. We believe that if we were able to dynamically adjust this value our system might adapt better to their requirements. In fact, we have not specifically stated how the replication depth is set, neither augmented or decreased. It can be adjusted by the system administrator but it can also be dynamically adjusted as a function of the demands coming from the computation unit. Moreover, it can be tuned autonomously in case of disaster or rapid evaluation of certain functions (e.g. accounting). Further, there might exist certain information that would need to be replicated in all nodes as it is rarely modified.

To this end, we are thinking about a cognitive system [18] based on supervised learning in order to (1) evaluate the whole system status and (2) predict the optimal value of the replication depth for each data item. Actually, we are implementing a learning classifier system (e.g., XCS or UCS) [37] able to adapt itself to the system dynamism due its online nature.

6. Conclusions and Future Work. This paper presents a novelty approach to take advantage of smart electric grids. We have defined a way to distribute and store information across the network so that the computation needed for certain smart functions can be greatly reduced. We have detailed the replication protocol based on epidemic updates and proofed its correctness. This work aims to provide some insight into the world of smart grids from a data perspective. For the sake of simplicity during the presentation of our system, we have outlined simple scenarios about the replication policy or fault-tolerance issues that need to be treated in detail in further works.

In addition, future work should be targeted at (1) implementing the architecture in a real-world scenario to obtain numerical values of its performance, and (2) defining the cognitive system behavior in order to tune and optimize the replication protocol according to real data access and update patterns.

REFERENCES

[1] AEP Ohio. About the gridsmart project, Feb 2011. Available in: `www.aepohio.com/save/demoproject/about/Default.aspx`.
[2] Marcos Kawazoe Aguilera and et al. Sinfonia: A new paradigm for building scalable distributed systems. *ACM Trans. Comput. Syst.*, 27(3), 2009.
[3] Yair Amir and Ciprian Tutu. From total order to database replication. In *ICDCS*, pages 494–, 2002.
[4] José Enrique Armendáriz-Iñigo, J. R. Juárez-Rodríguez, José Ramón González de Mendívil, Hendrik Decker, and Francesc D. Muñoz-Escoí. *k*-bound GSI: a flexible database replication protocol. In Yookun Cho, Roger L. Wainwright, Hisham Haddad, Sung Y. Shin, and Yong Wan Koo, editors, *SAC*, pages 556–560. ACM, 2007.
[5] Philip A. Bernstein and et al. *Concurrency Control and Recovery in Database Systems*. Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA, 1987.
[6] Irina Botan and et al. Secret: A model for analysis of the execution semantics of stream processing systems. *PVLDB*, 3(1):232–243, 2010.
[7] Eric A. Brewer. Towards robust distributed systems. In *PODC Conf.*, NY, USA, 2000. ACM.

