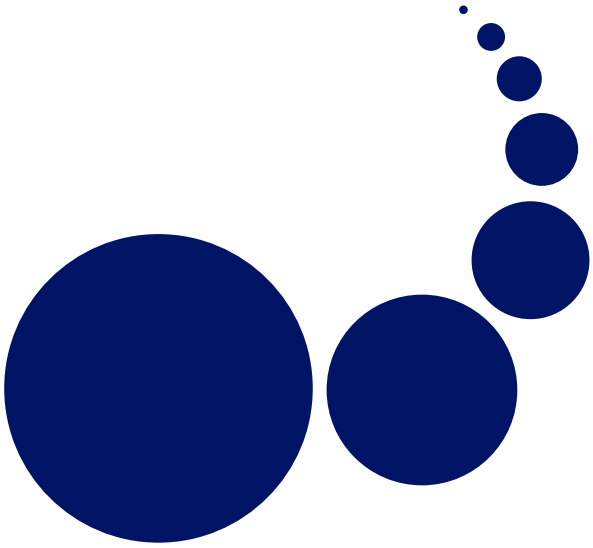
# SCALABLE COMPUTING
## Practice and Experience

## Special Issue: Selected Papers From 1st Workshop on Software Services

Editors: Dana Petcu and Alex Galis



Volume 12, Number 1, March 2011

ISSN 1895-1767

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

Scalable Computing: Practice and Experience

Volume 12, Number 1, March 2011

## TABLE OF CONTENTS

Selected Papers From 1st Workshop on Software Services:

Scalable Computing: Practice and Experience
Volume 12, Number 1, pp. iii–iv. http://www.scpe.org
ISSN 1895-1767


# INTRODUCTION TO THE SPECIAL ISSUE ON SELECTED PAPERS FROM 1ST WORKSHOP ON SOFTWARE SERVICES

Dear SCPE readers,

The series of Workshops on Software Services (acronym WoSS) was initiated in 2010 with the goal to create a framework for presenting and discussing the recent significant developments in the field of software services. Already the first edition WoSS proved to be an open forum for academics, practitioners, and vendors, allowing them to analyze the current trends and scientific and technological challenges, such as service quality assurance, adaptability, reliability, interoperability, and automating service-oriented application construction and management. A particularity of three editions of the series is the aim to highlight the achievements of the on-going European collaborative projects subscribing to European Commission FP7-ICT programme and the activities of the teams from new member states in software services.

The first edition of WoSS from September 2010, dedicated to Frameworks and Platforms, has attracted 31 contributions, part of them being accepted for presentation at the workshop and later on publication in the IEEE Computer Press proceedings of the SYNASC 2010 symposium.

This special issue presents the extended versions of 10 selected papers. The new papers are including extensive description of their contributions as well as novel approaches and results.

As an entire, the special issue intends to present an image of the current status of the research activities in the field of software services. Therefore the subjects are varying from workflow to resource management, from data migration to access control, from quality of services to public services, from file searching to trust, from surveillance applications to automated service compositions.

The last paper of this issue is part of the batch of papers presented in the previous number of SCPE 11(4), related to Network Management in Distributed Systems, but is presented in this number since its content matched better the aims of the current issue.

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

Alex Galis,
*University College London*
*Department of Electronic and Electrical Engineering*
*Torrington Place, London, United Kingdom*

Scalable Computing: Practice and Experience
Volume 12, Number 1, pp. 1–20. http://www.scpe.org
ISSN 1895-1767


# WORKFLOW AND ACCESS CONTROL RELOADED: A DECLARATIVE SPECIFICATION FRAMEWORK FOR THE AUTOMATED ANALYSIS OF WEB SERVICES

MICHELE BARLETTA*, ALBERTO CALVI†, SILVIO RANISE‡, LUCA VIGANÒ§, AND LUCA ZANETTI¶

**Abstract.** Web services supporting business and administrative transactions between several parties over the Internet are more and more widespread. Their development involves several security issues ranging from authentication to the management of the access to shared resources according to given business and legal models. The capability of validating designs against fast evolving requirements is of paramount importance for the adaptation of business and administrative models to changing regulations and rapidly evolving market needs. We present formal specification and analysis techniques that allow us to validate the designs of security-sensitive web services specified in the Business Process Execution Language and extensions of the Role-Based Access Control model. We also present a prototype tool, called WSSMT, mechanizing our approach and describe our experience in using it on two industrial case studies, on in the e-business and one in the e-government area.

**Key words:** Service-oriented architectures, Web services, Business Process Models, Workflow, Access Control, Policy Management, Formal Analysis

**AMS subject classifications.** 68Q60, 68Q85, 68N30, 68N01, 68M99

**1. Introduction.** Web services are becoming more and more widespread in many fields like e-commerce, e-health, and e-governance, where services support business and administrative transactions between several parties over the Internet. Their development involves several security issues ranging from authentication to the management of the access to shared resources according to given business and legal models. The capability of validating designs against fast evolving requirements is of paramount importance for the adaptation of business and administrative models to changing regulations and rapidly evolving market needs. So, techniques for the specification and automated analysis of dependable web services to be used in security-sensitive applications are crucial in the development of these systems.

To design practically usable specification and analysis techniques, it is important to have a closer look at the definition of web service. A web service is a piece of software with a clearly defined set of interface functionalities that can be invoked according to a certain ordering specified by a *workflow* (WF). The *WF level* of a service establishes whether a certain operation can be executed if the values of the state variables (the *data flow*) of the service satisfy certain conditions (the *control flow*), and the values stored in the state variables may be updated. This situation is further complicated by the fact that one may create several instances of the same service: all the instances will share the same behavior but, at any given time, they may be in different states, i.e., distinct control locations and values of the state variables. Thus, (unique) identifiers are required to name the different particular instances. Several (executable) specification languages are available: the data part can be described by, e.g., the Web Service Definition Language WSDL [10], the control part by, e.g., the Business Process Execution Language BPEL [1], and the identifiers by Uniform Resource Identifiers.

An additional source of security problems is the fact that many deployed services work over the Internet where identities should be certified and trusted so as to enable the deployment of flexible access policies. In fact, one of the most relevant and hard-to-design parts of the security level of services is their *policy management* (PM) level. Policies specify, for instance, what operations a service is granted or denied the right to execute, are usually expressed in terms of a set of basic facts, and are combined to form certain access rules. The basic facts depend on the particular application domain and are usually encapsulated in certificates whose possession enables the application of access rules. Since certificates can be produced or revoked at different time points, PM is an essentially dynamic activity. So, the PM level should be able to inspect part of the state of the WF of the service and, in turn, operations performed at the WF level can update the basic facts used to specify policies, so that we have an interplay from the WF to the PM and vice versa.

A widespread design approach to control the delicate interplay between the WF and PM levels of a service consists of clearly separating them and identifying where and how they interact. This separation is beneficial in

*Dipartimento di Informatica, Università di Verona, Italy, michele.barletta@univr.it
†Dipartimento di Informatica, Università di Verona, Italy, alberto.calvi@univr.it
‡FBK-Irst, Trento, Italy, ranise@fbk.eu
§Dipartimento di Informatica, Università di Verona, Italy, luca.vigano@univr.it
¶Dipartimento di Informatica, Sistemistica e Telematica, Università di Genova & FBK-Irst, Trento, Italy, luca.zanetti@unige.it

several respects for the design and maintenance of services, and also for their validation. So, in order to design usable specification and analysis techniques, it is desirable to reflect the two level structure of a service design. In this paper, we explain how this can be achieved by a suitable instantiation of the formal framework in [7]. In particular, we show how the WF level of web services described by BPEL processes can be mapped to Petri nets and how this, in turn, can be translated to a class of transition systems involving integer variables by using well-known results (see, e.g., [20]). For the PM level, we explain how RBAC4BPEL, a variant of the Role-Based Access Control (RBAC [19]) model, can be easily expressed as a certain symbolic transition system by using a fragment of first-order logic (FOL). Then, we explain how to combine the two (symbolic) transition systems to obtain one system, abstractly describing the whole web service. This allows us to introduce a technique for the exploration of the state space that is based on a symbolic execution procedure and exploits the capability of solving logical problems in certain fragments of FOL.

We also discuss how to mechanize the symbolic execution procedure by describing the architecture of a prototype tool, called WSSMT, which we have developed to allow designers of web services to gain confidence in their designs. Since our framework reduces verification problems to logical (satisfiability) problems, it is possible to use off-the-shelf state-of-the-art automated reasoning systems to automatically discharge the proof obligations encoding the satisfiability problems. The predictability of the behavior of the automated provers on the generated proof obligations is obtained by constraining the class of formulae used to describe the WF and PM levels together with those describing the possible executions of the service. In this way, it is possible to reuse decidability results for fragments of first-order logic that are supported by state-of-the-art theorem provers. We illustrate the practical viability of our approach by considering the validation of two case studies inspired by industrial systems, which have been considered in the context of the FP7 European project AVANTSSAR [3].

We proceed as follows. In Section 2, we first provide some background, in particular, on BPEL, Petri nets, and RBAC4BPEL, and then introduce a running example. In Section 3, we introduce our formal two-level specification framework, in particular, we discuss two-level transition systems and their symbolic execution. In Section 4, we explain how security-sensitive services specified by (a sub-set of) BPEL and RBAC4BPEL can be specified as two-level transition systems. In Section 5, we present our tool WSSMT and then, in Section 6, we report on the application of WSSMT to the specification and analysis of the two industrial-strength case studies we consider here. In Section 7, we draw some conclusions. Parts of the material included in this paper have appeared in preliminary form in [5, 9].

**2. Background and running example.** We describe the Business Process Execution Language (BPEL) [1] by using an example (a Purchase Ordering process, PO for short, taken from [16]) and explain how Petri nets can be used as an abstract semantics modelling the control flow of a security-sensitive service ignoring data and security issues. For a complete description of the semantics of BPEL and its relationship to Petri nets, the reader is pointed to, e.g., [25].

**2.1. BPEL and Petri nets.** In Fig. 2.1, we show a high-level BPEL specification of the WF level of the PO process. The `<process>` element wraps around the entire description of the PO process. The `<sequence>` element states that the activities contained in its scope must be executed sequentially. The `<flow>` element specifies concurrent threads of activities. The `<invoke>` element represents the invocation of an activity that is provided by an available web service. Finally, the `<receive>` element represents the invocation of an activity that is provided by the BPEL process being described. Indeed, BPEL provides a variety of constructs (e.g., to represent variables) that are ignored here for simplicity; the interested reader is pointed to [1]. In the case of the PO process, it is easy to see that the constraints on the execution described above are all satisfied by the nesting of control elements in Fig. 2.1. For example, because of the semantics of `<sequence>`, *crtPO* will be executed first while *apprPay* will be the activity finishing the PO process.

Fig. 2.1 shows also, on the right, a Petri net that can be seen as a (control-flow) abstraction of the BPEL process on the left of the figure. In order to sketch the mapping from BPEL processes to Petri nets, we first recall the basic notions concerning the latter.

A *Petri net* is a triple $\langle P, T, F\rangle$, where $P$ is a finite set of *places*, $T$ is a finite set of *transitions*, and $F$ (*flow relation*) is a set of arcs such that $P \cap T = \emptyset$ and $F \subseteq (P \times T) \cup (T \times P)$. Graphically, the Petri net $\langle P, T, F\rangle$ can be depicted as a directed bipartite graph with two types of nodes, places and transitions, represented by circles and rectangles, respectively; the nodes are connected via directed arcs according to $F$ (where arcs between two nodes of the same type are not allowed). A place $p$ is called an *input* (resp., *output*) place of a transition $t$ iff there exists a directed arc from $p$ to $t$ (resp., from $t$ to $p$). The set of input (resp., output) places of a transition

```
<process name="PO"/>
  <sequence>
    <receive operation="crtPO" ... > </receive>
    <invoke  operation="apprPO" ... > </invoke>
    <flow>
      <sequence>
        <invoke operation="signGRN" ... > </invoke>
        <invoke operation="ctrSignGRN" ... > </invoke>
      </sequence>
      <invoke operation="ctrPay" ... > </invoke>
    </flow>
    <invoke operation="apprPay" ... > </invoke>
  </sequence>
</process>
```

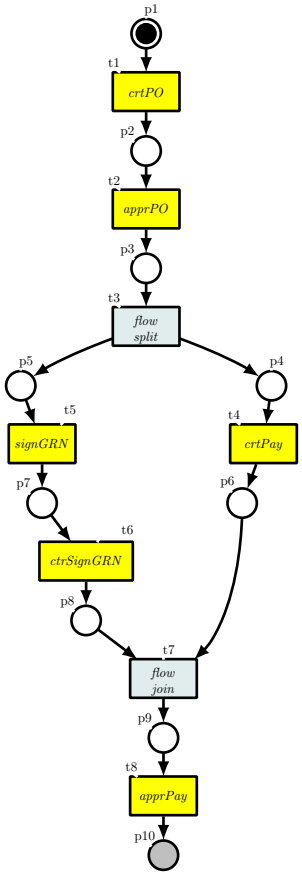



FIG. 2.1. *The PO process: BPEL (left) and corresponding Petri net (right). Legend: yellow transitions specify normal tasks, azure ones specify flow transitions (splits and joins) and grey places represents final places.*

$t$ is denoted by $\bullet t$ (resp., $t\bullet$); $\bullet p$ and $p\bullet$ are defined similarly. A *path* in a Petri net $\langle P, T, F\rangle$ is a finite sequence $e_0, ..., e_n$ of elements from $P \cup T$ such that $e_{i+1} \in e_i\bullet$ for each $i = 0, ..., n-1$; a path $e_0, ..., e_n$ in the net is a *cycle* if no element occurs more than once in it and $e_0 \in e_n\bullet$ for some $n \geq 1$. A Petri net is *acyclic* if none of its paths is a cycle. A *marking* of a Petri net $\langle P, T, F\rangle$ is a mapping from the set $P$ of places to the set of non-negative integers; graphically, it is depicted as a distribution of black dots in the circles of the graph representing the net. A transition $t$ is *enabled* in a marking $m$ iff each of its input places $p$ is such that $m(p) \geq 1$, i.e., $p$ contains at least one token. An enabled transition $t$ in a marking $m$ may *fire* by generating a new marking $m'$, in symbols $m \xrightarrow{t} m'$, where $m'(p) = m(p)$ if $p \notin (\bullet t \cup t\bullet)$, $m'(p) = m(p) - 1$ if $p \in \bullet t$, and $m'(p) = m(p) + 1$ if $p \in t\bullet$, i.e. $t$ consumes one token from each input place of $t$ and produces one token in each of its output places. A marking $m$ is *reachable* from $m_0$, in symbols $m_0 \rightarrow^* m$, iff there exists a sequence $m_1, ..., m_n$ of markings such that $m_i \xrightarrow{t} m_{i+1}$ for $i = 0, ..., n-1$ and $m_n = m$, for some $n \geq 0$. (For $n = 0$, we have that $m_0 = m$.) Given a Petri net $\langle P, T, F\rangle$ and a marking $m$, an instance of the *reachable problem* for Petri nets consists of checking whether $m_0 \rightarrow^* m$ or not. A *workflow (WF) net* [28] is a Petri net $\langle P, T, F\rangle$ such that (i) there exist two special places $i, o \in P$ with $\bullet i = \emptyset$ and $o\bullet = \emptyset$; and (ii) for each transition $t \in T$, there exists a path $\pi$ in the net beginning with $i$ and ending with $o$ in which $t$ occurs.

The idea underlying the Petri net semantics of BPEL is simple. Activities are mapped to transitions (the rectangles in Fig. 2.1) and their execution is modeled by the flow of tokens from input places to output places. When two BPEL operations are enclosed in a `<sequence>` element (e.g., *crtPO* and *apprPO*), two transitions are created (as in Fig. 2.1) with one input place (resp., $p_1$ and $p_2$) and one output place each (resp., $p_2$ and $p_3$), and the input place of the second is identified with the output place of the first one ($p_2$). When two BPEL operations are in a `<flow>` element (e.g., *crtPay* and the sequence of operations *signGRN* and *ctrSignGRN*),

$$
\begin{array}{rl}
U & := \{u_1, u_2, u_3, u_4, u_5\} \\
R & := \{\mathit{manager}, \mathit{finAdmin}, \mathit{finClerk}, \mathit{poAdmin}, \mathit{poClerk}\} \\
P & := \{p_1, ..., p_5\} \\
\mathit{ua} & := \{(u_1, \mathit{manager}), (u_2, \mathit{finAdmin}), (u_3, \mathit{finClerk}), (u_4, \mathit{poAdmin}), \quad (u_5, \mathit{poClerk})\} \\
\mathit{pa} & := \{(\mathit{finClerk}, p_4), (\mathit{finAdmin}, p_5), (\mathit{poClerk}, p_3), (\mathit{poAdmin}, p_1)\} \\
\succeq & \text{ least partial order s.t. } \mathit{manager} \succeq \mathit{finAdmin},\ \mathit{manager} \succeq \mathit{poAdmin}, \\
& \qquad\qquad \mathit{finAdmin} \succeq \mathit{finClerk}, \text{ and } \mathit{poAdmin} \succeq \mathit{poClerk}.
\end{array}
$$

FIG. 2.2. *The PM level of the PO process.*

four transitions are created: one to represent the split of the flow, one to represent its synchronization (join), and one for each activity that can be executed concurrently with the appropriate places to connect them (in Fig. 2.1, when a token is in place $p_3$, the *flow split* transition is enabled and its execution yields one token in place $p_4$ and one in $p_5$, which enables both transitions *signGRN* and *crtPay* that can be executed concurrently; the two independent threads of activities get synchronized again by the execution of *flow join*, which is enabled when a token is in $p_6$ and a token is in $p_8$). It is not difficult to see that the Petri net of Fig. 2.1 is an acyclic WF net where $p_1$ is the special input place $i$, $p_{10}$ is the special output place $o$, and each transition occurs in a path from $p_1$ to $p_{10}$.

**2.2. RBAC4BPEL.** In [16], Paci, Bertino and Crampton use RBAC4BPEL, an extension of the RBAC framework [19] adapted to work smoothly with BPEL, to specify the PM level of applications like the PO process. The components of RBAC4BPEL are: (i) a set $U$ of users, (ii) a set $R$ of roles, (iii) a set $P$ of permissions, (iv) a role hierarchy $\succeq$ (i.e. a partial-order relation on $R$), (v) a user-role assignment relation $ua$, (vi) a role-permission assignment relation $pa$, and (vii) a set $A$ of activities and a class of authorization constraints (such as separation-of-duty) to prevent some user to acquire permissions in certain executions of the application (see below for details). Note that components (i)–(vi) are standard in RBAC whereas (vii) has been added to obtain a better integration between the PM and the WF levels.

First, we describe components (i)–(vi) and some related notions. A user $u \in U$ is assigned a role $r \in R$ if $(u, r) \in ua$ and permissions are associated with roles when $(p, r) \in pa$. In RBAC4BPEL, a user $u \in U$ has a permission $p$ if there exists a role $r \in R$ such that $(u, r) \in ua$ and $(p, r) \in pa$. (We will see that each permission is associated to a right on a certain activity in $A$ of a BPEL process, e.g., its execution.) The role hierarchy $\succeq \subseteq R \times R$ is assumed to be a partial order (i.e., a reflexive, antisymmetric, and transitive relation) reflecting the rights associated to roles. More precisely, a user $u$ is an *explicit* member of role $r \in R$ if $(u, r) \in ua$ and it is an *implicit* member of role $r \in R$ if there exists a role $r' \in R$ such that $(r', r) \in \succeq$ (abbreviated as $r' \succeq r$), $r' \neq r$, and $(u, r') \in ua$. Thus, $\succeq$ induces a permission inheritance relation as follows: a user $u \in U$ *can get* permission $p$ if there exists a role $r \in R$ such that $u$ is a member (either implicit or explicit) of $r$ and $(p, r) \in pa$. For simplicity, we abstract away the definition of a role in terms of a set of attributes as done in [16].

Fig. 2.2 shows the sets $U, R, P$ and the relations $ua$, $pa$, and $\succeq$ for the PM level of the PO process. Although $(\mathit{manager}, p_i) \notin pa$ for any $i = 1, ..., 5$, we have that user $u_1$, which is explicitly assigned to role *manager* in $ua$, can get any permission $p_i$ for $i = 2, ..., 5$ as $\mathit{manager} \succeq r$ for any role $r \in R \setminus \{\mathit{manager}\}$, hence $u_1$ can be implicitly assigned to each role and then get the permission $p_i$.

In RBAC4BPEL, each permission in $P$ is associated with the right to handle a certain transition of $T$, uniquely identified by a label in $A$, for a Petri net $\langle P, T, F\rangle$. In many cases, this is particularly simple since only the right to execute a transition is considered as it is the case in the services considered in this paper. We bind permissions $p_i$ to different tasks as follows: $p_1$ is the permission for executing $\mathit{apprPO}$, $p_2$ for *signGRN*, $p_3$ for *ctrSignGRN*, $p_4$ for *crtPay*, and $p_5$ for $\mathit{apprPay}$. We are now in the position to describe component (vii) of RBAC4BPEL. Note that there are no permissions associated to *flow split* and *flow join* as these are performed by the BPEL engine and thus no particular authorization restriction must be enforced.

A *role* (resp., *user*) *authorization constraint* is a tuple $\langle D, (t_1, t_2), \rho\rangle$ if $D \subseteq R$ (resp., $D \subseteq U$) is the domain of the constraint, $\rho \subseteq R \times R$ (resp., $\rho \subseteq U \times U$), and $t_1, t_2$ are in $A$. An authorization constraint $\langle D, (t_1, t_2), \rho\rangle$ is *satisfied* if $(x, y) \in \rho$ when $x, y \in D$, $x$ performs $t_1$, and $y$ performs $t_2$. In other words, authorization constraints place further restrictions (besides those of the standard RBAC components) on the roles or users who can perform certain actions once others have been already executed by users belonging to certain roles.

Constraints of this kind allow one to specify *separation-of-duty (SoD)* by $\langle D, (t_1, t_2), \neq \rangle$, *binding-of-duty (BoD)* by $\langle D, (t_1, t_2), = \rangle$, or any other restrictions that can be specified by a binary relation over roles or users.

For the PO process, (vii) of RBAC4BPEL is instantiated as:

$$\langle U, (apprPO, signGRN), \neq \rangle,\ \langle U, (apprPO, ctrSignGRN), \neq \rangle,$$
$$\langle U, (signGRN, ctrSignGRN), \neq \rangle,\ \langle R, (crtPay, apprPay), \prec \rangle,$$

where $\prec := \{(r_1, r_2) \mid r_1, r_2 \in R,\ r_2 \succeq r_1,\ r_1 \neq r_2\}$ (recall that the sets $U$ and $R$ are defined in Fig. 2.2).

**3. A formal two-level specification framework.** Let us now consider the structure of the PO process introduced in Section 2. We can regard it as structured in two levels: the *WF level* dealing with the control of the flow (and the manipulation of data) and the *PM level* describing access control rules (and trust relationships). Each level is further structured in a *static* and a *dynamic* part; the former specifies the data structures manipulated by the service for the WF level or the relational structure used for the PM level, e.g., the user-role assignment relation of RBAC system, while the latter describes the possible executions of the system, e.g., how a certain integer variable storing the number of clients being served for the WF level or how a tuple is added to or deleted from a relation in a database for the PM level.

All the four components of our framework (static/dynamic parts of the WF/PM levels) are symbolically represented by formulae of many-sorted FOL with equality (see, e.g., [12] for definitions of the basic notions that we use in the following of the paper), which is a well-studied logic that comes equipped both with a rich catalogue of decidable fragments (i.e., classes of formulae for which there exist algorithms capable of solving their satisfiability problems) and with several well-engineered automated theorem provers to support mechanical reasoning in the logic or its fragments. The work in [6] has introduced a declarative framework that permits one to specify the static and dynamic parts of the WF and PM levels, and then to reduce interesting verification problems to satisfiability problems in decidable fragments. This paves the way to building push-button validation techniques for security-sensitive service applications as demonstrated by our prototype tool WSSMT, which is described in Section 5.

In this section, we briefly recall the main notions of the framework in [6] for specifying security-sensitive services composed of the WF and PM levels. Formally, these two levels are specified by a particular class of symbolic guarded assignment systems, called *two-level transition systems*, where first-order formulae are used to represent sets of states (i.e. the static part) and the actions (i.e. the dynamic part) of the system. More precisely, for this class of transition systems, the state variables are updated by applying a function to the actual values of the variables provided that a guard (expressed as a condition again on the values of the state variables) is satisfied. The state space of such transition systems can be explored by using a symbolic execution based on satisfiability solving of a class of FOL formulae. In the rest of this section, we recall the notion of two-level transition systems (Section 3.1) and then describe the symbolic execution technique (Section 3.2).

**3.1. Two-level transition systems.** A *two-level transition system Tr* is a tuple

$$\langle \underline{x}, \underline{p}, In(\underline{x}, \underline{p}), \{\tau_i(\underline{x}, \underline{p}, \underline{x}', \underline{p}') \mid i = 1, ..., n\} \rangle,$$

where $\underline{x}$ is a tuple of *WF state* variables; $\underline{p}$ is a tuple of *PM state* variables; the *initial condition* $In(\underline{x}, \underline{p})$ is a FOL formula whose only free variables are in $\underline{x}$ and where PM state variables in $\underline{p}$ may occur as predicate symbols; and for $i = 1, ..., n$ and $n \geq 1$ the *transition* $\tau_i(\underline{x}, \underline{p}, \underline{x}', \underline{p}')$ is a FOL formula whose only free variables are in $\underline{x}, \underline{x}'$ and where PM state variables in $\underline{p}, \underline{p}'$ may occur as predicate symbols (as it is customary, unprimed variables in $\tau_i$ refer to the values of the state before the execution of the transition while those primed refer to the values of the state afterward).

We assume there exists a so-called first-order *underlying structure* $\langle D, I \rangle$ *of the transition system Tr*, where $D$ is the domain of values and $I$ is the mapping from the signature to functions and relations over $D$, and in which the state variables and the symbols of the signature used to write the formulae $In$ and $\tau_i$ for $i = 1, ..., n$ are mapped. A *state* of *Tr* is a pair $v := (v_{\underline{x}}, v_{\underline{p}})$ of mappings: $v_{\underline{x}}$ from the WF state variables to $D$ and $v_{\underline{p}}$ from the PM state variables to relations over $D$.

A *run* of *Tr* is a (possibly infinite) sequence of states $v^0, v^1, ..., v^n, ...$ such that (i) $v^0$ satisfies $In$, in symbols $v^0 \models In$, and (ii) for every pair $v^i, v^{i+1}$ in the sequence, there exists $j \in \{1, ..., n\}$ such that $v^i, v^{i+1}$ satisfies $\tau_j$, in symbols $v^i, v^{i+1} \models \tau_j$, where the domain of $v^i$ is $\underline{x}, \underline{p}$ and that of $v^{i+1}$ is $\underline{x}', \underline{p}'$. Given a formula $G(\underline{x}, \underline{p})$,

called the *goal*, an instance of the *goal reachability problem* for $Tr$ consists of answering the following question: does there exist a natural number $\ell \geq 0$ such that the formula

$$In(\underline{x}_0, \underline{p}_0) \wedge \bigwedge_{i=o}^{\ell-1} \tau(\underline{x}_i, \underline{p}_i, \underline{x}_{i+1}, \underline{p}_{i+1}) \wedge G(\underline{x}_\ell, \underline{p}_\ell) \tag{3.1}$$

is satisfiable in the underlying structure of $Tr$, where $\underline{x}_i, \underline{p}_i$ are renamed copies of the state variables in $\underline{x}, \underline{p}$? (When $\ell = 0$, (3.1) is simply $In(\underline{x}_0, \underline{p}_0) \wedge G(\underline{x}_0, \underline{p}_0)$). The interest of the goal reachability problem lies in the fact that many verification problems for two-level transition systems, such as invariant checking, can be reduced to it.

**3.2. Symbolic execution of two-level transition systems.** If we were able to check automatically the satisfiability of (3.1), an idea to solve the goal reachability problem for two-level transition systems would be to generate instances of (3.1) for increasing values of $\ell$. However, this would only give us a semi-decision procedure for the reachability problem. In fact, this method terminates only when the goal is reachable from the initial state, i.e., when the instance of (3.1) for a certain value of $\ell$ is unsatisfiable in the underlying structure of the transition system $Tr$. But, when the goal is not reachable, the check will never detect the unsatisfiability and we will be bound to generating an infinite sequence of instances of (3.1) for increasing values of $\ell$. That is, the decidability of the satisfiability of (3.1) in the underlying structure of $Tr$ is only a necessary condition for ensuring the decidability of the goal reachability problem.

We can formalize this method as follows. The *post-image* of a formula $K(\underline{x}, \underline{p})$ with respect to a transition $\tau_i$ is

$$Post(K, \tau_i) := \exists \underline{x}', \underline{p}'.(K(\underline{x}', \underline{p}') \wedge \tau_i(\underline{x}', \underline{p}', \underline{x}, \underline{p})).$$

For the class of transition systems that we consider below, we are always able to find FOL formulae that are equivalent to $Post(K, \tau_i)$. Thus, the use of the second-order quantifier over the predicate symbols in $\underline{p}'$ should not worry the reader (see Section 4.2 for details).

Now, define the following sequence of formulae by recursion: $FR^0(K, \tau) := K$ and $FR^{i+1}(K, \tau) := Post^i(FR^i, \tau) \vee FR^i(K, \tau)$, for $i \geq 0$ and $\tau := \bigvee_{k=1}^n \tau_i$. The formula $FR^\ell(K, In)$ describes the set of states of the transition system $Tr$ that are *forward reachable in* $\ell \geq 0$ *steps*.

A *fix-point* is the least value of $\ell$ such that $FR^{\ell+1}(\tau, In) \Rightarrow FR^\ell(\tau, In)$ is true in the structure underlying $Tr$. Note also that $FR^\ell(\tau, In) \Rightarrow FR^{\ell+1}(\tau, In)$ by construction and hence if $FR^{\ell+1}(\tau, In) \Rightarrow FR^\ell(\tau, In)$ is valid, then also $FR^\ell(\tau, In) \Leftrightarrow FR^{\ell+1}(\tau, In)$ is so and $FR^\ell(\tau, In) \Leftrightarrow FR^{\ell'}(\tau, In)$ for each $\ell' \geq \ell$.

Using the sequence of formulae $FR^0(\tau, In), FR^1(\tau, In), ...$ it is possible to check if the goal property $G$ will be reached by checking whether $FR^\ell(\tau, In) \wedge G$ is satisfiable in the structure underlying $Tr$ for some $\ell \geq 0$. In case of satisfiability, we say that $G$ is *reachable*. Otherwise, if $FR^\ell(\tau, In)$ is a fix-point, the unsatisfiability of $FR^\ell(\tau, In) \wedge G$ implies that $G$ is *unreachable*.

Finally, if $FR^\ell(\tau, In)$ is not a fix-point and $FR^\ell(\tau, In) \wedge G$ is unsatisfiable, then we must increase the value of $\ell$ by 1 so as to compute the set of forward reachable states in $\ell + 1$ steps and perform the reachability checks again. Unfortunately, also this process is not guaranteed to terminate for arbitrary two-level transition systems. Fortunately, we are able to characterize a set of transition systems, corresponding to a relevant class of applications specified in BPEL and RBAC4BPEL, for which we can pre-compute an upper bound on $\ell$; this paves the way to solving automatically the goal reachability problem for these systems.

To this end, we consider three sufficient conditions to automate the solution of the goal reachability problem. First, the class $\mathcal{C}$ of formulae used to describe sets of states must be closed under post-image computation. Second, the satisfiability (in the structure underlying the transition system) of $\mathcal{C}$ must be decidable. Third, it must be possible to pre-compute a bound on the length of the sequence $FR^0, FR^1, ..., FR^\ell$ of formulae. Below, we show that these conditions are satisfied by a class of two-level transition systems to which applications specified in BPEL and RBAC4BPEL can be mapped. For ease of exposition, we first consider the WF and PM levels in isolation and then show how the results for each level can be modularly lifted when considering the two levels together. Before doing this, we introduce the notion of symbolic execution tree. The purpose of this is two-fold. First, it is crucial for the technical development of our decidability result. Second, it is the starting point for the implementation of our techniques as discussed in Section 5.

The *symbolic execution tree of the two-level transition system* $Tr$ is a labeled tree defined as follows: (i) the root node is labeled by the formula $In$, (ii) a node $n$ labeled by the formula $K$ has $d \leq n$ sons $n_1, ..., n_d$ labeled

by the formulae $Post(\tau_1, K), ..., Post(\tau_d, K)$ such that $Post(\tau_j, K)$ is satisfiable in the model underlying $Tr$ and the edge from $n$ to $n_j$ is labeled by $\tau_j$ for $j = 1, ..., d$, (iii) a node $n$ labeled by $K$ has no son, in which case $n$ is a *final node*, if $Post(\tau_j, K)$ is unsatisfiable in the underlying model of the VAS, for each $j = 1, ..., n$. A symbolic execution tree is 0-*complete* if it consists of the root node labeled by the formula $In$, it is $(d+1)$-*complete* for $d \geq 0$ if its depth is $d+1$ and for each node $n$ labeled by a formula $K_n$ at depth $d$, if $Post(\tau_j, K_n)$ is satisfiable, then there exists a node $n'$ at depth $d+1$ labeled by $Post(\tau_j, K_n)$. In other words, a symbolic execution tree is $d$-complete when all non-empty sets of forward states reachable in one step represented by formulae labeling nodes at depth $d-1$ have been generated. It is easy to see that the formula $FR^{\ell}(K, In)$, describing the set of states of the transition system $Tr$ forward reachable in $\ell \geq 0$ steps, is equivalent to the disjunction of the formulae labeling the nodes of an $\ell$-complete symbolic execution tree. This will be proved for the classes of two-level transition systems that we consider below.

**4. Mapping BPEL and RBAC4BPEL to two-level transition systems.** We now explain how security-sensitive services specified by (a sub-set of) BPEL and RBAC4BPEL (such as the PO process described in Section 2) can be specified as two-level transition systems. To simplify the task of the specifier as well as the technical development, we first describe how WF nets can be seen as a certain class of transition systems (Section 4.1), then we show how an RBAC4BPEL system can be mapped to a certain transition system (Section 4.2), and finally we explain how these specifications can be combined to obtain a complete specification (Section 4.3). We illustrate the notions by using the PO process as the running example. Although we do not show it for lack of space, transforming WF nets and RBAC4BPEL can be made automatic. For more details about this point, see also Section 6.2 where we briefly discuss how we have modified an available tool for generating Petri nets from BPEL files to derive the transition systems described in the following subsection.

**4.1. WF nets and terminating forward reachability.** We consider *Vector Addition System (VAS)*, a particular class $\langle \underline{x}, In(\underline{x}), \{\tau_i(\underline{x}, \underline{x}') \mid i = 1, ..., n\}\rangle$, of two-level transition systems such that (i) $\underline{p} = \emptyset$; (ii) their underlying structure is that of integers; (iii) each WF state variable in $\underline{x} = x_1, ..., x_m$ ranges over the set of non-negative integers; (iv) the initial condition $In(\underline{x})$ is a formula of the form $x_1 \bowtie c_1 \wedge \cdots \wedge x_m \bowtie c_m$, where $c_j$ is a natural number for $j = 1, ..., m$ and $\bowtie \in \{=, \neq, >, \geq\}$; and (v) each transition $\tau_i$, for $i = 1, ..., n$, is a formula of the form

$$\bigwedge_{i \in P} x_i \geq 0 \wedge \bigwedge_{j \in U^+} x'_j = x_j + 1 \wedge \bigwedge_{k \in U^-} x'_k = x_k - 1 \wedge \bigwedge_{l \in U^=} x'_l = x_l \,,$$

where $P, U^+, U^-, U^=$ are subsets of $\{1, ..., n\}$ such that $U^+, U^-, U^=$ form a partition of $\{1, ..., n\}$.

It is well-known that Petri nets and VASs are equivalent in the sense that analysis problems for the former can be transformed to problems of the latter whose solutions can be mapped back to solutions for the original problem and vice versa (see, e.g., [20]). We briefly describe the correspondence by considering the Petri net in Fig. 2.1. We associate an integer variable $x_i$ to each place $p_i$ for $i = 1, ..., 10$ whose value will be the number of tokens in the place. The state is given by the value of the integer variables representing the marking of the net, i.e., a mapping from the set of places to non-negative integers. Formulae can be used to represent sets of states (or, equivalently, of markings). So, for example, the formula $x_1 = 1 \wedge \bigwedge_{i=2}^{10} x_i = 0$ represents the marking where one token is in place $p_1$ and all the other places are empty (which is the one depicted in Fig. 2.1 where the token is represented by a solid circle inside that represents the place $p_1$ while all the other places do not contain any solid circle). The transition *crtPO* is represented by the formula

$$x_1 \geq 1 \wedge x'_1 = x_1 - 1 \wedge x'_2 = x_2 + 1 \wedge \bigwedge_{i=3}^{10} x'_i = x_i$$

saying that it is enabled when there is at least one token in $p_1$ ($x_1 \geq 1$) and the result of its execution is that a token is consumed at place $p_1$ ($x'_1 = x_1 - 1$), the tokens in $p_2$ are incremented by one ($x'_2 = x_2 + 1$), while the tokens in all the other places are unaffected ($x'_i = x_i$ for $i = 3, ..., 10$). The other transitions of the Petri net in Fig. 2.1 are translated in a similar way. In general, it is always possible to associate a state of a VAS to a marking of a Petri net and vice versa. This implies that solving the reachability problem for a VAS is equivalent to solving the reachability problem of the associated Petri net.

Now, we show that the three sufficient conditions (see Section 3.2) to mechanize the solution of the goal reachability problem are satisfied by VASs when using forward reachability. First, the class of formulae is closed under post-image computation:

FACT 1. *$Post(K, \tau_i)$ is equivalent to $K[x_j - 1, x_k + 1, x_l] \wedge \bigwedge_{i \in P} x_i \geq 0$, where $K[x_j - 1, x_k + 1, x_l]$ denotes the formula obtained by replacing $x'_j$ with $x_j - 1$ for $j \in U^+$, $x'_k$ with $x_k - 1$ for $k \in U^-$, and $x'_l$ with $x_l$ for $j \in U^=$.* ■

As a corollary, it is immediate to derive that if $K$ is a formula of Linear Arithmetic (LA) [8] — roughly, a formula where multiplication between variables is forbidden — then also $Post(K, \tau_i)$ is equivalent to an effectively computable formula of LA. Second, the satisfiability of the class of formulae of LA is decidable by well-known results [8]. Third, it is possible to pre-compute a bound on the length of the sequence $FR^0, FR^1, ..., FR^\ell$ of formulae. Using the notion of symbolic execution tree introduced above, once specialized to VASs, we can then prove:

LEMMA 4.1. *Let $PN := \langle P, T, F \rangle$ be an acyclic workflow net and $\Pi$ be the set of all its paths. Then, the set of forward reachable states of $\langle \underline{x}, In(\underline{x}), \{\tau_i(\underline{x}, \underline{x}') \mid i = 1, ..., n\} \rangle$, the VAS associated to PN, is identified by the formula $FR^\ell(\tau, In)$ for $\ell = max_{\pi \in \Pi}\{len(\pi|_T)\}$, where $\pi|_T$ is the sequence obtained from $\pi$ by forgetting each of its elements in $P$ and $len(\pi|_T)$ is the length of the sequence $\pi|_T$.* ■

This result is proved in [9] by using the notion of symbolic execution tree introduced above, specialized to VASs.

**4.2. RBAC4BPEL and terminating forward reachability.** Preliminarily, let $Enum(\{v_1, ..., v_n\}, S)$ be the following set of FOL formulae axiomatizing the enumerated datatype with values $v_1, ..., v_n$ for a given $n \geq 1$ over a type $S$: $v_i \neq v_j$ for each pair $(i, j)$ of numbers in $\{1, ..., n\}$ such that $i \neq j$ and $\forall x. (x = v_1 \vee \cdots \vee x = v_n)$, where $x$ is a variable of type $S$. The formulae in $Enum(\{v_1, ..., v_n\}, S)$ fix the number of elements of any interpretation to be $v_1, ..., v_n$; it is easy to see that the class of structures satisfying these formulae are closed under isomorphism. We consider RBAC4BPEL, a particular class $\langle \underline{p}, In(\underline{p}), \{\tau_i(\underline{p}, \underline{p}') \mid i = 1, ..., n\} \rangle$ of two-level transition systems such that (i) $\underline{x} = \emptyset$; (ii) the initial condition $In(\underline{p})$ is of the form $\forall \underline{w}. \varphi(\underline{w})$, where $\varphi$ is a quantifier-free formula where at most the variables in $\underline{w}$ may occur free; and (iii) the underlying structure is one in the (isomorphic) class of many-sorted structures axiomatized by the following sentences:

$$
\begin{array}{l}
Enum(U, User)\,, \\
Enum(R, Role)\,, \\
Enum(P, Permission)\,, \\
Enum(A, Action)\,, \\
\forall u, r.\, (ua(u, r) \Leftrightarrow \bigvee_{c_u \in U_{ua}, c_r \in R_{ua}} (u = c_u \wedge r = c_r))\,, \\
\forall r, p.\, (pa(r, p) \Leftrightarrow \bigvee_{c_r \in R_{pa}, c_p \in P_{pa}} (r = c_r \wedge p = c_p))\,, \\
c_r \succeq c'_r \text{ for } c_r, c'_r \in R\,, \\
\forall r.\, (r \succeq r)\,, \\
\forall r_1, r_2, r_3.\, (r_1 \succeq r_2 \wedge r_2 \succeq r_3 \Rightarrow r_1 \succeq r_3)\,, \\
\forall r_1, r_2.\, (r_1 \succeq r_2 \wedge r_2 \succeq r_1 \Rightarrow r_1 = r_2)\,,
\end{array}
$$

where $U$, $R$ and $P$ are finite sets of constants denoting users, roles, and permissions, respectively, $A$ is a finite set of actions, $u$ is a variable of type *User*, $r$ and its subscripted versions are variables of type *Role*, $p$ is a variable of type *Permission*, $U_{ua} \subseteq U$, $R_{ua} \subseteq R$, $R_{pa} \subseteq R$, and $P_{pa} \subseteq P$; (d) $\underline{p} = xcd$ is a predicate symbol of type $User \times Action$ abbreviating *executed*; and (e) each $\tau_i$ is of the form

$$\exists \underline{u}.\, (\xi(\underline{u}, xcd) \wedge \forall x, y.(xcd'(x, y) \Leftrightarrow ((x = u_j \wedge y = p) \vee xcd(x, y)))),$$

where $\underline{u}$ is a tuple of existentially quantified variables of type *User*, $u_j$ is the variable at position $j$ in $\underline{u}$, and $\xi(\underline{u}, xcd)$ is a quantifier-free formula (called the *guard* of the transition) where no function symbol of arity greater than 0 may occur (the part of $\tau_i$ specifying $xcd'$ is called the *update*).

To explain how a RBAC4BPEL system can be specified by the formulae above let us consider again the example described in Section 2. To constrain the sets of users, roles, and permissions to contain exactly the elements specified in Fig. 2.2, it is sufficient to use the following sets of formulae:

$$
\begin{array}{l}
Enum(\{u_1, u_2, u_3, u_4\}, User)\,, \\
Enum(\{manager, finAdmin, finClerk, poAdmin, poClerk\}, Role)\,, \\
Enum(\{p_1, p_2, p_3, p_4, p_5\}, Permission)\,.
\end{array}
$$

It is also easy to see that the formulae

$$\forall u, r.\,(ua(u,r) \Leftrightarrow \left(\begin{array}{l}(u = u_1 \wedge r = \mathit{manager}) \vee (u = u_2 \wedge r = \mathit{finAdmin}) \vee \\ (u = u_3 \wedge r = \mathit{finClerk}) \vee (u = u_4 \wedge r = \mathit{poAdmin}) \vee \\ (u = u_5 \wedge r = \mathit{poClerk})\end{array}\right)$$

and

$$\forall r, p.\,(pa(r,p) \Leftrightarrow \left(\begin{array}{l}(r = \mathit{finClerk} \wedge p = p_4) \vee (r = \mathit{finAdmin} \wedge p = p_5) \vee \\ (r = \mathit{poClerk} \wedge p = p_3) \vee (r = \mathit{poAdmin} \wedge p = p_1)\end{array}\right)$$

are satisfied by the interpretations of *ua* and *pa* in Fig. 2.2 and that $\mathit{manager} \succeq \mathit{finAdmin}$, $\mathit{manager} \succeq \mathit{poAdmin}$, $\mathit{finAdmin} \succeq \mathit{finClerk}$, and $\mathit{poAdmin} \succeq \mathit{poClerk}$ with the three formulae above for reflexivity, transitivity and antisymmetry make the interpretation of $\succeq$ the partial order considered in Fig. 2.2. The state variable *xcd* allows us to formalize component (vii) of the RBAC4BPEL system about the authorization constraints. The idea is to use *xcd* to store the pair user $u$ and action $a$ when $u$ has performed $a$ so that the authorization constraints can be formally expressed by a transition involving suitable pre-conditions on these variables. We illustrate the details on the first authorization constraint considered in Section 4.2, i.e., $\langle U, (\mathit{apprPO}, \mathit{signGRN}), \neq\rangle$. The corresponding transition can be formalized as follows:

$$\exists x_1, x_2.\,(xcd(x_1, \mathit{apprPO}) \wedge x_1 \neq x_2 \wedge \forall x, y.\,(xcd'(x,y) \Leftrightarrow ((x = x_2 \wedge y = \mathit{signGRN}) \vee xcd(x,y)))\,.$$

The guard of the transition prescribes that the user $x_2$ is not the same user $x_1$ that has previously performed the action *apprPO* and the update stores in *xcd* the new pair $(x_2, \mathit{signGRN})$. The following two constraints at the end of Section 4.2, namely $\langle U, (\mathit{apprPO}, \mathit{ctrSignGRN}), \neq\rangle$ and $\langle U, (\mathit{signGRN}, \mathit{ctrSignGRN}), \neq\rangle$, are formalized in a similar way. The encoding of the last constraint, i.e., $\langle R, (\mathit{crtPay}, \mathit{apprPay}), \prec\rangle$, is more complex and requires also the use of the user-role relation *ua* to represent the constraint on the role hierarchy:

$$\exists x_1, x_2, r_1, r_2.\,(xcd(x_1, \mathit{crtPay}) \wedge ua(x_1, r_1) \wedge ua(x_2, r_2) \wedge r_2 \succeq r_1 \wedge r_1 \neq r_2 \wedge \\ \forall x, y.\,(xcd'(x,y) \Leftrightarrow ((x = x_2 \wedge y = \mathit{apprPay}) \vee xcd(x,y)))\,.$$

The reader should now be convinced that every RBAC4BPEL specification can be translated into a RBAC4BPEL system.

We show that the three sufficient conditions to mechanize the solution of the goal reachability problem (see Section 3.2) are satisfied by RBAC4BPEL systems when using forward reachability. First, the class of formulae is closed under post-image computation.