[8] Richard E. Brown. Impact of Smart Grid on distribution system design. In *Power and Energy Society General Meeting - Conversion and Delivery of Electrical Energy in the 21st Century, 2008 IEEE*, pages 1–4, 2008.
[9] Miguel Castro and Barbara Liskov. Practical byzantine fault tolerance and proactive recovery. *ACM Trans. Comput. Syst.*, 20(4):398–461, 2002.
[10] A. Chuang and M. McGranaghan. Functions of a local controller to coordinate distributed resources in a smart grid. In *Power and Energy Society General Meeting - Conversion and Delivery of Electrical Energy in the 21st Century, 2008 IEEE*, pages 1 –6, 2008.
[11] Brian F. Cooper and et al. Pnuts: Yahoo!'s hosted data serving platform. *PVLDB*, 1(2):1277–1288, 2008.
[12] Flaviu Cristian. Understanding fault-tolerant distributed systems. *Commun. ACM*, 34(2):56–78, 1991.
[13] Sudipto Das, Divyakant Agrawal, and Amr El Abbadi. Elastras: An elastic transactional data store in the cloud. *CoRR*, abs/1008.3751, 2010.
[14] Jeffrey Dean and Sanjay Ghemawat. Mapreduce: a flexible data processing tool. *Commun. ACM*, 53(1):72–77, 2010.
[15] Giuseppe DeCandia, Deniz Hastorun, Madan Jampani, Gunavardhan Kakulapati, Avinash Lakshman, Alex Pilchin, Swaminathan Sivasubramanian, Peter Vosshall, and Werner Vogels. Dynamo: amazon's highly available key-value store. In *SOSP*, pages 205–220, 2007.
[16] Lukasz Golab and Theodore Johnson. Consistency in a Stream Warehouse. In *CIDR*, 2011.
[17] Jim Gray and et al. The dangers of replication and a solution. In H. V. Jagadish and Inderpal Singh Mumick, editors, *Proceedings of the 1996 ACM SIGMOD International Conference on Management of Data, Montreal, Quebec, Canada, June 4-6, 1996*, pages 173–182. ACM Press, 1996.
[18] J. Holland. *Adaptation in natural and artificial systems.* The MIT Press, second edition, 1992.
[19] Ricardo Jiménez-Peris, Marta Patiño-Martínez, Bettina Kemme, and Gustavo Alonso. Improving the scalability of fault-tolerant database clusters. In *ICDCS*, pages 477–484, 2002.
[20] Tatjana Kostic, Otto Preiss, and Christian Frei. Understanding and using the iec 61850: a case for meta-modelling. *Computer Standards & Interfaces*, 27(6):679–695, 2005.
[21] Konstantinos Krikellas, Sameh Elnikety, Zografoula Vagena, and Orion Hodson. Strongly consistent replication for a bargain. In *ICDE*, pages 52–63, 2010.
[22] Lakshman, Avinash and Malik, Prashant. Cassandra: a decentralized structured storage system. *SIGOPS Operating Systems Review*, 44(2), Apr 2010.
[23] Masdar. Masdar website, Feb 2011. Available in: `www.masdar.ae/en/home/index.aspx`.
[24] mThink. The smart grid in malta, Feb 2011. Available in: `mthink.com/utilities/utilities/smart-grid-malta`.
[25] Mayur R. Palankar, Adriana Iamnitchi, Matei Ripeanu, and Simson Garfinkel. Amazon s3 for science grids: a viable solution? In *DADC '08: Proceedings of the 2008 international workshop on Data-aware distributed computing*, pages 55–64, New York, NY, USA, 2008. ACM.
[26] Marta Patiño-Martínez, Ricardo Jiménez-Peris, Bettina Kemme, and Gustavo Alonso. Middle-r: Consistent database replication at the middleware level. *ACM Trans. Comput. Syst.*, 23(4):375–423, 2005.
[27] Alberto Paz, Francisco Perez-Sorrosal, Marta Patiño-Martínez, and Ricardo Jiménez-Peris. Scalability evaluation of the replication support of jonas, an industrial j2ee application server. In *EDCC*, pages 55–60. IEEE Computer Society, 2010.
[28] Fernando Pedone, Nicolas Schiper, and José Enrique Armendáriz-Iñigo. Byzantine fault-tolerant deferred update replication. In *LADC*, pages 7–16. IEEE Computer Society, 2011.
[29] Fernando Pedone, Matthias Wiesmann, André Schiper, Bettina Kemme, and Gustavo Alonso. Understanding replication in databases and distributed systems. In *ICDCS*, pages 464–474, 2000.
[30] Christian Plattner, Andreas Wapf, and Gustavo Alonso. Searching in time. In *SIGMOD Conference*, pages 754–756, 2006.
[31] Damián Serrano, Marta Patiño-Martínez, Ricardo Jiménez-Peris, and Bettina Kemme. Boosting database replication scalability through partial replication and 1-copy-snapshot-isolation. *Pacific rim international symposium on dependable computing, IEEE*, 0:290–297, 2007.
[32] Ricardo Manuel Pereira Vilaça, José Orlando Pereira, Rui Carlos Oliveira, José Enrique Armendáriz-Iñigo, and José Ramón González de Mendívil. On the cost of database clusters reconfiguration. In *SRDS*, pages 259–267, 2009.
[33] Werner Vogels. Eventually consistent. *Commun. ACM*, 52(1):40–44, 2009.
[34] White, Tom. *Hadoop: The Definitive Guide.* O'Reilly Media, 1 edition, June 2009.
[35] Matthias Wiesmann and André Schiper. Comparison of database replication techniques based on total order broadcast. *IEEE Trans. Knowl. Data Eng.*, 17(4):551–566, 2005.
[36] Matthias Wiesmann, André Schiper, Fernando Pedone, Bettina Kemme, and Gustavo Alonso. Database replication techniques: A three parameter classification. In *SRDS*, pages 206–215, 2000.
[37] S. Wilson. Classifier fitness based on accuracy. Technical report, The Rowland Institute for Science, 100 Edwin H. Land Blvd. Cambridge, MA 02142, Apr 1995.

Edited by: Dana Petcu and Jose Luis Vazquez-Poletti
Received: August 1, 2011
Accepted: August 31, 2011

Scalable Computing: Practice and Experience
Volume 12, Number 3, pp. 363–369. http://www.scpe.org
ISSN 1895-1767

A HYBRID FIREFLY-INSPIRED APPROACH FOR OPTIMAL SEMANTIC WEB SERVICE COMPOSITION

CRISTINA BIANCA POP, VIORICA ROZINA CHIFU, IOAN SALOMIE, RAMONA BIANCA BAICO, MIHAELA DINSOREANU, GEORGIANA COPIL *

Abstract. Inspired from biology, in this paper we propose a hybrid firefly method for selecting the optimal solution in semantic Web service composition. In our approach, the search space of the selection method is represented by an Enhanced Planning Graph structure which encodes all the Web service composition solutions for a given user request. As selection criteria we have considered the *QoS* attributes of the services involved in the composition as well as the semantic similarity between them. For the evaluation of the proposed selection method we have implemented an experimental prototype and carried out experiments on a scenario from the trip planning domain.

Key words: semantic Web service composition, firefly-based search, genetic operators, QoS, semantic similarity

AMS subject classifications.

1. Introduction and Related Work. The selection of the optimal solution in semantic Web service composition can be seen as an optimization problem which requires specific selection techniques that provide the desired results in an efficient way. Recent research studies demonstrated that principles inspired by the biological systems have lead to the design of efficient techniques that can be used to solve optimization problems. These biologically-inspired techniques are advantageous since they are capable of converging towards the optimal or a near-optimal solution in a short time without processing the entire search space. Such meta-heuristics include Genetic Algorithms, Ant Colony Optimization [3], Particle Swarm Optimization [5], or Artificial Immune Systems [1]. These meta-heuristics have been successfully applied to the problem of selecting the optimal solution in Web service composition.

In [8] genetic algorithms are used to find the optimal composition solution. The composition method is based on a given service abstract workflow, where each abstract service has a set of concrete services with different *QoS* values associated. Genetic algorithms are used to bind concrete services to the abstract ones aiming at identifying the optimal workflow instance in terms of *QoS* attributes. The genome is encoded as a binary string, where each position is associated to an abstract service in the workflow and indicates the concrete service which is selected to be used. The approach uses genetic operators and fitness functions applied on the genome to find the optimal composition solutions. The random mutation operator is used to generate new workflow instances. In the case of fitness functions, the approach presented in [8] proposes three fitness functions, one associated to each considered *QoS* attribute, in order to increase the probability of finding the optimal composition solution.