Fact 2. *$\mathit{Post}(K, \tau_i)$ is equivalent to*

$$(\exists \underline{u}.(K(xcd) \wedge xcd(u_j, t) \wedge \xi(\underline{u}, xcd))) \vee (\exists \underline{u}.(K[\lambda x, y.(\neg(x = u_j \wedge y = t) \wedge xcd(x,y))] \wedge \\ \xi[\underline{u}, \lambda x, y.(\neg(x = u_j \wedge y = t) \wedge xcd(x,y))]))\,,$$

*where $K[\lambda x, y.(\neg(x = u_j \wedge y = t) \wedge xcd(x,y))]$ is the formula obtained from $K$ by substituting each occurrence of $xcd'$ with the $\lambda$-expression in the square brackets and then performing the $\beta$-reduction and similarly for $\xi[\underline{u}, \lambda x, y.(\neg(x = u_j \wedge y = t) \wedge xcd(x,y))]$.* ■

As anticipated above when introducing the definition of post-image for two-level transition systems, we can eliminate the second-order quantifier over the predicate symbol *xcd*. Now, recall that a formula is in the *Bernays-Schönfinkel-Ramsey (BSR)* class if it has the form $\exists \underline{z} \forall \underline{w}.\,\phi(\underline{z}, \underline{w})$, for $\phi$ a quantifier-free formula and $\underline{z} \cap \underline{w} = \emptyset$ (see, e.g., [17]). As a corollary of Fact 2, it is immediate to see that if $K$ is a BSR formula, then also $\mathit{Post}(\tau_i, K)$ is equivalent, by trivial logical manipulations, to a formula in the BSR class. Since $\mathit{In}(xcd)$ is a formula in the BSR class, then all the formulae in the sequence $FR^0, FR^1, ...$ will also be BSR formulae. The second requirement is also fulfilled since the satisfiability of the BSR class is well-known to be decidable [17] and the formulae used to axiomatize the structures underlying the RBAC4BPEL transition systems are also in BSR. Third, it is possible to pre-compute a bound on the length of the sequence $FR^0, FR^1, ..., FR^\ell$ of formulae, although the existential prefix grows after each computation of the post-image when considering the formulae describing the set of forward reachable states. This is so because we consider only a finite and known set of

users so that the length of the existentially quantified prefix is bounded by $n_u^k \times n$, where $k$ is the maximal length of the existential prefixes of the transitions in the RBAC4BPEL system, $n_u$ is the number of users, and $n$ is the number of transitions.

PROPERTY 1. *Let $\langle \underline{p}, In(\underline{p}), \{\tau_i(\underline{p}, \underline{p}') \mid i = 1, ..., n\}\rangle$ be a RBAC4BPEL system, $k$ be the maximal length of the existential prefixes of $\tau_1, ..., \tau_n$, and $n_u$ be the cardinality of the set of users. Then, its symbolic execution tree is $\ell$-complete for every $\ell$ such that $\ell \geq n_u^k \times n$.* ■

The key idea of the proof is the observation that *xcd* is interpreted as a subset of the Cartesian product between the set of users and the set of actions whose cardinalities are bounded.

**4.3. Combining VASs and RBAC4BPEL systems.** We are now ready to fully specify applications that feature both the WF and the PM level. To do this, we consider *VAS+RBAC4BPEL* systems, two-level transition systems of the form

$$\langle \underline{x}, \underline{p}, In_V(\underline{x}) \wedge In_R(\underline{p}), \{\tau_i^V(\underline{x}, \underline{x}') \wedge \tau_i^R(\underline{p}, \underline{p}') \mid i = 1, ..., n\}\rangle,$$

where $\underline{x} = x_1, ..., x_n$ for some $n \geq 1$; $\underline{p} = xcd$, $In_V(\underline{x})$ is the initial condition of a VAS; $In_R(\underline{p})$ is the initial condition of a RBAC4BPEL system; $\tau_i^V(\underline{x}, \underline{x}')$ is a transition of a VAS; and $\tau_i^R(\underline{p}, \underline{p}')$ is a transition formula of a RBAC4BPEL system for $i = 1, ..., n$. Note that for some transition, the guard $\xi$ of $\tau_i^R(\underline{p}, \underline{p}')$ may be tautological since the operation involves no access-control policy restriction (e.g., the *flow split* and *flow join* of the Petri net in Fig. 2.1). It is natural to associate a VAS and an RBAC4BPEL system to a VAS+RBAC4BPEL system by projection, i.e., the associated VAS is $\langle \underline{x}, In_V(\underline{x}), \{\tau_i^V(\underline{x}, \underline{x}') \mid i = 1, ..., n\}\rangle$ and the associated RBAC4BPEL system is $\langle \underline{p}, In_R(\underline{p}), \{\tau_i^R(\underline{p}, \underline{p}') \mid i = 1, ..., n\}\rangle$. The structure underlying the VAS+RBAC4BPEL system is such that its reduct to the signature of the VAS is identical to the structure underlying the associated VAS, and its reduct to the signature of the RBAC4BPEL system is identical to the structure underlying the associated RBAC4BPEL system.

We now show how it is possible to modularly compute the post-image of a VAS+RBAC4BPEL system by combining the post-images of the associated VAS and RBAC4BPEL system.

FACT 3. *Let $K(\underline{x}, xcd) := K_V(\underline{x}) \wedge K_R(xcd)$. Then, $Post(K, \tau_i)$ is equivalent to*

$$K_V[x_j + 1, x_k - 1, x_l] \wedge \bigwedge_{i \in P} x_i \geq 0 \wedge ((\exists \underline{u}.(K_R(xcd) \wedge xcd(u_j, t) \wedge \xi(\underline{u}, xcd))) \vee (\exists \underline{u}.(K_R[\lambda x, y.(\neg(x = u_j \wedge y = t) \wedge xcd(x, y))] \wedge \xi[\underline{u}, \lambda x, y.(\neg(x = u_j \wedge y = t) \wedge xcd(x, y))]))),$$

*where the same notational conventions of Facts 1 and 2 have been adopted. In other words, the post-image of a VAS+RBAC4BPEL system is obtained as the conjunction of the post-images of the associated VAS, denoted with $Post_V(K, \tau_i) := Post(K_V, \tau_i^V)$, and the associated RBAC4BPEL system, denoted with $Post_R(K, \tau_i) := Post(K_R, \tau_i^R)$. Thus, we abbreviate the above formula as $Post_V(K, \tau_i) \wedge Post_R(K, \tau_i)$.* ■

The proof of this fact is obtained by simple manipulations minimizing the scope of applicability of $\exists \underline{x}$ and $\exists xcd$, respectively, and then realizing that the proofs of Facts 1 and 2 can be re-used verbatim. Because of the modularity of post-image computation, it is possible to modularly define the set of forward reachable states and the symbolic execution trees for VAS+RBAC4BPEL systems in the obvious way. By modularity, we can easily show the following property.

PROPERTY 2. *Let $PN := \langle P, T, F\rangle$ be a an acyclic WF net, $\langle \underline{x}, In_V(\underline{x}), \{\tau_i^V(\underline{x}, \underline{x}') \mid i = 1, ..., n\}\rangle$ be its associated VAS, and $\langle \underline{p}, In_R(\underline{p}), \{\tau_i^R(\underline{p}, \underline{p}') \mid i = 1, ..., n\}\rangle$ be the RBAC4BPEL system with $n_u$ users and $k$ be the maximal length of the existential prefixes of $\tau_1^R, ..., \tau_n^R$. Then, the symbolic reachability tree of the VAS+RBAC4BPEL system whose associated VAS and RBAC systems are those specified above is $\ell$-complete for every $\ell \geq min(max_{\pi \in \Pi}\{len(\pi|_T)\}, n_u^k \times |T|)$.* ■

The key observation in the proof of this property is that in order to take a transition, the preconditions of the associated VAS and of the associated RBAC4BPEL system must be satisfied. Because of the modularity of the post-image, the duality between the set of forward reachable states and the formulae labeling the symbolic execution tree can be lifted to VAS+RBAC4BPEL. We are now ready to state the decidability of the VAS+RBAC4BPEL system; see [9, extended version] for the proof.

THEOREM 4.2. *Let $PN := \langle P, T, F\rangle$ be a an acyclic WF net and let $\langle \underline{x}, In_V(\underline{x}), \{\tau_i^V(\underline{x}, \underline{x}') \mid i = 1, ..., n\}\rangle$ be its associated VAS. Further, let $\langle \underline{p}, In_R(\underline{p}), \{\tau_i^R(\underline{p}, \underline{p}') \mid i = 1, ..., n\}\rangle$ be a RBAC4BPEL system with a bounded*

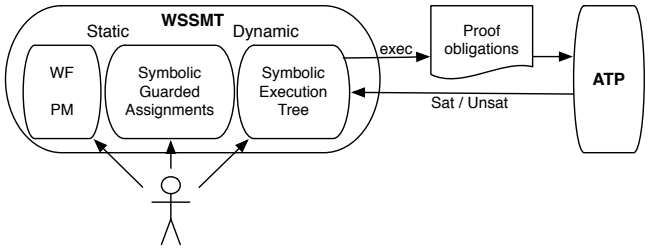



FIG. 4.1. *High level architecture view of WSSMT. All components of the architecture are described in Section 5.*

*number of users. Then, the symbolic reachability problem of the VAS+RBAC4BPEL system (whose associated VAS and RBAC4BPEL systems are those specified above) is decidable.* ■

To illustrate the kind of formulae arising in the application of Theorem 4.2, we consider the example specified in Fig. 2.1. In this case, we can restrict to consider three paths (projected over the transitions) in the WF net: *crtPO*, *apprPO*, *flow split*, *signGRN*, *ctrSignGRN*, *crtPay*, *flow join*, *apprPay*; *crtPO*, *apprPO*, *flow split*, *signGRN*, *crtPay*, *ctrSignGRN*, *flow join*, *apprPay*; and *crtPO*, *apprPO*, *flow split*, *crtPay*, *signGRN*, *ctrSignGRN*, *flow join*, *apprPay*; each one of length eight. It is easy to see that only the first path is to be considered as the other two produce states that are equivalent since it does not matter at what time *crtPay* is executed with respect to *signGRN* and *ctrSignGRN* (it is possible to mechanize also this check but we leave out the details for lack of space). So, for example, it is possible to check the so-called soundness of workflows [22], i.e., to check whether it is possible to terminate without "garbage" left. In terms of a WF net, this means that no tokens are left in places other than the special final place $o$ of the net. This can be checked by computing the post-images of the initial state of the VAS+RBAC4BPEL system of our motivating example along the lines of Facts 1, 2, and 3 and put this in conjunction with the formula characterizing the "no-garbage" condition, i.e.,

$$x_{10} \geq 1 \wedge \bigwedge_{i=1}^{9} x_i = 0\,.$$

Thanks to the closure under post-image computation of the VAS and the RBAC4BPEL systems, as well as the modularity of the post-image computation for the VAS+RBAC 4BPEL system, the resulting proof obligation is decidable as it can be put in the form $\varphi_V \wedge \varphi_R$ where $\varphi_V$ is a formula of LA (whose satisfiability is decidable) and $\varphi_R$ is a BSR formula (whose satisfiability is again decidable), and thus the satisfiability of their conjunction is also decidable.

**5. WSSMT: mechanizing the analysis of security-sensitive services.** There are two ways to mechanize the symbolic execution technique introduced in Section 3.2: the implementation of an *ad hoc* tool or the re-use of existing tools via a suitable front-end. Since the verification problems are reduced to satisfiability problems modulo theories, it is highly desirable to exploit the cornucopia of well-engineered and scalable Automated Theorem Proving (ATP) systems such as resolution-based provers and Satisfiability Modulo Theories (SMT) solvers. We chose the second option and implemented a tool called WSSMT, which is an acronym of "Web Service (validation by) Satisfiability Modulo Theories". A detailed description of the tool and its implementation can be found in [5, 31], here we only sketch its main functionalities and architecture, which is depicted in Fig. 4.1.

The main goal of WSSMT is to help users write specifications of security-sensitive services structured along the previously identified directions: WF/PM levels and static/dynamic parts. Once the algebraic structure of the WF and PM levels have been identified (e.g., Linear Arithmetic in the case of a VAS) by means of suitable *first-order theories* (see, e.g., [7] for a discussion of how to use theories to describe the state space of two-level transition systems) and a two-level transition system is specified to describe the possible actions of the system, the front-end will enable the user to create and manipulate a symbolic execution tree, which compactly represents several possible symbolic executions of the service under consideration. Indeed, to create such a tree, whose nodes are labeled with state formulae and edges with (instances of) transitions, the front-end must create the appropriate proof obligations (as explained in Section 3) and then invoke an available ATP system, chosen

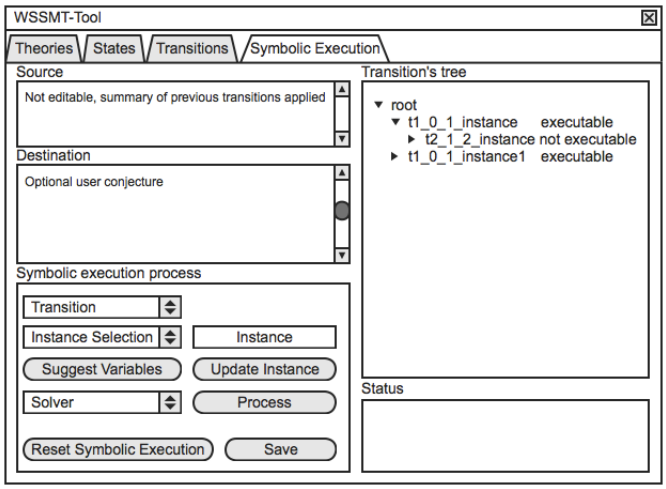



FIG. 5.1. *Symbolic Execution tab.*

by the user among those available in the back-end. Once the ATP has established the satisfiability of the proof obligation, the front-end updates the symbolic execution tree or complains about the impossibility of executing the chosen transition. The client-server architecture of WSSMT follows these observations as shown in Fig. 4.1.

The front-end is organized in several tabs corresponding to the various ingredients of our specification and verification framework: *Theories*, *States*, *Transitions*, and *Symbolic Execution*. The first two tabs describe the static part of the specification and are structured in such a way to specify the WF and PM levels independently. The *Transitions* tab allows the user to enter transitions as specified in Section 4.3. The *Symbolic Execution* tab, depicted in Fig. 5.1, is split in two parts: on the left, the user can enter the symbolic step to be checked for executability, while the right part shows the symbolic execution tree that represents one or more possible scenarios of execution. More precisely, the left part shows the state formula from which the symbolic execution step is taken (labeled *Source*) and allows the user to enter the formula to which the execution step should lead (labeled *Destination*) together with a transition chosen from the list of available transitions (combo labeled *Transition*), whose identifiers have been instantiated as explained in the combo labeled *Instance selection*. To send the resulting proof obligation to a back-end ATP system, the user should press the button *Process*.

To ease portability and modularity, WSSMT has been implemented in Java 1.5 as an Eclipse 3.5 plug-in by exploiting the SWT and JFace libraries [13, 27] for the creation of multi-platform graphical user interfaces. The concrete input language of state and transition formulae, as well as of axioms of the theories in WSSMT, is the DFG syntax [29]. It has been chosen because it supports many-sorted FOL, it is easy to extend, and several tools are available for its parsing and translation.

Currently, WSSMT has been used with SPASS [24], a state-of-the-art resolution-based prover, and the SMT solver Z3 [30]. The former was chosen because it has the same input language as the front-end so that it is trivial to generate the proof obligations to support symbolic execution. However, it would be easy to integrate any ATP system whose input language is the TPTP format [26] as there exists a translator from the DFG syntax to the TPTP format in the distribution of SPASS. This is left to future work. Z3 was chosen because it is one of the best SMT solvers (according to the last competition for such tools [23] at the state of writing) and complements the reasoning capabilities of SPASS by providing support for ubiquitous theories as decidable fragments of arithmetics (while SPASS only supports reasoning in pure FOL). It was easy to create a translator from the DFG syntax to the SMT-LIB input language [18], which is one of the input languages of Z3. Furthermore, integration of further SMT solvers can be done seamlessly as the SMT-LIB language is their common input language. The evaluation of the advantages of having several SMT solvers available as back-ends in WSSMT is also left to future work.

The ATP systems are invoked via calls to the operating system provided by Java and their results are parsed by the front-end in order to update the symbolic execution tree accordingly (along the lines explained in Section 3). Our experience in using WSSMT for the analysis of some specifications of security-sensitive services can be found in the next section, along with remarks about the performances of the ATP systems to discharge

```
<process name="LOP"/>
  <sequence>
    <invoke> inputCustData </invoke>
    <flow>
        <invoke> prepareContract </invoke>
         <sequence>
         <invoke> intRating </invoke>
         <if>
           <condition> ∼lowRisk </condition>
           <invoke> extRating </invoke>
         </if>
         </sequence>
     </flow>
   <invoke> approve </invoke>
     <if>
       <condition> ∼productOK </condition>
       <invoke> sign </invoke>
     </if>
  </sequence>
</process>
```

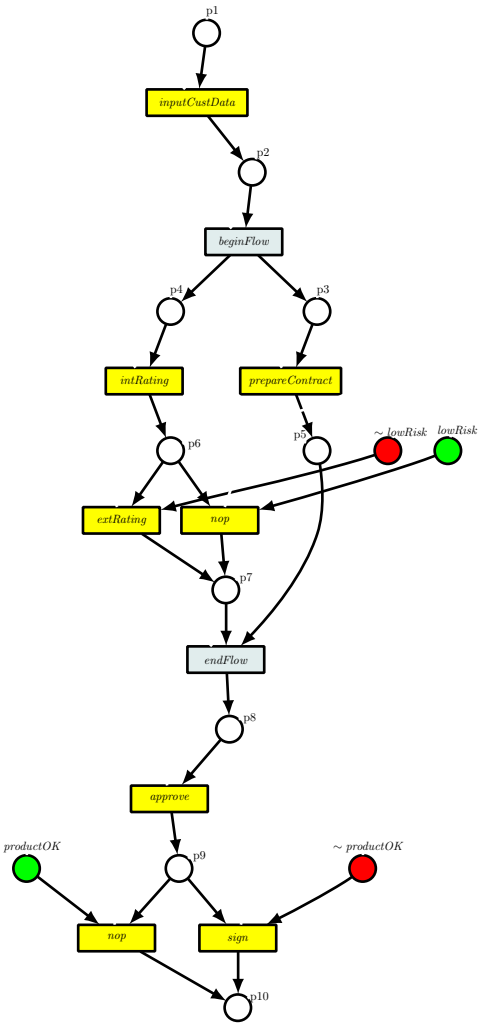



FIG. 6.1. *The LOP process: BPEL (left) and corresponding Petri net (right). Legend: see the legend in Fig. 2.1 and, in addition, consider that places filled in red (resp., green) specify Boolean variables whose value is false (resp., true).*

typical proof obligations.

**6. Case studies.** In this section, we report on the application of WSSMT to the specification and analysis of two industrial case studies considered in the FP7 European project AVANTSSAR (more details about these and other case studies can be found in [4]): the first one is called the Loan Origination Process and concerns a banking service in the e-business applications area (Section 6.1), whereas the second one is called the Digital Contract Signing and is a protocol in the e-government area (Section 6.2).

**6.1. Loan Origination Process.** The Loan Origination Process (LOP) is a bank's loan application process, which has been used as an example in many works (see, e.g., [2, 4, 11, 21]). We adapt the variant proposed in [2, 7], whose workflow is presented as a BPEL process and whose policy level is described by RBAC policies with delegation. Fig. 6.1 shows the BPEL description (left) and the corresponding Petri net translation (right) of the LOP. The Petri net has been obtained along the same lines as the one for PO.

Roughly, the LOP is composed of seven activities: the customer starts the process by providing its own data (*inputCustData*) and then the contract is prepared (*prepareContract*) and the customer's rating evaluation is run concurrently. The evaluation activity is performed by first performing an internal evaluation (*intRating*): if the risk associated with the loan is low, then the internal rating is sufficient (*nop*); otherwise, a credit

reporting agency is asked for an external rating (*extRating*). Afterwards, the bank approves the result of the loan evaluation (*approve*) and decides whether to sign the loan (*sign*) or not, entering again in the task (*nop*).

**6.1.1. WF level.** Similarly to the PO of Fig. 2.1, the order of the execution of the various activities of the LOP should satisfy some constraints to complete successfully. The *inputCustData* must be executed at the beginning of the process, then the flow splits in two so that parallel execution of two sets of tasks is possible. *intRating* and *prepareContract* are concurrent activities and while *prepareContract* is the only task in the right branch (see Fig.6.1), *intRating* in the left branch leads to a fork composed by *extRating* or *nop*. The choice to follow one of these branches will depend on the value of a Boolean variable (external to the flow) *lowRisk*. At this point the two parallel strings can synchronize by executing the task *endFlow*. Then, the task *approve* leads again to split the flow and the choice to follow the task *sign* or *nop* will be done depending on another external Boolean variable *productOK*. At this point, no more transitions are enabled and the execution of the workflow is terminated.

We can map the BPEL description to a WF net and this to a transition system as described in Section 4. In the following, for the sake of brevity, we sketch how to translate some of the markings of the Petri net and a few transitions.

Formally, we represent the set of places ($p_i$) using a map to the set of non-negative integers. To formalize the *Initial* (resp., *Final*) state of the flow where there is just one token in place $p_1$ and (resp., $p_{10}$) all other places are empty, is sufficient to write the following two formulae:

$$\mathit{Initial} \;:=\; p_1 = 1 \wedge \bigwedge_{i=2}^{10} p_i = 0 \;\text{ and }\; \mathit{Final} \;:=\; p_{10} = 1 \wedge \bigwedge_{i=1}^{9} p_i = 0\,.$$

Let us analyze *inputCustData*: the guard of the transition requires that there is one token in place $p_1$, its update consumes the token in $p_1$ and produces a token in $p_2$ while leaving all other places as they are. This can be formally represented by the following formula:

$$\mathit{inputCustData} := p_1 = 1 \wedge p_1' = p_1 - 1 \wedge p_2' = p_2 + 1 \wedge \bigwedge_{i=3}^{10} (p_i' = p_i)\,.$$

All the other transitions are formalized along the same lines.

**6.1.2. PM level.** Each task can be executed by an agent who is entitled to perform it. For example, the task *inputCustData* can be executed by an employee of the bank who has the role of *preprocessor*, while the task *sign* can be performed by the *director* of the bank. Indeed, the employees have certificates that can be used to play certain roles that are organized hierarchically, i.e., a certain role inherits all the rights owned by roles that are lower in the hierarchy. In this way, for example, the *director* of the bank has all the rights of one of its employees, so that the *director* can, in principle, process the loan application of a customer.

We also consider delegation of tasks. For example, for certain loan applications (e.g., for loans below a certain amount), the *director* of the bank can delegate one of his supervisors to sign the loan contract after its approval. To manage these issues we use the RBAC model, whose main motivation is to map security management to an organization's structure (such as the bank in the LOP). The underlying idea is that each user's role may be based on the user's job responsibilities in the organization. The key ingredients of an RBAC model are the same as the ones introduced in Section 4.2.

Similarly to the case of RBAC4BPEL, to formalize RBAC we have introduced three sort symbols $Id, Role$, and $Task$ for the sets $U, R$, and $T$, respectively. To formalize the PM level of the LOP, we assume that the three sort symbols are endowed with the following enumerated data-type theories:

$$T_{Id} := \mathit{Enum}(\{\mathit{davide}, \mathit{maria}, \mathit{marco}, \mathit{pierPaolo}, \mathit{pierSilvio}\}, Id),$$
$$T_{Role} := \mathit{Enum}(\{\mathit{preprocessor}, \mathit{postprocessor}, \mathit{supervisor}, \mathit{manager}, \mathit{director}\}, Role),$$
$$T_{Task} := \mathit{Enum}(\{\mathit{inputCustData}, \mathit{prepareContract}, \mathit{intRating}, \mathit{extRating}, \mathit{approve}, \mathit{sign}\}, Task).$$

Note that constant symbols of sort $Task$ have exactly the same names of transition labels of the Petri net in Fig. 6.1 (right).

Since a user can be associated to several roles, s/he must choose to activate a role (with the appropriate rights) to execute a task. Formally, we introduce a dynamic predicate symbol $\mathit{activated} : Id \times Role$ such that

$activated(i, r)$ holds whenever the user $i$ has chosen to become active in role $r$ (if this is possible according to the relation $ua$). We modeled activation and deactivation of roles with the following two transitions: the first formalizes the fact that an agent is active in a certain role if it was not already playing the given role and it can be assigned to it

$$\exists i.\ \left( \begin{array}{l} ua(i,\rho) \wedge \neg activated(i,\rho) \wedge \\ \forall j, r.(activated'(j,r) \Leftrightarrow \left( \begin{array}{l} \textit{if } (j = i \wedge r = \rho) \\ \textit{then true else } activated(j,r) \end{array} \right)) \end{array} \right) \wedge \upsilon$$

and, dually, the second formula models deactivation from a role

$$\exists i.\ \left( \begin{array}{l} activated(i,\rho) \wedge \\ \forall j, r.(activated'(j,r) \Leftrightarrow \left( \begin{array}{l} \textit{if } (j = i \wedge r = \rho) \\ \textit{then false else } activated(j,r) \end{array} \right)) \end{array} \right) \wedge \upsilon\,,$$

where, for both activation and deactivation, $\rho \in \Lambda$, $\Lambda := \{preprocessor, postprocessor, supervisor, manager, director\}$, and $\upsilon := \forall i, r.\ (ua'(i,r) \Leftrightarrow ua(i,r)) \wedge \forall i, t.\ (pa'(i,t) \Leftrightarrow pa(i,t))$.

The RBAC model has been widely adopted to organize policy access for large organizations (see, e.g., [21]). However, even when a task is typically handled by an employee with a certain role, it may be the case that, under certain conditions, the employee wishes to delegate its right to execute a task to another employee belonging to a role down in the hierarchy. So, delegation is a key for the flexibility and scalability of service-oriented applications. In the LOP, roles can be delegated according to the following rules:

1. if the loan has been approved internally, then a *manager* can delegate a *supervisor* to approve a loan,
2. if the loan has been approved internally, then a *manager* can delegate a *supervisor* to sign the loan,
3. if the loan does not require an external rating, then the *director* can delegate a *manager* to sign the loan,
4. if the loan does not require an external rating, then a *supervisor* can delegate a *postprocessor* to perform an external rating of a loan.

To formalize this, we add the dynamic predicate $delegated : Id \times Role \times Task$ to the PM level state, and the two Boolean variables $intRatingOK$ and $highValue$ to the WF level state. If the user $i$ is delegated to execute task $t$ by a user who has role $r$, then $delegated(i, r, t)$ holds; $intRatingOK$ is the result of the internal (to the bank) evaluation of the loan application (since the logic determining the value of this variable is not modeled, the value of $intRatingOK$ will not be modified by any transition); and $highValue$ is set to true if the amount of the loan is greater than a certain threshold established by the bank (since the precise value of the threshold is unimportant for our model, the value of $highValue$ will not be modified by any transition). By using this and the previously introduced dynamic predicates, the transitions formalizing the role delegation rules above

can be written as follows:

$$1.\ \exists i_1, i_2.\ \left(\begin{array}{l} intRatingOK \wedge ua(i_1, manager) \wedge \\ \quad ua(i_2, supervisor) \wedge pa(manager, approve) \wedge \\ \quad \forall i, r, t.\ delegated'(i, r, t) \Leftrightarrow \\ \quad \left(\begin{array}{l} if\ (i = i_2 \wedge r = manager \wedge t = approve) \\ then\ true\ else\ delegated(i, r, t) \end{array}\right) \wedge \upsilon \end{array}\right)$$

$$2.\ \exists i_1, i_2.\ \left(\begin{array}{l} intRatingOK \wedge ua(i_1, manager) \wedge \\ \quad ua(i_2, supervisor) \wedge pa(manager, sign) \wedge \\ \quad \forall i, r, t.\ delegated'(i, r, t) \Leftrightarrow \\ \quad \left(\begin{array}{l} if\ (i = i_2 \wedge r = manager \wedge t = sign) \\ then\ true\ else\ delegated(i, r, t) \end{array}\right) \wedge \upsilon \end{array}\right)$$

$$3.\ \exists i_1, i_2.\ \left(\begin{array}{l} \neg highValue \wedge ua(i_1, director) \wedge \\ \quad ua(i_2, manager) \wedge pa(director, sign) \wedge \\ \quad \forall i, r, t.\ delegated'(i, r, t) \Leftrightarrow \\ \quad \left(\begin{array}{l} if\ (i = i_2 \wedge r = director \wedge t = sign) \\ then\ true\ else\ delegated(i, r, t) \end{array}\right) \wedge \upsilon \end{array}\right)$$

$$4.\ \exists i_1, i_2.\ \left(\begin{array}{l} \neg highValue \wedge ua(i_1, supervisor) \wedge \\ \quad ua(i_2, postprocessor) \wedge pa(supervisor, extRating) \wedge \\ \quad \forall i, r, t.\ delegated'(i, r, t) \Leftrightarrow \\ \quad \left(\begin{array}{l} if\ (i = i_2 \wedge r = supervisor \wedge t = extRating) \\ then\ true\ else\ delegated(i, r, t) \end{array}\right) \wedge \upsilon \end{array}\right),$$

where $\upsilon := \forall i, r.\ (ua'(i, r) \Leftrightarrow ua(i, r)) \wedge \forall i, r.(activated'(i, r) \Leftrightarrow activated(i, r)) \wedge \forall i, t.\ (pa'(i, t) \Leftrightarrow pa(i, t))$.

At this point, we are able to characterize when an agent $i$ playing role $r$ is granted the right to execute task $t$: either $i$ is delegated to execute task $t$ by a user in role $r$ or user $i$ is assigned to role $r$, $i$ is active in that role, and there exists a role $\rho$ such that $r \succeq \rho$ and $\rho$ has the permission to perform task $t$. To formalize this, we add the predicate $granted\ :\ Id \times Role \times Tas$ as well as the axioms

$$\forall i, r, t.\ granted(i, r, t)\ \Leftrightarrow \left(\begin{array}{l} delegated(i, r, t) \vee \\ (ua(i, r) \wedge activated(i, r) \wedge (r \succeq \rho \wedge pa(\rho, t))) \end{array}\right)$$

for each $\rho \in \Lambda$. The predicate *granted* is the key to constrain the transitions describing the workflow of the application in Fig. 6.1. Formally, this is done by incorporating suitable applications of the predicate *granted* in the guards of the transitions. Before giving the transition system characterizing the interplay between the WF and the PM levels, we introduce a further dynamic predicate symbol $executed\ :\ Id \times Task$ to keep track of the fact that user $i$ has performed a task $t$, i.e., $executed(i, t)$. As it will be clear below, this enables us to specify certain crucial security properties of the Loan Origination Process.

*Symbolic execution and debugging.* We now show how the capability of automatically checking executability of scenarios can be useful for debugging a specification. In particular, we show that the specification of the LOP given above violates a crucial security property, namely SoD: "for each agent $i$, there exists a task in the workflow that is never executed by $i$". This property is violated if we can find a sequence of transitions leading the LOP from the initial to the final state and the following formula

$$\forall i. \exists t. \left(\begin{array}{l} ((t \neq extRating \wedge t \neq sign) \Rightarrow \neg executed(i, t)) \wedge \\ (lowRisk \Rightarrow \neg executed(i, extRating)) \wedge \\ (productOK \Rightarrow \neg executed(i, sign)) \end{array}\right) \tag{6.1}$$

expressing SoD for the workflow of the LOP, is unsatisfiable. To this end, we can show that the sequence of tasks: *inputCustData*, *beginFlow*, *prepareContract*, *intRating*, *nop*, *endFlow*, *approve*, and *sign*, starting from a certain initial state, lead the transition system to a final state by generating a trace that violates (6.1) because each task can be executed by just one user. The scenario is taken from [2] and can be easily checked by using WSSMT (all proof obligations are discharged almost immediately). For a full description of the symbolic execution of the scenario showing that user *pierSilvio* can execute all tasks specified above, thereby violating SoD, the reader is once more pointed to [7].

**6.2. Digital Contract Signing.** The Digital Contract Signing (DCS) case study concerns a protocol for secure digital contract signing between *two signers*, which are assumed to have secure access to a trusted third party, called the *Business Portal* (*BP*) so as to digitally sign a contract. To achieve this goal, each signer communicates the contract's conditions to *BP*, which creates a digital version of the contract, stores it, and coordinates the two signers so as to obtain their digitally signed copies of the contract, which will be stored for future reference. The *BP* relies on two trusted services: the *Security Server* (*SServ*) and the *Public Key Infrastructure* (*PKI*). The *SServ* provides operations for creating a unique record in a secure database (only *BP* can access the *SServ* and thus modify the database), updating fields of existing records (i.e., to add signatures provided by signers), and sealing the signed contract in the record. The *PKI* is invoked in order to double check signatures against a *Certificate Revocation List* (*CRL*) so to be sure that during the execution of the protocol one signer has not misbehaved (though we have formalized this aspect of the DCS case study with a high level of abstraction). The DCS protocol is successful when both signers provide two correctly signed copies of the same contract and the *BP* can permanently store the signed copies of the contract.

Instead of manually translating the BPEL files to Petri nets and then derive the associated VASs, as we were able to do for the PO and the LOP, we used the freely available tool BPEL2oWFN [15]. Since DCS consists of four BPEL processes (one corresponding to a signer, one for *BP*, one for *SServ*, and one for *PKI*), we used BPEL2oWFN to compose the instances of the BPEL processes for the execution of the protocol, and we used it also to compute the corresponding Petri net. We have then processed this to derive the VAS in the input syntax of WSSMT.

To have an idea of the dimension of the problem, we sketch in Fig. 6.2 the Petri net generated by BPEL2oWFN consisting of 51 places (orange places represent data exchange between instances of different BPEL processes) and 26 transitions, while we omit the specification of the BPEL processes because it would take too much space.

Once obtained an abstraction of the WF level, we have manually added the PM level in order to specify the access control rules for each of the four principals involved in the protocol. As for the PO process, we have used RBAC4BPEL: the formalization that we used is along the same lines as those in Section 4.2. For example, we have two relations: one associating users with roles and one associating roles with permissions (in our case, transitions since we consider only the right to execute an action). As before, by taking the join of the two relations, we can compute the access control relation so as to grant or deny the right to execute a transition to a certain user. Since it is well-known how to symbolically represent relations and the join operation in FOL, it was not difficult for us to create a suitable theory for the PM level and augmenting the guards and the updates of the transitions in order to integrate the constraints of the access control rules. Along the same lines, we have added SoD (e.g., the user signing the contract should not be the same as the one checking the validity of the signature on the contract)—as for the LOP process—and BoD (e.g., the users signing the contract should be same that have agreed on the conditions of the contract) authorization constraints. Further details can be found in [9].

Given the abstract specification of the DCS, we have used WSSMT to perform the symbolic steps corresponding to the typical scenario of execution described in [4]. Since we used a VAS for the WF level, we discharged the resulting proof obligations by invoking the Z3 SMT solver, which provides native support for arithmetics (whereas the resolution prover SPASS does not). Each proof obligation was discharged in few seconds on a standard laptop and augmented our confidence in the correctness of the specification.

However, the specification we created was quite abstract as it ignored the content of the messages exchanged among the various principals. This was so because we used the tool BPEL2oWFN to generate the specification of the WF level. In fact, such a tool creates a coarse abstraction of BPEL processes where it is only taken into account if messages are sent and or received. We decided to manually refine the specification by adding FIFO queues containing messages with sender, receiver, and payload. To encode this in first-order theories, several methods are possible (see, e.g., [14]). Once obtained the refined specification, we replayed the symbolic execution corresponding to the typical scenario previously considered by using again Z3 as the back-end ATP system. Again, all the proof obligations were discharged in less that a minute on a standard laptop. Finally, we have also verified some simple inductive invariant properties encoding the fact that the number of tokens in the Petri net remains constant.

**7. Conclusions.** To conclude the paper, let us briefly summarize our contributions. We have described automated formal analysis techniques for the validation of a class of web services specified in BPEL and

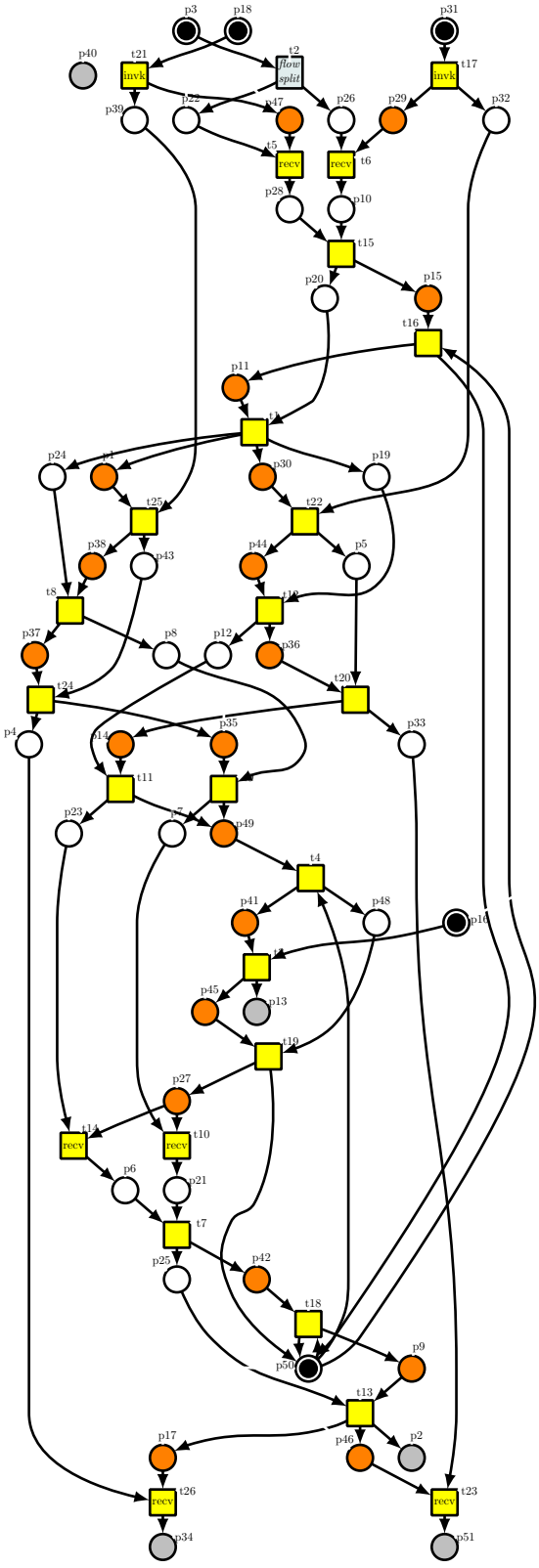



FIG. 6.2. *The Petri net representation of the WF level of the DCS case study. Legend: see Fig. 6.1 and, in addition, consider that orange places specify data exchange between instances of different BPEL processes.*

RBAC4BPEL. We also discussed our experience in using our prototype tool WSSMT on two industrial case studies, a loan origination process and a digital contract signing service, which have provided proof of concept of the flexibility of our specification and analysis framework, which allowed us to precisely capture the interplay between the workflow and the access-control level of the service. Throughout the paper, we have referred to related work and already mentioned several interesting directions for future work. In particular, besides for considering other case studies, we plan to extend our decidability results to WF nets containing restricted forms of loops and extensions of RBAC4BPEL with delegation, as well as to consider in more detail the interplay of the WF and PM levels with the data flow of the services under validation.

**Acknowledgment.** The work presented in this paper was partially supported by the FP7-ICT-2007-1 Project no. 216471, "AVANTSSAR: Automated Validation of Trust and Security of Service-oriented Architectures" and the "Automated Security Analysis of Identity and Access Management Systems (SIAM)" project funded by Provincia Autonoma di Trento in the context of the "Team 2009 - Incoming" COFUND action of the European Commission (FP7).

## REFERENCES

[1] A. Alves et al Web Services Business Description Language, Version 2.0, OASIS Standard, April 2007. Available at `http://docs.oasis-open.org/wsbpel/2.0/wsbpel-v2.0.pdf`.

[2] A. Armando and S. Ponta. Model checking of security-sensitive business processes. In P. Degano and J. Guttman, editors, *Formal Aspects in Security and Trust*, volume 5983 of *Lecture Notes in Computer Science*, pages 66–80. Springer Berlin / Heidelberg, 2010. `10.1007/978-3-642-12459-4\_6`.

[3] AVANTSSAR. The AVANTSSAR Project. `http://www.avantssar.eu`.

[4] AVANTSSAR. Deliverable 5.1: Problem cases and their trust and security requirements, 2008.

[5] M. Barletta, A. Calvi, S. Ranise, L. Viganò, and L. Zanetti. WSSMT: Towards the Automated Analysis of Security-Sensitive Services and Applications. In *12th International Symposium on Symbolic and Numeric Algorithms for Scientific Computing*, pages 417–424, Los Alamitos, CA, USA, 2010. IEEE Computer Society. Extended version available at arXiv: `http://arxiv.org/abs/1009.4625`.

[6] M. Barletta, S. Ranise, and L. Viganò. Verifying the Interplay of Authorization Policies and Workflow in Service-Oriented Architectures. In *IEEE CSE'09, 12th IEEE International Conference on Computational Science and Engineering*, pages 289–296. IEEE Computer Society, 2009.

[7] M. Barletta, S. Ranise, and L. Viganò. A declarative two-level framework to specify and verify workflow and authorization policies in service-oriented architectures. *Service-Oriented Computing and Applications*, pages 1–33, 2010. 10.1007/s11761-010-0073-4.

[8] A. Bockmayr and V. Weispfenning. *Handbook of Automated Reasoning, volume I*, chapter Chap. 12 "Solving numerical constraints", pages 751–842. Elsevier Science B.V., 2001.

[9] A. Calvi, S. Ranise, and L. Viganò. Automated Validation of Security-sensitive Web Services specified in BPEL and RBAC. In *12th International Symposium on Symbolic and Numeric Algorithms for Scientific Computing*, pages 456–464, Los Alamitos, CA, USA, 2010. IEEE Computer Society. Extended version available at arXiv: `http://arxiv.org/abs/1009.4625`.

[10] E. Christensen, F. Curbera, G. Meredith, and S. Weerawarana. Web Services Description Language (WSDL) 1.1. `http://www.w3.org/TR/wsdl`.

[11] A. Deutsch, L. Sui, V. Vianu, and D. Zhou. Verification of communicating data-driven web services. In *Proceedings of the twenty-fifth ACM SIGMOD-SIGACT-SIGART symposium on Principles of database systems*, PODS'06, pages 90–99, New York, NY, USA, 2006. ACM.

[12] H. B. Enderton. *A Mathematical Introduction to Logic*. Academic Press, 1972.

[13] JFace. `http://wiki.eclipse.org/index.php/JFace`.

[14] S. Lahiri, S. Seshia, and R. Bryant. Modeling and Verification of Out-of-Order Microprocessors in UCLID. In M. Aagaard and J. OLeary, editors, *Formal Methods in Computer-Aided Design*, volume 2517 of *Lecture Notes in Computer Science*, pages 142–159. Springer Berlin / Heidelberg, 2002.

[15] N. Lohmann and C. Gierds and M. Znamirowski. BPEL2OWFN. Available at `http://www.gnu.org/software/bpel2owfn`.

[16] F. Paci, E. Bertino, and J. Crampton. An Access-Control Framework for WS-BPEL. *Int. Journal of Web Services Research*, 5(3):20–43, 2008.

[17] R. Piskac, L. de Moura, and N. Bjoerner. Deciding Effectively Propositional Logic Using DPLL and Substitution Sets. *J. of Autom. Reas.*, 44(4):401–424, 2010.

[18] S. Ranise and C. Tinelli. The satisfiability modulo theories library (smt-lib). `http://www.smt-lib.org`, 2009.

[19] R. Sandhu, E. Coyne, H. Feinstein, and C. Youmann. Role-Based Access Control Models. *IEEE Computer*, 2(29):38–47, 1996.

[20] S. Sankaranarayanan, H. Sipma, and Z. Manna. Petri Net Analysis Using Invariant Generation. In N. Dershowitz, editor, *Verification: Theory and Practice*, volume 2772 of *Lecture Notes in Computer Science*, pages 188–189. Springer Berlin / Heidelberg, 2004. 10.1007/978-3-540-39910-0_29.

[21] A. Schaad, K. Sohr, and M. Drouineaud. A Workflow-based Model-checking Approach to Inter- and Intra-analysis of Organisational Controls in Service-oriented Business Processes. Journal of Information Assurance and Security, 2(1): 55–67, 2007.

[22] H. Schlingloff, A. Martens, and K. Schmidt. Modeling and Model Checking Web Services. *ENTCS*, 126:3–26, 2005.
[23] SMT-COMP. The SMT-COMP web-site. `http://www.smtcomp.org`.
[24] SPASS. The SPASS web-site. `http://www.spass-prover.org`.
[25] C. Stahl. A Petri Net Semantics for BPEL. Technical report, Humboldt-Universität zu Berlin, Institut für Informatik, 2004.
[26] G. Sutcliffe. The TPTP web-site. `http://www.cs.miami.edu/~tptp`.
[27] SWT: The Standard Widget Toolkit. `http://www.eclipse.org/swt`.
[28] W. van der Aalst. The application of Petri nets to workflow management. *J. of circuits, systems, and computers*, 8(1):21–66, 1998.
[29] C. Weidenbach. Spass input syntax version 1.5. `www.spass-prover.org/download/binaries/spass-input-syntax15.pdf`.
[30] Z3 theorem prover. `http://research.microsoft.com/projects/z3`.
[31] L. Zanetti. *Integrazione di tecniche SMT in un sistema di verifica per applicazioni orientate ai servizi.* Master Thesis, Faculty of Mathematical, Physical and Natural Sciences, University of Verona, Italy, 2009.

*Edited by:* Dana Petcu and Alex Galis
*Received:* March 1, 2011
*Accepted:* March 31, 2011

Scalable Computing: Practice and Experience
Volume 12, Number 1, pp. 21–34. http://www.scpe.org
ISSN 1895-1767


# RESOURCE MANAGEMENT BASED ON GOSSIP MONITORING ALGORITHM FOR LARGE SCALE DISTRIBUTED SYSTEMS

FLORIN POP *

**Abstract.** The optimization of resource management in large scale distributed systems (LSDS) with the capability of self-organization is a complex process. LSDS are highly dynamic systems, with permanent changes in their configurations, as peers may join and leave the system with no restriction or control. This paper present the architecture for monitoring and resource management based on existing middleware solutions through the design of algorithms and methods inspired by natural models. The architecture is decentralized and it aims to optimize resource management in different types of distributed systems such as Grid, P2P, and Cloud. The important components considered for the architecture are: allocation of resources, task scheduling, resource discovery, monitoring resources and provide fault tolerance. As the system may have a large number of nodes, we need a scalable algorithm for monitoring process, able to guarantee a fast convergence no matter what the structure of the network is. In this context, gossip-based algorithms offer solutions for various topics in LSDS. The project aims to highlights the original obtained results in internationally scientific community. The paper presents the expecting results and discusses the performance evaluation of proposed architecture. Secondly, the paper presents a gossip-based algorithm for monitoring large-scale distributed systems and analyzes its efficiency in a simulated environment provided by OverSim.

**Key words:** Resource Management, Large Scale Distributed Systems, Self-organizing Systems, Optimization, Task Scheduling

**AMS subject classifications.** 68M14, 68M15, 68Q10

## 1. Introduction.

A distributed system consists of multiple independent computers that interact with each other in order to achieve a common goal and more importantly, it appears to its users like a single coherent system. Andrew S. Tanembaum and Marteen van Steen mention in [2] the four critical goals that should be met when developing a distributed system: first, it should provide an easy and efficient way to access its available resources, its users should not be aware of the fact that its resources are distributed across the network, it should be open and also, it should be scalable.

The property of scalability can have different meanings, depending on the component that is intended to scale. The possibility of adding new users and resources with minimal changes refers to size scalability, but if it is considered the locations of the system's nodes, it is necessary have to ensure geographic scalability. Finally, a system can also be scalable regarding its administration, so that independent administrative organizations can be easily managed and controlled.

Several performance problems may occur when trying to ensure distributed system scalability: users running centralized applications may overwhelm the unique server that handles their requests, centralized algorithms that overload the network with the huge number of messages sent, global synchronization issues for decentralized algorithms, unreliability of wide-area networks, conflicting policies for resource usage, management and security [2].

The resource management is a very important key in Large Scale Distributes Systems (LSDS). Distributed computing was developed in recent years in LSDS in response to challenges raised by complex problems solving and resource sharing in collaborative, dynamic environments. LSDS computing concerns large-scale interconnected systems and have the main purpose to aggregate and to efficiently exploit the power of widely distributed resources. In LSDS, load-balancing plays an essential role, in cases where one is concerned with optimized use of resources. A well balanced task distribution contributes to reducing execution time for jobs and to using resources, such as processors, efficiently, in the system. On the other hand, the problem of scheduling heterogeneous tasks onto heterogeneous resources is intractable, thus making room for good heuristic solutions. Concerning the platforms, heterogeneity refers to hardware, software, communication characteristics and protocols, network irregularities, etc.

Bio-inspired models have been used for the decentralized construction of Grid information systems with adaptive and self-organizing characteristics. In [18] A. Forestiero and collaborators present a self-organizing Grid *SO-Grid* that basically provides two functionalities: logical reorganization of resources inspired from the behavior of some species of ants which move and collect items within their environment and resource discovery based on the ants ability of searching food by following pheromone traces left by the others. Gossiping protocols

*Computer Science Department, Faculty of Automatic Control and Computers, University POLITEHNICA of Bucharest, Romania, Email: florin.pop@cs.pub.ro

are attractive bio-inspired techniques commonly used in large-scale distributed systems. Not only their robustness, flexibility and simplicity, but also their efficiency in spreading information within a group, make them very interesting for enhancing distributed systems with self-organizing capabilities. Moreover, Baker and Shostak described in [20] a gossiping protocol between ladies through telephones, protocol which was later applied for computers and networks. Even though most of the time, these algorithms are continuously processes, one of their major drawbacks is the high number of sent messages across the network. This transforms convergence into a critical property. Convergence refers to the number of rounds executed during the gossiping process in order to propagate the information within the whole group of peers. In the context of large-scale distributed systems, we must ensure that the algorithm designed for monitoring the peers maintains its scalability even when the size of the network is huge.

All the work presented in this paper is based on position paper [1]. We extend here the previous work with experimental results and experimental validation of monitoring gossiping algorithm for LSDS. The paper is structured as following. Section 2 presents an overview of Large Scale Distributes Systems and defines different metrics used for characterizing networks from a structural point of view. In Section 3 is presented the problem of Resource Management in LSDS and existing solutions. Section 4 presents Resource Management in LSDS. Monitoring process is presented in Section 5, describing a gossiping our protocol for the monitoring process of a peer-to-peer system. Section 6 describes system evaluation and experimental results. Section 7 highlights the conclusions and future work. Moreover we analyze and explain the results obtained for various test scenarios. In the end we draw the conclusions and we emphasize the impact of our research.

**2. Overview of Large Scale Distributes Systems.** The development of SORMSYS project (Resource Management Optimization in Self-Organizing Large Scale Distributes Systems) refers to many types of distributed systems, including Grid Computing Systems, Cloud Computing Systems, Massive Parallel Machine (MPM) and Peer-to-Peer Systems (P2P).

**Grid Computing**. Grid Computing represents a distributed paradigm dedicated to high performance computing that brings together resources from different organizations in order to facilitate the collaboration of a group of people. The major challenges that hide behind the concept of Grid are large-scale resource sharing coordination and the ability of developing innovative applications by using the advantages of a system with a high degree of heterogeneity. The key issue in a grid computing system refers to the notion of Virtual Organization (VO) which represents a group of people or institutions having a collaboration based on a set of rules that defines the terms and conditions of sharing direct access to computers, applications, other resources.

Ian Foster, Carl Kesselman and Steven Tuecke analyze in their work [3] the Grid problems and challenges and present an extensible architecture with respect to protocols, services, application programming interfaces (API) and software development kits [4]. Grid's architecture is composed of four layers that can be easily mapped into Internet layers (protocol architecture). Grid computing can be seen as an answer to drawbacks such as overloading, failure, and low QoS, which are inherent to centralized service provisioning in client-server systems. Such problems can occur in the context of high-performance computing, for example, when a large set of remote users accesses a supercomputer.

Centralized computing systems based on paradigms such as the client-server or the master-worker have to deal with many drawbacks including overloading of the central node, single point of failure, low performance and QoS. Domenico Talia and Paolo Trunfio describe Grid computing in [5] as a solution to all of these problems, especially in the context of high-performance applications when a large set of remote users gains access to a supercomputer.

Grid's primary goal is to ensure access to remote resources and for this purpose have been developed toolkits that provide secure services for submitting batch jobs or executing interactive applications on remote machines, but they also include mechanisms for efficiently sharing and moving data.

Resource discovery and management is an important point of interest in Grid technology. Current centralized models in which a certain node running a server application is responsible for storing and publishing information about a certain organization's node set are not capable to satisfy the requests coming from a more dynamic and large-scale distributed system. Considering the dynamic nature of a Grid environment, it is necessary to provide fault tolerance mechanisms based on decentralized P2P algorithms that do not involve critical failure points. Grid technology includes many different forms of computing:

- *Semantic and Service-Oriented Grid* - resources and services described using semantic data model.
- *Ubiquitous and Pervasive Grid* - the result of combination between mobile and wireless devices with

wired Grid infrastructure.
- *Data Grid* - the controlled sharing and management of large amounts of distributed data.
- *eScience Grid* - computing dedicated for scientific applications from biology, medicine, astronomy, physics, finance, weather forecast.
- *Enterprise Grid computing* - enterprise data centers, managed by a single business entity.
- *Autonomic Grid computing* - Grid systems with self properties.
- *Knowledge Grid* - manage knowledge resources used in VOs by interoperability among users, applications and resources.
- *Economy Grid* - development of economic or market based resource management and contributory systems.

**Cloud Computing Systems**. This section presents the Cloud Computing System with its architectural features and the most important characteristics that distinguish this paradigm from other computing technologies.

Lizhe Wang and Gregor von Laszewski study the anatomy of a cloud system in [6] and define it as a set of network enabled services, providing scalable, QoS guaranteed, normally personalized, inexpensive computing platforms on demand, which could be accessed in a simple and pervasive way. All the applications delivered as services over the Internet and the hardware and systems software in the datacenters that provide those services defines the concept of cloud computing. Michael Armbrust and collaborators [7] mention three important aspects regarding cloud computing from a hardware point of view: the illusion of infinite computing resources available on demand, the elimination of an up-front commitment by Cloud users and the ability to pay for use of computing resources.

Mladen A. Vouk introduces cloud computing in [11] as the evaluated solution for on-demand information technology services and products based on virtualized resources and states the main properties of a cloud computing system: service oriented architecture, reduced information technology overhead for the end-user, great flexibility and reduced total cost of ownership [10]. As the user is the most important entity, a Cloud System has a clear hierarchy that groups users in different categories: system developers responsible with Cloud infrastructure, developers or authors of component services and underlying applications, domain personnel who integrates basic services into composite services and delivers them to end-users, users of simple and composite services. As in the case of Grid and P2P computing, a cloud system provides an integrated computing platform as a service, in a transparent way:

- *Hardware as a Service* - users can buy IT hardware or even an entire data center. Examples of flexible, scalable and manageable are represented by Amazon EC2, IBM's Blue Cloud project, Nimbus, Enomalism.
- *Software as a Service* - software applications are implemented as services and are provided to users across the Internet. In this manner, users are no longer required to buy software or to install and run the needed application on their local machines. An interesting example is Google's Chrome browser that offers a new desktop, through which applications can be delivered besides the traditional web browsing experience.
- *Data as a Service* - users connected to a certain network have share the opportunity of accessing data from multiple sources via services and manipulating in a transparent way, as it would be located on their local disk. A simple Web services interface used for storing and retrieving data is provided by Amazon Simple Storage Service. Other examples can be found at Google Docs, Adobe Buzzword, ElasticDrive.

Moreover, through cloud computing, users can subscribe to their favorite computing platforms with requirements of hardware configuration, software installation and data access demands, this feature representing the concept of Platform as a Service.

The main features for cloud computing are:

- User-centric interfaces - users are not required to learn new APIs and commands to access resources and services, like they have to do in the Grid systems case.
- *On-demand service provisioning* - users gain access to resources and services.
- *QoS guaranteed* - the computing environments can guarantee CPU speed, I/O bandwidth and memory size.
- *Autonomous system* - the system's management is transparent to its users
- *Scalability and flexibility* - computing clouds can be easily scaled regarding the geographic locations, hardware performance or software configurations, but they are also flexible to adapt to a large number

of users.

**P2P Computing Systems**. Peer-to-peer systems gained a significant importance in the last years due to their capability of cooperating and forming a network without the existence of a central point. They are distributed systems with no centralized control, where each node accomplishes the same functionality [8]. This decentralized nature makes them extremely robust against certain failures and also well-suited for high-performance computations or long-term storage.

In contrast with the Grid systems, P2P systems have been developed in open communities, in which security mechanisms are not required and do not involve authentication and content validation [9]. Users of P2P systems have common goals such as retrieving music from the Internet, so they only need protocols that offer anonymity.

Regarding the connectivity aspect, P2P systems prove to have another important difference from the Grids, as they are composed mainly of desktop computers connected to the network for a limited period of time and with reduced reliability [12]. Grids, on the other hand have powerful machines, statically connected through high-performance networks with high level of availability. Moreover, the number of nodes connected in a P2P network at a certain time is much greater than in a Grid which has a very restrictive access to resources due to certain accounting mechanisms.

Grid's major goal was to provide access and management to remote resources, fact which is not respected in P2P systems since they do not handle remote cycle's allocations and storage. A key element in P2P systems is the presence-management protocol that allows each node to periodically notify its presence, discovering, in the same time, its neighbors. Since in Grid environments resource discovery in is based mainly on centralized or hierarchical models, further implementations should be inspired from this P2P decentralized resource discovery model [13]. Even though fault tolerance is a very important aspect in a distributed environment, this issue remains unexplored in grid models and tools. As it has mentioned before, Globus is able to provide fault detection, but developers should implement fault tolerance at the application level. The solution comes from decentralized P2P algorithms, which do not have to handle centralized services with single points of failure.

P2P Computing appeared as the new paradigm after client-server and web-based computing. P2P systems became quite popular for file sharing among Internet users through Napster, Gnutella, FreeNet, BitTorrent and other similar systems. Besides its great impact over file sharing applications, P2P systems also have a strong contribution in high parallel computing: SETI@home and FightAIDS@home are parallel applications running on available nodes.

Differently from centralized or hierarchical models of Grid systems, in P2P systems, nodes (peers) have equivalent capabilities and responsibilities and can be both servers and clients. These systems are evolving beyond file sharing being thus the basis for the development of P2P applications.

Moreover, P2P systems have inspired the emergence and development of social networking for enabling human interaction at large scale, which are having a tremendous impact on today's information societies. Since the appearance of the P2P systems [14], new forms of such paradigm has appeared, including B2B (Business to Business), B2C (Business to Consumer), B2G (Business to Government), B2E (Business to Employee), etc.

Self-organization gains more and more importance with the fast expansion of the distributed systems. The permanent changes and unexpected events occurrences are extremely difficult to handle globally due to the very large size of the environment which can cover a huge number of different geographic locations. Moreover, traditional mechanisms for resource management and fault tolerance become inefficient, as centralized models will no longer represent viable solutions. In this context, the need of an flexible, robust, adaptive and self-organizing environment is obvious.

**Coefficients used to characterize a network** as a support for LSDS are important for systems self-organization. In assortative networks, well-connected nodes tend to join to other well-connected nodes, as in many social networks (for instance, Facebook). This means that an assortative network has the property that almost all nodes with the same degree are linked only between themselves. On the other hand, disassortative networks have the property that well-connected nodes join to a much larger number of less-well-connected nodes, fact which is typical of biological networks. Previous work in the field of assortativity present different methods for generating correlated networks with predefined correlations. In [22], R. Brunet and I.M. Sokolov propose a different approach based on link-restructuring, also called *rewiring process*, with either the goal of connecting nodes with similar degrees in order to obtain an assortative mixing, or the goal of connecting nodes having low degree with nodes having high degree in order to obtain a dissassortative mixing.

The clustering coefficient **C** [21] represents another basic metric for measuring the internal structure of a network. It relates to the local cohesiveness of the network and measures the probability that two verticies with

a common neighbor are connected. Considering an undirected network, a certain node $N_i$ with $k_i$ neighbors, there are:

$$\frac{k_i \times (k_i - 1)}{2}$$

possible edges between the neighbors.

The clustering coefficient of that node is given by the ratio of the actual number of edges $E_i$ between neighbors and the maximal number:

$$\mathbf{C} = \frac{2 \times E_i}{k_i \times (k_i - 1)}$$

The global value of the clustering coefficient is obtained as the average cluster coefficient of all nodes. In the context of large-scale distributed systems, the last two metrics are very important, because the algorithms we design must be scalable with respect to the system's size and also they must ensure the robustness of the system, as failures may occur at any moment of time.

**3. Resource Management in LSDS.** Resource management must take into account additional issues such as resource consumer and owner requirements, the need to continuously adapt to changes in the availability of resources, etc. Based on this Grid characteristic, a number of challenging issues need to be addressed: maximization of system throughput and user satisfaction, the sites' autonomy (the Grid is composed of resources owned by different users, which retain control over them), scalability, and fault-tolerance.

The design process of a resource management system for a given system and implicitly of a scheduling algorithm must consider all aspects of the system and applications that will run in it. The resource management process in LSDS has the main function as scheduling. Basically, we have local and global scheduling. A *local scheduler* considers a single CPU (a single machine). *Global scheduling* is dedicated to multiple resources. Scheduling for distributed systems such as the Grid is part of the global scheduling class. For global scheduling three classes of scheduling algorithms have been designed: *centralized* (the scheduling decisions are made by one central module which runs the scheduling algorithm), *hierarchical* (based on the principle of work division) and *decentralized* (algorithms for cooperatively or independently working) [15].

Various strategies for resource management have been developed, in order to achieve optimized task scheduling in distributed systems. In the static scheduling model, every task is assigned only once to a resource. A realistic prediction of the cost of the computation can be made before the actual execution. The static model adopts a "global view" of tasks and computational costs. One of the major benefits is the easy way of implementation. On the other hand, static strategies cannot be applied in a scenario where tasks appear a-periodically, and the environment undergoes various state changes. Cost estimate do not adapt to situations in which one of the nodes selected to perform a computation fails, becomes isolated from the system due to network failures, is so heavily loaded with jobs that its response time becomes longer than expected, or a new computing node enters the system. These changes are possible in Grids. In dynamic scheduling techniques, which have been widely explored in literature, tasks are allocated dynamically at their arrival. Dynamic scheduling is usually applied when it is difficult to estimate the cost of applications, or jobs are coming on-line dynamically (in this case, it is also called online scheduling). Dynamic task scheduling has two major components: one for system state estimation (other than cost estimation in static scheduling) and one for decision making. System state estimation involves collecting state information through Grid monitoring and constructing an estimate. Decisions are made to assign tasks to selected resources. Since the cost for an assignment is not always available, a natural way to keep the whole system healthy is by balancing the loads of all resources.

Another criterion used to classify the schedulers is the way they perform the state estimation of the system. Some of the schedulers attempt to predict the load on the resources in the future or the execution time of the jobs, others take into account only the present information. However, the available information is always partial or stale, due to the propagation delay in large distributed systems.

Some of the schedulers provide a rescheduling mechanism, which determines when the current schedule is re-examined and the job executions reordered. The rescheduling taxonomy divides this mechanism in two conceptual mechanisms: periodic/batch and event-driven on line. Periodic or batch mechanism approaches group resource request and system events which are then processed at intervals that may be periodically triggered

by certain system events. The other mechanism performs the rescheduling as soon the system receives the resource request [16].

The scheduling policy can be fixed or extensible. The fixed policies are system oriented or application oriented. The extensible policies are ad-hoc or structured. In a fixed approach, the policy implemented by the resource manager is predetermined. Extensible policy schemes allow external entities the ability to change the scheduling policy.

Genetic algorithms (GAs) have been widely used to solve difficult NP complete problems like scheduling problem. A genetic algorithm for scheduling independent tasks in Grid environment was developed. It can increase search efficiency with a limited number of iterations by improving the evolutionary process while meeting a feasible result. A fault tolerance-genetic algorithm for Grid task scheduling using check point was proposed. Another genetic algorithm based scheduler for computational grids is designed to minimize make-span, idle time of the available computational resources, turn-around time and the specified deadlines provided by users. The architecture is hierarchical and the scheduler is usable at either the lowest or the higher tiers. It can also be used in both the intra-grid of a large organization and in a research Grid consisting of large clusters, connected through a high bandwidth network.

Considering all the issues presented on the management of resources in different types of distributed systems and considering the remaining issues still unresolved, the motivation for the SORMSYS project is to design and develop a full decentralized architecture to optimize management of resources in large distributed systems. The need to optimize resources management is maintained by increasing the number of users and applications and their types. Thus, the main problem for the management of resources is to develop architecture with meta-scheduling capabilities aimed the resource dynamic comportment and heterogeneity. In terms of interdisciplinary, SORMSYS project considers self-organization distributed systems and use of nature-inspired techniques and methods for developing algorithms for resource allocation, task scheduling, resource discovery, resource monitoring and ensuring fault tolerance.

**4. System Architecture.** SORMSYS project reaches a scalable and flexible environment, able to detect complex events like distributed failures and to make a system converge to a desired state. A new approach for the LSDS infrastructure, represented by the *Virtual Middleware* (VM) is proposed. The VM is placed on top of the actual middleware and is responsible for the self-management and self-organization of the system.

Originality of the solution proposed by SORMSYS project is to use existing methods in various types of distributed systems (Grid, P2P, Cloud, MPM, Cluster), but also inspired by nature for the optimization process to realize the resource management architecture. The impact of the project in this context consists of the increase of resource availability while preserving scalability of large scale distributed systems, the increase of the maintainability of distributed systems using an architectural model based on a minimal set of functionalities, the extension of the methods for testing of the performances using simulation and real environment in the analysis of reliability, availability, safety and security for large scale distributed systems.

**4.1. Components Analysis.** The system's organization is a multi-layered one, with three major architectural components (see Figure 4.1):

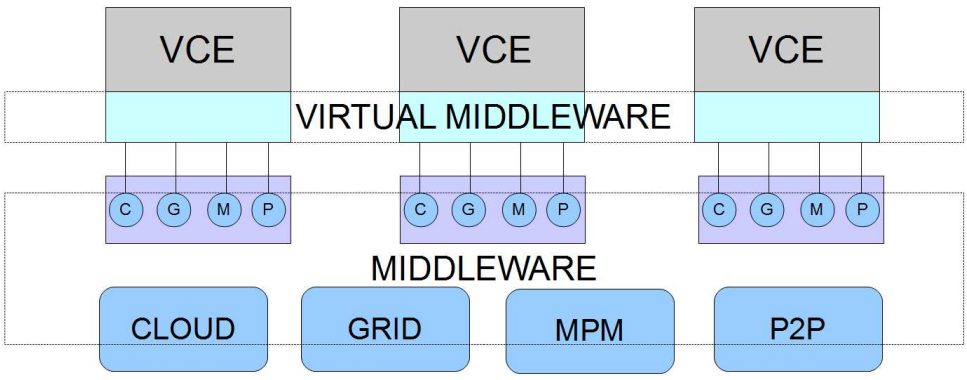



FIG. 4.1. *System Architecture.* ***C****loud,* ***G****rid,* ***P****2P,* ***M****PM*

- **Virtual Entity Manager**. The Virtual Controller Entity (VCE) is a piece of software running on each machine of the system that interacts with other similar entities in order to ensure an adaptive and self-organizing environment. Its implementation is based on bio-inspired and natural models such as ant computing and gossiping that are suitable in the context of a large-scale distributed system due to their good behavior in the presence of unexpected failures or other similar events.
- **Virtual Middleware** (VM). The VM is the layer that guarantees the collaboration and interoperability between VCEs, by providing communication channels and protocols necessary for their interactions. A complete description of this layer is offered in the next subsection.
- **Virtual Connectors**. The Virtual Connectors are responsible for providing the proper solutions for different types of distributed systems, according to their architectural structure and requirements: Grid, P2P, Massive Parallel Machine (MPM) or Cloud.

**4.2. Virtual Middleware in SORMSYS.** Middleware is a layer of software that connects users or applications working or running on different operating systems. It creates environments for developing systems that can be easily deployed on a large number of topologies and computing machines. This is achieved by providing necessary platforms and tools in order to monitor, validate, coordinate and manage resources, functions that actually ensure QoS for the running application and also users interoperability.

The VM of a large-scale distributed system is responsible for providing a self-organizing environment, with respect to any unexpected event that may occur. This VM can be viewed as an extension of the actual middleware, being mainly represented by the entities that control and monitor each node from the distributed system. The most important role of this layer is to ensure communication and interoperability between high level entities defined as VCEs that manage and control the whole system. Moreover, the services provided through decentralized gossip-based algorithms and other bio-inspired paradigms, prove the importance of the VM. Besides its strong connections with the agents running on top of it, the VM also provides connectors to the current available distributed systems technology: Cloud, Grid, P2P and MPM.

**4.3. Resources Management in SORMSYS.** A very difficult challenge in a large-scale distributed system is to realize the resources management. It is impossible to gather and maintain a detailed and up-to-date list of all nodes participating in any large computation. The solution comes from gossip-based algorithms that have an excellent behavior, proving very good convergence and resilience properties, in the present of continuous changes across the system.

Our first achievement in this research is a gossip based algorithm capable to monitor the nodes in terms of their performance, availability and reliability.

**5. SORMSYS Monitoring.** In the context of a large-scale distributed system, a major challenge is to discover each resource capabilities and to propagate them across the network. A resource's availability at a certain moment depends not only on its monitored parameters, but also it has to evaluate its neighbor's states. When running a distributed application, it is a critical requirement for the starting node to acknowledge the state of the resources from its neighborhood with respect to reliability and performance capabilities.

Gossip algorithms rely on simplicity, robustness and flexibility, features that make them suitable for a broad range of applications from data dissemination and aggregation [19] to overlay maintenance and resource allocation. Thus, the concept of gossiping is based on two key elements: repetition, as there is a continuous process of stochastic selection of two nodes and let them share information, and probabilistic choice which is involved in the selection process of the two nodes. The analogy with a real life behavior is obvious, as the information exchanged between two randomly chosen nodes can spread within a certain group. Moreover gossip-based information propagation can also be associated with the way a viral infection spreads in a biological population.

In the context of a large-scale distributed system, a major challenge is to be discover each node capabilities and to propagate them across the network. A node's processing availability at a certain moment depends not only on its parameters, but also it has to evaluate its neighbor's states.

When running a distributed application, it is a critical requirement for the starting node to acknowledge the state of the nodes from its neighborhood with respect to reliability and performance capabilities.

**5.1. Nodes Evaluation.** Each node is characterized by a set of parameters which describes its current state. The nodes are classified upon three key insights that are called the **ARP**-property:

1. **Availability** (A) - a node is considered available if it is not executing a task or if its current load is not too intensive. This property is also given as a percentage.
2. **Reliability** (R) - it is critical to have an evaluation of the nodes regarding their level of dependability, because unexpected failures must be avoided during a computation task. For this reason, the nodes are classified as following:
   - **Alive** - the node can be trusted and presents a very low risc of failure: $R = 1$ is the maximum value of this parameter indicates a great confidence in the node
   - **Dead** - the node is down and cannot be included in computation: $R = 0$ is the minimum value of this parameter and indicates a very low confidence in the node
   - **Transit** - a real value between 0 and 1 will indicate that a transit node is either overwhelmed due to some intensive computation, either in the process of recovering from a previous failure
3. **Performance** (P) - reflect the node's physical resources (CPU, memory, channels) and has three thresholds: *High*, *Medium* and *Low* given as real values between 0 and 1.

The node's evaluation is realized in respect to these three parameters by a function $f(A, R, P)$ which returns the ARP-value that is going to be associated with the node. There are several types of message that are going to be sent between system's nodes:

- **Gossip Request Message** (GRQ) - this message is sent by a node to its neighbors' with the goal of performing an evaluation of their ARP-property. This is always followed by a Gossip Reply Message sent to the initiator of the gossip.
- **Gossip Reply Message** (GRP) - this is the response that is received from neighbours with their parameters at a certain moment. At the first exchange it will contain only the records of the neighbor selected for gossiping.
- **Reliability Check Message** (RCM) - this message is sent by each node to its neighbors' in order to check if the node is still alive or not. It is useful for computing the reliability parameter from the ARP-property.

Both gossiping messages have the same layout, the only difference being a bit that indicates either a request or a reply - **GRx**. They also contain another two fields representing the peer that sent the message **SRC** and the round number registered at the moment when the message was created **RND**.

The last bytes store the gossip table **GT**, which is basically formed by the tuple $(Peer, ARP - value, Last\ Modified)$: the peer, its ARP-property and a bit that specifies if last round the recorded value has been modified. If the byte is unset, it is considered that the stored value is old and will have low priority during the process of solving conflicts between multiple GT. A proper field will also indicate the size of the transferred table.

**5.2. Decentralized Monitoring Algorithm.** In this paper is proposed a fully decentralized algorithm based on gossiping technique that ensures global monitoring in a distributed environment, with a large number of nodes.

The goal of this algorithm is to dynamically spread monitoring information about each node across the network, so that each node will estimate the ARP-value of the others.

Each node stores locally a *Gossip Table* that reflects its knowledge of the system in terms of availability, reliability and performance. The performance is measured according to the peer's physical resources and the availability is evaluated during the gossiping process. For reliability estimation, each node has to send a ping message at certain predefined *time samples* to each of its neighbors' and to update its current records from GT.

Peer-to-peer systems are highly dynamic systems, with permanent changes in their configurations, as peers may join and leave the system with no restriction or control. This makes monitoring an important element for several applications, esspecially for fault management: fault detection and fault recovery.

There are two major issues that occur in P2P systems, that represent important aspects for monitoring algorithm:

- first, as we consider large-scale distributed systems, the system's size, fact which influences the scalability and the efficiency of the algorithms that we design for managing the system.
  This gives the first requirement of any algorithm dedicated for such systems: it must be scalable with respect to the number of peers within the system.
- second, peer-to-peer systems are well-known for their dynamic behavior, as peers appear and disappear from the network all the time. This means, that the algorithm should be invariant to arbitrarly events.

Even though multiple nodes fail, or a considerable number of nodes join the system, the algorithm should be able to adapt itself in order to complete the same functions.

The algorithm is composed on several rounds, during which the nodes execute the following actions:

- *Step 1.* First, a certain node, randomly selects a gossiping partner, packs its knowledge existing in the gossiping table into a message labeled with GRQ and sends it to the selected peer.
- *Step 2.* For a predefined timeout, the node waits the reply from its partner. Based on the reply time, the initiator will evaluate the availability according to a certain pre-established threshold. In the mean time, the node can also receive other gossiping requests from its neighbors'.
- *Step 3.* When the timeout of current round expires, the node will have to handle the gossip tables received from its neighbors' during gossiping exchanges. It will solve the occurred conflicts, by storing in its local gossip table only the last modified records. After finalizing these updates, a new round can be performed.

The algorithm respects the scheme of a gossip algorithm, as it is based on both probabilistic choice for selecting the gossiping partner and on repetition, because several rounds are executed in order to ensure solution's convergence. This means that after a certain number of rounds, the monitoring information about each node is spread across the system and each node can evaluate the other peers in terms of availability, reliability and performance.

The number of rounds after the monitoring information from each peer have been propagated within the whole group of active represents the complexity of the algorithm.

**Algorithm 5.2.1** Active Thread on Peer $P_i$

```
P_j := peer within the network
map := hash table < peer, load >
f := computes the load of the peer based on its local parameters

P_j := selectRandomPeer()
local = f(w_1, w_2, ..., w_n)
map.put(P_i, local)
sendGossipRequest(P_j, map)

msg = receiveGossip(Q)
if (msg.type == REPLY) then
   map.update(msg)
end if
```

Each peer maintains a monitoring table that reflects the peer's local knowledge of the system it belongs to. Initially, this table contains only the monitoring information of the peer, but after several gossiping rounds it will be populated with data from different peers of the system. Each peer of the network executes two threads. The active thread, in which the peer play the role of the initiator, as it is the one that chooses a gossiping partner and sends it a request for exchanging information (algorithm 5.2.1). The passive thread, in which the peer simply waits for incoming gossiping requests and replies with its local knowledge (Algorithm 5.2.2). After each round, if new values have been exchanged, the table is updated and based on the current information, a peer can estimate the global load of the network:

$$\textbf{load} = \frac{\sum_{P_i} map.get(P_i)}{N}$$

**6. SORMSYS Evaluation and Use Cases for Experimental Tests.** Considering the topology in Figure 6.5 and the initial ARP-values in Figure 6.1, the behavior and the evolution of the gossiping algorithm can be described as following:

- $N_1$ selects the only neighbor it has, node 2 and sends a gossip request, which will be followed by a reply that is going to add a new entry in the local table.

**Algorithm 5.2.2** Background Thread on Peer $P_i$

```
P_j := peer within the network
map := hash table < peer, load >

msg = receiveGossip(Q)
if (msg.type = REQUEST) then
   sendGossipReply(Q, map)
   map.update(msg)
end if
```

| PEER | ARP |
|---|---|
| 1. | 0.5 |
| 2. | 0.6 |
| 3. | 0.7 |
| 4. | 0.3 |
| 5. | 0.2 |
| 6. | 0.2 |

FIG. 6.1. *Initial ARP Evaluation of Nodes*

- $N_2$ receives a request from $N_1$, adds a new entry according to the information it received and sends the reply message with its own data. It also has to initiate a gossip request, so let's assume that it selects $N_3$ to exchange information with.
- $N_3$ exchanges information with $N_2$ and sends a request message to a random neighbor: $N_4$. It will also receive requests from $N_5$ and $N_6$, so it will almost complete its table, except the first node.
- $N_4$ selects $N_3$, but they have already exchanged data, so no modifications appear.
- $N_5$ and $N_6$ send requests to $N_3$ and update their local tables with the replied data.

The algorithm converges towards a consistent solution after three rounds, when all peers will have information about each others. The complete evolution of the gossiping tables is illustrated in Figures 6.2, 6.3 and 6.4.

The peer number and its depth are colored to reflect the round they have been discovered by a certain node.

The overlay of the network in OverSim is randomly generated: each node that joins the system selects a random bootstrap node to connect with. The unique key associated with each node is determined based on the IP addresses as follows:

$$\mathbf{key} = IP\& \ (1 << 24)$$

After joining the overlay, each node sets up a timer that for scheduling a gossiping round. The gossips sent between peers are UDP messages of two types: REQUESTS and REPLIES.

After the timer expires, each node packs its current local knowledge and sends it through a gossip request message to a randomly selected peer within the system.

When receiving a gossip request, a node replies with its local knowledge and also updates its monitoring table with the information received. When receiving a gossip reply, a node will simply update its local knowledge.

For simulating multiple failures, we used a random churn model supported by OverSim.

Several tests were executed, with different network sizes and for each node was recorded the number of gossiping executed to reach a state where its local knowledge reflects the monitoring information of the whole network. Beside this, was encountered the total number of gossip rounds initiated by all peers and the average number of rounds after the algorithm converges to the desired state.

The number of gossips initiated by a node is represented by the number of rounds executed until its local monitoring table is completed.

As some of the peers reach the final state faster than other, we computed an average number of rounds using the weighted average.

| Peer | Depth | Round 1 | Round 2 | Round 3 |
|---|---|---|---|---|
| 2. | 1 | 0.6 | 0.6 | 0.6 |
| 3. | 2 | - | 0.7 | 0.7 |
| 4. | 2 | - | - | 0.3 |
| 5. | 3 | - | - | 0.2 |
| 6. | 3 | - | - | 0.2 |

| Peer | Depth | Round 1 | Round 2 | Round 3 |
|---|---|---|---|---|
| 1. | 1 | 0.5 | 0.5 | 0.5 |
| 3. | 1 | 0.7 | 0.7 | 0.7 |
| 4. | 1 | - | 0.3 | 0.3 |
| 5. | 2 | - | 0.2 | 0.2 |
| 6. | 2 | - | 0.2 | 0.2 |

FIG. 6.2. *Rounds evolution for nodes 1 and 2*

| Peer | Depth | Round 1 | Round 2 | Round 3 |
|---|---|---|---|---|
| 1. | 2 | - | 0.5 | 0.5 |
| 2. | 1 | 0.6 | 0.6 | 0.6 |
| 4. | 1 | 0.3 | 0.3 | 0.3 |
| 5. | 1 | 0.2 | 0.2 | 0.2 |
| 6. | 1 | 0.2 | 0.2 | 0.2 |

| Peer | Depth | Round 1 | Round 2 | Round 3 |
|---|---|---|---|---|
| 1. | 2 | - | 0.5 | 0.5 |
| 2. | 1 | - | 0.6 | 0.6 |
| 3. | 1 | 0.7 | 0.7 | 0.7 |
| 5. | 2 | - | - | 0.2 |
| 6. | 2 | - | - | 0.2 |

FIG. 6.3. *Rounds evolution for nodes 3 and 4*

Considering **N** the size of the network, **x** the dimensions of each group of nodes that complete the monitoring process in the same amount of time and **w** the number of rounds associated with each group of nodes.

$$\textbf{rounds} = \frac{\sum_{k}^{i=0} w_i \times x_i}{N}$$

For a network of 100 nodes, 3 of them reach the final state in 5 rounds, 55 in 6 rounds, 39 in 7 rounds and 3 in 8 rounds. The total number of gossips is 642 and the average number of rounds executed to ensure algorithm's convergence is 6.42.

In the case of 200 nodes, 6 rounds are required for 14 nodes to receive monitoring data from the whole system, 7 rounds for 141 nodes and 8 rounds for the rest of 45 nodes. The total number of gossips is 1431 and the average number rounds is 7.15.

With a network of 300 nodes, only 1 node completes the gossiping process in 6 rounds, 118 nodes in 7 rounds, 165 in 8 rounds and 16 nodes in 9 rounds. In this situation, he total number of gossips is 2296 and the average number rounds is 7.65.

For a network of 400 nodes, 130 of them reach the final state in 7 rounds, 250 in 8 rounds and 20 in 9 rounds. The total number of gossips is 3090 and the average number rounds is 7.72.

Considering a network of 500 nodes, 96 nodes received the complete set of monitoring data in 7 rounds, other 325 nodes in 8 rounds and the rest of 78 in 9 rounds. The total number of gossips is 3974 and the average number rounds is 7.95.

Finally, for a group of 1000 nodes, 7 nodes converged in 7 rounds, 485 nodes in 8 rounds, 488 nodes in 9 rounds and 20 nodes in 10 rounds. The total number of gossips is 8521 and the average number rounds is 8.52.

Figure 6.6 illustrates the evolution of algorithm's convergence for nodes belonging to networks of various sizes, while Figure 6.7 presents the average convergence rounds of the algorithm.

**6.1. Local ARP Estimation.** When selecting the node to become gossiping partner, the arbitrary selection will be eliminated and a more accurate approach is provided. For this, each node will associate to its neighbors a confidence parameter expressing the capacity of each subset of nodes dominated by each neighbor. The network can be viewed as a graph, so the previous statement is equivalent to: a node's confidence is calculated based on the capacity of each sub-graph determined by its children.

This value is important for scheduling a distributed application, because it estimates the capability of a certain set of nodes to execute a pool of tasks. In this context, the pre-defined *TIMEOUT* is now used to limit

| Peer | Depth | Round 1 | Round 2 | Round 3 |
|---|---|---|---|---|
| 1. | 3 | - | - | 0.5 |
| 2. | 2 | - | 0.6 | 0.6 |
| 3. | 1 | 0.7 | 0.7 | 0.7 |
| 4. | 2 | - | 0.3 | 0.3 |
| 5. | 2 | - | 0.2 | 0.2 |

| Peer | Depth | Round 1 | Round 2 | Round 3 |
|---|---|---|---|---|
| 1. | 3 | - | - | 0.5 |
| 2. | 2 | - | 0.6 | 0.6 |
| 3. | 1 | 0.7 | 0.7 | 0.7 |
| 4. | 2 | - | 0.3 | 0.3 |
| 5. | 2 | - | 0.2 | 0.2 |

FIG. 6.4. *Rounds evolution for nodes 5 and 6*

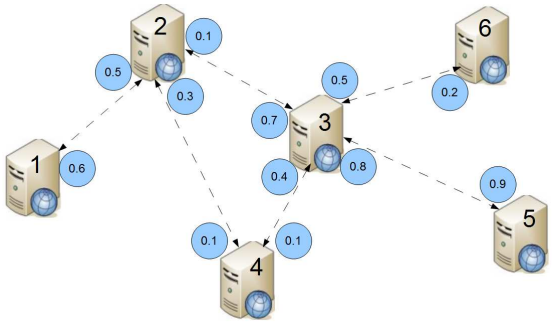



FIG. 6.5. *P2P Topology with Confidence Parameters*

the gossiping area for confidence computation. This is the reason to call it a local algorithm, because it returns a value that characterizes only a subset of nodes.

A node will locally store its confidence values associated for each connection it has with its neighbors. The *getConfidence(P, N)* function computes the confidence of a certain node $N$ in one of its neighbors $P$ according to the formula:

$$confidence(P,N) = \frac{1}{n}\sum confidence(child), child \neq P \tag{6.1}$$

**6.2. Hybrid Monitoring.** Now, considering the fact that some nodes might not be considered *interesting* from a distributed computation point of view, their neighbors should not choose them as gossiping partners. Thus, the probability of selecting a peer must be calculated according to the *confidence* that the node should have in that peer. The confidence was introduced above and is going to be the key element in the selecting phase of the monitoring algorithm.

This feature is related to ants behavior, as they are used to follow pheromone traces in their path towards food. In a similar way, the most persistent traces (the highest confidence) are going to indicate the selected peer.

Due to the fact that two bio-inspired approaches are combined, starting from a gossip-based model and adding ant colony features, proves that this algorithm is a hybrid one. It is relevant for optimizing the monitoring process, by focusing only on possible available nodes and ignoring those peers that are very unlikely to be able to participate in a distributed computing.

**7. Conclusion.** Developing the Virtual Middleware proposed by SORMSYS project highlight a new approach over the infrastructure of LSDS. SORMSYS project uses bio-inspired methods like gossip-based algorithms with ant colony features that are meant to ensure resource management and system monitoring. The advantages and the impact that gossip-based algorithms have over the LSDS will be analyzed. Moreover, the adaptive nature of the protocol, allows it to spread the information across the system, with no performance penalties, even when a high number of nodes join or leave the system. The excellent behavior of the monitoring algorithm is also strengthened by the results obtained in OverSim when monitoring systems of different sizes: from 100 to 1000 nodes.

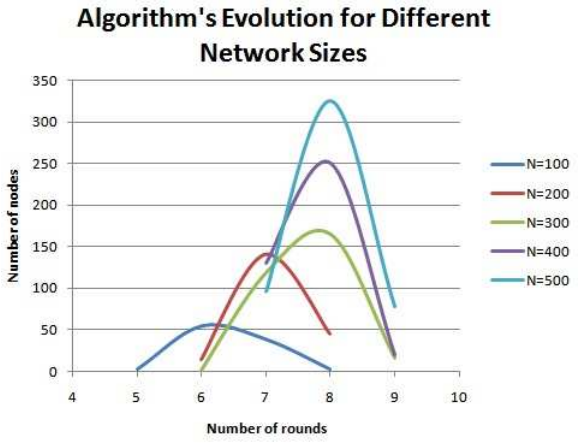



FIG. 6.6. *Number of rounds executed by nodes for different network sizes*

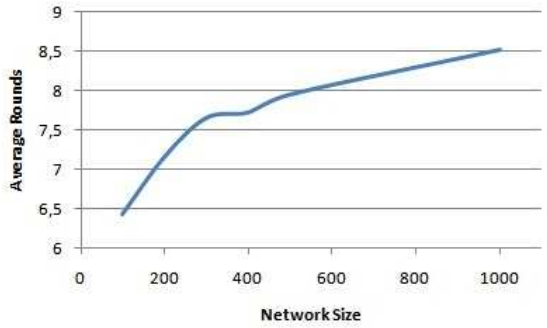



FIG. 6.7. *Average Rounds for Different Network Sizes*

Considering the fact that each node is limited as computing capacity in comparison with the system's size, the real power of computation is going to be the system as a whole, not the individual machines. In such an environment, centralized models based on master-worker or client-server paradigms are not compliant, as the huge number of requests coming from different peers will most probably overwhelm any central point considered. As a centralized model is not acceptable in the specified conditions, the future work will focus on fully-decentralized models that provide scalable solutions for monitoring and controlling the environment. The system will reach a high level of knowledge regarding its state in any node, so that it will be able to self-organize in order to deal with unexpected events. In this LSDS, it will be impossible to gather and maintain a detailed and up-to-date list of all nodes participating in any large computation. Gossip-based algorithms are known to exhibit a very good convergence and resilience properties in the present of constant unexpected events.

**Acknowledgment.** The research presented in this paper is supported by national project: "SORMSYS - Resource Management Optimization in Self-Organizing Large Scale Distributes Systems", Project CNCSIS-PN-II-RU-PD ID: 201 (Contract No. 5/28.07.2010).

REFERENCES

[1] FLORIN POP. *SORMSYS: Towards to a Resource Management Platform for Self-Organizing Large Scale Distributed Systems*, Proc. of SYNASC 2010, the12th International Symposium on Symbolic and Numeric Algorithms for Scientific Computing, Timisoara, Romania, September 23-26, 2010, pp: 534-540, ISBN:3-642-14798-4.

[2] A.S. TANENBAUM AND M. VAN STEEN. Distributed systems. Prentice Hall; 2 edition (Oct 12 2006), ISBN: 978-0132392273

[3] FOSTER, I., KESSELMAN, C., AND TUECKE, S.. The Anatomy of the Grid: Enabling Scalable Virtual Organizations. Int. J. High Perform. Comput. Appl. 15, 3 (Aug. 2001), 200-222.

[4] PEREIRA, A. L., MUPPAVARAPU, V., AND CHUNG, S. M.. Role-Based Access Control for Grid Database Services Using the Community Authorization Service. IEEE Trans. Dependable Secur. Comput. 3, 2 (Apr. 2006), 156-166.

[5] Trunfio, P., Talia, D., Papadakis, H., Fragopoulou, P., Mordacchini, M., Pennanen, M., Popov, K., Vlassov, V., and Haridi, S.. Peer-to-Peer resource discovery in Grids: Models and systems. Future Gener. Comput. Syst. 23, 7 (Aug. 2007), 864-878.

[6] Wang, L., Tao, J., Kunze, M., Castellanos, A. C., Kramer, D., and Karl, W.. Scientific Cloud Computing: Early Definition and Experience. In Proceedings of the 2008 10th IEEE international Conference on High Performance Computing and Communications (September 25 - 27, 2008). HPCC. IEEE Computer Society, Washington, DC, 825-830.

[7] M. Armbrust, A. Fox, R. Grith, A.D. Joseph, R.H. Katz, A. Konwinski, G. Lee, D.A. Patterson, A. Rabkin, I. Stoica, et al. Above the clouds: A berkeley view of cloud computing. EECS Department, University of California, Berkeley, Tech. Rep. UCB/EECS-2009-28, 2009.

[8] Liben-Nowell, D., Balakrishnan, H., and Karger, D.. Analysis of the evolution of peer-to-peer systems. In Proceedings of the Twenty-First Annual Symposium on Principles of Distributed Computing (Monterey, California, July 21 - 24, 2002). PODC '02. ACM, New York, NY, 233-242.

[9] Suomalainen, J., Pehrsson, A., and Nurminen, J. K.. A Security Analysis of a P2P Incentive Mechanisms for Mobile Devices. In Proceedings of the 2008 Third international Conference on internet and Web Applications and Services(June 08 - 13, 2008). ICIW. IEEE Computer Society, Washington, DC, 397-402.

[10] Wang, S. C., Yan, K. Q., Wang, S. S., and Huang, C. P.. Achieving high efficient agreement with malicious faulty nodes on a cloud computing environment. In Proceedings of the 2nd international Conference on interaction Sciences: information Technology, Culture and Human (Seoul, Korea, November 24 - 26, 2009). ICIS '09, vol. 403. ACM, New York, NY, 468-473.

[11] M.A. Vouk. Cloud computing - Issues, research and implementations. Journal of Computing and Information Technology, 16(4):235-246, 2008.

[12] Yu, J., Li, Z., Hu, J., Liu, F., and Zhou, L.. Structural Robustness in Peer to Peer Botnets. In Proceedings of the 2009 international Conference on Networks Security, Wireless Communications and Trusted Computing - Volume 02 (April 25 - 26, 2009). NSWCTC. IEEE Computer Society, Washington, DC, 860-863.

[13] Gehlen, G., Aijaz, F., Zhu, Y., and Walke, B.. Mobile P2P Web Services using SIP. Mob. Inf. Syst. 3, 3,4 (Dec. 2007), 165-185.

[14] Buchegger, S., Schiberg, D., Vu, L., and Datta, A.. PeerSoN: P2P social networking: early experiences and insights. InProceedings of the Second ACM Eurosys Workshop on Social Network Systems (Nuremberg, Germany, March 31 - 31, 2009). SNS '09. ACM, New York, NY, 46-52.

[15] M. Arora, S.K. Das, and R. Biswas. A decentralized scheduling and load balancing algorithm for heterogeneous Grid environments. In Proceedings of International Conference on Parallel Processing Workshops (ICPPW'02), Vancouver, British Columbia Canada, pages 499 - 505, 2002.

[16] Florin Pop, Ciprian Dobre, and Valentin Cristea. Performance analysis of Grid dag scheduling algorithms using monarc simulation tool. Proceedings of 7th International Symposium on Parallel and Distributed Computing (ISPDC'08), July 1-5, Krakaw, Poland, 2008.

[17] Forestiero, A., Mastroianni, C., and Spezzano, G.. So-Grid: A self-organizing Grid featuring bio-inspired algorithms.ACM Trans. Auton. Adapt. Syst. 3, 2 (May. 2008), 1-37.

[18] Forestiero, A., Mastroianni, C., and Meo, M.. Self-Chord: A Bio-inspired Algorithm for Structured P2P Systems. InProceedings of the 2009 9th IEEE/ACM international Symposium on Cluster Computing and the Grid (May 18 - 21, 2009). CCGRID. IEEE Computer Society, Washington, DC, 44-51.