In [11] a method based on Ant Colony Optimization has been proposed for selecting the optimal service composition which uses the *QoS* attributes as selection criteria. The approach applies an ant-based selection method on a composition graph, where the graph nodes represent abstract services having associated sets of concrete services, while the edges denote the interactions between services. The *QoS* model defined in [11] makes distinction between the *QoS* attributes that should be minimized or maximized. In addition, the multi-pheromone concept is introduced for representing the *QoS* attributes.

A hybrid method combining Particle Swarm Optimization (PSO) [5] with Simmulated Annealing is proposed in [4] for selecting the optimal or a near-optimal service composition solution based on *QoS* attributes. Authors model service composition using an abstract workflow on which concrete services are mapped. A composition solution is considered as the position of a particle in PSO, while velocity is used to modify a composition solution. To avoid the problem of premature stagnation in a local optimal solution, a Simmulated Annealing-based strategy is introduced which produces new composition solutions by randomly perturbing an initial solution.

In [9], an immune-inspired selection algorithm is proposed in the context of Web service composition. The approach uses an abstract composition plan which is mapped to concrete services, obtaining in this way a graph structure having services in the nodes and *QoS* values on the arcs. The immune-inspired algorithm is applied

*Department of Computer Science, Technical University of Cluj-Napoca, 26-28 Baritiu str., Cluj-Napoca, Romania, ({Cristina.Pop, Viorica.Chifu, Ioan.Salomie}@cs.utcluj.ro).

to select the optimal composition solution based on QoS attributes. A composition solution is encoded using a binary string and a mutation operator is used to generate new composition solutions.

In this paper we present a hybrid method for selecting the optimal composition solution that combines an extended version [6] of the Firefly Search algorithm [10] with genetic operators. The search space for the hybrid method is represented by an Enhanced Planning Graph (EPG) which encodes all the service composition solutions for a given user request. In our approach, a user request is described in terms of functional and non-functional requirements. To identify the optimal solution encoded in the EPG, we define a fitness function which uses the QoS attributes and the semantic quality of the services involved in composition as selection criteria. The proposed selection method was tested on a scenario from the trip planning domain.

The paper is structured as follows. Section 2 presents the formal model for representing semantic Web service composition. Section 3 details the Hybrid Firefly method for selecting the optimal solution in Web service composition. Section 4 presents the performance evaluation of the proposed selection method. We end our paper with conclusions.

2. Semantic Web Service Composition Model. In our approach, Web service composition is modeled using an *Enhanced Planning Graph* (EPG) structure [7]. This graph actually represents the search space of the hybrid firefly selection method. The EPG is obtained by mapping the classical AI planning graph to the semantic Web service composition domain and by adding new domain related structures to ease the service composition representation and service selection. The construction of the EPG is an iterative process which is applied at the semantic level by considering the ontology concepts that annotate the services functionality and their input/output parameters. In each iteration, a new layer consisting of a tuple (A_i, L_i) is added to the graph where: (1) A_i contains clusters of services whose inputs are provided by the services from the previous layers and (2) L_i contains clusters of service parameters. A cluster of services groups services which provide similar functionality, while a cluster of service parameters groups similar input and output parameters. The first graph layer is represented by the tuple (A_0, L_0), where A_0 is an empty set of service clusters and L_0 contains the user-provided input parameters. The construction of the EPG ends either when the user requested outputs are contained in the current set of service parameters or when the graph reaches a fixed point. Reaching a fixed point means that the sets of clusters of services and parameters are the same for the last two consecutive generated layers.

A composition solution encoded in the EPG consists of a set of services, one from each cluster from each EPG layer.

3. The Hybrid Firefly Selection Method. The hybrid method for selecting the optimal solution in semantic Web service composition combines a firefly algorithm [6] with principles from evolutionary computing. We have proposed such a hybrid method to maintain a good balance between exploration and exploitation thus eliminating the problem of local optimum stagnation.

3.1. Problem Formalization. The firefly meta-heuristic relies on a set of artificial fireflies which communicate with each other to solve optimization problems. The behavior of artificial fireflies is modeled according to the behavior of fireflies in nature, which search for a mating partner by emitting a flashing light. In this section we present how we mapped the concepts of the firefly meta-heuristic to the problem of Web service composition.

Just as the real fireflies search for a mating partner by means of flashing lights, we have a number of artificial fireflies which search for the optimal service composition solution. Thus, we map the attraction behavior of fireflies to the problem of selecting the optimal service composition as follows: (i) a firefly becomes an artificial firefly, (ii) the position of a firefly becomes a service composition solution, (iii) the brightness of a firefly becomes the quality of a composition solution evaluated with a multi-criteria fitness function, (iv) the attractiveness between two fireflies becomes the similarity between two composition solutions, (v) the movement of a firefly is mapped to a modification of the firefly's current composition solution, (vi) the environment in which fireflies fly is mapped to the EPG.

We formally define an artificial firefly as follows:

$$firefly = (sol, score) \tag{3.1}$$

where sol is a service composition solution and $score$ is the quality of sol.

A service composition solution is defined as:

$$sol = \{solElem_1, ..., solElem_n\} \tag{3.2}$$

where (i) $solElem_i$ is a solution element composed of a set of services, one service from each cluster of layer i; and (ii) n is the total number of layers in the EPG.