[19] Jan Sacha, Jeffrey Napper, Corina Stratan and Guillaume Pierre. Adam2: Reliable Distribution Estimation in Decentralized Environments, 30th International Conference on Distributed Computing Systems (ICDCS 2010), Genoe, Italy, June 2009

[20] B. Baker and R. Shostak. Gossips and telephones. *Discrete Mathematics*, 2(3):191–193, 1972.

[21] B.H. Junker and F. Schreiber. *Analysis of biological networks*. Wiley-Interscience, 2008.

[22] R. Xulvi-Brunet and IM Sokolov. Construction and properties of assortative random networks. *Arxiv preprint cond-mat/0405095*, 2004.

*Edited by:* Dana Petcu and Alex Galis
*Received:* March 1, 2011
*Accepted:* March 31, 2011

Scalable Computing: Practice and Experience
Volume 12, Number 1, pp. 35–49. http://www.scpe.org
ISSN 1895-1767


# MANAGEMENT OF ACCESS CONTROL IN INFORMATION SYSTEM BASED ON ROLE CONCEPT

ANETA PONISZEWSKA-MARANDA*

**Abstract.** Development of technology, progress and increase of information flow have the impact also on the development of enterprises and require rapid changes in their information systems. The growth and complexity of functionality that they currently should face cause that their design and realization become the difficult tasks and strategic for the enterprises at the same time. The informations systems store huge amount of data and allow to realize thousands of operations and business transactions on these data each day. In this case, it seems necessary to have the methods, techniques and tools that can make possibly the development of information system on level reflecting currently requirements.

The paper describes the aspects of access control management in information systems based on the concepts of roles. This concepts can be presented by the role-based access control model and its extensions defined during last years. The practical implementation of presented concepts was given in the form of platform for access control management that can be used by system developers and security administrators to support their job in assuring the security of data stored and processed in an information system and assuring the global coherence of access control rules in the whole system.

The proposed platform was based mainly on the approach connected with the access control model based on the role concept that reflects in the better way the company's organization on the access control level. The platform can be enrich with additional tool for access control administration with the use of other access control models.

**Key words:** security of information systems, access control management, access control models, role-based access control

**AMS subject classifications.** 68N02, 68U02

**1. Introduction.** Development of technology, progress and increase of information flow have the impact also on the development of enterprises and require rapid changes in their information systems. The growth and complexity of functionality that they currently should face cause that their design and realization become the difficult tasks and strategic for the enterprises at the same time. The informations systems store huge amount of data and allow to realize thousands of operations and business transactions on these data each day. In this case, it seems necessary to have the methods, techniques and tools that can make possibly the development of information system on level reflecting currently requirements.

It is important also for an enterprise to develop the security system that secure the information system against external threats. Very important stage of data protection building in information system is the creation of high level model, independent from the software, satisfying the needs of protection and security of a system. One of the basic concepts of protection models is access control. The purpose of access control to data in information system is a limitation of actions or operations that the system's users can execute. The access control based on role concept represents interesting alternative in relation to traditional systems of DAC (Discretionary Access Control) type or MAC (Mandatory Access Control) type. RBAC (Role-Based Access Control) model based on a role concept defines the user's access to information basing on activities that the user can perform in a system.

Security policies of information systems determine that it is necessary to define for each user a set of operations that it could be perform. Due to it the set of permissions should be defined for each system's user. It suffice to determine the permissions for execution of particular methods on each object accessible for that user. It is exists the need to create the tool, designated mainly for security administrator who could manage one of the security aspects of information systems, namely the control of users' access to data stored in a system.

To create the administration tool the access control model based on role concept in extended version (extended RABC) was chosen. It is necessary to deliver the tool allowing the definition of access control rules for any information system. It is exists the need to ensure the integrity of defined early access control rules in situation when we want to extend the existing information system by new components (i.e. applications). It is also necessary take the attention that two actors were distinguished in the design process of an information system and its associated security scheme: application/system developer and security administrator who cooperate with each other to define and apply the set of roles defined for particular system's users in according with security constraints assuring the global security strategy of an enterprise. The paper describes the platform created for access control management in field of information systems that can be used by these two actors

*Institute of Information Technology, Technical University of Lodz, Poland, anetap@ics.p.lodz.p

(i.e. application/system developer and security administrators) fulfilling their responsibilities in assuring the security of data stored and processed in information system and assuring the global coherence of access control on the global level.

The proposed platform was based mainly on the approach connected with the access control model based on the role concept that reflects in the better way the company's organization on the access control level. The created platform was based on the extended RBAC model [9] that provides the developers more flexibility and complex view of the organization security. However, the additional tool for administration of access control of information systems using the traditional access control models, as DAC model and classical RBAC model was added to this platform.

The paper is structured as follows. The first part presents the outline of access control approach and access control models, especially the extended RBAC model. The second part deals with the partition of responsibilities in creation process of access control rules realized by two main actors and describes in short the role engineering in creation of security schema, based on the extended RBAC model. The last part presents our proposition of a platform for management of access control in information systems and administration tool for access control in information systems with the use of DAC model and classical RBAC model.

**2. Access control and access control model based on role concept.** Access control represents one of the components of information system security, named logical security. Logical security contains three mutually supportive technologies that can be used to provide the system security: authentication, access control and audit. However, access control is the most important technique on logical security level and it is used frequently.

Access control allows to define the user's responsibilities and possibilities in a system. It can define what a user can do directly and also what programs executing on behalf of the user are allowed to do. Access control limits the activities of successfully authenticated users basing on the security constraints defined on the conception level and on the administration level. Access control approach consists of two components [1]:

- set of access policies and access principles that determine the possible access of system's users to data and information stored in a system using the access modes and
- set of control procedures (security mechanisms) that allow to verify the access requests sent by system's users in agreement with defined principles and rules; these access requests may be allowed, denied or modified.

The logical security system can contain two components that cooperate with each other to assure the global security of information systems:

- strategy of logical security that determines all the environments and specifications of the entire organization on the security level and
- access model with:
  - set of concepts to describe the system objects (data access) and system subjects (users),
  - definition of the users' rights to access the data,
  - access control policy that describes how users can manipulate data, defines data structure and manages the users' rights to access the data.

Three main access control policies were defined and used in practice during last years: Discretionary Access Control (DAC), Mandatory Access Control (MAC) and Role-Based Access Control (RBAC).

Discretionary Access Control (DAC) [1, 13, 14] manage the access of users to the information basing on user's identity and authorizations or rules that specify. The access to the object in the specific mode is granted only to subjects for which an authorization rule exists in ACL (Access Control List) and was verified. However, DAC has the weakness that information can be copied from one object to another and it is difficult for DAC to enforce the safety policy and protect the data against some security attacks.

The second access control policy is Mandatory Access Control (MAC) that was defined to enforce the lattice-based policies [1, 15]. MAC strategy manges the accesses to the data basing on the classification of subjects and objects in the system. Mandatory policy can also be defined as a flow-control policy because it prevents information flow towards objects of a lower classification.

Access control based on the concept of role - Role-Based Access Control (RBAC) [2, 3, 4] is an interesting alternative to the traditional approaches of DAC type or MAC type [1]. RBAC model requires the identification of roles in a system. The role is viewed as a main semantic structure around which the access control policy is formulated. The role can represent the user responsibility, the competency to do a specific task and it can embody the authority of system users. The roles can be defined for different job functions in an organization

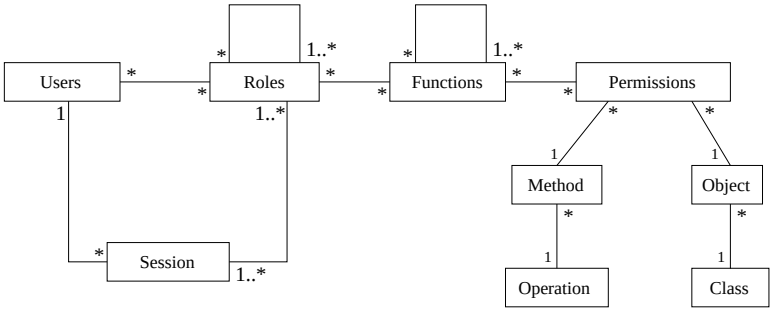



Fig. 2.1. *Extended RBAC model*

and the users can be assigned to the roles basing on their responsibilities and qualifications.

The RBAC model is very popular and stable in access control domain of information system security and for that reason it was extended during our studies. The obtained extension of RBAC model was next used as a base for creation of access control platform in field of access control management.

During our previous studies, we decided to extend the classical RBAC model. The reason for such extension was the need of better expressing the enterprise organization on the access control level, their relationships and dependences. In *extended RBAC (eRBAC)* model [9, 11, 22] each role realizes a specific task in the enterprise process and it contains many functions that the user can take and perform. It is possible to choose for each role the necessary system's functions. Thus, a role can be presented as a set of functions that this role can take and realize. Each function can be determined by one or more permissions that specify the possibilities of a function in a system. Therefore, a function can be defined as a set or sequence of permissions. The addition of function concept caused the necessity of new added relationships between the elements of model: R-F relations (i.e. relation between role an function), F-F relation (i.e. inheritance relation between functions) and F-P relations (i.e. assignment of permissions to a function).

If an access to an object is required, then the necessary permissions can be assigned to the function to complete the desired job. Specific access rights are necessary to realize a role or a particular function of this role. The permission determines the execution right for a particular method on the particular object. In order to access the data, stored in an object, a message has to be sent to this object. This message causes an execution of particular method on this object.

In extended RBAC model, the permission was presented as a function $p(o, m, c)$ where$o$ is an object, $m$ is a method which can be executed on this object and $c$ is a set of constraints which determine this permission.

Therefore, the extended RBAC mode is based on classical RBAC model and extended by addition of some elements, i.e. function, object, method, class, operation, to express more complex the elements of company's information system that are secured by these model (Fig. 2.1).

**3. Two actors in role engineering of information system security.** Two types of actors cooperate in the design and realization of security schema of information system [11]: on the one hand it is application/system developer who knows its specification that should be realized and on the other hand it is security administrator who knows the general security constraints that should be taken into consideration on the company level. We propose the partition of responsibilities between these two actors and their cooperation in order to fix the global access control (i.e. security schema) that fulfill the concepts of extended RBAC model. This partition of responsibilities is presented in figure 3.1 and they are divided into two stages: conception stage and exploitation stage.

**3.1. Conception stage.** The realization process of information system or simple application is provoked by a client's request or in general by client's needs to create a new information system or new application (i.e. to add a new component to the existing information system). Basing on the client's needs and requirements the application/system developer create the logical model of application/system, next define the project of this system that will be the base for its implementation. This model and next the project contain all the elements expressing the user's needs. These elements can be also presented in a form adequate to the access control concepts - it will be the extended RBAC model in our case. Therefore, the developer generates the sets of following elements: roles, functions, permissions and constraints. These sets of elements should be presented

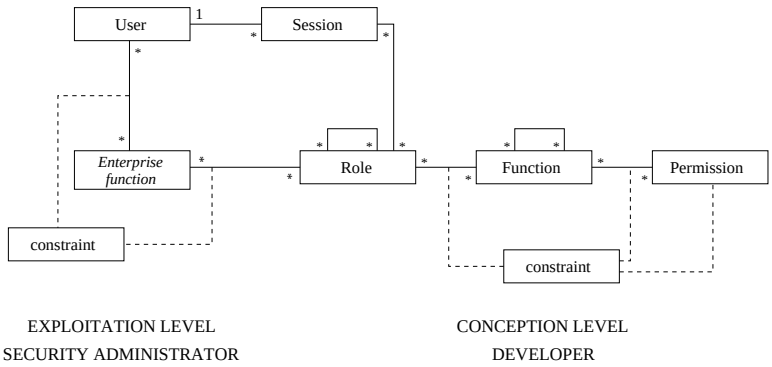



FIG. 3.1. *Two actors in creation of information system security - application developer and security administrator*

to the security administrator in a useful and legible form. The tasks of application developer basing on the extended RBAC model can be determined as follows:

- assignment of elements: permissions to functions and functions to roles,
- definition and setting up the security constraints associated to the elements of this application.

**3.2. Exploitation stage.** Next, the security administrator defines the administration rules and company constraints according to the global security policy and application/system rules received from the developer. He should also check if these new security constraints remain in agreement with the security constraints defined for the elements of existing information system in order to guarantee the global coherence of the new information system.

The security administrator received from the developer the sets of elements in the form adequate to extended RBAC model: set of of roles, set of functions, set of permissions and set of security constraints of the application. He uses these sets to mange the global security of the information system. First of all he defines the users' rights to use the particular applications. Two sets on company level are important to define the users' rights: persons working for the enterprise (users) and functions realized in the enterprise (enterprise functions).

An *enterprise function* is a task or a set of tasks that can be realized in a system by the same person. If the tasks of an enterprise function require the cooperation between two or more persons this function is realized by these persons taking into consideration the constraints defined on such function that specify the additional conditions for its execution [22].

The main task of security administrator is to assign the users to roles based on the relations between the enterprise functions and the roles (Figure 3.1). This assignment is based on user's responsibilities in the organization. Security administrator is also responsible for the definition and assignment the security constraints on global level. These constraints concern first of all the following relations: user-enterpriseFunction, enterpriseFunction-enterpriseFunction and enterpriseFunction-role.

**3.3. Role engineering in creation of security schema.** The most important stage in role engineering process of security schema creation is the complex and proper identification and definition of set of roles in the application or in the whole information system form the point of view of RBAC/eRBAC model. The role engineering process of the security scheme in information system using the RBAC/eRBAC model is proposed as follows (Fig. 3.2) [22]:

- Application/system developer creates the system application or a set of system applications (i.e. logical model, project) using object-oriented methodology (e.g. Unified Modeling Language - UML) for analyses and design of information systems. These methodology is used to define the application Model containing all elements that express the needs and requirements of users.
- Application developer initiates the process of user profile creation (e.g. role engineering) [11] based on the security rules concerning this application.
- The application Model created by application/system developer is translated into the concepts of access control models (i.e. RBAC or eRBAC) based on the connections of UML concepts with the concepts of RBAC/eRBAC model [10, 11]. Also the process of user profile creation is finished on the developer level.

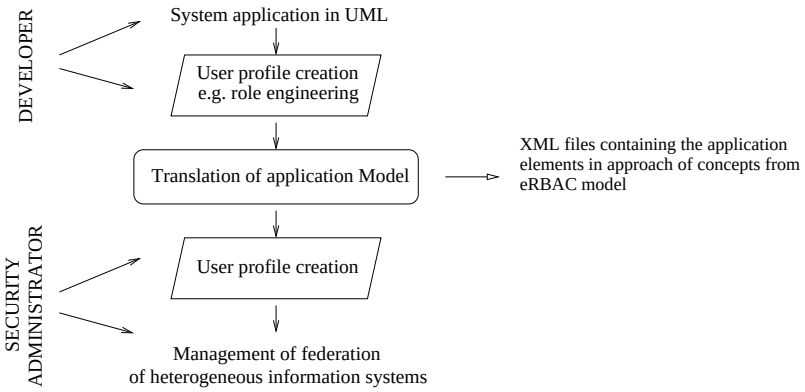



FIG. 3.2. *Role production in creation of security scheme*

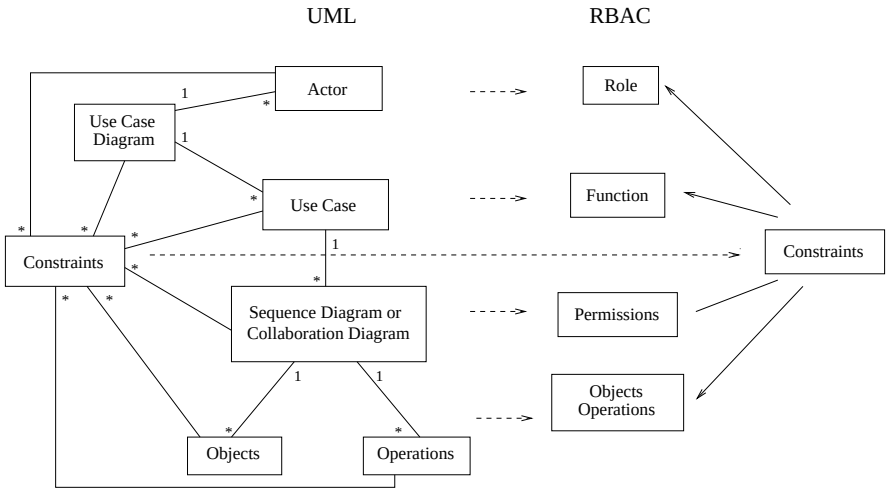



FIG. 3.3. *Concepts of extended RBAC model and their relationships with UML concepts*

- Security administrator receives the Model containing the lists of access control elements that are presented in a special form, e.g. in XML files. The administrator finishes the process of user profile creation using the rules of the global security policy.

**3.4. Association of concepts of extended RBAC model with UML concepts.** The UML (Unified Modeling Language) has been chosen for the analysis and design of information system. It is also used to design the security schema on access control level. Nowadays, UML is a standard modeling tool, properly reflecting the description of the information system and its requirements. UML proposes the set of design diagrams to present the modeling system. Two types of the UML diagrams have been chosen to present and implement the elements of extended RBAC model: use case diagram and sequence diagram. The concepts of extended RBAC model were joined with some chosen concepts of UML (Fig. 3.3). Description of connections between concepts of extended RBAC model and UML concepts was given widely in [9]:

- *RBAC role* is joined with UML actor,
- *RBAC function* joined with UML use case,
- *RBAC methods* and *objects* joined with methods and objects of UML,
- *RBAC permissions* can be found in the sequence diagrams,
- *RBAC constraints* are joined with constraint concept existing in UML [11],
- relations of different types that occur between the elements of extended RBAC model can be found in the use case diagrams and in the sequence diagrams.

**3.5. Creation of user profiles basing on extended RBAC model.** Each security policy demands in general the definition for each user a set of operations that the user will be allowed to execute. In consequence,

using extended RBAC model a set of permissions should be defined for each user. It suffices to specify the permissions for the execution of certain methods on each object accessible for that user. According to the connections between extended RBAC model and UML, a definition of user profiles on the developer level and security administrator level was proposed [22].

The implementation of extended RBAC model using UML concepts can be realized with the use of sequence diagrams, where the permissions are assigned to the rights of execution of methods realized in each use case [10, 22]. It is possible to identify and define the elements of extended RBAC model using the UML meta-model: the roles, the functions used by these roles and the permissions needed to realize the functions. The list of actors co-operating with the information system (i.e. roles) and the list of use cases (i.e. functions) can be received from the use case diagrams. Next, the analysis of these use case diagrams allows to find the relations between the particular elements: R-R (role-role) relation (from the generalization relation between the actors), R-F (role-function) relation (from the association relation between actors and the use cases) and F-F (function-function) relation (generalization relation between the use cases).

The description of each use case in the form of interaction diagrams (i.e. sequence diagram or communication diagram) gives the information about the actions and activities that allow to realize the functionality of the system's functions. Each activity represents the definition of a method on an object. Therefore, it is possible to specify the permission *execute* for each method attached to a message sending in a sequence diagram or communication diagram and next add to the set of permissions of a certain function. Therefore the F-P relations can be also automatically managed.

**3.5.1. Creation of roles.** Each user of an information system should have an assigned user profile which is defined by the set of roles played by him. A user profile is defined by a pair *(u, listRoles(u))*: *u* is a user, *listRoles(u)* is a set of roles assigned to this user [22].

It is possibly to give the rules of role creation and the assignments of these roles to users.

User profile should be defined for each user who interact with the information system

$$u_i \vdash userProfile_i$$

User profile is defined as a set of roles that the user can take and realize

$$userProfile_i \vdash setRoles_i$$

The process of role production in the security schema of information system with the use of UML concepts (the concepts of use case diagrams and interaction diagrams, i.e. sequence diagram or communication diagram) contains two stages [12, 22]:

- assignment of *set of privileges* (i.e. permissions) to a *use case* in order to define a function,
- assignment of *set of use cases* (i.e. functions) to an *actor* in order to define a role.

The security administrator is responsible for the creation of user profiles and assignment of users to the roles according to the elements and security constraints defined by the application/system developer and according to the security constraints defined on the global level.

**4. Management platform of access control.** The presented platform was created first of all for the security administrator to manage the logical security of information system (i.e. security on access control) on the global level. This platform was based on the approach presented in short in the previous sections. The second purpose of the platform was to make possible the cooperation between application/system developer and security administrator to realize and integrate the presented concepts on global level of access control. The result of such cooperation should be the validation of activities realized by the developer and by the administrator to assure the global coherence on access control level in the information system (Fig. 4.1).

**4.1. Main functionality of the platform.** The important stage of platform functionality is the process of role engineering. This stage is based on the concepts of extended RBAC model and uses the UML to design the logical model of an application or a system and uses the XML language to exchange the information came from different UML diagrams, assuring the independent formalism. Figure 4.2 presents the stages of the role engineering.

The first stage of the role engineering process is the creation of roles and definition of set of roles. This stage is realized basing on the associations between concepts of extended RBAC model and UML concepts. The algorithm of role construction of an information system was specified and implemented using the UML meta-model [22]. The stages of this algorithm are as follows (Algorithm 4.1.1):

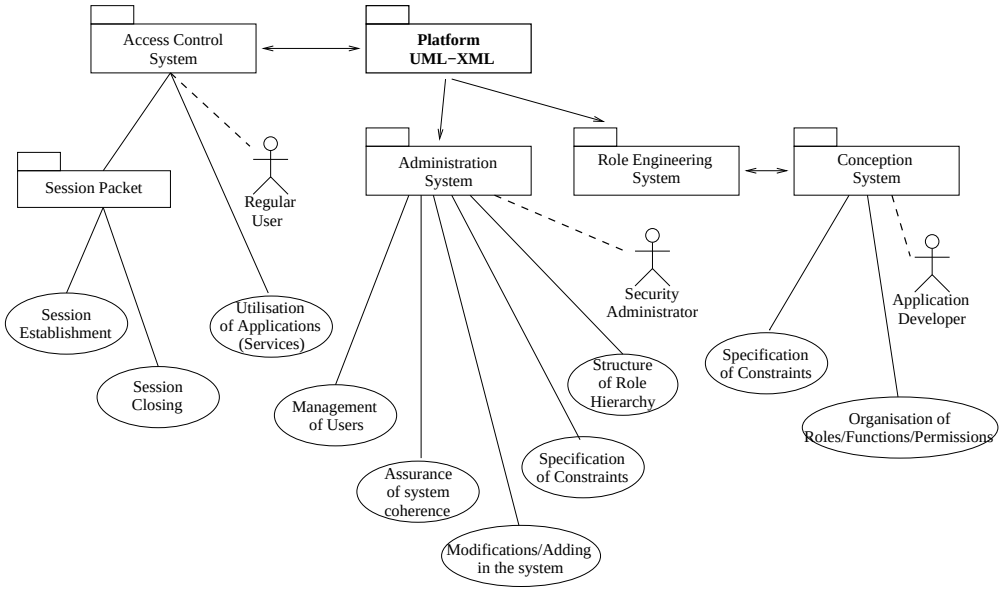



FIG. 4.1. *Functionality of platform for management of access control*

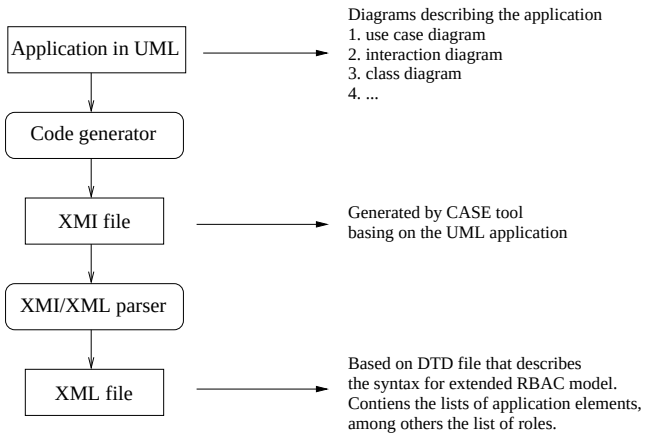



FIG. 4.2. *Role production of extended RBAC model*

- assignment of set of permissions to a function (use case of conceptual model) - Algorithm 4.1.2 and
- assignment of set of functions to a role (actor of conceptual model) - Algorithm 4.1.3.

**Algorithm 4.1.1** Algorithm for construction of roles

*ConstructionRoles(Model)*
**Begin**
AssignPermissionsToFunctions(Model)
AssignFunctionsToRoles(Model)
**End**

The first phase of role construction algorithm, i.e. assignment of set of permissions to a function (an use case of conceptual model), contains two main stages:

- searching of set of functions (use cases) in use case diagram and allocating it to *setUC* - Algorithm 4.1.4 and
- attaching of set of permissions to each function (use case) identified in the model that are necessary for

**Algorithm 4.1.2** First stage of role creation algorithm

*AssignPermissionsToFunctions(Model : Model)*
**Begin**
$setUC = SearchUC\,(Model)$
**for** *each* $cu_i \in setUC$ **do**
$setPermissions = CreationSetPermissions\,(uc_i)$
**done**
**End**

**Algorithm 4.1.3** Second stage of role creation algorithm

*AssignFunctionsToRoles(Model : Model)*
**Begin**
$setA = SearchA\,(Model)$
**for** *each* $a_i \in setA$ **do**
$setFA_{a_i} = CreationSetFunctions\,(a_i)$
**for** *each* $cu_j \in setFA_{a_i}$ **do**
$setRelationUC_{uc_j} = SearchRelationUC\,(uc_j)$
**for** *each* $uc_k \in setRelationUC_{uc_j}$ **do**
**if** $(relation\,(uc_j, uc_k)\ of\ type\ \ll include \gg)$
**then**
$setPermissions_{uc_j} = setPermissions_{uc_j} \cup$
$setPermissions_{uc_k}$
**endIfi**
**if** $(relation\,(uc_j, uc_k)\ of\ type\ \ll extend \gg)$
**then**
$setFA_{a_i} = setFA_{a_i} \cup uc_k$
**endIf**
**done**
**done**
$heritageSetA_{a_i} = SearchRelationA\,(a_i)$
**for** *each* $a_j \in heritageSetA_{a_i}$ **do**
$setFA_{a_i} = setFA_{a_i} \cup setFA_{a_j}$
**done**
**done**
**End**

function execution - Algorithm 4.1.5.

The second phase of role construction algorithm, i.e. assignment of set of functions to a role (an actor of conceptual model), contains the following stages:

- searching of set of roles (actors) in use case diagram and allocating it to *setA* - Algorithm 4.1.6,
- creation of set of functions for each role (actor) of the model (from the set *setA*) - Algorithm 4.1.7,
- completing of set of functions of each role by the set of functions being in relations with the particular use case and
- completing of set of roles for each user by the generalization relations that exist between the actors.

The second stage of role engineering process is the creation of XMI/XML file. This file can be generated automatically from the logical model of an application created in UML, using the CASE tool. Next, the created parser of XML file is used to analyze the obtained XMI files that contain the description of all elements of each UML diagram of the particular application using the XML syntax.

The functionality of XMI/XML parser was described by the following stages [22]:

- passage of XMI file and identification of elements existing in two types of diagrams (i.e. use case diagram and interaction diagram): actors, use cases, their relations, classes, objects and messages sent

**Algorithm 4.1.4** Searching of set of functions (use cases)

*SearchUC(Model : Model)*
**Begin**
**for** *each* $model_i \in Model$ **do**
**if** $(model_i\ of\ type\ UseCaseView)$ **then**
**for** *each* $modelElement_j \in model_i$ **do**
**if** $(modelElement_j\ of\ type\ UseCase)$ **then**
$set\_CU = set\_CU \cup modelElement_j$
**endIf**
**if** $(modelElement_j\ of\ type\ Package)$ **then**
$set\_CU = set\_CU \cup SearchUC\,(modelElement_j)$
**endIf**
**done**
**endIf**
**done**
return *set_CU*
**End**

**Algorithm 4.1.5** Creation of set of permission for each function (use case)

*CreationSetPermissions(uc : useCase)*
**Begin**
**for** *each collaboration* $\subset uc$ **do**
**for** *each interaction* **do**
**for** *each message* **do**
$permission = SearchPermissionElements\,(message)$
**if** $(permission \notin setpermissions)$ **then**
$setPermissions_{uc} = setPermissions_{uc} \cup permission$
**endIf**
**done**
**done**
**done**
return $setPermissions_{uc}$
**End**

**Algorithm 4.1.6** Search of set of roles (actors)

*SearchA(Model : Model)*
**Begin**
**for** *each* $model_i \in Model$ **do**
**if** $(model_i\ of\ type\ UseCaseView)$ **then**
**for** *each* $modelElement_j \in model_i$ **do**
**if** $(modelElement_j\ of\ type\ Actor)$ **then**
$set\_A = set\_A \cup modelElement_j$
**endIf**
**if** $(modelElement_j\ of\ type\ Package)$ **then**
$set\_A = set\_A \cup SearchA\,(modelElement_j)$
**endIf**
**done**
**endIf**
**done**
return *set_A*
**End**

**Algorithm 4.1.7** Creation of set of functions for each roles (actors)

*CreationSetFunctions(a : Actor)*
**Begin**
**for** *each* $associationEnd_i \subset a$ **do**
$association = Search\,(associationEnd_i)$
**for** *each* $associationEnd_j \subset association$ **do**
$classifier = associationEnd_j.type$
**if** $(classifier\ of\ type\ UseCase)$ **then**
$setFunctionsA = setFunctionsA \cup classifier$
**endIf**
**done**
**done**
return $setFunctionsA$
**End**

between the objects,
- definition of sets of identified elements,
- identification of these elements as the elements characteristic for the concepts of extended RBAC model,
- determination of sets of elements of particular application as the sets of elements characteristic for extended RBAC model,
- generation of XML file that contains the sets of identified and defined elements.

The parser returns the XML documents containing all the elements and their proprieties came from the logical model of the particular application. These elements are used next for the integration of extended RBAC model. The received XML document contains the elements of extended RBAC model such as: roles, functions, permission, objects and methods. The formal description of this file was given in file *eRBAC.dtd* which root element has the following form:

$<!ELEMENT\ \ RBAC(role+, function*, permission*, method*, object*,$
$operation*, class*) >$

The document described above is the result of developer job on local level of access control of an information system. It contains the elements of application logical model created by the developer and translated into concepts of extended RBAC model. Next, this document is given the security administrator who manages the security of the whole system. He should integrate the new elements, e.g. the new application, in the system on the logical security level. The XML document allows to realize such integration independent of the environment. It describes the elements of the new application on the RBAC level used by the administrator of access control. The security administrator uses the XML document to define the rights of the users in the system, so to define the user profiles [22].

The platform should assure the verification and validation of access control rules in the case of integration of new application in a system. It it realized taking into consideration the job of application/system developer and the job of security administrator. The validation should guarantee the global coherence of security schema defined for the organization.

**4.2. Additional tool for access control administration using traditional models.** Additional tool was created to manage the access control in a field of DAC model and classical RBAC model.

DAC model is a simple model containing first of all subject and objects - the aspects of DAC policy. For each object there is an access control list (ACL) defining privileges for defined user, application or process (Figure 4.3).

RBAC model is much more complicated model. Apart from basic entities: users, roles, permissions, there are some types of constraints (Figure 4.4):
- *Role-Object Constraints* - they limit some permissions for certain objects for defined roles,
- *User-Object Constraints* - they limit some permissions for certain objects for defined users,
- *Number of Members Constraints* - they limit the number of users that can be assigned to the role,
- *Prerequisite Roles* - they define which role is required to be assigned to the user beforehand,

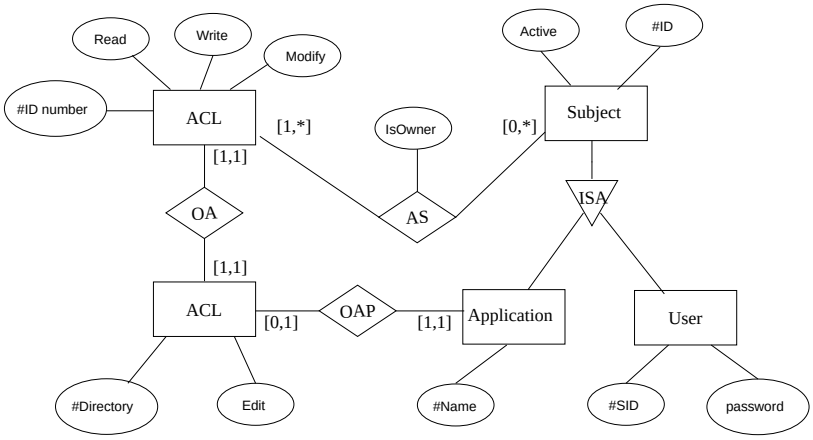



FIG. 4.3. *DAC entity relationship diagram*

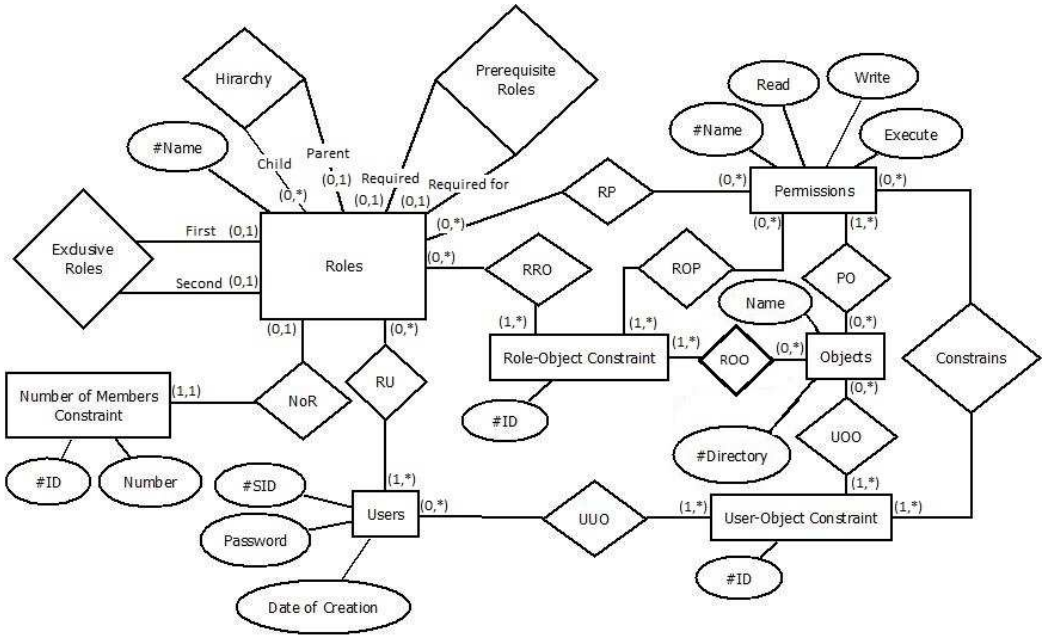



FIG. 4.4. *RBAC entity relationship diagram*

- *Exclusive Roles* - they define which roles cannot be assigned to the user at the same time.

The functionality of the tool for administration of DAC model is presented in form of UML use case diagram on Figure 4.5.

After the administrator chooses DAC model to manage, he is provided with a window consisting of user list, object list, their access control lists and the processes (Figure 4.6). He is able to add, modify and delete users and objects. When creating a new or modifying an object, the administrator has to define its directory and choose its owner from the list. After selecting a user and an object, their common access control list is presented; then the administrator gains ability to modify it as well. He can change which subject is connected with which object. Modifications include also read, write and execute permissions. In application/process panel, the administrator can choose either the process or the application which is allowed the access to chosen objects.

The functionality of the tool for administration of RBAC model is presented in form of UML use case diagram on Figure 4.7.

When the RBAC model is chosen for the administration, there is another window to manage this access control model (Figure 4.8). The security administrator gets the possibility of managing the roles defined in the

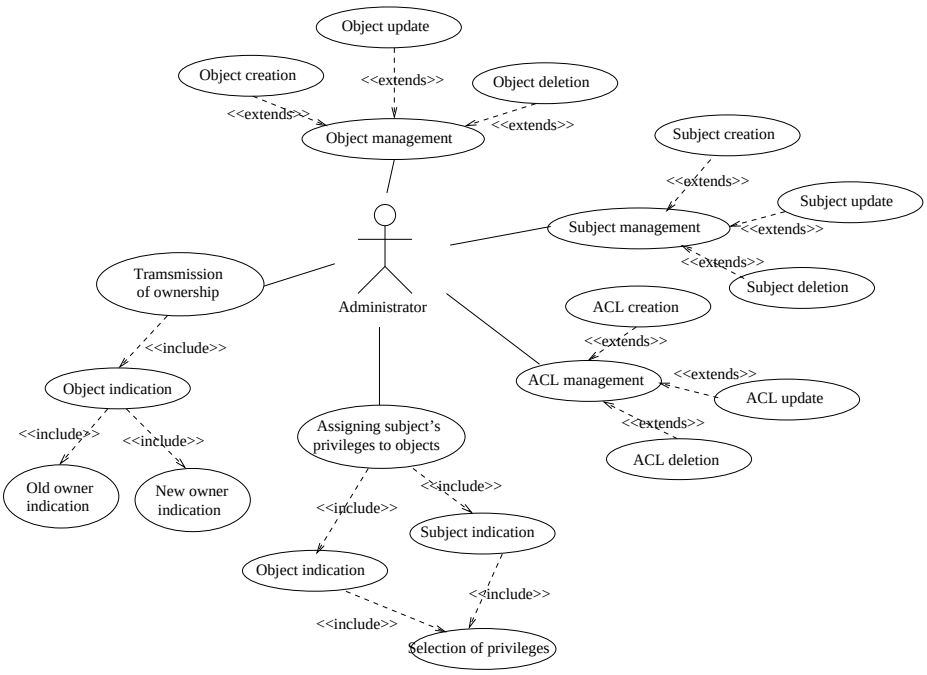



FIG. 4.5. *Functionality of administration tool for DAC model*

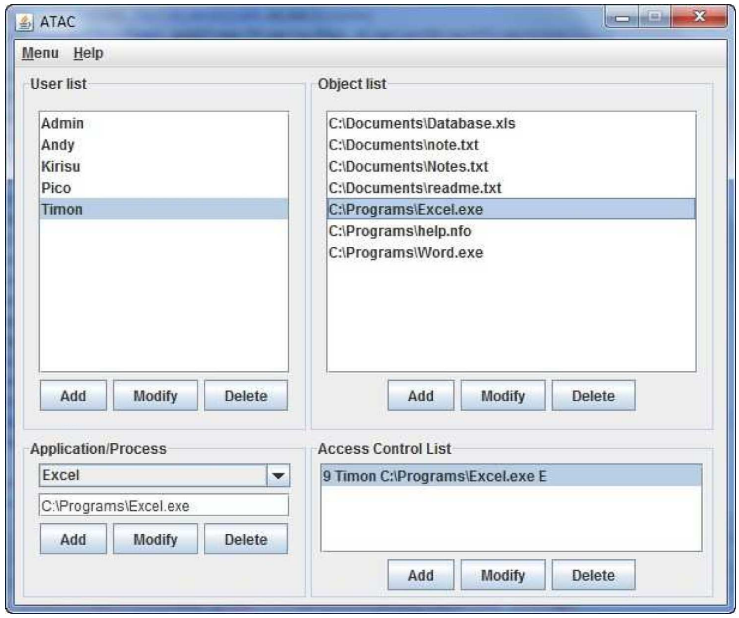



FIG. 4.6. *Tool - DAC model main window*

system obtained from the list. The list can be modified by adding new roles, modifying already existing ones or removing particular roles. The name of the role can be changed, permissions may be set and the role can be grouped in hierarchy by setting the parent role.

The administrator can manage the system users, defining the user profiles. He can add new users to the system, change the names, passwords and define the roles of existing users. There is also the possibility of deleting some of them from the list (Figure 4.9).

Another list in the main RBAC window consists of set of permissions. Modifications of the list include forming the new roles, eliminating old roles or changing the names or permissions of existing roles and references

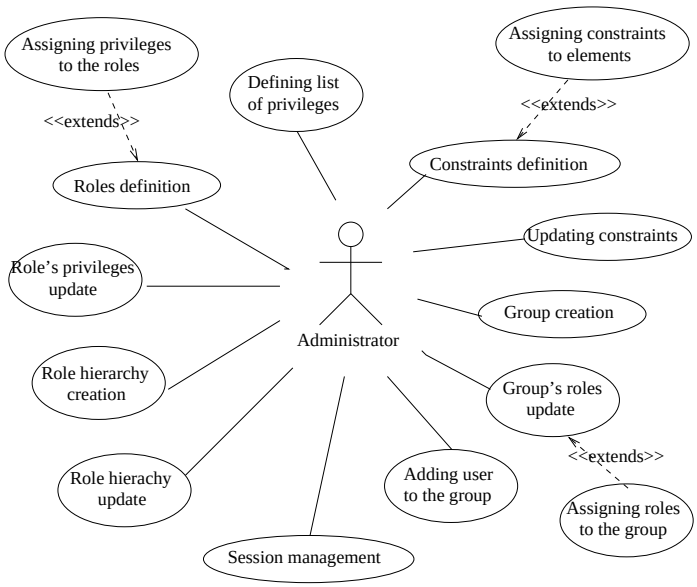



FIG. 4.7. *Functionality of administration tool for RBAC model*

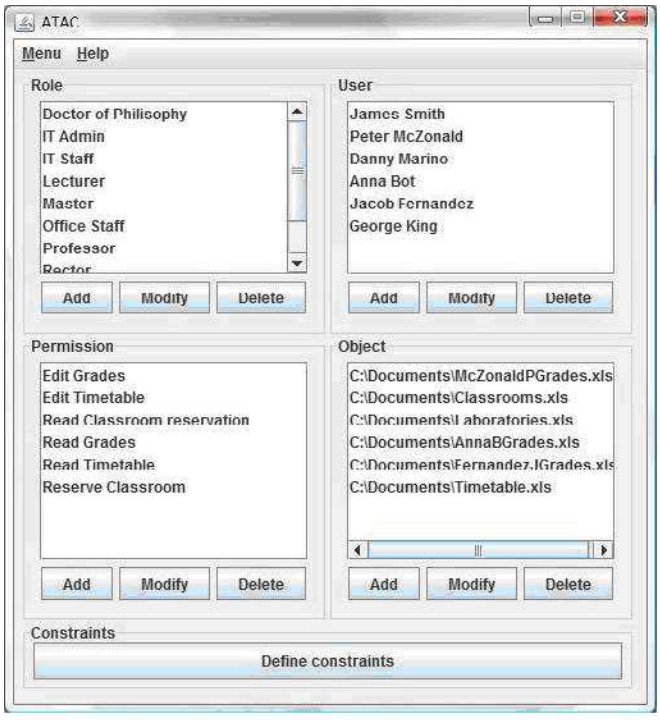



FIG. 4.8. *Platform - RBAC model main window*

to particular objects. The last one of the lists includes objects defined by their directories. Alterations of it embrace changing the directory of existing objects, removing undesired ones or creating new. Furthermore, our application enables the administrator to set various constraints that are the fifth part of the window. The constraints were divided into five groups, namely: role-object constraints, user-object constraints, number of members, exclusive roles and prerequisite roles.

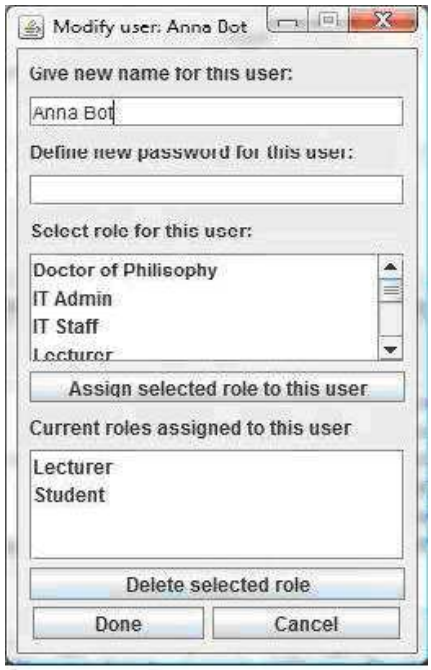



FIG. 4.9. *Platform - User modification window*

**5. Conclusions.** The presented paper describe the possibility to manage the one of aspects of logical security in the information systems. This management was based on the extended role-based access control model. The concepts of this model were used to define the process of role engineering for creation of user profiles in information systems. The process of role production is a very important stage in definition of logical security policy of an information system. It can be realized by two actors: application/system developer and security administrator who cooperate with each other to guarantee the global coherence on access control level.

The tool presented in the paper allows manage the logical security of company information system from the point of view of application/system developer and from the point of view of security administrator. Particularly, the security administrator, who is responsible for the information system security, can facilitate the management of access control of users to data stored in a system.

Currently, not all platform functionalities, shown on figure 4.1, were realized but the already implemented functions allow to manage the security of the information systems.

REFERENCES

[1] S. CASTARO, M. FUGINI, G. MARTELLA AND P. SAMARATI, *Database Security*, Addison-Wesley, 1994.
[2] R. S. SANDHU, E. J. COYNE, H. L. FEINSTEIN AND C. E. YOUMAN, *Role-Based Access Control Models*, IEEE Computer, Volume 29, No 2, s. 38-47, 1996.
[3] R. S. SANDHU AND P. SAMARATI, *Access Control: Principles and Practice*, IEEE Communication, Volume 32, No 9, s. 40-48, 1994.
[4] D. FERRAIOLO, R. S. SANDHU, S. GAVRILA, D. R. KUHN AND R. CHANDRAMOULI, *Proposed NIST Role-Based Access control*, ACM Transactions on Information and Systems Security, 2001.
[5] G. BOOCH, J. RUMBAUGH AND I. JACOBSON, *The Unified Modeling Language User Guide*, Addison-Wesley, 1998.
[6] G.-J. AHN, *The RCL 2000 Language for Specifying Role-Based Authorization Constraints*, ACM Transactions on Information and Systems Security, 1999.
[7] E. DISSON AND D. BOULANGER AND G. DUBOIS, *A Role-Based Model for Access Control in Database Federations*, Information and Communications Security, Proc. of 3th ICICS, China, 2001.
[8] G.-J. AHN AND R. S. SANDHU, *Role-based Authorization Constraints Specification*, ACM Transactions on Information and Systems Security, 2000.
[9] A. PONISZEWSKA-MARANDA, G. GONCALVES AND F. HEMERY, *Representation of extended RBAC model using UML language*, Proc. of SOFSEM 2005, LNCS 3381, Springer-Verlag, 2005.
[10] A. PONISZEWSKA-MARANDA, *Access Control Coherence of Information Systems Based on Security Constraints*, Proc. of 25th International Conference on Computer Safety, Security and Reliability, LNCS, Springer-Verlag, 2006.

[11] G. Goncalves and A. Poniszewska-Maranda, *Role engineering: from design to evaluation of security schemas*, Journal of Systems and Software, Elsevier, Vol 81, 2008.
[12] A. Poniszewska-Maranda, *Conception Approach of Access Control in Heterogeneous Information Systems using UML*, Journal of Telecommunication Systems Springer-Verlag, No 45, 2010.
[13] B.W. Lampson, *Database Security*, Addison-Wesley, 1974.
[14] D. Dows and J. Rub and K. Kung and C. Jordan, *Issues in discretionary access control*, Proc. of IEEE Syposium on Research in Security and Privacyv, 1985.
[15] D. Bell and L. LaPadulla, *Secure computer systems: Unified exposition and multics interpretation*, Mitre Corporation, 1975.
[16] E. Bertino and C. Bettini and P. Samarati, *A Temporal Access Control Mechanism for Database Systems*, IEEE Transitions on Knowledge and Data Engineering, 1996.
[17] E. Bertino and P. Bonatti and E. Ferrari, *A Temporal Role-based Access Control Model*, ACM Transaction on Information and System Security, 2001.
[18] A. Gal and V. Atluri, *An Authorization Model for Temporal Data*, ACM Transaction on Information and System Security, 2002.
[19] B. James and E. Joshi and U. Bertino and A. Latif and A. Ghafoo, *A Generalized Temporal Role-Based Access Control Model*, IEEE Transitions on Knowledge and Data Engineering, 2005.
[20] J. Park and R. Sandhu, *The UCON ABC Usage Control Model*, ACM Transactions on Information and System Security, 2004.
[21] J. Park and X. Zhang and R. Sandhu, *Attribute Mutability in Usage Control*, 18th Annual IFIP WG 11.3 Working Conference on Data and Applications Security, 2004.
[22] A. Poniszewska-Marada, *Platform for access control management in information system based on extended RBAC model*, IEEE Computer Press, Proceedings of the 12th IEEE International Symposium on Symbolic and Numeric Algorithms for Scientific Computing, Timisoara, Romania, September 2010.

*Edited by:* Dana Petcu and Alex Galis
*Received:* March 1, 2011
*Accepted:* March 31, 2011

Scalable Computing: Practice and Experience
Volume 12, Number 1, pp. 51–61. http://www.scpe.org
ISSN 1895-1767


# AUTOMATIC DATA MIGRATION E-SYSTEM FOR PUBLIC ADMINISTRATION E-SERVICES

CIPRIAN DOBRE* AND ANDREEA MARIN†

**Abstract.** Interoperability represents an important issue in developing successful e-Services applications. In such applications data is often produced by services specifics to one organization and is consumed by services belonging to another one. Still, automatic data migration and transfer utilities are currently insufficiently addressed. We present a solution for automatic data migration that is able to handle data transfers between different database instances. The solution works with dynamic data schemas, and requires minimal user input and interaction. The proposed e-System was successfully used to support public administration automatic collection of data about companies. Various government processes require information from businesses. Examples include statistics about number of employees, revenue data, etc. However, collecting such data generally requires it to be voluntarily transmitted by business organizations. We present results for evaluating in a real-world e-System the proposed solution, as a backbone designed to automatically collect reporting data.

**Key words:** data migration, automatic data conversion, interoperability, e-System

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

**1. Introduction.** Many e-Systems are made by composing the functions provided by services running in different organizations. Such services often use similar data, stored in different ways. Service composition often uses the data provided by one service as input for other service. In this case interoperability, the way data is semantically linked between these two systems, becomes a critical issue.

Organizations usually collect large volumes of data in their internal databases. In many cases, the stored data refers to the company structure, employees, customers and projects. With the development of new database systems, companies might want to change the database in use and transfer large amounts of data. Moreover changes in company structure and number of employees might generate modifications in the database schema. As a consequence, not only that the underlying database system is changed, but the data transfer would occur between two different schemas.

Data Migration is not an isolated process. A typical scenario involved the implementation of an application which requires data already existing in current storage devices (e.g., already used by other applications). Such a scenario is a typical premise for data migration. Recent studies show that the industry spends as much as $5 billion on data migration, considering software and consulting services [1].

This paper extends the results presented in [12]. We describe the design and implementation details for the system designed to automate the data migration between various data sources. As such, we describe how our solution is able to handle data transfers between different database instances. It is designed to work with dynamic data schemas, and requires minimal user input and interaction.

In addition, we present results for evaluating the proposed solution in a real-world e-System, as a backbone designed to automatically collect reporting data. An important problem with public administration is collecting data about companies. Various government processes require information from businesses. Examples include statistics about number of employees, revenue data, etc. However, collecting such data is not an easy task. It generally requires the data to be voluntarily transmitted by business organizations. In many cases on the business side the data, instead of being electronically transmitted, is copied manually from the database in the internal back-office system and then sent to the public administration entities. In this context the proposed e-System was successfully used to support public administration automatic collection of data about companies.

The paper is organized as follows. In Section 2 we present related work. Section 3 gives a broad view of data migration and presents its importance in any business environment. Section 4 presents the architecture of the data migration e-system. Details regarding the implementation of the system are provided in Section 5 and Section 6 describes a form of usage of the system. Section 7 gives conclusions and presents future work.

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

†Computer Science Department, Faculty of Automatic Control and Computers, University POLITEHNICA of Bucharest, Romania(andreea.marin@cti.pub.ro).

**2. Related work.** Data migration has been previously studied in research literature. A number of solutions ( [2] [3] [5]) were previously proposed, each having various limitations. A number of research papers ( [1]) analyzed the pros and cons of such solutions, presenting clear explanations about their differences and problems.

In [1] the authors make a distinction between data migration and traditional data moving solutions. Furthermore, the authors make an analysis about how large volumes of data can migrate between database systems and what are the steps in developing an efficient migration strategy. At present there are various methods for performing data import or export. Various software applications as well as database systems offer the possibility for exporting or importing data. Furthermore they allow exporting data in various known formats: csv, xls. Consequently they allow data import from these types of files also.

Previously, several research projects and software products tackled ideas related to our proposed solution. We next present details about several such solutions, which represent the state-of-the-art of this area. According to [2], SQL Server Integrated Services (SSIS) provide the developer the necessary means to develop workflows for data transfer. The workflows take the form of packages that contain data and control flow information, used both for creating mappings between source and destination, but also for connecting them. Moreover, the means for data transfers are provided also by SQL Server Management Studio Express, in the form of tasks that can be executed on top of the database. This utility transfers data in known file types (csv, xls) or in Oracle and SQL Server Databases having the same schema. On the other hand, the transfer depends on the Oracle Database System's version (10g, 9i).

Talend Open Studio is a very complex data integration and business modeling software product. According to [3] using specific jobs in Talend Open Studio can help transfer data between any two sources. It offers wide support for the most important database systems and for the best known file types. What it may be considered as an inconvenient is that the schemas of the source and destination entities have to be known at design time for the developer to be able to create mappings between them. However, Talend Open Studio provides a very user friendly graphical interfaces and offers broad connectivity for both target and source systems.

Unlike previous work, the solution presented in this paper tries to automate the steps in converting the data and automatic mapping of fields, even if they pertain to different data schemas. The solution can be used to automate many of the steps required when services require the data produced by other services, or the migration of data between different databases.

**3. Data Migration.** Data migration is a challenging process that aims to transfer existing data into a new environment. This transfer is made by companies when starting to use new software products or upgrading to a new type of database. These changes are frequent in the business world as both databases and software products are evolving quickly.

As described in [7], a series of factors should be taken into account when planning data migration within a business environment. On the first hand a high volume of data is involved. The migration process must be supervised and also capable of treating interruptions. The problems that might arise during the migration process might affect the integrity and consistency of data. Furthermore, the majority of migration processes often occur between heterogeneous environments. An aspect that should be taken into account is the set of transformations required by certain data types. This aspect is treated by the software solution presented in the paper.

Usually, data migration is a static lengthy process that requires careful planning. According to [8], the following steps should be followed when planning the data migration process:

1. Analyze and define the source database structure: tables, data types.
2. Analyze an define the destination database structure: tables, data types.
3. Perform field mapping: mapping between the source and destination data, data type transformation.
4. Define the migration process.

As it is described in the paper, the proposed software system provides a dynamic environment for executing the above tasks. The end user is presented with a very friendly and easy interface, and in the same time with a product that can perform data migration using a dynamic approach.The importance of data migration is explained in [9]. One of the most important factors that determine this process is its cost. A significant amount of the IT budget, as well as labour, is spent on data migration. Data migration is carried on frequently in companies thus increasing the amounts of money invested in such processes. Projects regarding data migration can become very complex creating a very large market for labor, consulting, software and hardware.

**4. The architecture of the automatic data migration system.** The architecture of the Automatic Data Migration System is presented in Figure 4.1. The system maps data between two data sources. It can then dynamically transfer the data whenever required.

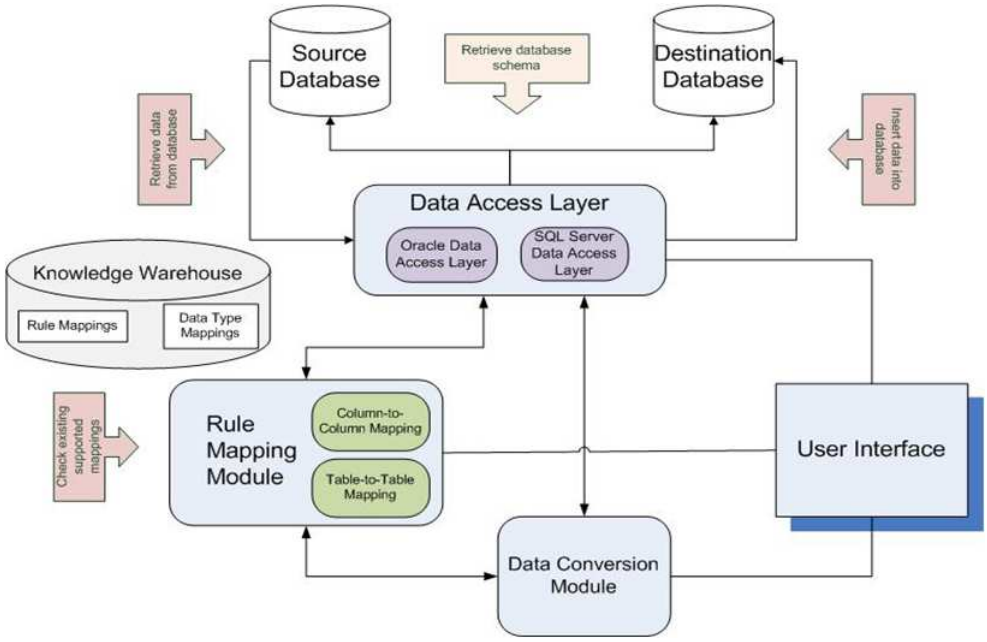



FIG. 4.1. *The proposed architecture.*

The system first loads the data schema (fields, relationships, etc) from the first data source. The user is then presented with an intuitive representation of the data schema. The user has full control over the data migration process using this data interface. On the other hand, the system includes mechanisms to automatic recognize data types and map data fields belonging to different data sources. It can be instructed to move data on various triggered actions, even if conversions among data types are required.

The architecture consists of specialized modules that interact with each other maintaining a continuous flow of data. The main components are as follows. The three main modules of the software application are the rule mapping module, the data conversion module and the data access layer. These modules are controlled indirectly by the graphical user interface.

The data access layer retrieves the database schemas in order to be displayed by the user interface. In addition it is used for database specific operations such as data retrieval and insertion.

The rule mapping module can be used to register mapping rules in the system, to manage and send for execution against the involved databases. The rule mapping module checks the validity of the rules using a repository. The repository consists of several implicit and explicit mappings that are supported by the application.

The data conversion module is used for the data conversion that must occur during data migration. In addition, a knowledge warehouse is used for storing conversion mappings between known data sources.

**5. Implementation details.** The implementation of the e-System supports data migration/transfer between data sources without having prior details about the structure of the data. The implementation of the automatic data migration system considers several of the most known data sources.

In the majority of batch data transfer applications the schemas must be defined at design time. For example, the Typed Data Sets from .NET is a data mapping that can be used in design [6]. However, such an approach cannot be used for implementing the migration system. In our system no schema is defined at design time and this increases the usability of our system for non-technical users. This is an important improvement because many data migration projects have to be developed and run by database specialist.

The system allows the user to perform data migration activities on his own, without him having advanced technical knowledge over database systems. Furthermore the application supports migration between similar

and different databases. For example, one of the first features supported was data migration between two SQL Server databases. In addition, the second important feature supported was migration between an Oracle database instance and a SQL Server one.

Concerning security issues, the only information the user has to know is the credentials of these databases. As a consequence, both databases must allow remote connections and the user must have administrator rights on the system on which he is running the data migration system.

Software requirements for the proposed system include Microsoft .NET Framework 3.5, Microsoft IIS 7.0 and Microsoft SQL Server 2008 Express. Microsoft .NET framework is used for the implementation of all the three modules. The database is the main component of the Knowledge Warehouse, used to store mapping rules information.

In the following paragraph we present details about the implementation of the main components.

**5.1. User interface.** The user interface supports several functions:

1. Authentication.
2. Creation and management of mapping rules.
3. Display of data schemas, conflicts or resolutions.
4. Display of migration results.

The user inserts the credentials for the source and destination data sources. The user interface supports a wide range of authentication mechanisms, designed to support connections to various databases and flat file sources. After the credentials are verified and a connection is created, the data schemas are retrieved from the both data sources and they are displayed. All information is displayed in a web portal, designed using .NET platform TreeViews. An example of this portal is presented in Figure 5.1.

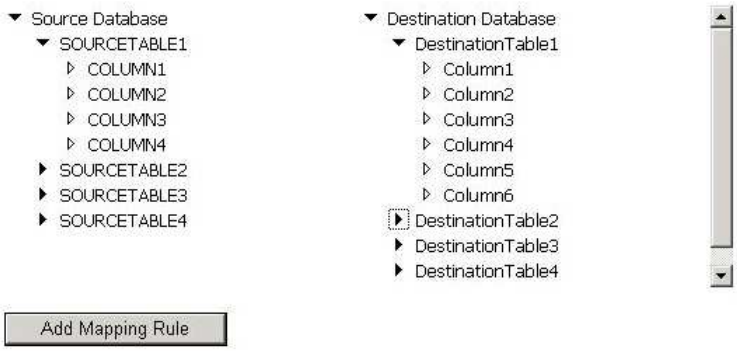



FIG. 5.1. *The data sources structure.*

Using the interface, the user can browse through the tree structure and select the columns and tables he wants to use for constructing the mapping rules. The mapping rules are dynamically created by selecting a source end point and a destination end point. The end points are selected from a list of available data sources.

After the mapping rules are created they are added into a list and can be further reviewed and managed. The list, presented in Figure 5.2, is available for the user to delete the rules he does not want to continue with. After the rules are saved the user can proceed to the rules checking and execution. If all the mapping rules have structures similar to the supported mapping rules, the migration can occur. If not the user is displayed with a list of conflicts and the possibilities to resolve them. This feature is explained in the following sections.

| | SourceTable | SourceColumn | DestinationTable | DestinationColumn |
|---|---|---|---|---|
| Delete | SOURCETABLE1 | COLUMN1 | DestinationTable1 | Column2 |
| Delete | SOURCETABLE2 | COLUMN3 | DestinationTable3 | Column1 |
| Delete | SOURCETABLE4 | COLUMN9 | DestinationTable1 | Column6 |

FIG. 5.2. *Examples of mapping rules.*

**5.2. Rule mapping module.** The rule mapping component is used to create mapping rules according to the specifications of the user, to verify the correctness of the rules and, if all the conditions are verified, to further send the rules for execution against the database. The system supports two types of mappings: column-to-column mappings and table-to-table mappings.

The column-to-column mapping is described as follows. By means provided by the interface the user can select the source and destination location of the data. The user can choose any column from the first database (the source column) and any column from the destination database (the destination column). Also, this module uses the Knowledge Warehouse component to automate the process of column mapping. This component includes various predefined rules of mappings columns. For example, if two columns have the same name (for example the name of an employee or salary) then the user is already presented with the mapping rule. Also, if the user previously selected a mapping rule and the same two columns are again used in another selection, again the two columns are already mapped. However, the user has full control over these predefined mappings.

After the two columns are selected the user can register his mapping in the system. In the end, when all the mappings are registered the user instructs the system to send the rules to be processed against the databases.

The table-to-table mapping is similar to the column approach, but instead of columns, the end point entities are tables. This feature was added to support the case study reporting system that is presented in the next Section. This reporting system considers that there is a slight possibility that users have similar data structures in both source and destination databases. When this kind of mapping is performed the system checks that the involved tables have the same structure exactly. This is equivalent with comparing two hash tables having the same key values, in this case, the keys being the column names. If the column names and data types are not the same the migration cannot be performed and the user is notified accordingly, being required to act in concordance with the best suited solution for his needs.

The rule mapping module also allows managing the created rules. Before starting the data conversion, the user can check the created rules and decide whether he wants to use all of them or not. Both, the user interface and the rule mapping module provide the ability to delete the rules considered unsuitable.

The module also supports the management of user generated errors, such as creating an incorrect mapping. An incorrect mapping can be defined as a mapping whose end point entities do not have the same type. For example a user can create a table-to-column mapping, which would generate serious damage to the data migration process if not discovered prior to the start of rule execution.

**5.3. Data conversion module.** This module assists the rule mapping and migration functions of the e-System with data conversions. This has to be performed according to the user needs and in such a way that data is not altered during the conversion. An incorrect data conversion can negatively affect the applications relying on that data as well.

The module supports two types of conversions:

1. Implicit conversions.
2. Explicit conversions.

Implicit conversions are made between well-known database data types [4]. Examples include conversions between SQL Server, Oracle or other databases, such as Binary to Raw, Image to Long raw and others (see Table 5.1). Implicit data conversions are automatically created by using predefined rules provided by the Knowledge Warehouse component.

The explicit conversions are used for conversions between non-compatible data types. For example, for some situations the user might want to convert numerical data types into strings or vice-versa. Such conversions are part of the requirements that the user is presented with an large collection of data conversions. Table 5.2 presents several such explicit data conversions.

To prevent possible data losses, when the user requires such a conversion, he is presented with a warning message stating that continuing this action might alter the data. As a consequence, the user must agree on continuing with the data conversion. If the user does not find the data conversion suitable for his requirements he has two choices:

1. To perform just the data conversions that are considered safe and then alter the destination database; or
2. To stop the whole migration process, alter the destination database and restart the migration process.

This component is also capable of automating the conversion processes. For implicit conversions the system is able to recognize formats and correctly manage data migration between columns of compatible data types.

TABLE 5.1
*Examples of implicit data conversions.*

| Data type | SQL Server | Oracle | MySQL |
|---|---|---|---|
| boolean | Bit | Byte | N/A |
| integer | Int | Number | Int<br>Integer |
| float | Float<br>Real | Number | Float |
| currency | Money | N/A | N/A |
| String(fixed) | Char | Char | Char |
| String(variable) | Varchar<br>Nvarchar | Varchar<br>Varchar2 | Varchar |
| Binary object | Binary<br>Varbinary<br>Image | Long<br>Raw | Blob<br>Text |

TABLE 5.2
*Examples of explicit data conversions.*

| Source data type | Destination data type |
|---|---|
| String | Int |
| Boolean | Int |
| Boolean | Bit |
| Int | String |
| Boolean | String |
| Char | Boolean |

Also, the Knowledge Warehouse stores information for both implicit and explicit data conversions. Whenever a data conversion rule that does not exist in the repository is created by a user, and the rule is validated by the data conversion module, it is saved for further usage in a temporary storage space. The system administrator checks the temporary tables periodically and if the rules are in concordance with the system's requirements and purpose, they can be added to the permanent mapping tables.

**6. A case study.** The application presented in this paper was tested and used in an e-Services system for public administration reporting services. The e-Services system is able to provide optimized automatic data transfers, document workflows and business reporting of business organizations. Its main purpose is to optimize businesses improving management of routine tasks such as periodical reporting data or automatically managing interactions between the organization and the public administration.

In many countries companies are required by government to periodically report their fiscal, social and statistics information to various institutions. Many data required by these reports are redundant and in many cases the company has to report the same information to different institutions. Such reports and the methodology for filling them are well documented by the local legislation. Many believe the public administration will soon be affected by various changes and challenges due to the private sector's expectations [10]. In Romania, in particular, many institutions adopted their own "in-house" systems for managing data using different technologies. The existing heterogeneous means of communication can have a negative impact on the reporting workflow between enterprises and public authorities. Due to this issue it is important to model the reporting process as close to administrative reality as possible.

The architecture of the e-System (presented in Fig. 6.1) involves a high degree of interaction between the system and the public institutions. It consists of several software components, called Processors, responsible with the automatic generation of reports based on the data available within a business organization. A Central Authority is responsible with communicating with all processors, updating their rules and report templates and managing their processes. In addition, public institutions interact with the Central Authority and provide information regarding the reports using a natural language.

The role of the Central Authority is to represent a connection between the companies that use a Processor and the Public Institution. From a general view the Central Authority takes input from the Public Institutions and provides as output information for the Processors. Any public institution involved in the workflow can send information regarding a report to the Central Authority. This type of information can contain the structure of a report, as a pdf form and the methodology describing how the form must be filled in, which information it

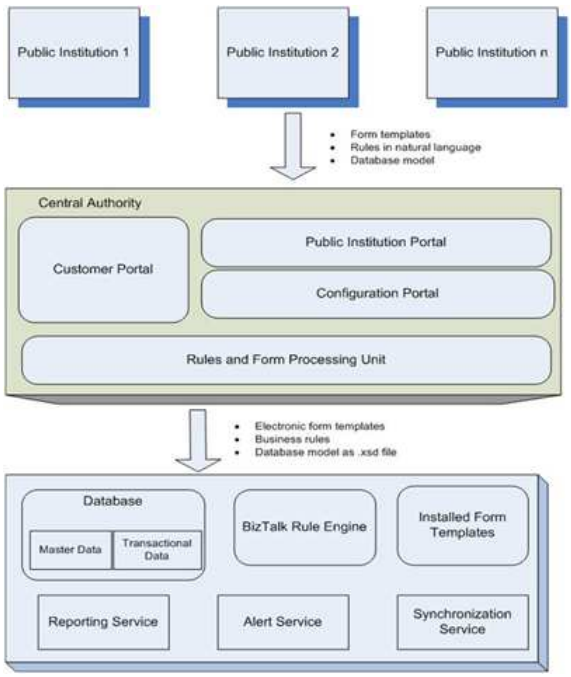



FIG. 6.1. *The architecture of the reporting e-System.*

should contain, the deadline until the completed form should be returned to the public body and the periodicity of the form (yearly, quarterly, monthly, weekly). The Central Authority creates three components that are sent to the Processors registered with it in a single communication package. The first component is an InfoPath form that tries to resemble closely the original form sent by the Public Authority. InfoPath is a Microsoft technology used to easily create and fill in forms. However it is based on open source standards such as XML and XSLT. Characteristics of this type of form include navigation, web browser compatibility and automatic completion of data based on a specified data source. The second component is an XSD schema that describes the types of data used by the report. The last component is the formal set of rules created to represent the methodology in an electronic format. These rules are written and interpreted using the BizTalk engine. BizTalk is a technology that allows integration and communication between various heterogeneous applications. In the current project it was used to integrate a series of web services and to create workflows inside the Processor according to the rules mentioned above.

The Processor, that is called the Pro Processor or simply Pro in the project, represents the core of the reporting system. Its components work together in order to automate tasks and complete periodical workflows. Each Processor has its own database. This database contains two types of data: master data and transactional data. Master data represents data that does not change frequently during a company's lifetime, like company name, address, different elements of identification given by the public bodies, employees' information. Transactional data represents data that has to be inserted periodically (weekly, monthly) into the Processor's database or data that can be computed using a combination of master data and other transactional data. One of the roles of the Processor is to infer transactional data when possible. For describing a typical workflow in the Processor we will consider a moment in its lifetime when data resides in the database, the transfer of data during the initialization of the reporting system will be explained in a following paragraph in the current section. The Processor will receive from the Central Authority a communication package containing an InfoPath form, a XSD template and a set of BizTalk business rules as mentioned above. The form will be installed in the system with its additional information, such as periodicity, data needed and the roles of the persons responsible for filling in the form. The XSD template is taken as input by one of the services of the Processor. This service creates a new database according to the contents of the template. Furthermore, the Synchronization Service is responsible for copying necessary data in the new database from the existing one. The third component of the communication package, the set of BizTalk business rules, is installed in the BizTalk Rule Engine. The Alert

Service is responsible for starting a reporting workflow. This service is aware of the deadlines of each report, information that resides in the Processor's database. Whenever a certain deadline is met, this service notices the Reporting Service which tries to fill in the report with the data needed. After filling in the report it is send to the person inside the company that is responsible with it. This person can verify, correct or fill in missing information. Lastly, the form is signed. If any information was added or changed in the form, the Processor is aware of these changes and propagates them into the database. After the form is signed, the Processor sends it to the Public Administration body responsible with collecting that type of information. The described workflow is presented in Fig. 6.2.

Several modules are involved in the implementation of the processor. The first one is the configuration module and is responsible for configuring the master data inside the PRO. The second module is the data acquisition one, which allows the PRO to interconnect to other local systems inside the companies. Next, the data processing module generates the reports accordingly to the business rules implemented. The eSafe module stores all the required data securely. The last module is report dispatching, which accomplishes the transportation of reports to the corresponding public institutions [11].

The main modules shown in Fig. 6.3 are described further. The Portal is responsible for data entry for both master and transactional data and consists of a number of sections, each section with a corresponding web form and a corresponding XML mapping considering the PRO internal XML representation of master and transactional data. Each section has one or more persons responsible for filling the data. There are two types of sections: master sections and transactional sections, for the corresponding data types. There may be a third section for "user and role management", where a designated administrator can manage the users authorized to fill in and sign the form. The *Data Warehouse* is responsible for storing both master and transactional data. Here, a module stores and selects the different versions of the business rules.

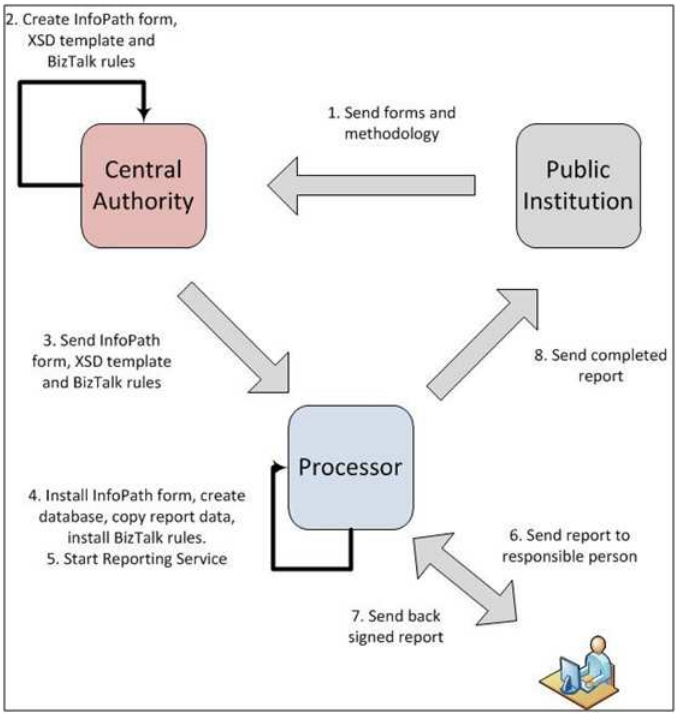



FIG. 6.2. *Workflow for reporting services.*

The *Central Management Unit* is responsible for starting and scheduling jobs. For starting a job the services involved in this unit test if all the required data is available and if so the job is started. If the data is not available yet, PRO sends alerts via email to the users responsible for providing the missing data. This unit's roles include logging system's states and managing functional reports. Furthermore, it communicates with the Central Authority according to the protocol defined for updates and deployment. It also sends completed reports to the Public Institutions. The jobs are executed by the *Business Rules Processing Unit*. Security issues

and certificates are managed by the *Security Module*. The *Report archiving module* is responsible for archiving the XML reports following predefined rules.

Two types of adapters are shown in Fig. 6.3. The *enterprise adapter* implements the enterprise specific interfaces to PRO. It is responsible for mapping the PRO internal XML representation of reports to enterprise specific forms and vice versa. It also maps the enterprise specific representations of master and transactional data to PRO representations. The *public institution adapter* is a module that implements the public institution specific interfaces to PRO. As in the case of the previous adapter, it is responsible for mapping the PRO internal XML representation of reports to the public institution specific forms and vice versa.

The role of the *Data Interpretation Service* is collecting data for the Processor. For achieving this, the Service handles two types of data: master data and transactional data, which were described before in the current section. Master data is submitted in the Customer Site of the Central Authority and is transferred along with the processor. Transactional data, on the other hand, is obtained by aggregating master data and other transactional data using the Business Process Rules that are received from the Central Authority. The *Data Interpretation Service* uses the migration system described in the paper. The migration system is configured through its interface and helps the company set up connections between local back-office systems and the Processor. The system further connects to the company's database and to the database inside the Processor. The user must provide credentials and localization data for its own database for the system to be able to connect. After the connections are completed, the user can see the extended structure of the both databases. Using the web interface, the user can define mappings of columns and tables, as previously described in the paper.

The communication between the Public Institutions and the Processor is done through means of email. An email usually contains a completed form.

The reporting system was developed using Microsoft .NET platform, with C# as the main language. Other technologies used are InfoPath forms and XML, which is involved in the communication process between the entities mentioned. Furthermore, for implementing such a system in a real life environment, the entities involved should have the following system configurations.

For the client side, the Processor consists of two servers. Both servers should have as operating system Windows Server 2008 SP1 Standard Edition. The first server is responsible with the security of the Processor and has the following software products installed: Active Directory Domain Services, Active Directory Certificate Service and Internet Information Services (IIS) 7.0. The second sever of the Processor is responsible with the reporting workflow and the software installed consists of the following products: Internet Information Services (IIS) 7.0, Microsoft SQL Server 2005, Microsoft BizTalk Server 2009 and Microsoft Office SharePoint 2007.

The Central Authority can be implemented using a single server running Windows Server 2008 SP1 Standard Edition operating system. The following software products have to be installed on the server in order that it works properly and can fulfill its designated tasks: Internet Information Services (IIS) 7.0, Microsoft SQL Server 2005, Microsoft BizTalk Server 2009, Microsoft Office SharePoint 2007, Active Directory Domain Services and Active Directory Certificate Service.

The Public Administration should have an operating system that permits the installation of Microsoft Outlook 2007 and Microsoft InfoPath 2007. The type of operating system used in the experiments was Windows Vista SP2.

The role of the data migration software is to provide the initial data for the reporting system. The reporting system in discussion has its own database system. At the first use of this system, the only information available is the database schema. For the initialization a series of internal data is needed. The data is considered to reside partially or totally in the userś own database. For the transfer of the data between the two databases, the presented software is used. The issues arising from this approach are: which data is needed for the transfer, how the data is transferred and what is the structure of the source and destination database schemas.

Based on this architecture, we developed a pilot implementation based on the business realities in Romania. The basis of such an implementation consisted of a series of specific Romanian public administration documents used in domains such as social insurance, environment and fiscal reporting. The analysis of the reporting processes between businesses and public authorities revealed the existence of 4 classes of reports, for reporting fiscal, social, environmental and statistical data.

Such reports can generate a lot of documents and bureaucracy that can have a negative impact on performance of the private business enterprises. For the pilot implementation we concentrated, in particular, on the 010 Fiscal Registration Declaration, officially known as declaration of amendments for judicial persons, associa-

tions, and other entities without judicial personality. The sensitive data that has to be inserted into this report consists of the identification data of the taxpayer and the categories of declaration tax obligations according to the law, hereinafter called fiscal vector. The fiscal vector can be defined as the permanent obligation of the taxpayer and consists of data regarding the following financial aspects: VAT, excises, petrol tax and natural gases from the internal production tax, gambling tax, profit tax, fiscal royalties, micro corporation income tax, wages income tax, special taxes such as: social health insurance tax, unemployment tax, professional illness and accidents tax, social insurance tax. The migration was done between two different database types. The source database was Oracle 11g and the destination database was Microsoft SQL Server 2008. In this context the migration system optimized the overall performance of the reporting system by ensuring a transparent data retrieval process. It allows the user to easily manage the data mapping processes for filling the required reports.

Without the migration service, the user should be presented with mechanisms to export the data required by the reporting system manually. The data cannot even be directly imported, manually conversions being needed for many of the fields required. Using the migration service there is no need to have a data operator to deal with the data, many of its tasks being performed automatically. Therefore, the system ensures a lower operating time and fewer errors when moving the data.

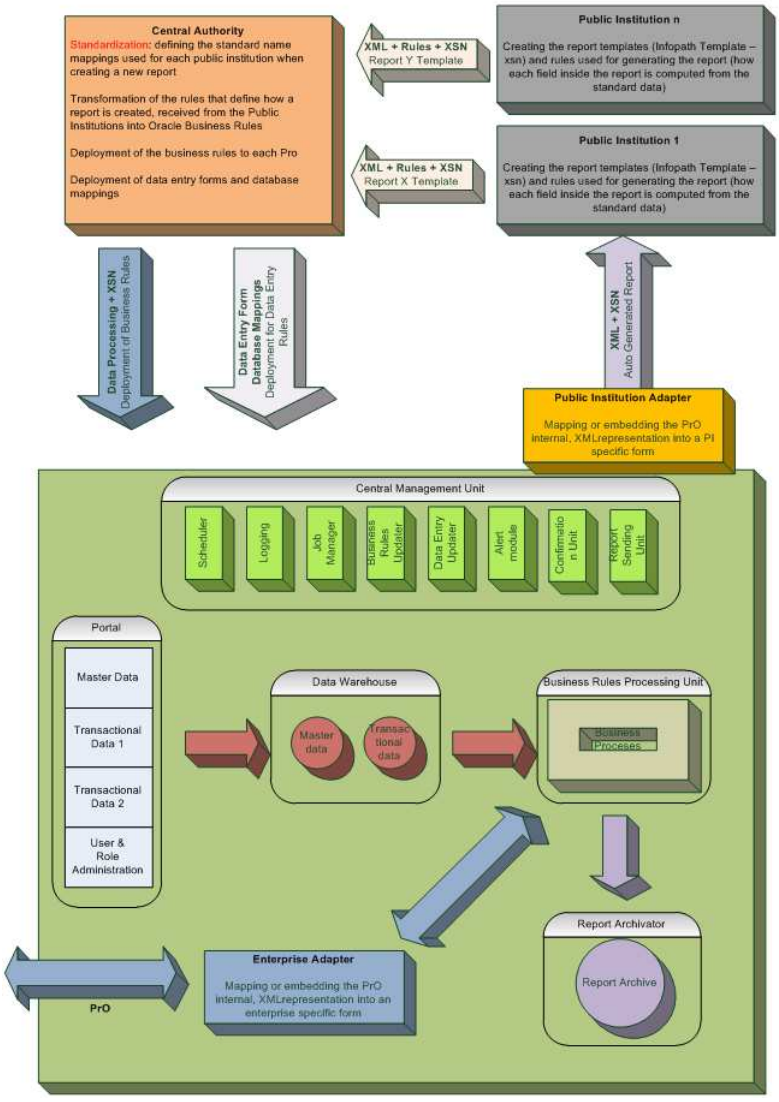



FIG. 6.3. *The Architecture of the reporting PRO type e-System.*

**7. Conclusions and future work.** In this paper we presented a system designed to automate the data migration between various data sources. The solution is able to handle data transfers between different database instances. It is designed to work with dynamic data schemas, and requires minimal user input and interaction.

Organizations usually collect large volumes of data in their internal databases. In many cases, the stored data refers to the company structure, employees, customers and projects. With the development of new database systems, companies might want to change the database in use and transfer large amounts of data. Moreover changes in company structure and number of employees might generate modifications in the database schema. As a consequence, not only that the underlying database system is changed, but the data transfer would occur between two different schemas. In this context, the system presented in this paper can optimize the effort needed to correctly manage the migration process. As a case study, for example, we evaluated the solution in a real-world system implementation designed to automatically collect report data. The migration system leaded to the optimization of the overall performance of the reporting system by ensuring a transparent data retrieval process. It allows the user to easily manage the data mapping processes for filling the required reports.

In the future we plan to further extend the migration system with advanced mechanisms for data type recognition and conversions. We plan to design a solution where the Warehouse repository can be dynamically updated from a central repository of knowledge. Furthermore, in order to provide platform independence, the software will be ported to Java. The process might be lengthy as there is a significant difference between the facilities Java and C# provide as programming languages. A possible outcome of having two software products written in separate programming languages is that the capabilities offered by these languages with respect to database connections can result in a significant efficiency gain. That is to say both Java and C# have improved database connectors for certain types of databases. Due to this a user can choose the software product to use according to the types of the source and destination databases. Platform independence is a desired goal of the project. Another feature that should be implemented in the near future is support for migration interruption. There are cases when software is subject to failures and for a migration system this type of event is critical. Interrupting the migration system may result in data loss or data inconsistency. Therefore a module that takes care of any unwanted interruptions and helps restore the migration process from the point it was interrupted is necessary.

**Acknowledgments.** The research presented in this paper is supported 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. The work has been co-funded 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.

REFERENCES

[1] P. Howard, *Data Migration*, A White Paper by Bloor Research, Informatica, October 2007, Last accessed March 12, 2011, from http://vip.informatica.com/?elqPURLPage=552.
[2] M. Otey, and D. Otey, *Microsoft SQL Server 2005 developerś guide*, Microsoft, *McGraw-Hill Osborne Media*, 2005.
[3] Talend, *Talend Open Studio, User guide*, Last accessed June 29, 2010, from http://www.talend.com/index.php.
[4] J. Paul, *SQL Data Types*, Last accessed July 1, 2010, from http://onlamp.com/pub/a/onlamp/2001/09/ 13/aboutSQL.html.
[5] J. Morris, *Practical Data Migration*, British Computer Society Ltd, 2006.
[6] Microsoft, *Implementing Data Transfer Object in .NET with a Typed DataSet*, Last accessed July 4, 2010, from http://msdn.microsoft.com/en-us/library /ff650461.aspx.
[7] Talend, *Data Migration*, Last accessed March 13, 2011, from http://www.talend.com/solutions-data-integration/data-migration.php.
[8] M. Tungare, P. S. Pyla, M. Sampat, and M. A. Perez-Quinones, *Syncables: A Framework to Support Seamless Data Migration Across Multiple Platforms*, IEEE International Conference on Portable Information Devices (PORTABLE07), Orlando, FL, 2007, pp. 1–5.
[9] P. Allaire, J. Augat, J. Jose, and D. Merrill, *Reducing Costs and Risks for Data Migrations*, Hitachi, Last accessed February, 2010, from http://www.hds.com/assets/pdf/white-paper-reducing-costs-and-risks-for-data-migrations.pdf.
[10] M. Chatzidimitriou, and A. Koumpis, *Marketing One-stop e-Government Solutions: the European OneStopGov Project*, IAENG International Journal of Computer Science, 35:1 (2008).
[11] F. Pop, C. Dobre, D. Popescu, V. Ciobanu, and V. Cristea, *Digital Certificate Management for Document Workflows in E-Government Services Infrastructure*, in Proc. of The 9th IFIP WG 8.5 International Conference on Electronic Government (EGOV 2010) Lausanne, Switzerland, 2010, pp. 363–374.
[12] A. Marin, C. Dobre, D. Popescu, and V. Cristea, *E-System for Automatic Data Migration*, in Proc. of the 12th Intl. Symposium on Symbolic and Numeric Algorithms for Scientific Computing, Timisoara, Romania, 2010, pp. 479–485.

*Edited by:* Dana Petcu and Alex Galis
*Received:* March 1, 2011
*Accepted:* March 31, 2011

Scalable Computing: Practice and Experience
Volume 12, Number 1, pp. 63–78. http://www.scpe.org
ISSN 1895-1767


# A QOS-BASED FRAMEWORK FOR THE ADAPTATION OF SERVICE-BASED SYSTEMS

RAFFAELA MIRANDOLA* AND PASQUALINA POTENA†

**Abstract.** Since a system may require dynamic adaptation for several reasons (e.g., a new version may be available and a new functionality or a different level of quality of service) it should be possible to dynamically adapt a service-based system in an automated manner. In this paper we give a general overview of the main components of a framework, based on an optimization model, that dynamically adapts a service based system (i.e., both the structural and behavioral software and hardware architecture) while minimizing the adaptation costs and guaranteeing a required level of the system qualities. Adaptation actions can be triggered both by a user request and/or automatically after the runtime violation of system quality constraints, or the appearing/disappearing of services into the environment. In this paper we provide also a deeper discussion of the optimization model that is the core of the framework by providing an example of instantiation of the model together with a first experimentation.

**Key words:** service-based adaptation; quality of services and non-functional aspects; optimization model.

**1. Introduction.** Due to the frequent changes required to software after its release, the development processes are rapidly going towards small effort dedicated to the early phases and large effort for the after-deployment phases. Since a system may require dynamic adaptation for several reasons, such service evolution (e.g., a new version may be available), hardware volatility (e.g., network quality changes) and varying users demands requiring new requirements (e.g., a new functionality or a different level of quality of service) it should be possible to dynamically adapt a service-based system in an automated manner. An application should be self-adaptive in order to automatically and autonomously adapt its behavior to respond to changes. For example, in the pervasive computing domain [13], an application should be context-aware self-adaptive, i.e., the context capturing the notion of information about the physical environment should trigger certain changes. The switch from a Wifi network to a GSM one could, for example, be require (survey on context-aware systems can be found in [13, 24, 35, 39]).

In this paper we introduce a framework, based on an optimization model, that dynamically adapts a Service-Based System (SBS) while minimizing the adaptation costs and guaranteeing a required level of the system qualities. Adaptation actions can be triggered both by a user request and by an automatic request raised after the runtime violation of system quality constraints, or the appearing/disappearing of services into the environment. In particular, in this paper we give a general overview of the main components of the framework by providing a deeper discussion of the optimization model that is the core of the framework.

Specifically, to modify the software structure the framework replaces existing software services with different available instances and embeds into the system new software services if necessary. With respect to changes in the system behavior it modifies the system scenarios (expressed, for example, as UML Sequence Diagrams) by removing or introducing interaction(s) between existing services and between these latter and new services. Finally, to modify the hardware architecture it may re-deploy (deploy) existing/new services on the hardware nodes. Furthermore, it may improve (adopt new) hardware resources (e.g., CPU, disk, memory, etc) and modify the interaction between hardware nodes (e.g., introducing a new link). Note that, these hardware improvements can be done only if the service belongs to the application developers or from the provider itself when composing some complex SBSs.

This paper extends the work in [30] by providing a more detailed presentation of the framework. In particular, we provide a deeper discussion for the example of instantiation of the model by providing a first numerical experimentation.

The paper is organized as follows: Section 2 introduces related work and discusses the novelty of our contribution; in Section 3 we introduce our framework; in Section 4 we provide the formulation of the optimization model that represents the core of our framework; in Section 5 we provide an example of instantiation of the model together with a first experimentation; conclusions are presented in Section 6.

**2. Related Work.** A wide range of approaches and frameworks for adaptation have been proposed (see surveys [4, 14, 21]). Most of them typically adapt a system by (i) service selection (see, for example, [9, 10, 34]

*Politecnico di Milano, Dipartimento di Elettronica e Informazione, Piazza Leonardo da Vinci, 32, Milano, 20133 Italy (mirandola@elet.polimi.it).

†Università degli Studi di Bergamo, Dipartimento di Ingegneria dell'Informazione e Metodi Matematici, Viale Marconi, 5, Dalmine (BG), 24044 Italy (pasqualina.potena@unibg.it).

detailed below), (ii) parametrization (see, for example, [8]) or (iii) exploiting the inherent redundancy of the SOA environment (see, for example, [17]).

Paper [9] presents a runtime selection of services composing a web service while satisfying QoS constraints, maximizing some QoS (e.g., reliability) and minimizing other attributes, such as the price. It leverages both on formulas for estimating QoS attributes with respect to some workflow constructs (i.e., sequence, switch, flow and loop) and genetic algorithms for solving the problem.

In [34] an approach for supporting the service selection driven by QoS attributes is presented. It enhances the classical service scenario (i.e., consumer, provider and registers) by assuming that the service register stores, apart from the information of the services (e.g., QoS data and interfaces), information about services usage and service trees. A service tree is a binary search tree generated by adding a node for each service that could be selected by the customer. Each node is labeled with the service score, which is defined as the weighted sum of the qualities of the service. A service score could be updated, for example, if the service receives a good feedback from other consumers. The algorithm for building the service tree and for updating and selecting (or re-selecting) the best service is presented. In fact, a service could be re-selected, for example, if either the currently selected service is not available or the QoS of the service changes.

In [10] it is presented an approach for the self-adaption of a SOA system in order to meet reliability and availability requirements while dynamically changing the environment. The approach allows adopting simultaneously (for different users, but also for different requests generated by the same user) adaptation actions based on the service selection and architecture selection.

In [19] an approach supporting the impact of the replacement of a service on other services is presented (e.g., how the change of a service provider impact on the other services). It is based on workflow patterns and on the value of changed information [18] for sequence and different parallel pattern scenarios.

Our framework takes advantage of the use of monitoring technique, using approaches like the ones detailed below.

In [31] the VieDAME (Vienna Dynamic Adaptation and Monitoring Environment forWS-BPEL) tool is introduced. It supports the monitoring of BPEL processes according to certain QoS criteria and the replacement of a service with one of its available functional equivalent alternatives. Moreover, it allows the handling of interface mismatches at a SOAP message level.

In [6] an approach for the dynamic monitoring and recovery of BPEL processes is introduced. The monitoring exploits WSCoL (Web Service Constraint Language), whereas the recovering is implemented by introducing WSReL (Web Service Reaction Language) providing a library of atomic actions that can be combined to create reaction strategies, which are activated whenever an anomaly is discovered by the monitoring. An implementation of the approach is proposed using AOP techniques to activate both dynamic monitoring and recovery during process execution.

Lodi et al. [27] describe a QoS-aware middleware that can be included in application servers in order to implement Service Level Agreement (SLA)-driven clustering of servers. The middleware operates by dynamically configuring, monitoring and balancing the load among different servers, in order to provide strong QoS guarantees despite high variability in the request rates.

In order to adapt a system, for example, while changing the requirements (e.g., new functionalities could be claimed or a different level of quality of service), it might be necessary to modify the structure of the service composition, e.g., new services have to be embedded into the system and the interactions between existing services have to be changed for assuring a certain level for the system qualities. An application should automatically and autonomously adapt its behavior (i.e., its service composition) to respond to changes under the guide of the system qualities.

Some frameworks have been introduced for dynamically generating a service composition, e.g., [36] surveys automated web service composition methods. Usually they adapt a system only in a proactive way, after the adaptation request triggered by a user. These frameworks support the service selection with respect to a service composition defined by the user (see, for example, the VRESCo runtime environment [37] where a user can require a service composition satisfying quality constraints) or choose the service composition, which they have generated together with a finite set of other candidates, that better fulfills the required quality (see, for example, [11], where it is presented a framework composing services at run time for answering a user query and the survey [21] where it is defined a service composition middleware model describing the existing service composition middleware in pervasive environments).

In our view, a service composition should be dynamically generated, other than after a user request, sponta-

neously by the application itself. The spontaneous service composition is typically not supported by the existing frameworks, such as the ones for pervasive environments (see survey [21]) where instead it should be one of the fundamental properties of an application. In [22] a spontaneous service integrator middleware is presented supporting the extension of system functionalities by integrating new services appearing into the environment with existing services. Moreover, it supports the spontaneous service selection while services appear (disappear) into the environment. For the selection activity the authors uses a metric measuring the degree of non-functional QoS similarities between two equivalent syntactic or semantic services.