To evaluate the score of a composition solution, we define a fitness function QF which considers the QoS attributes of the associated services as well as the semantic quality of the connections between these services:

$$QF(sol) = \frac{w_{QoS} * QoS(sol) + w_{Sem} * Sem(sol)}{(w_{QoS} + w_{Sem}) * |sol|} \tag{3.3}$$

where: (i) $QoS(sol)$ citepop2010 is the QoS score of the composition solution sol; (ii) $Sem(sol)$ [7] is the semantic quality score of the solution sol; (iii) w_{QoS} and w_{Sem} are the weights corresponding to user preference related to the relevance of QoS and semantic quality.

3.2. The Hybrid Firefly Selection Algorithm. A prerequisite of the hybrid selection method is to establish the number of fireflies that will be used in the search process so as to obtain the optimal solution in a short time interval and without processing the entire search space. We have defined the number of fireflies based on the total number of solutions encoded in the EPG:

$$noF = Round(\sqrt[n]{noSol}) \tag{3.4}$$

where: (i) $noSol$ is the number of possible composition solutions encoded in the EPG; and (ii) n is a positive integer determined experimentally.

The inputs of the selection algorithm (see *Algorithm_1*) are: (i) the EPG resulted from the Web service composition process, (ii) the weights w_{QoS} and w_{Sem} which state the relevance of a solution's QoS quality compared to its semantic quality, and (iii) a number noF (formula 3.4) of artificial fireflies used to search for the best composition. The algorithm returns the optimal or a near-optimal composition solution.

Algorithm 3.2.1 Hybrid_Firefly_Web_Service_Selection

```
1  Input: EPG, w_QoS, w_Sem, noF
2  Output: fSol_best
3  begin
4  FSOL = ∅
5  for i = 1 to noF do FSOL = FSOL ∪ Gen_Random_Solution(EPG)
6  repeat
7      for i = 1 to noF do
8          for j = 1 to noF do
9              if (QF(FSOL[i]) < QF(FSOL[j])) then
10                 r = Compute_Distance(FSOL[i], FSOL[j])
11                 FSOL[i] = Crossover(FSOL[i], FSOL[j], r)
12                 u = Generate_Random_Vector(|FSOL[i]|)
13                 FSOL[i] = Mutation(FSOL[i], u)
14             end if
15         end for
16     end for
17     fSol_best = Get_Best_Solution(FSOL)
18     SOL_best = SOL_best ∪ fSol_best
19     u = Generate_Random_Vector(|fSol_best|)
20     FSOL = Modify_Best_Firefly(FSOL, u)
21 until (Stopping_Condition())
22 return Get_Best_Solution(SOL_best)
23 end
```

In the first step of the selection algorithm each firefly is associated with a randomly generated composition solution (see line 5). These initial solutions are further improved in an iterative process which stops when the best solution has been the same over the last $noIt$ iterations (see line 21).

In each iteration, if the score of the solution associated to a firefly is better than the score of the solution associated to another firefly it means that the latter firefly will be attracted towards the first one and thus it will have its solution improved. The steps for improving the solution associated to the less bright firefly are the following:

1. The distance r between the two composition solutions is computed (see line 10) as the difference of their scores.
2. A crossover operator is applied between the two composition solutions in a number of points depending on the value of the distance r compared to three thresholds r_1, r_2, r_3 (see line 11). As a result of the crossover operation, two new solutions will be obtained and the one having the highest score according to the QF function will be kept.
3. A mutation operation (see line 13) is performed on the best solution obtained within crossover to introduce diversity. In the mutation process, a mutation vector is randomly generated (see line 12) to specify the points where services will be replaced with other services from the same clusters.

After all solutions have been processed, the best one is determined (see line 17), added to the set of the best composition solutions and then mutated (see line 20) according to a randomly generated mutation vector (see line 19). This last mutation is performed to enlarge the search space and to avoid the stagnation in a local optimal solution.

4. Performance Evaluation. To validate our selection approach we have implemented an experimental framework which has been tested on a set of scenarios from the trip planning domain. First we performed a series of experiments aiming to identify the optimal values of the selection method's adjustable parameters. Then, using the optimal configuration of the adjustable parameters we performed further tests to evaluate the performance of the proposed method in contrast with a Bee-inspired selection method.

4.1. Experimental Framework. The architecture of the experimental framework is presented in Figure 4.1.

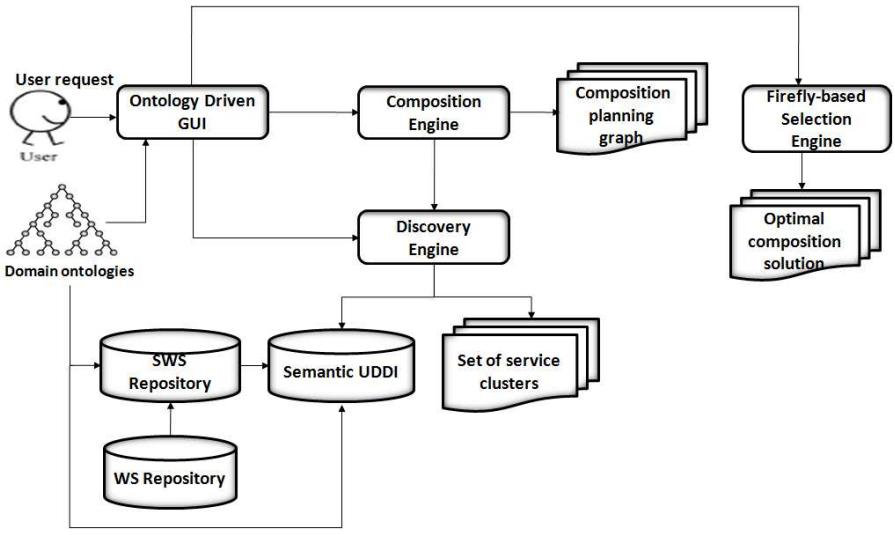


Fig. 4.1: The architecture of the experimental framework.