They do not consider the impact of replacing a service on the quality of the whole system. Moreover, they do not consider the possibility of replacing also the other system elementary services, i.e., the service selection with respect to all services. As opposite, our framework supports the spontaneous adaptation of the hardware and structural and behavioral software architecture if system quality constraints are violated at run time or services appear (disappear) in the environment. Our framework, based on an optimization model, minimizes also the cost of adapting the system while assuring a required level of the system qualities.

The system quality depends, both on the software features, and on the hardware features. A set of software adaptation actions may differ for the system quality achieved after their application while changing the hardware structure. In fact, for example, the replacement of a service with a certain instance may be a good decision for the system reliability, but at the same time it could increase the system response time because of the execution time of the instance, which, in this case, may be decreased by modifying the characteristics of the hardware resources (e.g., CPU, disk, memory, network throughput, etc). Therefore, an application should automatically and autonomously adapt, other than its software behavior, its hardware architecture to respond to changes, whenever this is possible.

*Novelty of our approach*

With respect to existing approaches, the following major aspects characterize the novelty of the proposed method:

- This is one of the few papers (to the best of our knowledge our previous work is the first [30]) introducing a dynamic self-adaptive framework supporting both the software (including both static and dynamic models) and hardware architecture adaptation using an optimization model minimizing the adaptation costs and assuring a required level of the system qualities.
- Adaptation actions can be triggered both by users, for example, claiming the introduction of new functional (non-functional) requirements or changing already implemented system functionalities (non-functional requirements satisfaction) and spontaneously by the framework itself after the runtime violation of system quality constraints or the appearing/disappearing of services into the environment.
- The proposed approach is general and does not rely on specific architectural style, development process or service application domain. It could be specialized with respect to the application domain chosen. For example, it could be enhanced to support the context-aware property of the pervasive systems exploiting the properties discussed in [4].
- The proposed optimization model is independent from the methodology adopted to represent system architectural models and from the strategy used to generate the adaptation plans for each adaptation requirement. In fact, we assume that each scenario of a functionality offered by the system is represented by an UML Sequence Diagram. However, since our model uses as parameters of a scenario the number of busy periods of an elementary element (see Section 5), a scenario could be represented with whatever notation that permits to describe the system scenarios (e.g., the Message Sequence Charts, MSCs [2], or the UML Collaboration Diagrams).
- Our framework can facilitate the work of a software engineer. In fact, a user does not have to insert as input value architectures satisfying all changes required, but possible sets of (software and hardware) adaptation actions (called *adaptation plans*) for each new requirement (called *adaptation requirement*), which he/she want to implement. When the computation time becomes too big, e.g., while increasing the number of adaptation plans, the adoption of metaheuristic techniques possibly combined with the introductions of dependencies between adaptations plans of different adaptation requirements could allow the reduction of the research space.

**3. Framework.** Figure 3.1 depicts the main modules of our framework. The framework, based on an optimization model (generated and solved by the *Generator and Evaluator* module), dynamically adapts a service based system (by changing software and hardware features by means of the *Executor* module) while

minimizing the adaptation costs and guaranteeing a required level of the system qualities. Adaptation actions can be triggered both by requests of a user (which can be defined using the *User Requests Manager* module that can also interact with the *Monitor* to know if the system quality constraint are violated by the current runtime system) and/or automatically by the framework itself (after it receives alerts from the *Monitor* using probes for monitoring the system and interacting with services repositories by means of the *Provider Info* module).

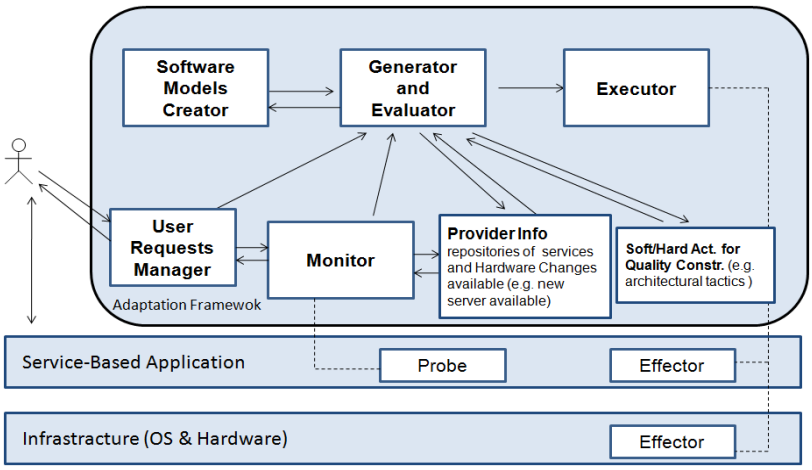



FIG. 3.1. *The Adaptation Framework*

**3.1. Software Models Creator.** Goal of the Software Models Creator is to generate system models (e.g., component diagram, sequence diagrams and deployment diagram), from the system implementation (by assuring consistency between the architecture and implementation [33]) or by modifying the existing models as suggested by the *Generator and Evaluator* module. The optimization model, generated and solved by the *Generator and Evaluator* module, is independent from the methodology adopted to represent system architectural models and from the strategy used to generate the adaptation plans for each adaptation requirement.

**3.2. User Requests Manager.** A user can use the User Requests Manager module to make adaptation requests. He/She may claim the introduction of new services offered by the system and modify the dynamics of existing services [1]. Furthermore, he/she may either require a value threshold (e.g., as lower bound) for a new system quality or modify the value threshold for an already required system quality. Before making his/her requests, the user can know (by means of the module *Monitor* interacting with the *User Requests Manager*) if there are requirements violated by the current runtime system. For each adaptation requirement, the *User Requests Manager* allows users to annotate the software architecture diagrams for defining different set of adaptation actions able to guarantee this new requirement.

In order to keep as simple as possible the modeling aspects of our work, we assume that plans of different adaptation requirements are independent from each other, namely the modifications claimed by a plan do not comprise both the satisfaction of the existing functional requirements, and the application of plans for other adaptation requirements. We intend to work for relaxing such a assumption. To this extent, we are investigating how to enhance the framework using the guidelines of the existing tools (like the CoDesign tool [5]) allowing architects geographically distributed to cooperate in order to design a system.

A software adaptation plan may suggest how to change both the structure and the behavior of a system. Specifically, to modify the system structure it suggests how to replace existing software services with different available instances and if the adoption of new software services is necessary. With respect to changes in the system behavior it may suggest how to modify the system scenarios (expressed, for example, as UML Sequence Diagrams) by removing or introducing interaction(s) between existing services and between these latter and new services.

[1] Throughout the paper, the services offered by the system will be named external services.

To define the software adaptation plans, for example, initial solutions based on the use of meta-heuristic techniques can be found in [28]. In fact, in [28] it is suggested how to explore the design space for generating architecture candidates with required performance features. It is basically based on two methods executed in parallel for generating the best architectural candidates from a (some) architecture(s) fulfilling the functional requirements. The first one uses the classic metaheuristic search techniques [7], whereas the second one uses specific knowledge of the performance analysis to specifically solve performance problems in current candidates. As last step, a detailed performance simulation is executed for the best architecture candidates.

To generate software adaptation plans for non-functional requirements several approaches can be adopted, for example, looking for "architecture bad smells", i.e., recurring software designs solutions that negatively impact software quality [15] and/or the application of architectural tactics, i.e., reusable architectural building blocks that provide a generic solution to address issues pertaining to quality attributes [25].

Finally, for each adaptation requirement, the *User Requests Manager* allows users to annotate the hardware architecture diagrams to suggest also hardware architecture changes (such as how to deploy existing/new services and if adopt new hardware nodes or links). For example, if he/she is a service provider, in order to improve the system response time he/she could suggest to try to improve its hardware resources (e.g., CPU, disk, etc.).

**3.3. Monitor.** Goal of the Monitor is to monitor the system at runtime and raise the violation of non-functional requirements, e.g., the system reliability is under the minimum level required. Moreover, it claims the appearing/disappearing of services in the environment by interacting with service repositories managed by service providers. Such repositories are accessed by interacting with the *Provider Info* module.

For implementing such a module the guidelines of the several approaches presented in literature leveraging on monitoring technique can be used (e.g., [27, 31, 41]).

Moreover, the monitor could continually measure the QoS attributes of the services (see, for example, the monitor used in [37] ) or the providers could improve the estimate of the services non-functional property by monitoring them e.g., as suggested in [29], the providers monitor the resources utilization as client applications access them, and dynamically adjust the advertised average service times as better estimates are computed.

**3.4. Provider Info.** Goal of the Provider Info module is to allow accessing the service repositories managed by service providers (or by the brokers). These latter ones can use such a module to notify possible hardware changes that could be performed (e.g. possible deployments/re-deployments of services, how can be improved hardware resources, such as CPU, disk, etc.). By using such information the framework automatically suggests additional hardware actions by introducing variables and constraints in the optimization model generated and solved by the *Generator and Evaluator* module.

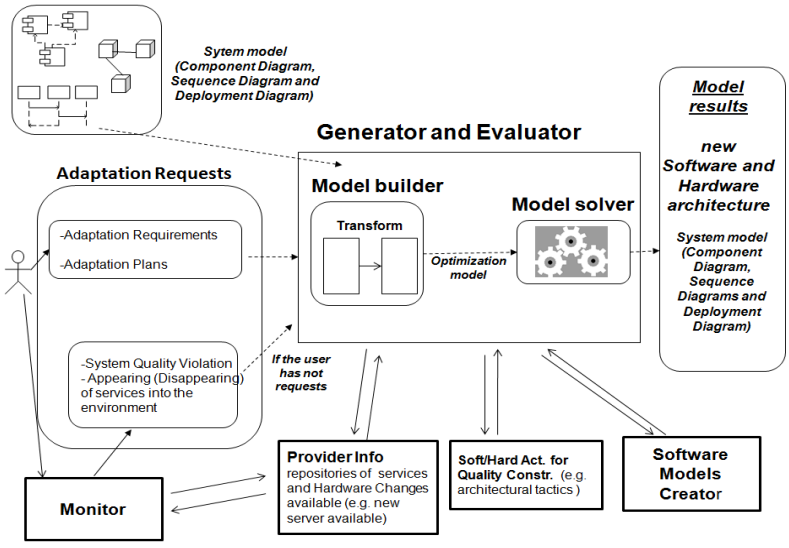



FIG. 3.2. *The Generator and Evaluator module and its environment*

**3.5. Generator and Evaluator.** Figure 3.2 shows the *Generator and Evaluator* module within its working environment. It is made of two components, which are a *Model builder* and a *Model solver*.

The *Model builder* after receiving either adaptation requests from the user (by means of the *User Requests Manager* module) or alerts from the *Monitor* generates the optimization model in the format accepted from the solver (e.g., LINGO [1]).
The user defines adaptations plans for each new requirement, whereas if the *Model builder* gets alerts from the *Monitor* about the violation of non-functional requirements, it itself generates a set of adaptation plans for each non-fulfilled requirement using the *Software/Hardware Actions for Quality Constraints* module. Such a module is a repository of software and hardware adaptations actions that may be used for improving the system qualities (e.g. general solutions as the architectural tactics or software/hardware actions defined by the system maintainer based on his/her experiences). Instead, if the *Monitor* notifies the appearing (disappearing) of services in the environment, the *Model builder* generates the optimization model for finding the best service selection actions while minimizing the replacement costs and guaranteeing a certain level for the system qualities. Note that the framework could be enhanced for generating adaptation plans also for each non-functional requirement of the system.

The *Model solver* processes the optimization model received from the builder and produces the results, that consists in a set of software and hardware adaptation actions. It suggests how to adapt both the static and dynamic software architecture and the hardware architecture. The combined use with the *Software Models Creator* module generates the new software (hardware) architectural models (e.g. component, sequence and deployment diagrams).

The framework could be enhanced for supporting the service level agreement (re-)negotiation, for example, if there is not a configuration of services able to guarantee the required system quality.

**3.6. Executor.** Goal of Executor module is to implement the adaptation actions suggested by the module *Generator and Evaluator*, i.e., it implements the system as described by the new software (hardware) diagrams defined by the optimization model. To this extent, existing guidelines of the approaches supporting the dynamic service invocation (e.g. [26]) and the ones for dynamically adapting the system behavior (e.g. the context-oriented programming [20]) could be exploited.

**4. Optimization Model Formulation.** In this section we introduce the general formulation of the model, that is generated and solved by the *Generator and Evaluator* module, aimed at finding the optimal set of adaptation actions needed to tackle required changes to the software and hardware architecture of a service based system. "Optimal" is here intended as the actions that incur in a minimum cost while guaranteeing a required level of a number $Q$ of quality attributes (such as response time, availability, etc.) for the whole system.

Since our model may support different service application domain, we adopt a general definition of software service: it is a self-contained deployable software module containing data and operations, which provides/requires services to/from other elementary elements. A service instance is a specific implementation of a service.

Through the composition of its $n$ software services, the system offers external services to users. Let us assume that for each external service we dispose of a diagram (e.g. a UML Sequence Diagram (SD)) describing its dynamics in terms of interactions that take place between software services to provide the external service. Let us also assume that we dispose of SDs for external services that are not active in the current system implementation, but they can be activated to satisfy new requirements.

Let $s_i$ be the $i$-th service ($1 \leq i \leq n$).

Let $K$ be the total number of SDs, i.e., the ones related to active external services plus the ones related to external services that may be introduced.

Let $pexec_k$ be the probability that the $k$-th system functionality will be invoked. It must hold: $pexec_k \geq 0$ for all $k = 1 \dots K$ and $\sum_{k=1}^{K} pexec_k = 1$. This information can be synthesized from the operational profile [32][2]. It is obvious that for a non-active functionality $k$ we get $pexec_k = 0$.

*Getting User Requests* - An *user adaptation scenario* is a set of new requirements to be satisfied, which are claimed by the user. The new requirements are: (i) a new functional requirement, i.e., either introducing a new external service or modifying the dynamics of an existing external service; (ii) a new non-functional requirement, i.e., either requiring a value threshold (e.g. as lower bound) for a new system quality or modifying the value threshold for an already required system quality.

[2]Note that such assumption might be not realistic in all cases the operational profile may be not (fully) available. However, in such cases the domain knowledge and the information provided by the software architecture could be used for estimating it, as suggested for example in [38].

In order to introduce a new external service one of the SDs that are not active must be activated, namely it enters the set of SDs that contribute to the costs, reliability, response time etc. of the whole software architecture. In order to modify the dynamics of an existing external service certain interactions of the SD related to the service have to be added/removed. Finally, to introduce (modify) new/existing non-functional requirements certain interactions of the SDs may be added/removed, for example, by following the guidelines of the "architectural tactics" (see Section 3).

*Getting Alerts by the Monitor* - An alert raised by the *Monitor* can claim: (i) the violation of a non-functional requirement, e.g. the system reliability is under the minimum level required; (ii) the appearing/disappearing of a service in the environment.

In order to satisfy either the new requirements required by the user or exploit the alerts raised by the *Monitor* some adaptation actions have to be performed. The user defines adaptation plans for each requirement, whereas the framework itself defines adaptation plans when it gets alerts about the violation of non-functional constraints. An adaptation plan is a set of actions modifying the static and dynamic structure of the software architecture and the hardware architecture to (exploit a certain alert) address a certain requirement. Obviously, for each requirement (alert) several adaptation plans may be available. Since they suggest different adaptation actions, they may differ for adaptation cost and/or for the system quality achieved after the application of their actions. Let $AP_r$ be the set of adaptation plans available for the $r$-th requirement in the user adaptation scenario (the $r$-th alert raised by the *Monitor*)[3].

We consider the following software adaptation actions:

1. *Introducing new software service*: An adaptation action may suggest to embed into the system one or more new software services [4]. We call $NewS$ the set of new available services (accessed by the *Provider Info* module) that can provide different functionalities, whereas $news_h$ represents the $h$-th service of $NewS$.
2. *Replacing existing service instances with functionally equivalent ones*: An adaptation action may suggest to replace a software service with one of additional instances available for it (e.g. a web service available in the market). We assume that the additional instances available for the service $s_i$ are functionally compliant with it, i.e., each instance provides at least all services provided by $s_i$ and requires at most all services required by $s_i$ [5]. The instances may differ from $s_i$ for cost and quality attribute (e.g. reliability and response time). We call $Avail_i$ the set of instances available for the $s_i$, while $s_{ij}$ is the $j$-th instance of $Avail_i$.
3. *Modifying the interactions between software services in a certain external service*: An adaptation action may suggest to modify the system dynamics by introducing/removing interactions between software services within a certain external service.

The system quality depends on the hardware features other than on the software features. In fact, a set of software adaptation actions may differ for the system quality achieved after the application of their actions while changing the hardware structure (e.g. the system response time may decrease while improving the processing capacity of a hardware node). Since the services are not acquired in terms of their binaries and/or source code, but they are simply used while they run within their own execution environment (that is not necessarily under the control of the system using them), hardware changes can be suggested by the service providers. To this extent, these latter can insert such information by means of the *User Requests Manager* module while requiring adaptation requirements and notify them by means of the *Provider Info* module.

In the following we discuss possible hardware adaptation actions.

- *Defining the deployment of software service on hardware nodes*: An adaptation action may suggest how to re-deploy existing services and/or deploy the new ones on the hardware nodes.
- *Introducing hardware resources*: An adaptation action may suggest to embed into the system one or more new hardware resources, e.g. a new hardware node where to deploy the services[6].
- *Modifying hardware resources*: An adaptation action may suggest to modify the characteristics of the underlying hardware resources (e.g. CPU, disk, memory, network throughput, etc).

[3] In the remainder of the paper a plan $p \in AP_r$ is also called $ap_{rp}$.

[4] Note that such type of action has to be associated to another action that indicates how this software service interacts with existing services, therefore it modifies the interactions within certain functionalities (see last type of software action).

[5] As we have remarked in [12] such an assumption could be relaxed by introducing integration/adaptation costs.

[6] Note that such type of action has to be associated to another action that indicates how this node interacts with existing nodes (see last type of hardware action).

TABLE 4.1
*Example of adaptation plans*

| Requirement ID | Adaptation Plan ID | Adaptation Plan Description |
|---|---|---|
| $req_1$ | $ap_{11}$ | *Replacing $s_3$ with its first instance AND Replacing $s_4$ with its second instance* |
| | $ap_{12}$ | *Adding a new service $news_2$* |
| $req_2$ | $ap_{21}$ | *Replacing $s_2$ with its first instance* |
| | $ap_{22}$ | *Adding a new service $news_1$ AND Replacing $s_5$ with its first intance* |
| | $ap_{23}$ | *Adding a new service $news_2$* |

- *Modifying the interactions between hardware nodes*: An adaptation action may suggest to introduce/remove connection links between hardware nodes.

It is obvious that any combination of such software and hardware adaptation actions may have a considerable impact on the cost and quality of the system. Therefore, our optimization model aims at quantifying such impact to suggest the best adaptation plans that still minimize the costs while satisfying the quality constraints defined according to the quality properties of interest. These latter predict the quality of the system after the adaptation phase, i.e., after applying the adaptation plans for implementing the new requirements (exploiting the alerts of the monitors). Therefore, for each non-functional property required for the system (i.e., existing properties or new ones required by the user) a constraint is defined verifying the impact of the changes on the system quality.

Finally, in order to open the solution space to additional possibilities, it may suggest additional service replacement actions and hardware actions. In other words, in the optimization model we leave to the solver the possibility to choose additional service replacement actions and hardware actions that have not been embedded in any selected plan.

**4.1. An example of Output of the optimization model.** Let us assume that a user claims two new requirements for a system composed by five services (i.e., $s_1$, $s_2$, $s_3$, $s_4$ and $s_5$). Through the composition of these software services, the system offers external services to users. Table 4.1 describes the user adaptation scenario, where for each new requirement the adaptation plans suggested by the user are reported. For example, $req_1$ can be managed by either "Replacing $s_3$ with its first instance AND Replacing $s_4$ with its second instance" ($ap_{11}$) or "Adding a new service $news_2$"($ap_{12}$). For the sake of simplicity, we have used the natural language for defining the requirements and the adaptation plans.

Let us suppose that to achieve the optimal cost of adaptation and assuring a required level of the system qualities the optimization model suggests to adopt the following plans: adaptation plan $ap_{12}$ for the first requirement and adaptation plan $ap_{22}$ for the second requirement. In this case, the static structure of the software architecture, describing dependencies between two services of the system and the behavior of the system with respect to the external service $\overline{k}$ would change as shown in Figure 4.1. In particular, the model could suggest all the replacements of existing services (i.e., $s_{ij}$), as well as the adoption of the new services $news_1$ and $news_2$. In addition, the model could also suggest to modify the hardware architecture as shown in Figure 4.2. Note that a new hardware node is introduced and some links are removed or adopted [7].

**4.2. Model Variables and Elementary Constraints.** The following variables help to select an adaptation plan for each requirement of a user adaptation scenario (for each alert raised by the *Monitor*). For each requirement $r$ of a user adaptation scenario, exactly one plan must be chosen if it is a functional requirement. If it is a non-functional requirement it may be not necessary to select one of the plans suggested for it by the users since for each required quality of the system a constraint, predicting the quality resulting after the adaptation phase, is defined. For example, the reliability for the system is assured above a certain threshold without applying one of the "architectural tactic" suggested by the user. Similarly, for each alert claiming the violation of non-functional constraints it may be not necessary to select one of the plans generated by the framework itself.

[7] In the figures we have marked with bold the modifications brought after the application of the plans and we have put a cross on the interactions that are removed.

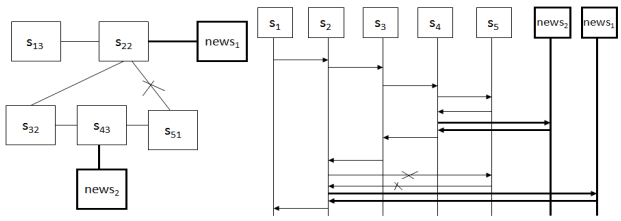



FIG. 4.1. *Resulting static structure and a SD of the system after the application of* $ap_{12}$, $ap_{22}$

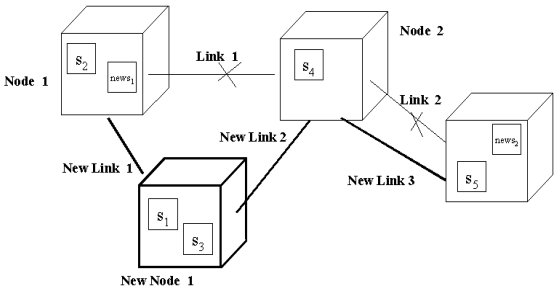



FIG. 4.2. *Resulting Hardware Architecture of the example*

$$y_{rp} = \begin{cases} 1 & \text{if } p \text{ is the plan chosen for requirement } r \\ & \text{(for the alert } r\text{) } (p \in AP_r) \\ 0 & \text{otherwise} \end{cases}$$

The following variables help to select an instance available for the $i$-th existing service. For each existing service $i$ still used after the adaptation phase, exactly one instance must be selected.

$$x_{ij} = \begin{cases} 1 & \text{if the instance } j \text{ is chosen for the service } i \\ & (j \in Avail_i) \\ 0 & \text{otherwise} \end{cases}$$

If there are no available instances, we assume that $Avail_i$ includes the element $i$ itself. In addition, if a plan is part of the model solution then all instances that it suggests for the existing services have to belong to the solution.

The following variables help to select the new software services to be introduced.

$$z_h = \begin{cases} 1 & \text{if the service } h \text{ is chosen } (h \in NewS) \\ 0 & \text{otherwise} \end{cases}$$

Similarly, if a plan is part of the model solution then all new services that it suggests to introduce have to belong to the solution.

Additional constraints, which can be expressed as contingent decisions [23], may be added for claiming incompatibility between services due to problems such as technology and licensing.

Different kinds of variables and constraints can be defined to support hardware actions (e.g. variables to introduce new hardware nodes and links). In the following, for example, we define the variables and discuss constraints to suggest the deployment of the existing/new services on hardware nodes.

The following variables suggest how to deploy the $i$-th existing/new service on an hardware node. Each existing service $i$, which is still used after the adaptation phase, must be deployed on exactly one node $t$. If the new service $h$ is selected, it must be deployed on exactly one node $t$.

$$S_{it} = \begin{cases} 1 & \text{if the existing/new } i\text{-th service} \\ & \text{is deployed on node } t \ (t \in HN) \\ 0 & \text{otherwise} \end{cases}$$

where $HN$ is the set of hardware nodes. Additional constraints can be defined for existing services that cannot be re-deployed on a different node for legacy reasons (i.e., their deployment does not have to be modified). Also, if a plan is part of the model solution then all the mappings of existing/new services on hardware nodes that it suggests have to belong to the solution. Finally, additional constraints have to be defined to guarantee that a path exists, i.e., a set of links exists, between two (existing and/or new) interacting services.

**4.3. Cost Objective Function and Quality Constraints.** We want to adapt a system while minimizing the adaptation cost under constraints on a number $Q$ of quality attributes (such as response time, availability, etc.) of the whole system.

Let us suppose that the value of each attribute of any service composing the system after the adaptation phase depends on the value of parameters $u$, $v$ and $w$.

Let $u$ denote the maximum number of software architecture observable parameters, such as the number average of busy periods (i.e., the number of invocations of the service within a certain SD, and it can be easily estimated by parsing the diagram and counting the number of activations along its lifeline). Let $v$ represent the maximum number of hardware observable parameters, e.g. the processing capacity of the node hosting the service measured, for example, as the number of instructions per time unit that the resource can execute, under the assumption that all instructions require equal time complete. Finally, let $w$ be the maximum number of parameters expressing the specific features of its implementation (e.g. the reliability of the instance used for replacing an existing service).

Let $\Gamma_{kq} : \Re^u \times \Re^v \times \Re^w \rightarrow \Re$ ($\bar{\Gamma}_{kq} : \Re^u \times \Re^v \times \Re^w \rightarrow \Re$) be the function that predicts the $q$-th quality attribute ($1 \leq q \leq Q$) of an existing/new service from the $u$, $v$ and $w$ parameters after the adaptation phase with respect to the $k$-th external service. For the sake of readability, we introduce here a formulation without correlations among different quality attributes and services interacting between each other (in Section 6 we discuss how relaxing such assumptions).

We can represent the value of the $q$-th quality attribute of the $i$-th existing service as a function of the decisional strategy:

$$\theta^q_{ki} = \sum_{j=1}^{|Avail_i|} x_{ij}\Gamma_{kq}(\lambda^q_{i1}, \ldots, \lambda^q_{iu}, \lambda'^q_{i1}, \ldots, \lambda'^q_{iv}, \Lambda^q_{ij1}, \ldots, \Lambda^q_{ijw})$$

$\theta^q_{ki}$ is a function of the value of the software architecture observable parameters $\lambda^q_{it}$'s denoting, for example, the added/canceled busy periods that the chosen adaptation plans suggest for the service $i$ within the $k$ external service. $\theta^q_{ki}$ depends also on the value of the hardware observable parameters $\lambda'^q_{it}$'s, denoting, for example, the processing capacity of the hardware node, that is suggested by one of the chosen adaptation plans or that could be one of the additional actions of the model, where the service is deployed after the adaptation phase. Finally, $\theta^q_{ki}$ depends also on features $\Lambda^q_{ijt}$'s indicating, for example, the probability of failure of its selected instance or the size of the executable file containing this instance after the adaptation phase.

We can represent the value of the $q$-th quality attribute of the $h$-th new service as a function of the decisional strategy:

$$\bar{\theta}^q_{kh} = z_h\bar{\Gamma}_{kq}(\lambda^q_{h1}, \ldots, \lambda^q_{hu}, \lambda'^q_{h1}, \ldots, \lambda'^q_{hv}, \bar{\Lambda}^q_{h1}, \ldots, \bar{\Lambda}^q_{hw})$$

$\bar{\theta}^q_{kh}$ is a function of the value of the software architecture observable parameters $\lambda^q_{ht}$'s denoting, for example, the number of busy periods that the chosen adaptation plans suggest for the $h$-th new service within the $k$-th external service. $\bar{\theta}^q_{kh}$ depends also on the value of the hardware observable parameters $\lambda'^q_{ht}$'s, denoting, for example, the processing capacity of the hardware node, that is suggested by one of the chosen adaptation plans or that could be one of the additional actions of the model, where the service is deployed after that the adaptation phase. Finally, $\bar{\theta}^q_{kh}$ depends also on features $\bar{\Lambda}^q_{hw}$'s indicating, for example, the probability of failure of the service.

Let $G_{kq} : \Re^n \times \Re^{|NewS|} \rightarrow \Re$, with $1 \leq q \leq Q$, be the function that computes the $q$-th quality attribute of the $k$-th external service as a function of the same attribute of each existing/new service. And let us assume (without loss of generality) that, for each quality attribute, a threshold value $\Theta^q$ has been required as a lower bound.

Let *Cost* represent the cost function of the whole system depending on the costs of the existing/new services that we have represented as $\theta_i^0$ ($\bar{\theta}_i^0$). Different cost models could be used to define *Cost*, such as it could be as a function of the cost to replace the existing services, the one to introduce new services and the one for integrating services. For the sake of readability, we introduce here a formulation without correlation between $\theta_i^0$ ($\bar{\theta}_h^0$) and the software (hardware) observable parameters. In fact, for example, $\theta_i^0$ ($\bar{\theta}_h^0$) could be a function of the price charged for each invocation (measured in money per invocation) of the service $s_{ij}$ ($news_h$).

We can summarize the formulation of our optimization model as follows:

$$
\begin{aligned}
&minCost(\theta_1^0,\ldots,\theta_n^0,\bar{\theta}_1^0,\ldots,\bar{\theta}_{|NewS|}^0)\\
&\sum_{k\in K} pexec_k \cdot G_{kq}(\theta_{k1}^q,\ldots,\theta_{kn}^q,\bar{\theta}_{k1}^q,\ldots,\bar{\theta}_{k|NewS|}^q) \geq \Theta^q\\
&\forall q = 1\ldots Q\\
&\text{Other constraints(e.g., equations}\\
&\text{to predict } \theta_{ki}^q\text{'s and } \bar{\theta}_{kh}^q\text{'s)}
\end{aligned}
$$

**5. An instantiation of the model.** An example of optimization model instantiation can be found in our previous work [12], where an optimization model suggests how to change the (structural and behavioral) software architecture in order to minimize the maintenance cost while guaranteeing a required level of software reliability. In the following we discuss the cost and the reliability function used in [12] in order to show an example of application of the functions *Cost* and $G_{kq}$.

The function *Cost* to be minimized, as the sum of the costs of all the instances selected for the existing services and the ones for introducing new services is given by: $Cost = \sum_{i=1}^{n}\sum_{j=1}^{|Avail_i|} c_{ij}x_{ij} + \sum_{h=1}^{|NewS|} \bar{c_h}z_h$, where $\theta_i^0 = \sum_{j=1}^{|Avail_i|} c_{ij}x_{ij}$ and $c_{ij}$ is the cost of the $j$-th instance available for the service $i$; $\bar{\theta}_h^0 = \bar{c_h}z_h$ and $\bar{c_h}$ is the cost to adopt the $h$-th new service into the system. In [12] we provide suggestions for estimating the parameters $c_{ij}$ and $\bar{c_h}$.

Since we assume that for each external service provided by the system we dispose of a Sequence Diagram, the reliability of the $k$-th external service $G_{kq}$ can be expressed as follows:

$$G_{kq} = \prod_{i=1}^{n}\Big(\sum_{j=1}^{|Avail_i|} x_{ij}(1-\Lambda_{ij1}^q)^{\lambda_{i1}^q}\Big) \cdot \prod_{h=1}^{|NewS|}(1-\bar{\Lambda}_{h1}^q z_h)^{\lambda_{h1}^q}$$

$G_{kq}$ is function of the software parameter observable $\lambda_{i1}^q$ equals to the number of busy periods that the existing service $i$ shows in the Sequence Diagram $k$ after the adaptation phase (i.e., after the application of the set of adaptation plans), and $\lambda_{h1}^{\prime q}$ equals to the number of busy periods that the new service $h$ shows in the Sequence Diagram $k$ after the adaptation phase.
The parameter $\lambda_{i1}^q$ can be expressed as follows:

$$\lambda_{i1}^q = bp_{ki} + \sum_{r=1}^{m}\sum_{p\in AP_r} VBP_p(i,k)\cdot y_{rp}$$

where $m$ is the number of new requirements claimed by the user (alerts raised by the *Monitor* about the violation of non-functional constraints). $\lambda_{i1}^q$ depends on the added/canceled busy periods $VBP_p(i,k)$ [8] that the chosen adaptation plans for the adaptation requirements suggest for the service $i$ within the $k$-th external service. $bp_{ki}$ is the number of busy periods that the service $i$ shows in the Sequence Diagram $k$ before the adaptation phase.
The parameter $\lambda_{h1}^q$ can be expressed as follows:

$$\lambda_{h1}^q = \sum_{r=1}^{m}\sum_{p\in AP_r} BP_p(h,k)\cdot y_{rp}$$

$\lambda_{h1}^q$ depends on the number of busy periods $BP_p(h,k)$ that the chosen adaptation plans suggest for the $h$-th

[8] $VBP_p$ is a $n \times K$ matrix, where an element $VBP_p(i,k)$ represents the variation of the number of busy periods of the $i$-th existing service in the $k$-th external service. Note that if $p$ suggests to remove interactions then $VBP_p(i,k)$ is a negative number.

TABLE 5.1
*Parameters of the available instances for the existing services.*

| Service ID | Service alternatives | Cost $c_{ij}$ | Prob. of fail. on demand $\theta_{ij}$ | Num. of busy in scen. $\bar{k}$ $bp_{\bar{k}i}$ |
|---|---|---|---|---|
| $s_1$ | $s_{11}$ | 8 | 0.008 | 1 |
| | $s_{12}$ | 10 | 0.0009 | |
| | $s_{13}$ | 14 | 0.00001 | |
| $s_2$ | $s_{21}$ | 5 | 0.004 | 3 |
| | $s_{22}$ | 10 | 0.0002 | |
| $s_3$ | $s_{31}$ | 7 | 0.008 | 2 |
| | $s_{32}$ | 10 | 0.0004 | |
| $s_4$ | $s_{41}$ | 6 | 0.009 | 2 |
| | $s_{42}$ | 11 | 0.008 | |
| | $s_{43}$ | 13 | 0.0001 | |
| $s_5$ | $s_{51}$ | 7 | 0.008 | 2 |
| | $s_{52}$ | 10 | 0.0001 | |
| | $s_{53}$ | 12 | 0.000003 | |

TABLE 5.2
*Parameters of the new services available.*

| Service ID | Cost $\bar{c_i}$ | Prob. of fail. on demand $\bar{\theta_i}$ |
|---|---|---|
| $news_1$ | 7 | 0.00003 |
| $news_2$ | 5 | 0.000002 |
| $news_3$ | 4 | 0.00006 |
| $news_4$ | 7 | 0.005 |

new service within the $k$-th external service.
Finally, $G_{kq}$ depends also on the probability of failure on demand $\Lambda^q_{ij1}$ of the $j$-th instance available for the service $i$ and the probability of failure on demand $\bar{\Lambda}^q_{h1}$ of the new service $h$.

***An example of application of the model***

In order to show the practical usage of our optimization model we have applied the optimization model to the example considered in Section 4.1. We have solved the optimization model with respect to different configurations of the system for multiple values of the reliability bound $R$. For the sake of space we do not detail the results of all experiments that we have performed. We have observed several aspects that our model is able to capture and quantify, such as the choice to either keeping or replacing a service and how it helps to combine (and, in general, to reason about) the decisions to be taken for each requirement. A deeper experimentation of the application of an instantiation of the optimization model for a smartphone mobile application can be found in [12].

In the following we report a sample of configuration of the system for the example considered in Section 4.1.

Table 5.1 shows the parameter values of the available instances for the existing services, likewise Table 5.2 does for the new services that can be adopted into the system.

The second column of Table 5.1 lists, for each existing service, the set of alternatives. For each alternative: the buying cost $c_{ij}$ (in KiloEuros, KE) is given in the third column, the probability of failure on demand $\theta_{ij}$ is given in the fourth column, the number of busy periods $bp_{\bar{k}i}$ that the service $i$ shows in the scenario $\bar{k}$ is given in the fifth column.

The first column of Table 5.2 lists the new services available. For each new service: the buying cost $\bar{c_i}$ (in KiloEuros, KE) is given in the second column and the probability of failure on demand $\bar{\theta_i}$ is given in the third column.

Tables 5.3 and 5.4 summarize how the adaptation plans available for each requirement suggest to change the system behavior, i.e., to remove(introduce) interaction(s) between existing units and between these latter and new units, with respect to the external service $\bar{k}$.

The first column of Table 5.3 lists the existing services. For each service $i$: the variation of the number of busy periods $VBP_p(i, \bar{k})$ that the adaptation plans available for the first requirement (i.e., $ap_{11}$ and $ap_{12}$) suggest for the service $i$ in the scenario $\bar{k}$ is given in the third column; the variation of the number of busy periods $VBP_p(i, \bar{k})$ that the adapation plans available for the second requirement (i.e., $ap_{21}$, $ap_{22}$ and $ap_{23}$) suggest for the service $i$ in the scenario $\bar{k}$ is given in the fifth column.

The first column of Table 5.4 lists the name of the new services available. For each service $h$: the number of

TABLE 5.3
*Variation of the number of busy periods of the existing services with respect to the adapation plans.*

| Service ID | Adaptation Plan ID $ap_{1p}$ | Value for the scenario $\bar{k}$ $VBP_p(i,\bar{k})$ | Adaptation Plan ID $ap_{2p}$ | Value for the scenario $\bar{k}$ $VBP_p(i,\bar{k})$ |
|---|---|---|---|---|
| $s_1$ | $ap_{11}$ | 0 | $ap_{21}$ | 0 |
| | $ap_{12}$ | 0 | $ap_{22}$ | 0 |
| | | | $ap_{23}$ | 0 |
| $s_2$ | $ap_{11}$ | 0 | $ap_{21}$ | 0 |
| | $ap_{12}$ | 0 | $ap_{22}$ | 0 |
| | | | $ap_{23}$ | 0 |
| $s_3$ | $ap_{11}$ | 0 | $ap_{21}$ | 0 |
| | $ap_{12}$ | 0 | $ap_{22}$ | 0 |
| | | | $ap_{23}$ | 0 |
| $s_4$ | $ap_{11}$ | -1 | $ap_{21}$ | 0 |
| | $ap_{12}$ | 1 | $ap_{22}$ | 0 |
| | | | $ap_{23}$ | 0 |
| $s_5$ | $ap_{11}$ | -1 | $ap_{21}$ | 0 |
| | $ap_{12}$ | 0 | $ap_{22}$ | -1 |
| | | | $ap_{23}$ | 1 |

TABLE 5.4
*Number of busy periods of the new services with respect to the adaptation plans.*

| Service ID | Adaptation Plan ID $ap_{1p}$ | Value for the scen. $\bar{k}$ $BP_p(h,\bar{k})$ | Adaptation Plan ID $ap_{2p}$ | Value for the scen. $\bar{k}$ $BP_p(h,\bar{k})$ |
|---|---|---|---|---|
| $news_1$ | $ap_{11}$ | 0 | $ap_{21}$ | 0 |
| | $ap_{12}$ | 0 | $ap_{22}$ | 1 |
| | | | $ap_{23}$ | 0 |
| $news_2$ | $ap_{11}$ | 0 | $ap_{21}$ | 0 |
| | $ap_{12}$ | 1 | $ap_{22}$ | 0 |
| | | | $ap_{23}$ | 1 |
| $news_3$ | $ap_{11}$ | 0 | $ap_{21}$ | 0 |
| | $ap_{12}$ | 0 | $ap_{22}$ | 0 |
| | | | $ap_{23}$ | 0 |
| $news_4$ | $ap_{11}$ | 0 | $ap_{21}$ | 0 |
| | $ap_{12}$ | 0 | $ap_{22}$ | 0 |
| | | | $ap_{23}$ | 0 |

busy periods $BP_p(h,\bar{k})$ that the adaptation plans available for the first requirement (i.e., $ap_{11}$ and $ap_{12}$) suggest for the service $h$ in the scenario $\bar{k}$ is given in the third column; the number of busy periods $BP_p(h,\bar{k})$ that the adaptation plans available for the second requirement (i.e., $ap_{21}$, $ap_{22}$ and $ap_{23}$) suggest for the service $h$ in the scenario $\bar{k}$ is given in the fifth column.

We have solved the optimization model by varying the reliability bound $R$. The results highlight, in general, that the adaptation cost increases when higher reliability thresholds have to be guaranteed.

For example, if $R = 0.97$ the model provides the following solution [$s_{12}$, $s_{22}$, $s_{31}$, $s_{42}$, $s_{51}$] [$news_1$], [$ap_{11}$, $ap_{22}$]. This means that, in order to achieve the optimal cost of adaptation the following plans have to be adopted: the adaptation plan $ap_{11}$ for the first requirement and the adaptation plan $ap_{22}$ for the second requirement. In addition, all the replacements of existing services (i.e., $s_{ij}$) are specified, as well as the adoption of the new service $news_1$ is claimed. The adaptation cost is equal to 52 KE, whereas the system reliability is equal to 0.974698.

If we would increase $R = 0.98$ the model provides the following solution [$s_{12}$, $s_{21}$, $s_{32}$, $s_{43}$, $s_{52}$] [$news_2$], [$ap_{12}$, $ap_{21}$]. In order to satisfy the reliability constraint for R = 0.98 a different solution, which is also more expensive, is suggested. In fact, the adaptation cost is equal to 53 KE, whereas the system reliability is equal to 0.985874. The two solutions differ also for the instances selected for the existing services and the adaptation plans suggested for the requirements. If $R = 0.97$ the model suggests to include $news_1$, whereas if $R = 0.98$ the new service $news_2$. The introduction of either $news_1$ or $news_2$, following the plans suggested by the solutions, would involve different modifications on the structure and the behavior of the system. For example, by applying the solution for $R = 0.97$ the existing service $s_5$ would not be used any more after the adaptation phase, whereas by applying the solution for $R = 0.98$ the service $s_5$ would be replaced by its second available instance (i.e., $s_{52}$).

Figures 5.1 and 5.2, respectively, show how the static and dynamic structure of the system with respect to the external service $\bar{k}$, respectively, change after the application of the adaptation actions suggested by the solution for $R = 0.97$ and $R = 0.98$. In the figures we have marked as bold the modifications brought after the

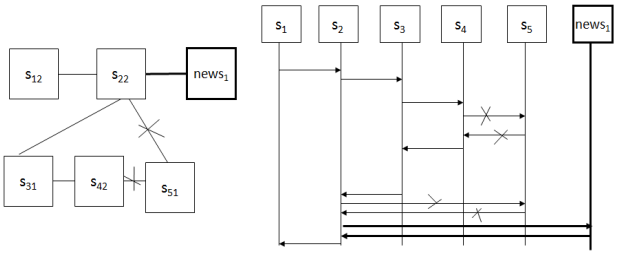



FIG. 5.1. *Resulting static structure and a SD of the system after the application of* $ap_{11}$, $ap_{22}$

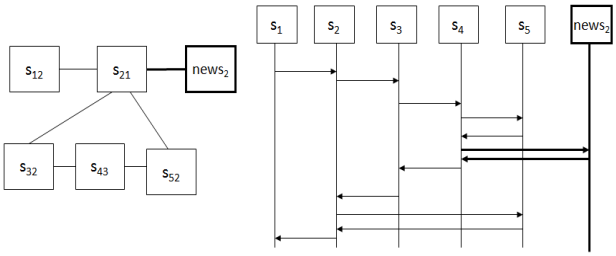



FIG. 5.2. *Resulting static structure and a SD of the system after the application of* $ap_{12}$, $ap_{21}$

application of the plans and we have put a cross on the interactions (i.e., messages in the Sequence Diagram and dependencies between services) that are removed.

Finally, if $R = 0.99$ the model provides the following solution [$s_{13}$, $s_{22}$, $s_{32}$, $s_{43}$, $s_{51}$] [$news_1$, $news_2$], [$ap_{12}$, $ap_{22}$]. The adaptation cost is equal to 66 KE, whereas the system reliability is equal to 0.990273. It is necessary to increase the cost from 53 KE to 66 KE in order to satisfy the reliability constraint. Note that in this case the model suggest to add both the new service $news_1$ and the new service $news_2$, whereas for $R = 0.97$ and $R = 0.98$, $news_1$ and $news_2$, respectively. The introduction of both $news_1$ or $news_2$, following the plans suggested by the solutions, would involve different modifications to the structure and behavior of the system. Figure 4.1 shows how the static and dynamic structure of the system with respect to the external service $\bar{k}$ change after the application of the adaptation actions suggested by such a solution.

Note that one of two requirement $req_r$ claimed by the user could claim to modify the system in order to modify (introduce) one or more non-functional requirements at the same time, e.g., to improve the response time and the availability of the system. Therefore the model can be used to evaluate the tradeoff among the system reliability and other non-functional requirements (e.g. response time and availability).

**6. Conclusion.** We have presented a framework, based on an optimization model, that dynamically adapts both the software and hardware features of a service based system while minimizing the adaptation costs and guaranteeing a required level of the system qualities. Adaptation actions can be triggered both by a user request and/or automatically after the runtime violation of system quality constraints, or the appearing/disappearing of services into the environment. In addition, we have provided an example of instantiation together with a numerical experimentation. Experiments show that the use of a the proposed method is very helpful to make decisions during the activity of system adaptation.

We are investigating several future directions, for example, we intend to specialize our framework by enhancing it for guaranteeing specific properties of a service application domain, such as the context-aware property of the pervasive systems. In particular, we are implementing a prototype to apply our approach on realistic examples. We want to instantiate the optimization model by considering different system qualities (e.g. software reliability, availability, and response time) by enhancing our reliability model. In this direction, we also intend to take into account dependencies between different quality attributes and dependencies between services. To this extent, additional constraints may be needed, which can be expressed as *contingent decisions* [23] and, for example, the error propagation property [3] for expressing dependencies between failures of services interacting between each other could be embedded into the reliability model.

Since the adaptation of a single service may impact on the other services the propagation of the changes along the service composition should be analyzed. We also intend to leverage on several methods that have been introduced for estimating the propagation of the uncertainty of the model input parameters on the quality

system (e.g. [40])

In order to solve possible problems due, for example, to the model solution too large computation time we intend to investigate the use of the meta-heuristic techniques (e.g. the tabu-search algorithm) to improve the overall model complexity and scalability combined with the simulation techniques [16].

Other interesting research directions we intend to investigate concern the introduction/evaluation of risk factors associated to user adaptation requirements and the evaluation of dependencies among different requirements. Moreover, we intend to introduce dependencies between adaptation plans, by leveraging, for example, the contingent decisions.

## REFERENCES


[1] [Online]. Available: www.lindo.com.

[2] Itu-T Recommendation Z.120 Message Sequence Charts (msc99). In *Technical report, ITU Telecommunication Standardization Sector*, 1996.

[3] W. Abdelmoez, D. M. Nassar, M. Shereshevsky, N. Gradetsky, R. Gunnalan, H. H. Ammar, B. Yu, and A. Mili. Error Propagation In Software Architectures. *Software Metrics, IEEE International Symposium on*, pages 384–393, 2004.

[4] F. André, E. Daubert, and G. Gauvrit. Towards a Generic Context-Aware Framework for Self-Adaptation of Service-Oriented Architectures. In *Proc. of the Fifth International Conference on Internet and Web Applications and Services*, 2010.

[5] J. Bang, D. Popescu, G. Edwards, N. Medvidovic, N. Kulkarni, G. Rama, and S. Padmanabhuni. Codesign: a highly extensible collaborative software modeling framework. In *ICSE '10: Proc. of the 32nd ACM/IEEE International Conference on Software Engineering*, pages 243–246, 2010.

[6] L. Baresi and S. Guinea. A dynamic and reactive approach to the supervision of bpel processes. In *ISEC '08: Proc. of the 1st conference on India software engineering conference*, pages 39–48. ACM, 2008.

[7] C. Blum and A. Roli. Metaheuristics in combinatorial optimization: Overview and conceptual comparison. *ACM Comput. Surv.*, 35(3):268–308, 2003.

[8] R. Calinescu and M. Kwiatkowska. Using quantitative analysis to implement autonomic IT systems. In *ICSE*, pages 100–110, 2009.

[9] G. Canfora, M. D. Penta, R. Esposito, and M. Villani. A framework for QoS-aware binding and re-binding of composite web services. *Journal of Systems and Software*, 81(10):1754–1769, 2008.

[10] V. Cardellini, E. Casalicchio, V. Grassi, F. L. Presti, and R. Mirandola. Towards Self-adaptation for Dependable Service-Oriented Systems. In *Architecting Dependable Systems VI*, pages 24–48, 2009.

[11] D. Chiu, S. Deshpande, G. Agrawal, and R. Li. A Dynamic Approach toward QoS-Aware Service Workflow Composition. In *ICWS*, pages 655–662, 2009.

[12] V. Cortellessa, R. Mirandola, and P. Potena. Selecting Optimal Maintenance Plans based on Cost/Reliability Tradeoffs for Software Subject to Structural and Behavioral Changes. In *Proc. of the 14th European Conference on Software Maintenance and Reengineering (CSMR 2010)*.

[13] E. Di Nitto, C. Ghezzi, A. Metzger, M. Papazoglou, and K. Pohl. A journey to highly dynamic, self-adaptive service-based applications. *Autom. Softw. Eng.*, 15(3-4):313–341, 2008.

[14] J. Fox and S. Clarke. Exploring approaches to dynamic adaptation. In *MAI '09: Proceedings of the 3rd International DiscCoTec Workshop on Middleware-Application Interaction*, pages 19–24, 2009.

[15] J. Garcia, D. Popescu, G. Edwards, and N. Medvidovic. Toward a Catalogue of Architectural Bad Smells. In *QoSA*, pages 146–162, 2009.

[16] F. Glover, J. Kelly, and M. Laguna. New advances for wedding optimization and simulation. volume 1, pages 255–260, 1999.

[17] H. Guo, J. Huai, H. Li, T. Deng, Y. Li, and Z. Du. ANGEL: Optimal Configuration for High Available Service Composition. *Web Services, IEEE International Conference on*, pages 280–287, 2007.

[18] J. Harney and P. Doshi. Adaptive web processes using value of changed information. In *In International Conference on Service-Oriented Computing (ICSOC)*, pages 179–190, 2006.

[19] Q. He, J. Yan, H. Jin, and Y. Yang. Adaptation of Web Service Composition Based on Workflow Patterns. In *ICSOC*, 2008.

[20] R. Hirschfeld, P. Costanza, and O. Nierstrasz. Context-oriented Programming. *Journal of Object Technology*, 7(3):125–151, 2008.

[21] N. Ibrahim and F. L. Mouël. A Survey on Service Composition Middleware in Pervasive Environments. *CoRR*, abs/0909.2183, 2009.

[22] N. Ibrahim, F. L. Mouël, and S. Frénot. MySIM: a spontaneous service integration middleware for pervasive environments. In *ICPS '09: Proceedings of the 2009 international conference on Pervasive services*, pages 1–10, 2009.

[23] H.-W. Jung and B. Choi. Optimization models for quality and cost of modular software systems. *European Journal of Operational Research*, 112(3):613 – 619, 1999.

[24] G. Kapitsaki, G. Prezerakos, N. Tselikas, and I. Venieris. Context-aware service engineering: A survey. *Journal of Systems and Software*, 82(8):1285–1297, 2009.

[25] S. Kim, D. Kim, L. Lu, and S. Park. Quality-driven architecture development using architectural tactics. *Journal of Systems and Software*, 82(8):1211–1231, 2009.

[26] P. Leitner, F. Rosenberg, and S. Dustdar. Daios: Efficient Dynamic Web Service Invocation. *IEEE Internet Computing*, 13:72–80, 2009.

[27] G. Lodi, F. Panzieri, D. Rossi, and E. Turrini. SLA-Driven Clustering of QoS-Aware Application Servers. *IEEE Trans. Software Eng.*, 33(3):186–197, 2007.
[28] A. Martens and H. Koziolek. Performance-oriented Design Space Exploration. In *Proc. of the 13th Int. Workshop on Component Oriented Programming (WCOP'08)*, 2008.
[29] M. Marzolla and R. Mirandola. Performance Prediction of Web Service Workflows. In *Proc. of third International Conference on Quality of Software Architectures, QoSA 2007*, volume 4880 of *Lecture Notes in Computer Science*, pages 127–144, 2007.
[30] R. Mirandola and P. Potena. Self-adaptation of service based systems based on cost/quality attributes tradeoffs. *Symbolic and Numeric Algorithms for Scientific Computing, International Symposium on*, pages 493–501, 2010.
[31] O. Moser, F. Rosenberg, and S. Dustdar. Non-intrusive monitoring and service adaptation for WS-BPEL. In *WWW*, pages 815–824, 2008.
[32] J. Musa. Operational profiles in software-reliability engineering. *Software, IEEE*, 10(2):14–32, Mar 1993.
[33] E. Nistor, J. Erenkrantz, S. Hendrickson, and A. van der Hoek. ArchEvol: versioning architectural-implementation relationships. In *SCM '05: Proceedings of the 12th international workshop on Software configuration management*, pages 99–111, 2005.
[34] M. Oh, J. Baik, S. Kang, and H. Choi. An Efficient Approach for QoS-Aware Service Selection Based on a Tree-Based Algorithm. In *ACIS-ICIS*, pages 605–610, 2008.
[35] M. Papazoglou, P. Traverso, S. Dustdar, and F. Leymann. Service-Oriented Computing: State of the Art and Research Challenges. *IEEE Computer*, 40(11):38–45, 2007.
[36] J. Rao and X. Su. A Survey of Automated Web Service Composition Methods. In *SWSWPC*, pages 43–54, 2004.
[37] F. Rosenberg, P. Celikovic, A. Michlmayr, P. Leitner, and S. Dustdar. An End-to-End Approach for QoS-Aware Service Composition. In *EDOC*, pages 151–160, 2009.
[38] R. Roshandel, N. Medvidovic, and L. Golubchik. A Bayesian Model for Predicting Reliability of Software Systems at the Architectural Level. In *QoSA*, pages 108–126, 2007.
[39] H. Truong and S. Dustdar. A survey on context-aware web service systems. *International Journal of Web Information Systems (IJWIS)*, 5(1):5–31, 2009.
[40] N. Wattanapongskorna and D. Coit. Fault-tolerant embedded system design and optimization considering reliability estimation uncertainty. *Reliability Engineering and System Safety*, 92:395–407, 2007.
[41] J. Zhu, C. Guo, Q. Yin, J. Bo, and Q. Wu. A Runtime-Monitoring-Based Dependable Software Construction Method. In *ICYCS*, pages 1093–1100, 2008.

*Edited by:* Dana Petcu and Alex Galis
*Received:* March 1, 2011
*Accepted:* March 31, 2011

Scalable Computing: Practice and Experience
Volume 12, Number 1, pp. 79–92. http://www.scpe.org
ISSN 1895-1767


# FLEXIBLE ORGANIZATION IN THE *ORCFS* RELATIONAL FILE SYSTEM FOR EFFICIENT FILE SEARCHING

ADRIAN COLEŞA*, ALEXANDRA COLDEA†, AND IOSIF IGNAT‡

**Abstract.** The need for efficient organization of files grows with the computer storage capabilities. However, a classical hierarchical file system offers little help in this matter, excepting maybe the case of links and shortcuts. OrcFS proposes a solution to this problem. By redefining several file system concepts, it allows the user to set custom metadata, in the form of "$(property, value)$" pairs, to express relationships between files. Using it, the system automatically creates a classified view of its components, in which those having similar characteristics are grouped together. A virtual hierarchy is generated, which provides multiple access paths to the same file. This assures that the data can be classified in a more flexible way and the navigation can be done more intuitively. The traditional concept of directory is extended, to accommodate the user-defined properties. In OrcFS, both classical navigation and query interrogation are possible. The enhanced system is compliant with the Linux's VFS interface, thus no changes need to be made to existing applications and they may be used in OrcFS. A prototype of the project was implemented in user-space using the FUSE library to reimplement system calls. The performed tests proved that even if introducing new data in the OrcFS implies some overhead, this is negligible compared with the gain obtained when searching for files in an immense file tree structure.

**Key words:** file-system; file relationships; metadata; fast retrieval; compatibility

**1. Introduction.** In the last years notable enhancements have been obtained in the field of data storage. The most modern computers are a collection of impressive numbers. The disk capacity increased up to the order of terabytes, the data can now be stored more efficiently and retrieved faster. However, these improvements at a low-level bring along with them the necessity of a higher-level data organization revolution.

The classical operating system offers only basic help for data organization at user level. Moreover, there is almost no flexibility in expressing relationships between files.

A relational file system′s objective is to find a manner to overcome the fore mentioned limitations. Due to the extensive research in this domain several directions have been outlined. All projects associate semantic information with files [11] in order to describe their contents and interrelationships. Some solutions, like [15, 12, 3] store this metadata in databases, while others [10] use the already existent i-node and directory structures. While in most implementations a file is presented to the user as an atomic element, there are some [14, 12, 3] which split this concept and work with different parts of the file. The latter extract from file contents different pieces of information, interpret and manage them to provide the user the possibility to see and interact with the stored data at a more abstract and complex level. Regarding the user capabilities, the projects can be classified in two categories: some, [11, 3], simply index the metadata extracted from different file formats while others [15, 12, 10, 9, 8, 7] allow the definition of custom properties.

We propose a system compatible with the already existent applications, that provides a classification in which files and file containers are treated similarly. From the point of view of the direction taken, our project belongs to the path that treats the file as an indivisible item. The focus is not on analyzing the file content, but on extending the organization and search methods. In this respect, the project does not head towards becoming a data storage enhancer, like [12], but towards a relational file system that introduces new file container concepts in order to obtain the desired flexibility. The two concepts are not mutually exclusive, the proof of which is [15], and thus the analysis of the file content can be seen as a possible extension of OrcFS.

Our system allows the user to define metadata, in order to describe files and file containers on two levels. Firstly, properties specify the criteria by which the files in the current location will be classified. Next, each file may or may not assign values to these criteria. Using this information the system will dynamically create virtual directories to reflect the classification. Moving further, the user will be able do define his own virtual containers, which are the result of boolean queries on properties and values. Because this implementation needs flexibility in manipulating relationships, we will be using a relational database to store the metadata, similar to [15].

An important characteristic of our system is that it runs over the classical file system, respecting the VFS [5] interface and maintaining an apparent hierarchical view.

*Computer Science Department, Technical University of Cluj-Napoca, Romania (adrian.colesa@cs.utcluj.ro).
†Computer Science Department, Technical University of Cluj-Napoca, Romania (alexandra.coldea@student.utcluj.ro).
‡Computer Science Department, Technical University of Cluj-Napoca, Romania (iosif.ignat@cs.utcluj.ro).

The following sections describe in greater details the implementation and testing of the system. As such, Section 2 and Section 3 present how several concepts of the classical file system have been redefined in order to serve the needs of OrcFS. Section 4 analyzes the project architecture and individual modules. Section 5 presents an example of how the system can be used. Section 6 contains a description of the tests performed and an examination of their outcomes. Section 7 summarizes the approach taken by other projects in this area. Section 8 contains several conclusions.

**2. Redefinition of File System Concepts.** The integration of the proposed relational model in the classical file system is done by redefining the old concepts and defining new ones. Due to the fact that the most affected is the logical organization layer, the main concept which was extended is that of a file container, or directory. The following sections describe how each concept has been altered.

**2.1. Category.** Categories are the only physical containers in the file system. Every other type of container presented from here on is virtual. They are the true file holders in the sense that every file must belong to a category. The notion remains from the hierarchical organization and its functionality is similar to that of a directory. Therefore, a category and a directory both define a file container, specified by the user by certain logical criteria. In the case of the traditional directory these logical criteria are the name and, indirectly, the hierarchy created. However, for the category, the classification criteria will be more detailed. The next sections will explain these criteria.

Similarly to directories, each category may contain several subcategories. However, the difference between subcategories is not only based on their names, like for subdirectories, but also on specific properties each item possesses.

Some examples are the category of books, papers, projects.

The notion of absolute and relative path maintain their meaning from the classical operating systems. However, the difference is that a file is no longer uniquely identified by its absolute path, but by a combination between it and properties. The following section will detail on this change.

**2.2. Property and Value.** In order to obtain more flexibility in organizing the categories, properties (tags, attributes) are associated to each of them. Every subcategory or file has the properties of the category it belongs to. Therefore, inside a category, the significance of properties is classification criteria. In order to classify a file by a certain criteria the user must assign a value to the corresponding property. Hence, the elements of a category are described using pairs "$(property, value)$". An item may have several pairs "$(property, value)$" associated with it and it may associate several values to the same property.

Thus, a property is associated to a category, while a value is associated to a file or a subcategory. Each category contains several properties and each of its files or subcategories, may or may not assign one or more values to any property associated with that category. In this organization a file is uniquely identified by its absolute path — indicated by the sequence of categories from root to the file location concatenated with file name— and a list of "$(property, value)$" pairs.

Let's assume the example of the *Articles* category. Some properties (classification criteria) which may be defined are: *author*, *year*, *keyWord*. Different files in this category may assign different values to the same property — they may have different authors. At the same time, the same file may assign different values to the same property — for example an article may be associated with several key words. The user may search through the files of a category using different classification criteria. Thus, he is able to search for all the articles written by a certain author, or all the articles written by several authors. Moreover he may ask for all the files "*having the authors X and Y and the year Z*". This leads to a very flexible and intuitive way of navigation through the file system.

Both properties and values are seen by the user as directories. However, they are only virtual containers because their content is not stored on disk, it is generated. In order to improve the performance and not be compelled to create every time the content of these types of directories, indexing will be used.

The difference between these types of containers and a category is that in the latter the user groups together files that, according to him, have a logical connection and a common set of properties. On the other hand, in property and value containers, the system groups together files that have the same subset of metadata fields.

**2.3. The $Classification$ Directory.** In order to link the categories with the property and value directories, the $Classification$ directory was introduced. It represents an isolation of the classified view from the

non-classified view. This distinction is introduced in order to make the system and part of its facilities accessible through the interface of existing applications.

Each category contains a virtual directory called *Classification* which consists of the files in the current location in a classified hierarchy. At the first level, this directory contains all properties associated with the current category.

At the next level, each property will contain directories corresponding to the values each of them may take. A value subdirectory contains all files with which it is associated but also, another *Classification* directory. The latter contains a list of all the properties of the category that are not in the path of the current location (all properties not visited yet). Therefore, this organization treats the navigation through the classified view as a query composed of property and value constraints imposed on the files of the current category. By going deeper in the tree, one permanently refines the search, by adding more constraints to the query.

With this organization it may seem that the same file is present in several places. In fact, what this implementation of a relational file system does is provide several virtual paths to the same file.

Take for example a category *Articles* which contains the properties *author* and *year*. The file *File1* gives the values "*Author1*" and "*Author2*" to the property *author* and "*Year1*" to the property *year*. The generated *Classification* directory contains two virtual directories *author* and *year*, corresponding to the two properties. The directory *author* contains two subdirectories called *Author1* and *Author2*, corresponding to the two values the property author may take. The directory *year* contains one subdirectory called *Year1*. The directory *Author1* contains the file called *File1* and another *Classification* subdirectory. The latter contains the subdirectory *year*, corresponding to the property on which no filtering has been done yet on this path. In this way, *File1* can be found by taking any of the following paths:

- `/Articles/File1`
- `/Articles/Classification/Author/Author1/File1`
- `/Articles/Classification/Author/Author1/Classification/Year/Year1/File1`
- `/Articles/Classification/Author/Author2/File1`
- `/Articles/Classification/Author/Author2/Classification/Year/Year1/File1`
- `/Articles/Classification/Year/Year1/File1`
- `/Articles/Classification/Year/Year1/Classification/Author/Author1/File1`
- `/Articles/Classification/Year/Year1/Classification/Author/Author2/File1`

The structure that is created is very useful for retrieving files on systems with large amount of data and on which multiple users operate. Because this entire structure is generated, it does not take up additional disk space.

**2.4. Query.** The notion of query represents a formula used for locating files that satisfy certain conditions.

A query places constraints on the files in the current location, and only those items which satisfy the restrictions will be part of the result. A constraint has the form: '$p1 = v1$' and it is translated in natural language as: "*Find all files for which property* p1 *has value* v1". The individual constraints are connected using boolean operators: & (and), | (or), ! (not).

In this way, the file system can be queried like a relational database. A concrete example of such a query is: "$(p1 = v1)\&(p2 = v2)$", which will return all the items which give value 'v1' to property 'p1' and value 'v2' to property 'p2'. This result can also be obtained by classical navigation and an example of such a location is `Classification/p1/v1/Classification/p2/v2`. However, the result of the query "$(p1 = v1)|(p2 = v2)$" may not be obtained by navigating through the file system because it corresponds to the reunion of two directories (`Classification/p1/v1` and `Classification/p2/v2`). The query "$(p1 = v1)!(p2 = v2)$" can be translated in natural language as "*Find all files which give value* v1 *to property* p1 *but which do not give value* v2 *to property* p2". This is another example of query which may not be simulated by classical navigation.

The only difference between a query directory and a value directory is that the former represents a more complex way of expressing constraints. In fact, a query directory must be seen and manipulated like any location on the disk. This is why queries do not have to be implemented as new system calls, but can be integrated in the operating system by using the existing ones and changing their in-kernel implementation. Therefore, the `chdir` system call was used (its correspondent in shell terms is the `cd` command) and did not interpret anymore its parameter as a simple path, but as a query.

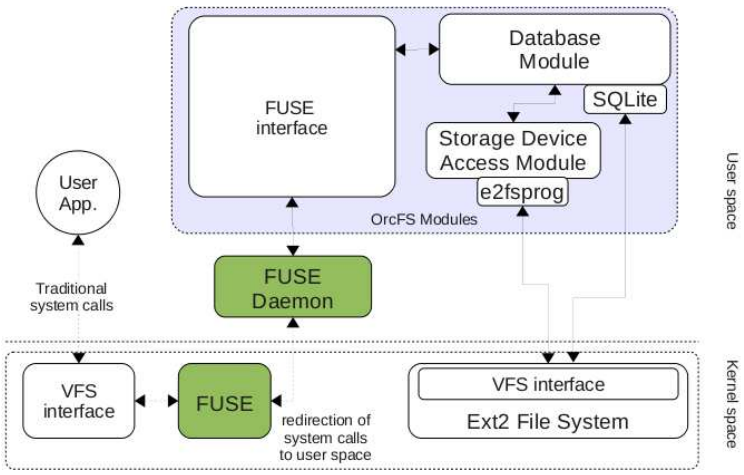



FIG. 3.1. *OrcFS user-space architecture*

Therefore, one will be able to write "`cd p1=v1`" in a terminal and the system will display all files for which this condition holds. Going further with the idea of increasing the navigability through the file system, the result will also contain a $Classification$ directory which will provide further options for search refinement.

The way our system integrates the queries into the operating system provides compatibility with the exiting applications, allowing them to use our system as a normal one, but providing many different paths to the same file.

**2.5. File.** The concept of file has not changed relative to its meaning from the classical file systems. All the changes made are at a higher level, thus maintaining the file implementation unaltered. However, what has changed is how files are being manipulated. A user is now able to associate with them properties and values which convey meaning. This way, it is possible to treat files like entities of a database and use queries to group them into virtual directories. The same are treated subcategories. From now on, the term file system element will represent either a file or a subcategory.

**3. System Architecture.** The system is divided into three modules that communicate with each other in order to perform an action on the file system, required by the user or an application.

At the highest level, there is the VFS interface module. This receives requests from the user space, delegates them and then returns the result. At the lower level it is placed the module for storage device access. It can use an exiting file system storage strategy like Ext2 for example, provided in Linux kernel by the *e2fsprogs* library [1]. The metadata management module connects the other two modules. It manages the properties, values and relationships between them. It is implemented using an in-kernel database engine like in [13]. We used for that the *SqLite* [4] library. We chose SqLite because it is fast and not based on the client-server model and small enough to be easily integrated in kernel. The metadata management module stores and loads the database using the storage access module and answers metadata requests received from the VFS interface. All the commands that do not involve metadata operations are forwarded directly to the storage interface, without interference from the database module.

In a final implementation, all three modules will have been placed in the operating system's kernel. However, in order to ease the development process, a testing prototype using the FUSE [2] library was used. The latter is an application that runs both in the kernel — as a module which implements the VFS interface — and in the user space — as a daemon. It allows a developer to rewrite system calls in the user space and test them without recompiling of the kernel.

Figure 3.1 illustrates the way the system is placed in user space and how different parts of it interact with each other.

A user application issues a request which must me resolved by the VFS. The FUSE in-kernel module captures the request and generates a FUSE event. The FUSE daemon in the user space listens for such events and when one occurs, it delegates it to the OrcFS implementation. This application contains a database module (i.e. the metadata management module) and a storage device access module, both of which communicate with

the underlying file system using specific libraries.

**4. Transparent Integration of OrcFS in an OS.** Our file system can be integrated in an OS completely transparent for the existent applications. This is because we do not change the traditional file-system interface adding new system calls, but just extend the functionality of the system calls already belonging to it. This way, an existing application can access our file system with no modification of its code. Nevertheless, if the application is aware of the fact that our file system provides extended functionality, it can take advantage of them. We see how this can be done in the following subsection.

**4.1. Backward compatibility: Browsing a Category Using a Traditional File-System Explorer.** Assume the file system contains a category named *Books*, which we want to browse using a file-system explorer. Table 4.1 describes its contents in terms of "$(property, value)$" pairs. There are six files, one subcategory and two properties, i.e. *Author* and *Year*. Each file or subcategory is given zero or one values to each category's property. Based on these associations, the category's contents will be classified and viewed in different ways. The multiple ways a category's element can be found can be easily seen in Figure 4.1, which illustrates the file tree generated for the *Book* category. Still, there could be files given no value to any property. We call such files unclassified files. Consequently, they belong only to the unclassified view of the analyzed category and can be found in just a single way and place. We also note that files and subcategories are treated similarly.

| | Type | Property *Author* | Property $Year$ |
|---|---|---|---|
| Book1 | file | Author1 | 2009 |
| Book2 | file | Author1 | 2010 |
| Book3 | file | Author1 | 2011 |
| Book4 | file | Author2 | 2009 |
| Book5 | file | Author2 | 2009 |
| Book6 | file | Author2 | |
| Book7 | file | | |
| Books2009 | category | | 2009 |

TABLE 4.1
*Books category description*

Let us start the navigation through the *Book* category from its root directory. It corresponds to the *unclassified area*. This is the place where all category's elements are visible like in any other traditional directory. Yet, we can also find here the special virtual directory $Classification$, which gives us access to the *classified area*. Entering the *Classification* directory, we can find two subdirectories *Author* and $Year$, corresponding to the two properties of the category. They are also virtual directories, while they do not correspond to physical containers. Their contents is generated when the user explicitly asks it or, like in our example, he accesses them. The $Year$ subdirectory gives the user the possibility to see all its books classified based on the year when they were published. The *Author* subdirectory is where books are classified based on their author. The two directories' contents consists only in subdirectories, each one corresponding to a different value given to those properties by different category's elements. Entering one of it, the user can find category elements having associated the corresponding "$(property, value)$" pair. For example, following the path `Books/Classification/Author/Author1`, the user can find files *Book1*, *Book2* and *Book3*. Besides the elements belonging to a value subdirectory, our system also generates a new $Classification$ subdirectory, to be used to further classify the current directory contents. This lower level $Classification$ subdirectory contains as subdirectories only the properties not considered on the path to it. Thus, we can find only the $Year$ subdirectory in the `Books/Classification/Author/Author1/ Classification` directory. The $Year$ subdirectory contains subdirectories corresponding to its possible values. Entering, for example, the 2009 subdirectory, the user can find again the *Book1* file. This will be in fact the only file found here, because it is the only one having associated both "$(Author, Author1)$" and "$(Year, 2009)$" pairs. The other files can be found on different paths, while they give different values to the $Year$ property.

The $Classification$ subdirectory also belonging to the `Books/Classification/Author/Author1/Classification/Year/2009/` is empty, because there is no other property not yet considered on that path, i.e. there is no further classification criterion.

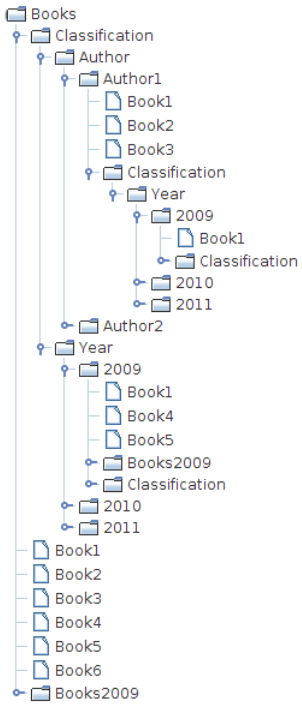



FIG. 4.1. *Generated structure of the Books category*

A similar way of finding the *Book1* file could be the one starting with the other property in the first *Classification* directory, which is `Books/Classification/Year/ 2009/Classification/Author/Author1/`. This is because the queries corresponding to the two different paths, i.e. "*Which are all the books written by* Author1 *in* 2009?" and "*Which are all the books written in* 2009 *by* Author1?", are logically equivalent and must return the same answer. The logical equivalence of the two queries results from the fact that the two conditions they include are linked by a logical `AND` operator, which is commutative. This gives the user the possibility to find the same set of files following different paths in the file tree. Although, we must note that these navigating alternatives correspond to queries containing only the `AND` operator. Still, the real queries can also be expressed using some other different logical operators, like `OR` and `NOT`, and actually there are files corresponding to such queries. For instance, the answer of the "*Which are the books written by* Author1 *in* 2009 *or* 2010?" query, would contain both *Book1* and *Book2* files. As we can easily observe, this file set cannot be located in any directory in the file tree illustrated in Figure 4.1. We eliminated this limitation, giving the user applications the possibility to directly jump to virtual directories containing the answers of questions formed by different logical operators, directories which do not correspond to any node in the file tree, still being located on a sort of a file path. This remaining limitation is that such directories cannot be touched navigating down in the file tree level by level. The way we did this is described in the following subsections.

**4.2. Reuse and reinterpret the existent file-system system calls.** We have reimplemented the system calls related to directory access, but left unchanged the ones providing access to files' contents. This is because we gave a new interpretation of the directory concept — related to data organization, but kept that of the file concept — related to storing of data. Still, the traditional effect of the system calls we reinterpreted is changed only in the classification area, not in the unclassified area. Thus, if a file is removed from the root directory of a category, it is completely removed from that category, different from removing the same file from one of the subdirectories in classification area, which has another effect. Let us see exactly how each change on a directory is interpreted.

*Creating a file* in the classification has also the additional effect of classifying the new file based on the "$(property, value)$" pairs included in the path specified at file creation. For example, creating a new file *Book7* in the directory `Books/Classification/ Author/Author1/Year/2011` will create the file in the *Books* category, but also associate to it the "$(Author, Author2)$" and "$(Year, 2011)$" pairs. We must note that the same effect is

obtained using the similar path `Books/Classification/Year/2011/ Author/Author1`. If a file with the same name already exists in the category, the file contents is truncated and its metadata is replaced with the one specified in the creation path. If the file cannot be truncated, because of permission rights restrictions, the user is reported an error. This can be confusing if the existent file is not visible in the path specified at file creation. In order to avoid such situations, it is recommended to create new files only in the root directory of a category, where all its files are visible. After that, the file can be associated explicitly "$(property, value)$" pairs, by performing other operations on it, interpreted as file classification operations.

*Removing a file* from a directory located in the classification area will not result in the physical removal of that file, but just in removing some of its associated metadata. More precisely, the "$(property, value)$" pairs included in the path of the directory the file is removed from will be the one removed. For example, removing `Books/Classification/Author/Author1/Book3` results in removing the "$(Author, Author1)$" pair from those associated to file $Book3$. A file remains in its category until it is removed from the root directory of that category.

*Creating a directory* in the unclassified area just creates a new element, i.e. subcategory, in the corresponding category. The result of creating a new subdirectory in the classification area depends on the exact directory (i.e. parent directory) the operation is performed. There exist the following cases:

- If the parent directory is one of the $Classification$ directories, then the new directory is interpreted as a new property of the category. For example, creating the directory `Books/Classification/Publisher/` creates a new property, named $Publisher$, in category $Books$.
- If the parent directory is property directory, then the new directory is taken as a new value of that property. For example, creating the directory `Books/ Classification/Year/2003` creates the new value 2003 for the existent property $Year$. Obviously, there will be at that moment no file having associated the "$(Year, 2003)$" pair, but this can be done later, using other file operations.
- If the parent directory is a value directory, the new directory is an element of the category, classified based on the properties and values in the path used on its creation.

We must note that a problem similar with that from file creation can also occur in the case of directory creation: using directories names already existent in a category. We already described the reason they are not visible on any level of the category file tree. So, in order to avoid confusion, it is recommended for the user to create properties or values on the highest levels of the category. Thus, new properties should be created in the top level $Classification$ directory, where all category's properties are visible.

*Removing a directory* is very similar with the directory creation operation, in the way its parameters are interpreted. There are more cases:

- If the removed directory is a $Classification$ one, an error is reported, since it does not correspond to any data in the file system, but it is used just to provide specialized access to files.
- If the removed directory is a property one, then the files that could be found on the subtree having as root that directory are removed their associations with the removed property. Although, the property itself is not removed unless it is removed from the $Classification$ directory on the top level.
- Removing a value directory is similar with removing its corresponding property directory, but its effect is reduced only to that value, not to all the values of that property, like in the previous case.

*Creating a hard link* to a file in the same category classifies that file, i.e associates to it new $(property, value)$ pairs extracted from the path of the new hard link. There are two restrictions when a new hard link is created: the same name must be used for the new link and it cannot be created in a property directory, where only value directories are allowed.

*Creating a symbolic link* is identical to creating a new file, since a symbolic link is actually a new file, different by the referred one.

Another system call we reinterpreted is *chdir*. It is not really a file operation. Yet, we used it in order to give the user access to sets of files that cannot necessarily be found just navigating in a category's file tree. It also allows the user "jumps" in physically non-locatable directories, corresponding to queries including logical operators like "OR" and "NOT".

**4.3. The Query Language.** We have seen that both the navigation through the file tree going down level by level and the direct "jump" to a virtual directory are made using the *chdir* system call. We interpreted the path specified for it as a request (query) addressed to our system to generate the contents of a directory, being it an immediate subdirectory in the file tree, relative to the current directory, or even one having no corresponding

node in the file tree. The difference between the two cases is that in the former, the query contains only the "AND" logical operator, while in the latter, any logical operator could be used. We illustrate in the following examples, executed in the *Books* directory, the possible forms of such queries.

Navigation through the category's file tree can be done in the classical way, by moving up or down the tree, searching for some files. Moving down a new level means refining the result of the previous query, by classifying the files in the current directory by the remaining properties. This is very useful in cases the user does not remember from the beginning all (or many of) the properties and values associated to a file.

```
> cd Books/Classification
> cd Author
> ls
Author1 Author2
> cd Author1
> ls
Book1 Book2 Book3 Classification
> cd Classification
> ls
Book6 Year
> cd Year
> ls
2007 2008 2009
> cd 2007
> ls
Book1 Classification
> cd Classification
> ls
>                 # no file
```

In case the user remembers from the beginning more information about the needed files, he can specify directly more complex queries corresponding to paths in the category's file tree. The advantage provided by our system is that the user must not specify the "$(property, value)$" pairs in a fixed order, having more alternatives to do that, as illustrated below.

```
> cd Books/Classification/
> cd Author/Author1/Year/2009
> ls
> Book1
> cd -  # change to the previous location
> cd Year/2009/Author/Author1
> ls
> Book1
```

Although, a more complex way to search files is by using the specialized query syntax supported in the path specified to *chdir* system call or *cd* command in the command line. The following examples illustrate several possible queries together with their outcome.

```
> cd Books
> cd 'Author=Author1&Year=2009'
> ls
Book1  Classification
> cd 'Author=Author1&Year=2009|Year=2010'
> ls
Book1  Book2 Classification
> cd 'Author=Author1!Year=2009'
```

```
> ls
Book2  Book3 Classification
```

The examples described has proven that in our enhanced file system, both files and categories can be manipulated easier and located faster.

**5. Tests and Results.** The previous sections have proven the utility of having a metadata improved file system that automatically constructs a classified organization of files and categories. However, in order to prove that such a system can be used without introducing significant overhead, several usage tests were performed. They measured the time of completing different tasks in a classical file system, i.e. Linux Ext2, compared to the same operations in our enhanced file system.

We run each test using the Linux *time* command. This command takes as parameter the name of the testing application, runs it and returns at the end of that test's execution the following time information: (1) the *real time*, which is the wall-clock time elapsed from the start of the testing application until its termination, (2) the *user time*, which is the time the testing application spent using the processor in user mode to execute its own code, and (3) the *system time*, which is the time the processor was used in kernel mode executing system code on behalf of the testing application. We named the sum of the user and system times *effective time* and the difference between the real time and the effective time *waiting time*. The waiting time is the time spent by the testing application waiting for processors to become available and is highly dependent on the system's scheduler and load. Because the OrcFS testing prototype was implemented in user space using Fuse, most of what should have otherwise been run as system code in kernel mode, was actually run in user mode in an user application, whose execution could have been interleaved with the execution of other running applications. Consequently, the execution of our file system's code could have been suspended many times by the operation system's scheduler in order to also let other applications run. That is why we observed, especially for long running tests, that their waiting time accounts for a large percentage from their real time. We did not consider however the waiting time to be either important or relevant for our tests, as long as our system is ultimately intended for the kernel space implementation. Thus, the overhead introduced by our system is mainly indicated by the comparison of effective times of the tests. Although, we also illustrated in the result tables the overhead at the real time level, just to see the disadvantage of having a user space implementation of our system.

The tests performed can be divided into several categories which are presented next.

**5.1. Creation Tests.** The first two tests performed concern the creation of files and directories.

In **Test1** we created $N$ subcategories in the same category of OrcFS, opposed to creating $N$ subdirectories in the same directory in the classical file system.

The results are depicted in Table 5.1. The last column contains the mean percentage the OrcFS time represents from the Ext2 time, for different types of times we illustrated.

It must be emphasized that when $N$ is rather small (100 or 1000 directories) the difference between the real creation times is not very big. The only overhead here is the one introduced by the FUSE indirection and by the extra logic. However, as $N$ increases and reaches the values of 10.000 or 50.000, another factor must be taken into consideration, which is, as we already mentioned, the Linux scheduler. The problem that occurs is that, by using FUSE the system calls are redirected in user space. Therefore, they are now run in user processes that have a smaller priority. Even more, the Linux scheduler decreases the priority of a process if that process uses intensively the processor. It can be seen from the Table 5.1 that while creating 50.000 directories, the process spends the majority of time waiting for processor. In fact, while performing the test in the classical file system there was a noticeable decrease of the speed of the other processes during the execution of system calls in kernel, while in OrcFS no such event was observed, because most of the code was executed in user space. On the other hand, taking into account only the effective time, we can observe that the overhead is rather small, i.e. up to about 31% for 50.000 directories.

This is why, after a certain number of directories created, the real times obtained are no longer relevant. As a consequence, the following tests will no longer use very large values. We must also take into account that in practice new elements are very rarely introduced simultaneously in the system in such a huge number.

In **Test2** we created $N$ files in the same category in OrcFS, compared to creating $N$ files in the same directory in the classical file system. The result is depicted in Table 5.2.

In the case of file creation the situation is similar to that of directory creation. It is expected that once the project is introduced inside the kernel, the difference will decrease.

| No. of directories | Time type | Ext2 time | OrcFS time | Mean percentage |
|---|---|---|---|---|
| 100 | real | 0m0.408s | 0m0.499s | 122% |
| | user | 0m0.156s | 0m0.176s | |
| | sys | 0m0.144s | 0m0.152s | |
| | effective | 0m0.300s | 0m0.328s | 109% |
| 1000 | real | 0m3.194s | 0m6.315s | 197% |
| | user | 0m1.116s | 0m1.212s | |
| | sys | 0m1.168s | 0m1.388s | |
| | effective | 0m2.284s | 0m2.600s | 113% |
| 10000 | real | 0m38.723s | 3m4.346s | 485% |
| | user | 0m10.969s | 0m12.545s | |
| | sys | 0m10.817s | 0m14.121s | |
| | effective | 0m21.786s | 0m26.666s | 122% |
| 50000 | real | 3m11.378s | 68m4.267s | 2136% |
| | user | 0m50.863s | 1m03.220s | |
| | sys | 0m52.711s | 1m13.473s | |
| | effective | 1m43.574s | 2m16.693s | 131% |

TABLE 5.1
*Test1 results*

| No. of files | Time type | Ext2 time | OrcFS time | Mean percentage |
|---|---|---|---|---|
| 100 | real | 0m0.355s | 0m0.631s | 177% |
| | user | 0m0.132s | 0m0.148s | |
| | sys | 0m0.112s | 0m0.116s | |
| | effective | 0m0.244s | 0m0.264s | 108% |
| 1000 | real | 0m3.144s | 0m7.048s | 224% |
| | user | 0m1.048s | 0m1.312s | |
| | sys | 0m1.164s | 0m1.364s | |
| | effective | 0m2.212s | 0m2.676s | 121% |
| 5000 | real | 0m16.202s | 0m57.611s | 356% |
| | user | 0m05.120s | 0m06.228s | |
| | sys | 0m05.192s | 0m06.656s | |
| | effective | 0m10.312s | 0m12.884s | 125% |

TABLE 5.2
*Test2 results*

**5.2. Directory Opening Tests. Test3** opens a category containing $N$ subcategories, opposed to opening a directory containing $N$ subdirectories. Table 5.3 presents the result of these tests.

**Test4** opens a category containing $N$ files, opposed to opening a directory containing $N$ files. Table 5.4 presents the result of these tests.

**Test5** opens a *Classification* directory containing $N$ properties, compared to opening a directory with $N$ subdirectories. Table 5.5 presents the result of these tests.

**Test6** opens a property directory with $N$ values, versus opening a directory with $N$ subdirectories. Table 5.6 presents the result of these tests.

**5.3. System Simulation Test.** In **Test7** we created a category with $N$ properties and $M$ values associated with each property, compared to creating in the classical file system (Ext2) the entire structure generated by OrcFS in the *Classification* directory in the same situation. We measure only the real time, as it was fully relevant. Table 5.7 illustrates the results.

For the first part of Test 7 there was created a category with 3 properties each of which contain 3 values. In

| No. of directories | Time type | Ext2 time | OrcFS time | Mean percentage |
|---|---|---|---|---|
| 100 | real | 0m0.008s | 0m0.025s | 315% |
| | user | 0m0.000s | 0m0.000s | |
| | sys | 0m0.008s | 0m0.008s | |
| | effective | 0m0.008s | 0m0.008s | 100% |
| 1000 | real | 0m0.032s | 0m0.164s | 512% |
| | user | 0m0.004s | 0m0.004s | |
| | sys | 0m0.012s | 0m0.016s | |
| | effective | 0m0.016s | 0m0.020s | 125% |

TABLE 5.3
*Test3 results*

| No. of files | Time type | Ext2 time | OrcFS time | Mean percentage |
|---|---|---|---|---|
| 100 | real | 0m0.008s | 0m0.014s | 175% |
| | user | 0m0.008s | 0m0.006s | |
| | sys | 0m0.004s | 0m0.008s | |
| | effective | 0m0.012s | 0m0.014s | 117% |
| 1000 | real | 0m0.020s | 0m0.081s | 202% |
| | user | 0m0.012s | 0m0.008s | |
| | sys | 0m0.008s | 0m0.016s | |
| | effective | 0m0.020s | 0m0.024s | 120% |

TABLE 5.4
*Test4 results*

| No. of directories | Time type | Ext2 time | OrcFS time | Mean percentage |
|---|---|---|---|---|
| 100 | real | 0m0.008s | 0m0.060s | 750% |
| | user | 0m0.000s | 0m0.004s | |
| | sys | 0m0.008s | 0m0.008s | |
| | effective | 0m0.008s | 0m0.012s | 150% |
| 1000 | real | 0m0.022s | 0m0.107s | 486% |
| | user | 0m0.004s | 0m0.010s | |
| | sys | 0m0.012s | 0m0.018s | |
| | effective | 0m0.016s | 0m0.028s | 175% |

TABLE 5.5
*Test5 results*

| No. of directories | Time type | Ext2 time | OrcFS time | Mean percentage |
|---|---|---|---|---|
| 100 | real | 0m0.008s | 0m0.055s | 687% |
| | user | 0m0.000s | 0m0.000s | |
| | sys | 0m0.008s | 0m0.004s | |
| | effective | 0m0.008s | 0m0.004s | 150% |
| 1000 | real | 0m0.022s | 0m0.107s | 486% |
| | user | 0m0.004s | 0m0.006s | |
| | sys | 0m0.012s | 0m0.020s | |
| | effective | 0m0.016s | 0m0.026s | 163% |

TABLE 5.6
*Test6 results*

| No. of properties | No. of values | Ext2 time | OrcFS time | Mean percentage |
|---|---|---|---|---|
| $N = 3$ | $M = 3$ | 0m2.502s ($526 dirs$) | 0m0.079s ($10 dirs$) | 2.65% |
| $N = 4$ | $M = 4$ | 1m18.218s ($17.749 dirs$) | 0m0.176s ($17 dirs$) | 0.22% |

TABLE 5.7
*Test7 results*

the OrcFS, this means creating 10 directories. However, simulating the classified structure that is automatically created means generating 526 directories in the classical file system. Therefore, the resulting time difference between these two operations is great. The second part emphasizes even more this difference, since in the case of 4 properties and 4 values for each of them means creating 17 directories in OrcFS and 17,749 directories in the classical file system. This is the test that shows the full power of the OrcFS. A single file may be found by taking several paths and thus, the risk of taking a wrong path while knowing a few characteristics of the desired file is significantly reduced.

In conclusion, while operations of introducing new data in the file system are delayed in the OrcFS, these time differences are not as important as the gain for creating an immense structure with only a few number of operations.

**6. Related Work.** The first project that approached this issue is the Semantic File System [11]. It was designed to provide associative attribute-based access to the contents of an information storage system. The attributes are automatically extracted from the files stored on disk with the help of transducers. A transducer is a filter that takes as an input the contents of a file and outputs the files entities and their corresponding attributes. A user does not have the capability to define attributes, he may only write queries, whose result are virtual directories. The solutions implemented in [11] represent the starting point of OrcFS. Some of the concepts proposed by the Semantic File System, such as virtual directories or automatic classification were adapted by OrcFS as the base of the entire project. However, what was not implemented is the automatic attribute extraction, which was left as a future improvement.

An improvement of the Semantic File System is Nebula [6], in which the user is able to perform CRUD (create, read, update, delete) operations on attributes. Nebula implements files as sets of attributes. Each attribute describes some property, such as owner, protection or file type. The content of a file is represented by a special text attribute. The traditional directories are replaced with database views. When a file is created, it is placed in an index, not a directory. When the file system is refreshed, the file will appear in the appropriate views. This is how views dynamically classify files. As opposed to this approach, OrcFS treats the file as an indivisible item.

A further improvement was made by Be File System (BFS) [10]. It stores the list of attributes associated with a file as a directory, the address of which is stored in the attributes field of the BFS i-node structure. The attribute directory of a file is similar to a regular directory. Programs can open it and iterate through it to enumerate all the attributes of a file. The attribute directory solution is rather slow in the case of having several small attributes. This it why, the spare area of the i-node disk block is used to store small attributes. Because the focus of OrcFS is to increase the flexibility of expressing relationships, it does not employ the i-node structure, but a database to store the file attributes.

A notable approach to the relational organization of the file system is LisFS (Logical File System) [15]. In this approach the storage structure, called formal context, holds objects that are described by logical formulas. These objects represent files or parts of files. Both paths and queries are seen as formulas. LisFS allows boolean querying. There are two types of proprieties: intrinsic, which are computed from the content of the object and extrinsic, which are assigned by the user. A key feature is the fact that LisFS runs over the Linux kernel, respecting the VFS (Virtual File System) interface. This assures compatibility with the current applications. This project is the one that has the most elements in common with OrcFS. They both use a database to store metadata, work in user space and enable query navigation. The main difference is that in OrcFS treats both files and directories in the same way.

WinFS (Windows Future Storage) [12] was first presented in 2003 as an advanced storage subsystem for

the Microsoft Windows operating system. It breaks down the concept of files into subdivisions, extending the possibility of data processing at a lower level. Furthermore, it treats all data as objects. The user can create relationships or sets of objects through an application and reuse them in other applications and define functions on objects.

Although it is not a file system, Google Desktop [3] is worth mentioning. It is a desktop search application that uses indexes to manage the information on the computer. Once installed, it starts indexing the email, files and Web history stored. This operation runs only once, when the computer is first idle for 30 seconds. Google Desktop also ensures that the index stays up to date, adding new e-mail as it is received, files as they are updated them and web pages as they are accessed.

The most similar projects with OrcFS are those in [9] and [8] on whose architecture the OrcFS is based on, but which we extended to homogeneously treat files and subcategories in a category as generic file system elements. We also developed the query command line syntax to support complex interrogation and navigation in the OrcFS file system using existing shells and file browsers.

**7. Conclusions.** An architecture of the file system enhancing the organizing and searching capabilities has been developed. It is based on allowing the user to associate descriptive metadata in the form of property-value pairs to both files and directories. The system creates virtual file containers forming a classified organization, in order to provide multiple access paths to the same file. Navigation down this tree structure represents a permanent query refinement, in which the set of results is diminished until the user either finds the desired file or an empty directory is obtained.

The usability of OrcFS has been proven at a subjective level, by creating a specific system and navigating through the classification. It was also proven at an objective level, by measuring and comparing the times of performing different tasks both in the classical file system and this enhanced system.

Several steps that will be taken in the future for improving even further the navigation, search and overall accessibility are:

- Automatically extract property-value pairs from different file formats.
- Allow the user to define category templates with specific properties. This would shorten the initial job of environment creation. Therefore, the user will be able to use the same set of properties in different locations of the system, without having to specify them every time.
- Maintaining a record of the paths where a template has been applied, so that the user is able to formulate queries such as "Find all locations for the template where certain restraining conditions are valid".
- Develop a specialized file browser which takes advantage of the enhancements of OrcFS, in order to present a richer file system. This application must clearly differentiate between property, value and category directory types. It must allow the user to specify property-value pairs using a form and not through file operations like in the classical browser.
- Integrate OrcFS in the Linux kernel, in order to decrease the overhead introduced by FUSE redirection.
- Develop a distributed relational file system. A client should be able to make queries and navigate like in OrcFS but the answer will come from different servers, transparently to the user. The individual responses of each server will be merged, in order to present the user with a uniform view of the entire system.