The ontology driven graphical user interface guides the users in the processes of searching and composing Web services by providing a controlled language that uses the ontology concepts. The SWS Repository is a repository of semantically annotated services based on the Domain Ontology. The Semantic UDDI extends the classical UDDI structure by (*i*) storing semantic Web service descriptions and (*ii*) providing means to semantically inquire the UDDI repository. The Discovery Engine receives a service request from a user or from the Composition Engine. In the case of a user request, the Discovery Engine provides an ordered set of atomic Web services which match the request. In the case of a service request from the Composition Engine, the Discovery Engine provides a set of atomic Web services organized in clusters. To satisfy the service requests,

the Discovery Engine inquires the Semantic UDDI. The Composition Engine is the component responsible for the construction of the EPG. The Composition Engine interacts with the Discovery Engine which provides the appropriate service clusters. The EPG is then used by the Firefly-based selection engine which provides the optimal or a near optimal composition solution according to the user preferences.

4.2. Setting the Optimal Values of the Adjustable Parameters. The framework has been tested on a scenario (see in Table 4.1 the associated user request) from the trip planning domain for which the EPG is organized on 3 layers consisting of 51 services grouped in 11 clusters. This EPG encodes 13996800 composition solutions.

Table 4.1: User request for planning a holiday

User inputs	Requested outputs	*QoS* weights	Semantic quality weight
SourceCity, DestinationCity, StartDate, EndDate, HotelType, NumberOfPersons, NumberOfRooms, CarType, ActivityType	AccomodationInvoice, FlightInvoice, CarInvoice	Total $QoS = 0.55$, Availability = 0.30, Reliability = 0.30, Cost = 0.15, Response time = 0.25	0.35

To identify the optimal values of the selection method's adjustable parameters (the number of stagnations, noS, the three thresholds, r_1, r_2 and r_3, for computing the number of crossover points) we have performed 100 runs of the associated selection algorithm for each configuration and analyzed how the average number of processed solutions ($noSolGen_{avg}$), the average simulation time ($tExec_{avg}$), and the standard deviation ($stDev_{avg}$) are affected. Table 4.2 presents the average values obtained for each configuration considered.

Table 4.2: Tests summary for the Hybrid Firefly Algorithm

#	noS	r_1	r_2	r_3	$noSolGen_{avg}$	$tExec_{avg}$	$stDev_{avg}$
1	3	0.001	0.005	0.01	2103	8	0.0084
2	3	0.003	0.006	0.008	2186	9	0.0089
3	3	0.03	0.06	0.09	2146	8	0.0126
4	3	0.01	0.05	0.099	2192	8	0.0111
5	6	0.001	0.005	0.01	2145	9	0.0076
6	6	0.003	0.006	0.008	2120	9	0.0089
7	6	0.03	0.06	0.09	2119	8	0.0089
8	6	0.01	0.05	0.099	2098	7	0.0111
9	9	0.001	0.005	0.01	2079	8	0.0098
10	9	0.003	0.006	0.008	2079	8	0.0107
11	9	0.03	0.06	0.09	2070	8	0.0125
12	9	0.01	0.05	0.099	2140	8	0.0108

When choosing the final values of the adjustable parameters a tradeoff between obtaining the optimal solution and a very low execution time has to be considered. By analyzing the results from Table 4.2 it can be noticed that the smallest average execution time, 7 seconds, corresponds to a standard deviation of the

identified optimal solution of 0.0111, but for a standard deviation of 0.0076, which is the smallest one, the average execution can get up to 9 seconds.

By analyzing the experimental results we conclude that the Hybrid Firefly selection algorithm returns the optimal or a near-optimal solution (the average standard deviation is 0.089) on average in 8 seconds by processing around 0.015% of the search space.

4.3. Comparative Analysis. To assess the performance of the Hybrid Firefly selection algorithm we have compared it with a Bee-inspired selection algorithm we previously introduced in [2]. The two selection algorithms have been comparatively evaluated according to the following criteria: the average number of processed solutions, the average simulation time, and the standard deviation of the score of the best solution returned by the algorithm related to the score of the global optimal composition solution for the considered scenario.

For each algorithm we performed the same number of simulations on the same scenario and on the same machine. In Table 4.3 we summarize the experimental results obtained for the two selection algorithms taking into consideration the optimal values of their adjustable parameters.

By analyzing the experimental results we conclude that the Hybrid Firefly algorithm outperforms the Bee-inspired algorithm in terms of number of processed solutions and executon time, but in terms of standard deviation it provides slightly higher values which are acceptable.

Table 4.3: Comparative analysis between the Hybrid Firefly Selection Algorithm and the Bee-inspired Selection Algorithm

Method	Average number of processed solutions	Average execution time	Average standard deviation
Hybrid Firefly	2145	9	0.007
Bee-inspired	2867	13	0.002

5. Conclusions and Future Work. This paper proposed a firefly-inspired method for selecting the optimal or a near optimal solution in semantic Web service composition. The selection method has been applied on Enhanced Planning Graph which encodes the set of composition solutions for a given user request. By combining the firefly-based selection approach with genetic operators we ensure a good balance between exploration and exploitation thus avoiding the problem of stagnation in a local optimum. To demonstrate the feasibility of our approach, we have implemented an experimental prototype which has been tested on a scenario from the trip planning domain. To assess the performance of the Hybrid Firefly method we have compared it with a Bee-inspired selection method.

REFERENCES

[1] L. N. Castro and F. von Zuben, *Learning and Optimization using the Clonal Selection Principle*, IEEE Transactions on Evolutionary Computation, Volume 6, Issue 3, pp. 239-251, 2002.

[2] V. R. Chifu, C. B. Pop, I. Salomie, M. Dinsoreanu, A. N. Niculici and D. S. Suia, *Selecting the Optimal Web Service Composition based on a Multi-criteria Bee-inspired Method*, Proceedings of the 12th International Conference on Information Integration and Web-based Applications & Services, pp. 40-47, 2010.

[3] M. Dorigo, M. Birattari, and T. Stutzle, *Ant Colony Optimization - Artificial Ants as a Computational Intelligence Technique*, IEEE Computational Intelligence Magazine, 1(4), pp. 28-39, 2006.

[4] X. Fan and X. Fang, *On Optimal Decision for QoS-Aware Composite Service Selection*, Information Technology Journal, Volume 9, Issue 6, pp. 1207-1211, 2010.

[5] J. Kennedy, and R. Eberhart, *Particle swarm optimization*, Proceedings of the IEEE International Conference on Neural Networks, pp. 1942-1948, 1995.

[6] S. Lukasik and S. Zak, *Firefly Algorithm for Continuous Constrained Optimization Tasks*, Computational Collective Intelligence. Semantic Web, Social Networks and Multiagent Systems, LNCS vol. 5796, pp. 97-106, 2009.

[7] C. B. Pop,V. R. Chifu, I. Salomie, and M. Dinsoreanu, *Immune-Inspired Method for Selecting the Optimal Solution in Web Service Composition*, LNCS, vol. 6162, pp. 1-17, 2010.

[8] J. Wang and Y. Hou, *Optimal Web Service Selection based on Multi-Objective Genetic Algorithm*, Proceedings of the International Symposium on Computational Intelligence and Design - Volume 01, pp. 553-556, 2008.

[9] J. Xu and S. Reiff-Marganiec, *Towards Heuristic Web Services Composition Using Immune Algorithm*, Proceedings of the International Conference on Web Services, pp. 238-245, 2008.

[10] X.S. Yang, *Nature-Inspired Metaheuristic Algorithms*, Luniver Press, 2008.
[11] W. Zhang, C. K. Chang, T. Feng and H. Jiang, *QoS-based Dynamic Web Service Composition with Ant Colony Optimization*, Proceedings of the 34th Annual IEEE Computer Software and Applications Conference, pp. 493-502, 2010.

Edited by: Dana Petcu and Jose Luis Vazquez-Poletti
Received: August 1, 2011
Accepted: August 31, 2011

Scalable Computing: Practice and Experience
Volume 12, Number 3, pp. 371–374. http://www.scpe.org
ISSN 1895-1767

ABSTRACTION LAYER FOR CLOUD COMPUTING*

BINH MINH NGUYEN, VIET TRAN AND LADISLAV HLUCHY†

Abstract. In this paper, we will present an abstraction layer for cloud computing. The abstraction layer will allow users to manipulate virtual machines as objects and simplify the process of porting applications to cloud computing. This approach could improve the flexibility of cloud computing, enable interoperability and simplify the creation of more complex systems in the cloud.

Key words: abstraction, cloud computing, object-oriented programming

1. Introduction. In the recent years, cloud computing has become an attractive option for scientific communities as well for industry. The illusion of unlimited resources that are available immediately at users' request, as well as the flexibility, elasticity, reliability that cloud computing offers, are really interesting for short-term testing and development, as well as for long-term flexible infrastructures. More and more institutions and companies start to build private clouds as well as use public cloud offered by large providers.

As present, there are several large providers including Amazon, Microsoft, ElasticHosts, and so on. There are also open-source cloud middleware for building clouds like Eucalyptus [1], OpenNebula [2], as well as proprietary cloud software from VMWare, Citrix, IBM, and so on. Unfortunately, these softwares are often incompatible from each other, that can increase the cost of porting applications to clouds as well as create potential vendor lock-in. There are efforts to standardize cloud middleware, mostly notable by Open Grid Forum [7] with OCCI [4] (Open Cloud Computing Interface).

In this paper, we present an abstraction layer for cloud computing. The abstraction layer could simplify the creation and use of virtual machines in cloud, and also make interoperability between providers from the view of users. The abstraction also enables opportunities for creating optimization layer (substituting, brokering) between the abstraction layer and cloud middleware. At the moment, we primarily focus on IaaS (Infrastructure as a Service) type of cloud computing, i.e. like Amazon EC2 [3], Microsoft Azure [6], OpenStack [5] and OCCI.

2. Overview of cloud interfaces. Basically, cloud interfaces can be divided into three categories: graphical portals, command line clients and web services. Most of cloud middlewares offer two or all three kinds of interfaces.

Command line clients are the most frequent clients and that offer every middleware. Basically, they offer command for image management (upload/download images) and virtual machine management (create, monitor, shutdown). For example for Amazon EC2 command line client [10], a command *"ec2-run-instances ami-12345678 -k ec2-keypair"* will create a virtual machine with image with ID *ami-12345678* and keypair named *ec2-keypair*. The command will return the machine ID that users can use for late manipulation with the machine. If users want to connect to the machine, they first use command *"ec2-describe-instance"* to find the public IP address (or public machine name) of the virtual machine, then use SSH command to connect to the machine using the IP address and the private key from the used key pair. Other middlewares like OpenNebula or OpenStack have similar command line clients.

The graphical portals offer web-based graphical interfaces where users can choose parameters for each virtual machine (types of machines, images, key pairs, network configuration), and control these virtual machines (start, shutdown, monitor their status). These interfaces can also offer additional actions specific for each middleware, e.g. attaching additional virtual disk, making images/snapshots of the running machines, migration, replication and so on. The portals are suitable for learning and deploying simple systems, but for more complex actions, they may require users to make a lot of mouse clicking. Web service interfaces are primarily used by developers. For example, for creating a virtual machine via Amazon EC2 web service [8], users can send a request to the web service with action and parameters like *"https://ec2.amazonaws.com/? Action=RunInstances&ImageId=ami-12345678"*. OGF [7] also define OCCI as RESTful web service [9], where parameters of actions are defined in XML files. Command line clients practically use the web services for communication with servers.

*This work is supported by projects DMM VMSP-P-0048-09, SMART II ITMS: 26240120029, VEGA 2/0184/10, VEGA No. 2/0211/09.

†Institute of Informatics, SAS, Dubravska cesta 9, Bratislava, Slovakia (minh.ui@savba.sk, viet.ui@savba.sk, hluchy.ui@savba.sk).

3. Design of abstraction layer. In this paper, we will present an abstraction layer for cloud computing. The aims of the abstraction layer are as follows:

- Abstraction of cloud resources: resources in the clouds (virtual machines, images, storages) are abstracted as objects and users can manipulate with them via methods provided by the objects. The abstraction will allow changes in the backend without affecting functionalities and modification of developed applications in the abstraction layer.
- Abstraction of complex systems: via mechanism like inheritance, composition and polymorphisms, developers can make abstraction of more complex systems with several components easily, and deploy them with single click.
- Simplification of use interface: Users can manipulate resources as objects without dealing with implementation details.
- Interoperability: Applications and user scripts developed in the abstraction layer will work for different cloud middleware from different providers.
- Optimization: The abstraction layer will allow optimization mechanisms like brokering, substitutions, load balancing and so on. For example, when the user create a new virtual machine, the optimization layer can choose which provider is best for the current instance.

In our design, we use object-oriented approach for abstraction of computing resources:

- The resource is represented as an object where all information related to the resource is encapsulated as data member of the object.
- Manipulation with the resource will be done via member methods of the object. Assume that a virtual machine in the cloud is represented by an object vm, then starting the machine is done by vm.start(), uploading data/application code to the machine is done by vm.upload(data, destination), execution of a program on the machine is done by vm.exec(command-line), and so on.
- Users can concretize and add more details to resource description using derived class and inheritance in OOP. For example, a Cluster class is used for representation of generic cluster, a derived class HadoopCluster can be used for abstraction of cluster with Hadoop software installed.
- Default values will be used whenever possible for quick learning. Sometime, the users just want to create a virtual machine for running their applications, they do not care about concrete Linux flavor, or which key pair should be used. The interface should not force users to specify every parameter even if the users do not care about it.
- Abstraction also makes space for resource optimization. The optimization layer can decide which options are best for users.

4. Examples.

4.1. Running applications on Clouds. We started with a simple example how to create a virtual machine in the cloud and execute an application on the newly created machine. The commands look as follows:

```
t = Instance()
t.start()
t.upload("appdata.dat appcode.exe","")
t.run("appcode.exe -I appdata.dat -o result.dat")
t.download("result.dat", "")
t.shutdown()
t.delete()
```

As Python is a scripting language, users can choose if they will execute the commands one by one in interactive mode of Python shell, or create a script from the commands. The command in the first line will create an instance (a virtual machine) with default parameters (defined in configuration files or in the *defaultInstance* variable). The users can customize the instance by adding parameters e.g. *t = Instance(type=large)* or even more complex *t = Instance (type=medium, image=myImage, keypair=myKeypair)*. If the users want to create more instances with the similar parameters, they can set common parameters to *defaultInstance*. Note that the instance is created without starting, so users can change parameters, e.g. *t.setKeypair(public, private)*.

The next commands in the example will start the virtual machine, upload the application and its data, execute the application on the virtual machine, download output data and terminate the machines. Users can get information about the virtual machines simply by command *print t*. The information given by the command is similar to the *xxx-describe-instance* in Amazon EC2 or Eucalyptus. As it is shown in the example above, users

do not have to deal with detailed information like IP address, SSH commands connection to the virtual machines and so on. They simply upload data, run application or download data with simple, intuitive command like *t.upload(), t.run(), t.download()* and so on. Of course, if the users really need to run its own SSH connection, they can do it by information (IP address, SSH key) from the *print t* command.

Now we can go further in abstraction by creating a function *runapp(inputdata, commandline, outputdata, type=medium)* from the commands. From this point, users can execute an application in the cloud only with single command *runapp()* with input/output data and command line as parameters.

Note that the abstraction (like *Instance* class or *runapp()* command) does not only simplify the process of using cloud computing, but also allows experts (e.g. IT support staff of institutes/companies) to do optimization for users. The actual users of Cloud computing do not have to be IT professionals but may be scientists, researchers, experts from other branches. For example, the IT staff can customize the virtual machines creation by checking if there is free capacity in the private clouds first before going to public clouds.

4.2. Abstract object for clusters. In this section, we will demonstrate how to create a new abstract object within our abstraction layer. We will define a new object for abstraction of clusters. The initialization of the cluster look as follows:

```
class Cluster:
    def _init_()
        head = Instance()
        for i in rang (1, N)
           worker[i] = Instance()

    def start()
        head.start()
        for i in rang (1, N)
            worker[i].start()
        config()

    def config()
        nodefile = nodefile + worker[i].privateIP
        f = open('/tmp/nodefile', 'w')
        f.write(nodefile)
        f.close()
        head.upload('/tmp/nodefile', '');

    def upload(source, dest)
        head.upload(source, dest);
```

After defining the cluster object, users then can create a cluster and use them as follows:

```
t = Cluster()
t.start()
t.upload()
```

Of course, we can define derived classes from Cluster, e.g. PBSCluster, HadoopCluster, by modifying the configuration method in the Cluster class config(). We can go further by create a abstract object ElasticPBSCluster, a cluster with PBS where worker nodes are dynamically added and removed according to the actual loads of the PBS.

5. Conclusion. In the paper, we have presented an approach for abstraction of cloud computing. The abstraction layer could improve the flexibility of cloud computing, enable interoperability and simplify the creation of more complex systems in the cloud.

REFERENCES

[1] Daniel Nurmi, Rich Wolski, Chris Grzegorczyk, Graziano Obertelli, Sunil Soman, Lamia Youseff, Dmitrii Zagorodnov, *The Eucalyptus Open-Source Cloud-Computing System*. In Proceedings of the 2009 9th IEEE/ACM International Symposium on Cluster Computing and the Grid (CCGRID '09) pp. 124-131, IEEE Computer Society, Washington, DC, USA.

[2] Dejan Milojii, Ignacio M. Llorente, Ruben S. Montero, *OpenNebula: A Cloud Management Tool*. IEEE Internet Computing, vol. 15, no. 2, pp. 11-14, 2011.
[3] *Amazon Elastic Compute Cloud (Amazon EC2)*. http://aws.amazon.com/ec2/. Last visited on August 2011.
[4] *Open Cloud Computing Interface*. http://occi-wg.org/. Last visited on August 2011.
[5] *OpenStack: Open source cloud computing software*. http://openstack.org/. Last visited on August 2011.
[6] *Windows Azure*. http://www.microsoft.com/windowsazure/. Last visited on August 2011.
[7] *Open Grid Forum*. http://ogf.org/. Last visited on August 2011.
[8] *Amazon Elastic Compute Cloud API Reference*. http://docs.amazonwebservices.com/AWSEC2/2011-07-15/APIReference/. Last visited on August 2011.
[9] *Open Cloud Computing Interface - Infrastructure*. http://www.gridforum.org/PublicCommentDocs/Documents/2010-12/ogf_draft_occi_infrastructure.pdf. Last visited on August 2011.
[10] *Amazon Elastic Compute Cloud Command Line Reference*. http://docs.amazonwebservices.com/AWSEC2/latest/CommandLineReference/. Last visited on August 2011.

Edited by: Dana Petcu and Jose Luis Vazquez-Poletti
Received: August 1, 2011
Accepted: August 31, 2011

AIMS AND SCOPE

The area of scalable computing has matured and reached a point where new issues and trends require a professional forum. SCPE will provide this avenue by publishing original refereed papers that address the present as well as the future of parallel and distributed computing. The journal will focus on algorithm development, implementation and execution on real-world parallel architectures, and application of parallel and distributed computing to the solution of real-life problems. Of particular interest are:

Expressiveness:

- high level languages,
- object oriented techniques,
- compiler technology for parallel computing,
- implementation techniques and their efficiency.

System engineering:

- programming environments,
- debugging tools,
- software libraries.

Performance:

- performance measurement: metrics, evaluation, visualization,
- performance improvement: resource allocation and scheduling, I/O, network throughput.

Applications:

- database,
- control systems,
- embedded systems,
- fault tolerance,
- industrial and business,
- real-time,
- scientific computing,
- visualization.

Future:

- limitations of current approaches,
- engineering trends and their consequences,
- novel parallel architectures.

Taking into account the extremely rapid pace of changes in the field SCPE is committed to fast turnaround of papers and a short publication time of accepted papers.

INSTRUCTIONS FOR CONTRIBUTORS

Proposals of Special Issues should be submitted to the editor-in-chief.

The language of the journal is English. SCPE publishes three categories of papers: overview papers, research papers and short communications. Electronic submissions are preferred. Overview papers and short communications should be submitted to the editor-in-chief. Research papers should be submitted to the editor whose research interests match the subject of the paper most closely. The list of editors' research interests can be found at the journal WWW site (`http://www.scpe.org`). Each paper appropriate to the journal will be refereed by a minimum of two referees.

There is no a priori limit on the length of overview papers. Research papers should be limited to approximately 20 pages, while short communications should not exceed 5 pages. A 50–100 word abstract should be included.

Upon acceptance the authors will be asked to transfer copyright of the article to the publisher. The authors will be required to prepare the text in LaTeX 2$_\varepsilon$ using the journal document class file (based on the SIAM's `siamltex.clo` document class, available at the journal WWW site). Figures must be prepared in encapsulated PostScript and appropriately incorporated into the text. The bibliography should be formatted using the SIAM convention. Detailed instructions for the Authors are available on the SCPE WWW site at `http://www.scpe.org`.

Contributions are accepted for review on the understanding that the same work has not been published and that it is not being considered for publication elsewhere. Technical reports can be submitted. Substantially revised versions of papers published in not easily accessible conference proceedings can also be submitted. The editor-in-chief should be notified at the time of submission and the author is responsible for obtaining the necessary copyright releases for all copyrighted material.