REFERENCES

[1] Ext2 file system utilities. Internet page last accesed on July 2010.
[2] Fuse. Internet page last accesed on July 2010.
[3] Google desktop. Internet page last accesed on July 2010.
[4] Sqlite. Internet page last accesed on July 2010.
[5] Daniel Bovet and Cesati Marco. *Understanding the Linux Kernel*. O'Reilly Media, 3 edition, November 2005.
[6] Mic Bowman, A. Dharap, Mrinal Baruah, Bill Camargo, and Sunil Potti. A file system for information management. Technical report, The Pennsylvania State University, 1994.
[7] Alexandra Coldea, Adrian Colesa, and Iosif Ignat. Orcfs: Organized relationships between components of the file system for efficient file retrieval. In *Proceedings of The 12th International Symposium on Symbolic and Numeric Algorithms for Scientific Computing (SYNASC '10)*, pages 434–441. IEEE Computer Society, 2010. (DBPL, IEEExplore).
[8] Adrian Colesa, Victor Cionca, Alexandru Ţaţa, and Iosif Ignat. A meta-data enhanced file system. In *IEEE International Conference on Intelligent Computer Communication and Processing (ICCP'07)*, 2009.

[9] Adrian Coleşa, Iosif Ignat, Zoltán Majó, and Victor Cionca. A meta-data enhanced file system using the classical interface. In *Proceedings of The Tenth International Conference on Applied Mathematics, Computer Science and Mechanics*, Cluj-Napoca/Băişoara, Romania, June 2006.

[10] Dominic Giampaolo. *Practical File System Design with the Be File System*. Morgan Kaufmann Publishers, Inc.

[11] David K. Gifford, Pierre Jouvelot, Mark A. Sheldon, and James W. O'Toole Jr. Semantic file systems. In *Proceedings of 13th ACM Symposium on Operating Systems Principles*, pages 16–25, October 1991.

[12] Richard Grimes. Revolutionary file storage system lets users search and manage files based on content. *MSDN Magazine*, January 2004.

[13] Aditya Kashyap. File system extensibility and reliability using an in-kernel database. Technical Report FSL-04-06, Stony Brook University, December 2004.

[14] Yoann Padioleau and Olivier Ridoux. A parts-of-file file system. In *Proceedings of the annual conference on USENIX Annual Technical Conference (ATEC '05)*, page 17. USENIX Association, 2005.

[15] Yoann Padioleau, Benjamin Sigonneau, and Olivier Ridoux. Lisfs: a logical information system as a file system. In *Proceedings of the 28th International Conference on Software Engineering (ICSE '06)*, pages 803–806. ACM, 2006.

*Edited by:* Dana Petcu and Alex Galis
*Received:* March 1, 2011
*Accepted:* March 31, 2011

Scalable Computing: Practice and Experience
Volume 12, Number 1, pp. 93–103. http://www.scpe.org
ISSN 1895-1767


# VIDEO AND SENSOR DATA INTEGRATION IN A SERVICE-ORIENTED SURVEILLANCE SYSTEM

DAMIAN MIERZWINSKI, DARIUSZ WALCZAK, MARCIN WOLSKI, MARCIN WRZOS
NETWORK SERVICES DEPARTMENT
ICHB PAN - POZNAN SUPERCOMPUTING AND NETWORKING CENTER
POZNAN, POLAND
{DAMIANM, DWALCZAK, MARCIN.WOLSKI, MARCIN.WRZOS}@MAN.POZNAN.PL

**Abstract.** Video surveillance systems have become very popular these days. In households, enterprises and even cities the monitoring systems have been deployed to improve the sense of safety. Additional sensors which produce large amounts of data are very often used independently. A real challenge is a meaningful integration of video and sensor streams, which could assist people operating these systems. This challenge requires a new, smart approach to browsing, analyzing and integrating information of various modalities and from various sources. Nowadays motion detection, face recognition and integration with external sensors are the popular subject of research. We notice the lack of efficient and flexible solutions for managing video surveillance data for public or private security. The problem is scalability and manageability of more and more complicated systems connecting large amounts of devices. In this paper we would like to focus on managing video and other data sources in the service-oriented manner for a large scale domain. Simplifying monitoring configuration and usage is also in the scope of our interest.

**Key words:** video surveillance system; sensor data integration; video sources management; mobile surveillance; SOA; service integration; event processing

**AMS subject classifications.** 68M10,68M11,68M14,68U35

**1. Introduction.** Monitoring solutions, like those deployed in cities or in large companies, grow together with the area under surveillance. Public and private security forces very often have access to many surveillance systems, provided by various vendors. There exist many additional sensors and devices which could add new and useful features to the system if they could only be integrated with it. These require additional management tools and applications. An integrated approach would be desired.

Direct integration is possible for a few different systems but is nearly impossible on a large scale like in the case of integrating private surveillance systems. IT industry is aware of this problem and tries to solve it with Service Oriented Architectures (SOA) where each module exposes a standard interface for communication with other systems. Therefore, SOA principles should be investigated in a distributed video surveillance environment. Another emerging trend in software integration that is suitable to be used in surveillance and public security domains is Event Driven Architecture (EDA) and Complex Event Processing (CEP) [1].

There is a number of requirements for monitoring systems in the literature. They focus on video image quality, simple event detection, annotating and modelling, detecting complex events using simpler ones or finding patterns of events. Other requirements include scalability issues or lack of ontologies for visual surveillance systems. The quality and performance measurement of new surveillance systems should be addressed here as well [2].

Visual monitoring alone is very often not enough to satisfy the growing expectations put on the surveillance systems. Research interests are put on integrating video, audio and data originating from other sensors, e.g. motion sensors, door locks, RFID scanners. Methods of combining visual data from various video sources or tracking objects between them are also subject of research [3].

Scalability, effective management and QoS in large and distributed monitoring systems are the popular topics of research efforts [4, 5, 6]. Surveillance of highways, cities, forests and industrial terrains managed by the police, public forces or security companies are the best examples of systems that find the above properties important.

The paper presents how the SOA and CEP principles can be applied to large and distributed monitoring systems which have to be scalable and easily configurable. The contribution summarizes research and development achievements on mobile surveillance platform at Poznan Supercomputing and Networking Center[1] (PSNC). The work on the service oriented surveillance system has been started in collaboration with the Polish police and has gained a positive feedback from the target users. This work was earlier described in [7]. As a promising extension to this topic video and sensor streams integration is being conducted within the Future

[1] http://www.man.poznan.pl

Internet Engineering project[2]. The whole design has been extended with event processing mechanisms that simplify management of various types of devices.

This paper is organized as follows. Section 2 describes state of the art in large video surveillance systems. Section 3 describes the system architecture, video and event processing, services registration, video browsing, video description. Implementation details can be found in section 4, and a brief description of first tests in section 5. Section 6 concludes the paper.

**2. Related work.** First video surveillance solutions were based on an analogue technology called Closed Circuit Television (CCTV). In the most trivial case these were cameras connected with TV sets located in one room. Currently most of the video cameras use a digital Charge Coupled Device (CCD) to capture images and analogue techniques to distribute and store data. The conversion between digital and analogue signal causes video quality degradation. Thanks to technology evolution fully digital systems powered by computers can be used without conversion. With high performance computers additional features can be expected, e.g. real time events detection and license plate recognition. In [8] three main generations of surveillance systems were distinguished: analogue CCTV systems, automated visual surveillance done by combining computer vision technology with CCTV systems, and automated wide-area surveillance systems. This contribution focuses on 2nd and 3rd generation systems especially in distributed architectures. Image processing is out of the scope of this paper.

Highway monitoring solutions are probably the first and the most distributed surveillance systems. At the beginning simple inductive loops were used to detect cars. Nowadays similar sensors count traffic, measure queue lengths on ramps and at intersections or help at parking lots to find a free place. In [9] authors describe VDS240, a vehicle detection system based on wireless sensors. The sensor infrastructure consists of two different nodes, the magnetic one, placed in the center of the lane and the Access Point (AP) located on the side of the road. The AP aggregates data from magnetic sensors and sends it to the Traffic Management Center (TMC) or a local controller. The radio interface enables direct communication with up to 96 nodes within the range of 1500m. Similar systems are built using laser [10], radar [11], acoustic [12] and optical sensors [13]. Because of the hardware development and great improvements in the video analysis, sensors are replaced or supported with video cameras. It makes highway monitoring systems one of the most significant examples of distributed video surveillance solutions. Captured images are used to estimate all parameters usually measured by sensors. Depending on the quality of the video stream and the algorithms used the system can classify vehicles, trace lane changes or even recognise license plates. Usually the system requires only one camera per monitored location. The video scene must be segmented and transformed into objects. Their behaviour is recognised and tracked. An example of such a system was described in [14]. High resolution cameras can be applied to monitor objects smaller than cars (e.g. in Stockholm, Sweden cyclists are under surveillance). In [11] authors present a video analysis system applied to detect biking in the "wrong direction". They also analyse traffic conflicts between cyclists and other road users. They use the foreground-background segmentation for the trajectories estimation and the shape analysis of points of interest for the speed estimation.

Some new surveillance solutions are characterized by the fact that they use visual surveillance methods combined with information originating in additional systems, such as sensor devices or external knowledge bases. They are gathered under the terms smart surveillance or multisensor surveillance systems. The key idea is to enhance monitoring with sensors or integrate it with whole sensor networks where video cameras are just one type of sensors.

In [15] authors present a data fusion system for objects tracking. They use optical and infrared sensors to monitor the same outdoor scene. Redundant information is fused together to obtain a more accurate estimate. Physical sensors are connected directly to first level nodes according to their location. Nodes create a hierarchical structure. Data fusion is performed in a centralized fashion considering sensor reliability every time. Authors compared this to a single camera system and noticed more accurate trajectories.

Surveillance systems are getting bigger and more complicated. Software architects deal with this issue using service-oriented architecture. According to the SOA paradigm, software should be delivered as loosely coupled, and cooperating services which should be described, published and easily discovered. In such an environment new applications called business processes can be created by composition of existing services [16]. SOA is mature in the business world [17, 18, 19] and is expanding to new domains like video conferencing [20] or the public security sector [21].

[2]http://www.iip.net.pl

N.E.S.T [22] is an example of an SOA-based surveillance system. It was built by the Institute for Information and Data Processing, Fraunhofer IITB. Authors worked on decentralization, expandability and upgradability of today's monitoring solutions. They developed an architecture and set of services for video surveillance, e.g. motion detection and tracking, and abandoned luggage detection. In N.E.S.T an operator defines different surveillance tasks as automated processes, sometimes with human interaction. The tasks are modelled in BPEL language and executed in the BPEL-engine connected to services through a service-bus. There is a second JMS-based system bus for very frequent notifications. A third bus is planned for the future, it will be an infrastructure for streaming. Static and dynamic data are stored in the World Model through the Model Access Service. Presented services are suitable for a hotel scenario. In this case, the receptionist starts the new guest tracking process. It estimates his/her route to the room and checks if the guest did not get lost.

Another example of a smart surveillance system is developed by the SAMURAI project, funded by the EU FP7. One of the project objectives is object detection in multi-camera environment under real world conditions and on using multi-modal data fusion. The detection system is based on heterogeneous sensor network consisting of fixed position legacy CCTV cameras and mobile cameras with GPS receivers.

IBM in [23] highlights three key challenges that need to be addressed to enable the widespread deployment of smart surveillance systems:

1. The multi-scale challenge better information acquisition based on video analysis, e.g. face recognition, person tracking
2. The large system deployment challenge.
3. The contextual event detection challenge better interpretation of gathered information to detect events and identify trends

The Exploratory Computer Vision Group in IBM addressed these challenges in their solution [23]. The IBM Smart Surveillance System has two key components: the IBM Smart Surveillance Engine (IBM:SSE) and the IBM Middleware for Large Scale Surveillance (IBM:MILS). The first one is a software-only event detection technology based on video analysis likes: object detection, object tracking, object classification. It also provides real time alerts and creates Viewable Video Index description of all interesting activities in the video, implemented as a set of XML files. MILS converts this index to relational tables and provides powerful Query Services mechanism. Thanks to it user will be able to query for activities providing time period, object size, object class, object motion description. It will be also possible to query by context-based content similarity, e.g. [23] "Show all activities where there were blue cars or cars similar to this car here the user specifies an example car through an image".

This paper addresses configurability, flexibility and sensor integration issues. Based on previous experience and state of the art in video surveillance, an SOA-based architecture for monitoring video delivery and CEP-based sensor data processing has been proposed.

## 3. System description.

**3.1. Architecture.** The proposed solution is based on SOA principles in order to address flexibility and integration issues of present video surveillance systems. SOA is an architectural style that supports service orchestration [24]. In particular, SOA defines the find-bind-execute paradigm to differentiate between service providers and service consumers and their loose coupling. We propose to apply this main paradigm to the video surveillance system. Instead of creating a direct relationship between the provider (video source) and the consumer (typically a person who is watching the video content) each video source is seen in the system as a single video service. All registered video services create the service cloud. Then someone who acts a service consumer explores the cloud in order to find relevant service, binds to its endpoint and executes the service.

Referring to SOA principles a prototype solution for a mobile monitoring station was prepared (see Section 4.1). The main part of the architecture is the Tiberinus server. The server consists of three main components (see Figure 3.1). The first one is an application enabling to register external video sources in the system. The only requirement for a device or application to register and become a video source is to expose RTP/RTSP video streams. The second component is a streaming server reflecting video streams originating from the video sources and a video recorder, an additional application that records video streams and stores them as files in the file system. The third component is an application that allows users to search the repository of registered video sources and archived video material. It also has the capability to playback and download archived and live videos. It provides methods of searching for the desired video sources and archived material, registering video sources, accessing their descriptions and services provided by them. Both the register and search application

expose web service interfaces named Tiberinus-Manager and Tiberinus-Search, respectively.

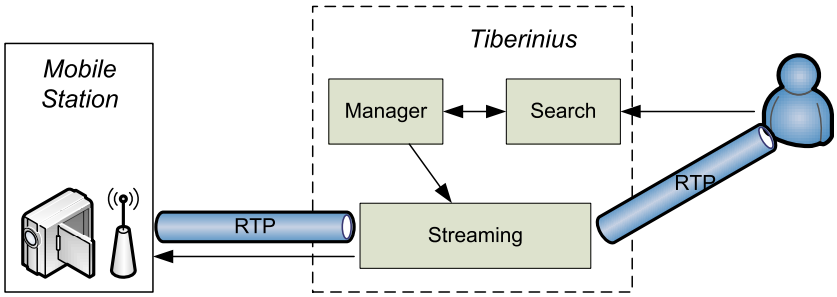



FIG. 3.1. *System architecture*

Server modules are developed as loosely coupled services so configuration is as easy as SOA infrastructure management. One can use standard tools like BPEL editors and runtime engines. In case of adding new features like face or license plate recognition service one has to define a new BPEL process and deploy it on the BPEL engine.

While SOA is proper for service integration there exist more convenient architectures for sensor data integration. Sensor devices are usually capable of sending simple events including appropriate data. In order to achieve high efficiency and low energy consumption sensor gathers data, sends it and goes to sleep mode its network interfaces are turned off. Low level events sent by sensors can be easily mapped to the events in the IT world. This task is performed by the event streaming layer which is dedicated to sensor data integration.

Figure 3.2 presents an extended view of the architecture with Tiberinus in a service ecosystem together with video analysis services and event processing layer.

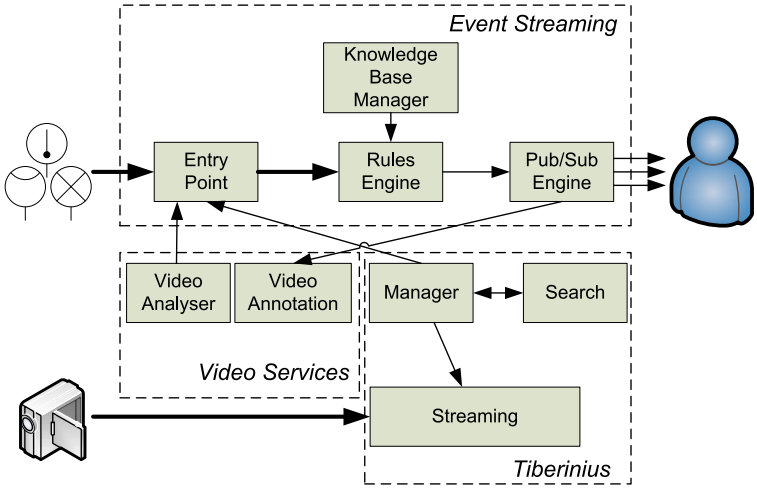



FIG. 3.2. *Tiberinus ecosystem*

The event processing layer allows users of a surveillance system to subscribe for the events they are interested in. The system will send notifications to the user in case of an alert is raised or a specified event on sensor device happens. It can be done on different levels of abstraction. The example of the low level event in the surveillance domain is "the light is turned on", on the other hand the high level event could be "employee X came to work". The first module in this layer collects events and exposes entry points for sensors and services. Events are grouped by time windows and passed to the event processing module. This module is based on Business Rules (BR) engine which applies user defined actions to a particular event configuration. BR engine can check if the event occurred or not, what are the event details and if there was any other associated event raised before. All actions are defined by rules. They can be managed by a special module with a user friendly interface. User can define his own events, rules and actions through this UI. All data is stored in the shared repository. Rules in BR engine generate new events which are distributed by the notification module. New

event enters the event processing layer by the entry point or can notify services that were subscribed for this type of event.

**3.2. Video source registration.** Registration of video sources on the server is done using the Tiberinus-Manager web-service located on the Tiberinus server. The web-service can be accessed using either the SOAP or REST interface.

Registration of video sources on the server consists of two parts. The first part is submitting all the metadata to the Tiberinus server. This metadata includes descriptors such as the name of the video source, its text description, geographic location, registration date, etc. Additionally, information about all of the services that are provided by the video source is included, e.g. PTZ control, image parameter selection. The second part is sending the RTP/RTSP video stream to the server where it can be archived or reflected to client applications. When the video source is successfully registered on the server its description is made available through the Tiberinus-Manager web-service. Information about the URI location of the live video stream and locations of the archived material (either as a URL to a static file or to a video stream) is also made available in the same way.

Very few video devices have the capability to push and announce video streams that they produce. In order to broaden the spectrum of compatible devices, the process of pushing the stream to the server was replaced with pulling the stream from the video devices. After the first part of registration finishes successfully, Tiberinus connects to the address provided with the registration metadata and negotiates the connection using RTSP.

Video sources can be registered in many Tiberinus servers at the same time. This could be done in order to send the video sequences to different locations/institutions or to store the video material in more than one place. However, regarding that the mobile video source would often have limited bandwidth capacity (e.g. when using a modem or GSM/3G wireless connection) this would not be the optimal choice. Instead, video sources once registered in one Tiberinus server could be registered in other Tiberinus servers thus building a hierarchical structure. Registering video streams in Tiberinus servers that are higher in the hierarchy is done analogously. All of the metadata submitted during the registration will be the same. Optionally, it can be enriched with information about node hierarchy and a description of each node. The only element that will be different is the RTSP address of the video source, now pointing to the Tiberinus video server (see Section 3.5).

**3.3. Searching and playing streams.** In traditional CCTV recorders one has to connect the screen to the device and find relevant recording using remote control. In our opinion it is not enough, nowadays people are used to Video on Demand served via a web browser. Most TV shows, series and clips are available on web pages. Modern surveillance systems should adapt this common way of watching videos.

In presented system an end user can browse and search for the desired data using the web application. The search can be performed using multiple criteria. The user can input the name or a description of the video source, the date of the beginning or end of the recording. The user can select if s/he is only interested in finding live streams, registered videos or both. Additionally s/he can enter the type of the video source: either a static legacy CCTV camera or a mobile video source. The repository can also be searched using location criteria. The user can select a location on the map and enter search distance or enter the geographical coordinates manually. If the location criteria are used, only the results originating from video sources located in the searched locations will be presented.

Results returned by the web application contain all the metadata related to the video source, URLs to the video streams and, in case of archived videos, URLs to the files that can be downloaded. The user can view all of the videos using the QuickTime movie player embedded in the webpage.

Tiberinus provides also machine-oriented method of searching the videos which can be accessed using the REST or SOAP interface of the Tiberinus-Search web-services. Web-service exposes similar methods of searching for live video sources and archived material to those found in the user interface. All the search criteria mentioned earlier can be used.

**3.4. Event processing.** After registering a new video source the manager module generates an event. This event goes through the entry point to the rule engine. BR engine results go to the Pub/Sub module which notifies proper services (see Figure 3.3). Next, depending on the service functionality configuration the right action is taken. Similar scenario takes place when a new sensor is announced in the system and starts sending data. The multimedia processing services connect to the streaming server and immediately start to stream the audio-visual content in order to perform the analysis. After the analysis of the part of data (e.g. a single video frame) is completed a new system event containing its result is produced (see Figure 3.3). Depending

on the nature of the service the event can be generated every time or can only be generated when the analysis result is positive. The example video processing services could consist of a face detection, automatic number plates recognition or movement detection services. In more complex systems those services could perform the unattended baggage detection, person identification or could perform image analysis on an x-ray, infrared or MMW images. The second set of services is configured to listen to the other system events, e.g. containing the data analysis results. It is possible to launch the services only if certain conditions occur, e.g. when two events of given types occur in a given time window or on given time and system conditions. At this point there is a possibility for a user to implement advanced processing rules combining events concerning different types of data or to use external data sources and context information. The video annotation tool is an example of such service. After a face detection event is published the video annotation tool updates the description of the video stream where the face was detected with time and spatial information about the event. When the human voice event was published this information is also included in the description. When another service identifies the person information in the video stream description can be further specified and extended. The annotation tool could be configured to listen to any kind of events and to extend the video file description in appropriate manner. The video annotator service is subscribed to receive the recognition event but not the sensor event. Additionally in the event processing engine there is a rule that will generate a complex event when a recognition and a sensor event appear in the system in a given time window. The video annotator is subscribed for the complex event and when it appears in the system the service is notified.

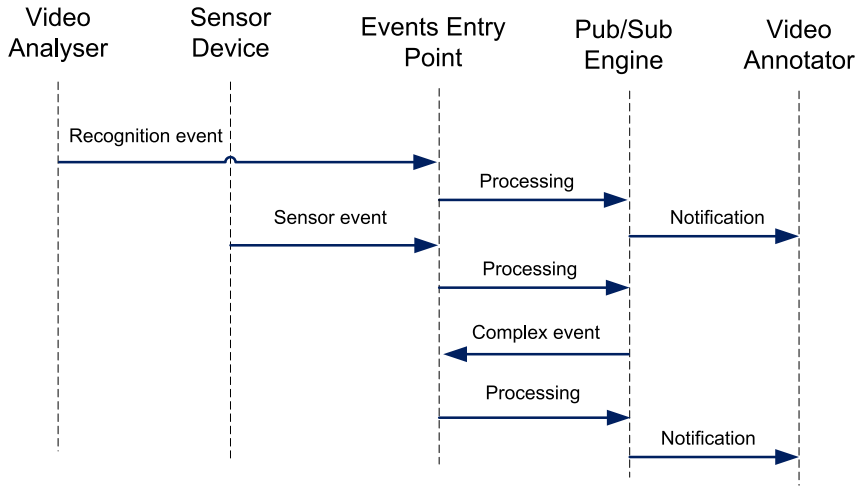



FIG. 3.3. *Video annotation processing path*

More complex service is the tailgating detection service. It is configured to listen for the "door opened" and "person detected" events. When the system detects one event of the first type and more than one event of the second type on the configured window in the video image from the camera pointed at the door then it produces an alert. The alert is an event of the new kind: "tailgate detected". Other services can listen for that event and react, e.g. confirm it or notify selected people. All of the events in the system can have specified priority level and confidence factor. Other more complex scenarios could include the use of information from the sensors, other image processing services and context services.

**3.5. Distributed architecture.** The previous sections present the basic configuration of the video surveillance system and sensor fusion ecosystem.

However, to address requirements and challenges posed by large scale domains, where multiple video sources may exist, distributed configuration needs to be used. A hierarchical architecture similar to [25] has been created but in presented case structure is not forced by geographical location. It is configured based on business rules and user requirements. Individual user or company might wish to investigate different location from one place and pass streams to proper institutions or a cloud service. Two types of nodes can be distinguished: leaves and internal nodes. A leaf can work as a fully featured local surveillance system. Internal nodes aggregate, archive and pass streams according to user requirements (Figure 3.4). Their connections depend on use cases. Bigger companies with more than one location would probably like to aggregate streams and archive them in one place.

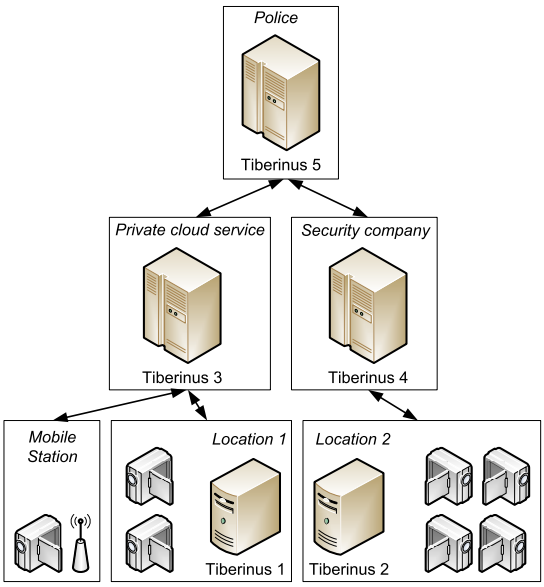



FIG. 3.4. *Example of Tiberinus distributed configuration*

Small businesses usually outsource security services and give them access to streams from the local surveillance system.

## 4. Prototype.

**4.1. Ad-hoc monitoring station.** As proof of concept, a Mobile Monitoring Station (MMS) prototype has been prepared. In general it can be described as an autonomous and manageable streaming source. It can be used to monitor areas not being covered by standard monitoring solutions.

The prototype MMS is a car with installed equipment which streams video over wireless networks: namely 802.11g or UMTS (see Figure 4.1). The main part of MMS is an access point, OSBRIDGE 3GN[3]. It connects to the remote server and establishes an encrypted vTun[4] channel. Other devices can communicate and upload video streams through this channel. Axis M7001 video server, which allows connection of an arbitrary analogue video camera has been used as the video source. Every device is supplied with power from the primary battery used in the car, or an additional extended battery if needed.

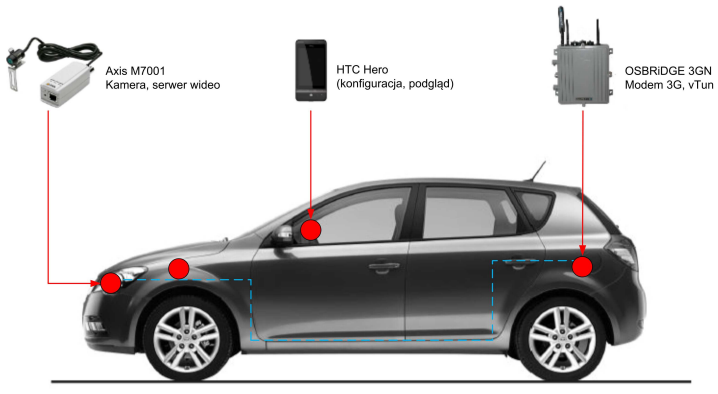



FIG. 4.1. *General MMS prototype overview*

The MMS is managed over the WIFI network by an HTC Hero mobile phone based on Google Android 1.6, with a dedicated application installed. It is simplified to two main actions: registering and unregistering video

[3]http://www.osbridge.com/?q=en/node/93
[4]http://vtun.sourceforge.net/

sources at the remote server. Moreover, it is possible to view video streams that are generated by currently registered cameras. This gives a possibility to verify the camera configuration (zoom, angle, focus, etc.). When registering video sources using the mobile phone application, information about the phone location (taken from the phone's GPS receiver) and direction (taken from the phone's digital compass) are sent to the server. Therefore, it is advised to align the phone with the camera's line of view to store information about the camera's direction.

**4.2. Tiberinus server.** The Tiberinus server consists of three components that were briefly described in Section 3.1. The software managing video streams and storing the metadata information was developed by PSNC. Darwin Streaming Server[5] was used as the streaming server enabling live and stored video material playback. VLC[6] media player was used to fetch the stream from video source, send it to video server and optionally store it in the file system. All of the web services and the web application allowing search and playback of video streams were also developed for this project.

All of the components of the Tiberinus server were installed and configured to work on a virtual machine running on the Linux operating system. This solution enabled testing the system in different network topologies and configurations.

**4.3. Sensor data integration.** As a proof of concept laboratory configuration of the event processing layer was build. Implementation is based on JBoss Drools[7] - Business Logic integration Platform. This package consists of three modules: Event processing, Rules and Workflow. First two were crucial while the third one was useful for integration with services layer. Knowledge base was build through Drools Guvnor application which is a web based rule manager for Drools. Notification module "Pub/Sub Engine" is based on XMPP protocol and implemented upon Jabber server.

Sensor network is based on SunSPOT[8] devices and Axis IP Cameras. SunSPOTs provide data about the environment like temperature, light level and acceleration of the devices. The Axis IP Camera's provide basic movement detection functionality. Video services were developed on basic functionality of OpenCV library. The annotations on the surveillance material are performed using the MPEG-7 standard.

**5. Proof of concept.** The proof of concept performed in the laboratory and in car environment aimed to confirm if the system architecture, as well as software solutions are capable of handling a basic set of video streams in a service oriented manner.

**5.1. Laboratory configuration.** In the first scenario one set of devices was located in an office building in Poznan. It consisted of an analogue camera and a video encoder. The camera was placed outside and attached to a balcony facing cars stopping at traffic lights. The parameters of the camera were set to focus on the cars' licence plates. The camera was connected to a video server which was connected to the Tiberinus server using an Ethernet connection.

The second set of devices was located in the same office building. It consisted of a Wi-Fi capable IP camera and a wireless 3G router. The camera was placed inside the building and was pointing at the same road junction, but was configured to capture the whole scene. The camera was connected to the wireless router using a Wi-Fi connection. The router was connected with the same Tiberinus server as before using the 3G connection, and a VPN tunnel was established between them.

After the infrastructure was set, a testing procedure was performed. The video sources were registered using the mobile application. After that the system was searched for desired video materials: currently registered live video sources and video sources located near the coordinates of the office building. In the second scenario an additional Tiberinus server was deployed. The second set of devices was connected to this additional server. Both of the Tiberinus servers were connected to the ESB server. Similar testing procedure was performed. The video sources were searched using two different Tiberinus web applications.

Finally, the system was configured as in the first scenario but with Ethernet connection instead of 3G. The server was set to record the video material. In this configuration it was left unmodified for a week. About 13GB of video material was obtained together from both cameras, compressed with the h264 video codec. This video material could be used for further analysis.

[5] http://dss.macosforge.org/
[6] http://www.videolan.org/vlc/
[7] http://www.jboss.org/drools/
[8] http://www.sunspotworld.com/

**5.2. Mobile configuration.** Accordingly, a wide scope of tests has been performed using the MMS as video source and uploading data through UMTS. The car with a surveillance system installed streamed video from many different locations which resulted in variable strength of a 3G signal. Static tests were conducted at a parking lot. Moving objects like cars or people walking on the pavement were recorded. Mobile tests were performed from the car being in motion, riding through the streets of the center of Poznan.

As expected, the quality of gathered records is noticeably lower in comparison to video streamed over wired networks. Subjective experience can be described as low resolution but smooth video, which was achieved for TCP as the transport protocol for RTP stream. Surprisingly, the UDP protocol had to be abandoned because of a very high ratio of lost frames when testing in poor network conditions, i.e. high ratio of lost packets and high latency.

**5.3. Event processing.** Using architecture described in 3.4 and implementation described in 4.3 several tests were conducted. Axis Video camera and its motion detection functionality was used. SunSPOT sensor network was installed. Simple video analysis service which detects faces on the image was developed. With this tools complex events related to conference room were detected, e.g. possible conference room occupancy or room condition changes. Integration with services layer was also tested thanks to video and notification services.

Complex events add several additional dimensions to the monitoring process. It was proven that in the test scenario the decisions based on extra assumptions are more accurate. A turned on light does not imply meeting inside a conference room  someone might forgot to turn it off. Motion detection also does not imply meeting  maybe someone is doing cleaning. Face detection might be helpful and face detection event might indicate meeting if faces are looking at the screen together. Usually one sensor is not enough to make the right decision and take specified action. When using the combination of those events the false positive rate decreases.

In order to detect some real life context events rules were written in special Drools domain specific language. One of that rules is shown below.

```
1: rule "possible-meeting"
2: when
3:    LightEvent(status==LightEvent.ON)
4:    $m : MotionEvent()
5:    FaceDetectedEvent(counter > 1, this after $m)
6: then
7:    PossibleMeetingEvent e = new PossibleMeetingEvent(new DateTime())
8:    insert(e)
9: end
```

FIG. 5.1. *Drools rule example*

Each rule contains two main sections, the conditions section after word *when* and actions section after word *then*. Declarations start with *rule* keyword followed by name of the rule.

Figure 5.1 shows *possible-meeting* rule which detects a potential meeting in a conference room. Rule conditions say when the rule actions have to be taken. In this example all three events have to occur: the light must be turned on, motion has to be detected and then the face detection service has to find more than one face. These real life events are mapped to events in the system, accordingly: LightEvent, MotionEvent and FaceDetectedEvent. When all conditions are fulfilled the new event is created. New event notifies proper services that the conference room might be occupied. During the creation the current time is saved within the event. Last line of the actions section inserts this event to the rule engine. This way the new event can be used by other rules.

Rules give a user great flexibility in combining low level events into higher level events which should trigger proper services or notify user. Drools Guvnor provides a user friendly interface for building rules so the user do not have to know all the language details.

**6. Summary.** This paper described the way the SOA principles and tools can enhance modern video surveillance systems. Easy configuration and integration of distributed and differentiated video solutions can enhance public and private security forces work. Flexible composition of smart surveillance services like face or licence plate recognition is a great advantage of loosely coupled CCTV implementation. In addition, an

IP-based and loosely coupled architecture is ready to scale, which has been proven in many enterprise solutions. Easy access to recorded and live streams will improve the efficiency and usability of CCTV. The user friendly interface should encourage home and small business users to invest in modern surveillance solutions.

The event processing subsystem is flexible and can be extended to fit the requirements of the particular surveillance site. The system events and event processing rules can be extended during the development process. The simple video analyzing services as described are to demonstrate the basic capabilities and show the future opportunities of such services in the domain of visual surveillance using the architecture presented in this paper.

The crucial part of further development works will be mobile station tests in operational conditions. The first evaluation has discovered that mobile video sources very often have weak processing abilities. Additionally network connections in a mobile environment often have limited bandwidth and therefore the image quality is strongly dependant on the network provider and technology used. In such conditions management and description of video sources is crucial.

The first attempt described in the related work paragraph and presented proof of concept solution confirmed that SOA is a promising direction of research for large surveillance systems. Scalability tests, sensor's data fusion, semantic description of systems and content are the most interesting topics for further work based on the presented prototype.

## REFERENCES


[1] D. Luckham, *The Power of Events: An Introduction to Complex Event Processing in Distributed Enterprise Systems*, 2002

[2] W. Liu, P. Miller, J. Ma, W. Yan, *Challenges of distributed intelligent surveillance system with heterogenous information*, 2009

[3] W. Hu, T. Tan, L. Wang, S. Maybank, *A survey on visual surveillance of object motion and behaviors*, IEEE Transactions on Systems, Man and Cybernetics, Volume 34, 2004

[4] S.S. Yau, N. Ye, H. Sarjoughian, D. Huang, *Developing Service based Software Systems with QoS Monitoring and Adaptation*, Proceeding of the 12th IEEE Int'l Workshop on Future Trends of Distributed Computing Systems, Honolulu, Hawaii, USA, October 2008

[5] S. Sutor, F. Matusek, F. Kruse, K. Kraus and R. Reda, *Large-Scale Video Surveillance Systems: New Performance Parameters and Metrics*, The Third International Conference on Internet Monitoring and Protection, Bucharest, July 2008

[6] F. Licandro, G. Schembra, *Wireless Mesh Networks to Support Video Surveillance: Architecture, Protocol, and Implementation Issues*, EURASIP Journal on Wireless Communications and Networking Hindawi Publishing Corporation, Volume 2007, Article ID 31976

[7] D. Mierzwinski, D. Walczak, M. Wolski, M. Wrzos, *Surveillance System in Service-Oriented Manner*, synasc, pp.427-433, 12th International Symposium on Symbolic and Numeric Algorithms for Scientific Computing, 2010

[8] M. Valera, S.A. Velastin, *Intelligent distributed surveillance systems: a review*, IEEE Proc.-Vis. Image Signal Process, April 2005

[9] A. Haoui. et al., *Wireless magnetic sensors for traffic surveillance*, Transport. Res. Part C (2007), doi:10.1016/j.trc.2007.10.004

[10] A. Rakusz, T. Lovas, A. Barsi, *LIDAR-Based Vehicle Segmentation*, International Conference on Photgrammetry and Remote Sensing, Istanbul, Turkey, 2004.

[11] J. Fang, H. Meng, H. Zhang, X. Wang, *A low-cost vehicle detection and classification system based on unmodulated continuos wave radar*, Proceedings of the 2007 ITSC, Seattle, WA, 2007

[12] R. López-Valcarce, C. Mosquera, F. Pérez-González, *Estimation of Road Vehicle Speed Using Two Omnidirectional Microphones: a Maximum Likelihood Approach*, EURASIP Journal of Applied Signal Processing, vol. 8 pp.1059-1077, 2004.

[13] G. Gritsch, M. Litzenberger, N. Donath, B. Kohn, *Real-Time Vehicle Classification using a Smart Embedded Device with a 'Silicon Retina' Optical Sensor*, Proceedings of the 11th International IEEE Conference on Intelligent Transportation Systems Beijing, China, October 12-15, 2008

[14] L. Vibha, M. Venkatesha, G.R. Prasanth, N. Suhas, P. Deepa Shenoy, K.R. Venugopal, L.M. Patnaik, *Moving Vehicle Identification using Background Registration Technique for Traffic Surveillance*, Proceedings of the International MultiConference of Engineers and Computer Scientists 2008 Vol I IMECS 2008, 19-21 March, 2008, Hong Kong

[15] L. Snidaro, G.L. Foresti, G. Luca, R. Niu, P.K. Varshney, *Sensor Fusion for Video Surveillance*, 7th Int. Conf. on Information Fusion, 2004

[16] M.P. Papazoglou, P. Traverso, S. Dustdar, F. Leymann, *Service-Oriented Computing: State of the Art and Research Challenges*, 64-71, November 2007

[17] D. Krafzig, K. Banke, D. Slama, *Enterprise SOA: Service-Oriented Architecture Best Practices (The Coad Series)*, Prentice Hall PTR Upper Saddle River, NJ, USA, 2004

[18] C. Legner, R. Heutschi, *SOA Adoption in Practice-Findings from Early SOA Implementations*, European Conference on Information Systems, p. 16431654, 2007

[19] C. Baroudi, F. Halper, *Executive Survey: SOA Implementation Satisfaction*, Hurwitz and Associates, 2006

[20] J. Alcober, G. Cabrera, X. Calvo, E. Eliasson, K. Groth, P. Pawalowski, *High Definition Videoconferencing: The Future of Collaboration in Healthcare and Education*, eChallenges e-2009 Conference, October 21-23 2009, Istanbul,

Turkey
[21] C. Mazurek, M. Stroiñski, D. Walczak, M. Wolski, *Supporting high-tech crime investigation through dynamic service integration*, Fifth International Conference on Networking and Services, 2009
[22] A. Bauer, S. Eckel, T. Emter, A. Laubenheimer, E. Monari, J. Mosgraber, F. Reinert, *N.E.S.T. - Network Enabled Surveillance and Tracking*, Future security: 3rd Security Research Conference Karlsruhe; 10th-11th September 2008
[23] A. Hampapur, L. Brown, J. Connell, A. Ekin, N. Haas, M. Lu, H. Merkl, S. Pankanti, A. Senior, C. Shu, and Y. Tian, *Smart Video Surveillance*, IEEE Signal Processing Magazine, march 2005
[24] The Open Group, *SOA Source Book*, Van Haren Publishing, 2009
[25] M. Pellegrini, P. Tonani, *Highway traffic monitoring: Main problems and current solutions*, Advanced video-based surveillance systems, 27-33, 1999

*Edited by:* Dana Petcu and Alex Galis
*Received:* March 1, 2011
*Accepted:* March 31, 2011

Scalable Computing: Practice and Experience
Volume 12, Number 1, pp. 105–120. http://www.scpe.org
ISSN 1895-1767


# TRUSTED BELIEFS FOR HELPFUL BEHAVIOR WHEN BUILDING WEB SERVICES *

IOAN ALFRED LEŢIA, RADU RĂZVAN SLĂVESCU †

**Abstract.** Composite software services often present uncertainty over their non-functional properties. To tackle this, one could model them as shared goals of an agent team which aims at maximizing the likelihood of success of the joint task. An architect is in charge with picking services and providers, while a consultant helps him, when possible, by suggesting alternative approaches. The multinomial version of the "Belief Recipe Tree" structure relies on beliefs built based upon prior mutual experiences of the consultant with various providers and/or abstract plans and revised after each interaction, exhibiting a higher flexibility in several web service building scenarios.

**Key words:** collaborative behavior, multi-agent systems, web services, trust, beliefs, subjective probability, agent teams

**AMS subject classifications.** 68T37, 68T42

**1. Introduction.** In the last decade, the development of web applications which employ e-services as basic blocks for deploying a complex functionality gained much momentum. Within this context, the paradigm of web services (WS) makes generous promises regarding the development of flexible and highly customized applications. Some particular tasks do find their solution in an existing web service, but this is the exception rather than the rule, as it is highly unlikely that one can find an already built WS able to solve her specific problem in each and every detail. In the vast majority of cases, matching the user's specific requirements demands designing a composite web service. Most of the time, this is made of simpler web services offered by a bunch of different providers. Building such a WS (i.e. a composite web service), demands solving at least two issues. On one hand, the developer must decompose the complex goal task into simpler ones, up to the basic level, where each of them can be solved by an existing WS. On the other hand, every basic task must be assigned to a concrete WS provider which must be selected among more providers of this type. Both issues requires considering functional as well as non-functional criteria. Hence, the system requires producing the intended output, but also guaranteeing some quality parameters like response time, reliability or cost.

A possible approach for this problem has been proposed, for example, in [3]. The cited paper describes a workflow based solution which takes into account the Quality of Service (QoS) of each component. A composite web service is shown to be reducible to a sequential flow whose QoS parameters are easy to analyze and predict. To achieve this, the system designer will need accurate data about the non-functional aspects of web services we mentioned above. The intrinsic volatility of such information makes it potentially unreliable though, as the values might change rapidly and alter the subsequent decisions. In the same time, the available information is inherently incomplete. This, combined with the possibility of having inconsistencies in the knowledge about the decomposition recipes lead to the conclusion that uncertainty must be accommodated when assembling web services.

Withing this context, developing a composite WS with appropriate reliability becomes a challenging task. To deal with it, one might regard this problem as the process of a joint goal being solved by a set of cooperative agents, and among them, WS architects and consultants. Typically, a consultant is employed by a WS architect in order to provide him/her with guidelines about designing a composite WS. This involves delivering both an abstract plan and a concrete list of WS providers for each plan step in order to maximize the likelihood of the whole WS to become up and running. Both agents above are interested in having this goal achieved, but the type of decisions they can make are quite different. The architect will be focused on choosing each service and its provider. The consultant will have to decide whether to adopt a helpful behavior towards her peer or to refrain from it. This behavior consists of suggesting his partner an alternative. The consultant must do this such that the global likelihood of success is maximized, whilst not providing its partner with more information than he/she paid for.

We argue the aforementioned issues can be solved by taking trust into consideration when an agent starts building composite web services. In order to prove this, we presented in [15] a possible extension of the

*Part of this work was supported by CNCSIS-UEFISCSU, National Research Council of the Romanian Ministry for Education, Research, Youth and Sport, under project ID_170/2009.

†Department of Computer Science, Technical University of Cluj-Napoca, Bariţiu 28, RO-400027 Cluj-Napoca, Romania ( {letia,srazvan}@cs-gw.utcluj.ro)

Probabilistic Recipe Tree (PRT) to a structure we called Belief Recipe Tree (BRT) able to accommodate trust concerning web services and to allow making decisions based on this. Trust integrates a historical sequence of experiences into a single value and measures the confidence one should have in a service to meet its advertised non-functional parameters (e.g. the likelihood of success). We show how to develop and employ the BRT structure, which is actually a belief-based extension of the "Probabilistic Recipe Tree" in [14], for making the above type of decisions. Then, we suggest a way for building the beliefs an agent needs for cooperation over the WS design, based on the set of experiences the consultant has had with different providers and/or design plans. Since the beliefs are revised after each experience, it follows that the flexibility offered is higher due to the fact the providers which are not always truthful can be punished and the system can adjust its beliefs, as well as its subsequent decisions, accordingly.

The present paper elaborates on the idea by extending the BRT to multinomial opinions. This aims allowing agent to deal with a finer grained set of mutually exclusive sentences describing an agent's performance, like "The level of success is $poor/average/good$ as opposite to a mere $failure/success$, in order to allow a more realistic model of the domain. The main contribution made is to extend the operations on BRT to the multinomial case. First, we revisit the structure of PRT and BRT and present some examples in more details. Then, we introduce the multinomial versions of the BRT operations. Finally, we present an example to illustrate the type of decisions which are allowed by the enriched BRT.

The rest of this paper is organized in the following way. Section 2 introduces the basic blocks of the Subjective Logic which are further needed for building and manipulating the BRT. Section 3 details the structure, components and operations on BRT, starting from a brief presentation of its purely probabilistic counterpart PRT. Section 4 illustrates in detail the ideas in a scenario of building a composite web service which generates car routes. The resulting web service ought to take into consideration both map details as well as traffic information and weather forcast when generating an autoroute. The ideas are then extended to multinomial opinions. Section 5 positions the approach among some other works in the field, while Section 6 concludes and sketches future lines of investigation.

**2. Subjective Logic based Trust.** Since uncertainty of sentences describing the world is inherent, several formalisms were developed in order to deal wit it. One such formalism is the Subjective Logic (SL) theory, introduced in [9] and extensively updated in [10]. This logic could be seen as a superset of the classical probability theory because, as its parameters approach some limit values, the theory is reduced to the classical one. SL deals with beliefs and includes operations on sentences similar to those in the Dempster-Shafer theory of beliefs [5, 22]. In the same time, it is compatible with Nilsson's probabilistic logic [18]. The present section briefly presents the ideas of SL we used for trust estimation and web service selection. For further details on the topic of SL, we refer to [10].

**2.1. Subjective Logic Opinions.** SL argues that a perfectly accurate assessment of probability is beyond the scope of human nature. Therefore, uncertainty involved in estimating the values of probabilities must be considered and assessed. The theory's basic block is the concept of *opinion* over a sentence. Given a proposition $x$, the opinion $\omega_x$ regarding the truth value of $x$ is defined as a quadruple $\omega_x = (b_x, d_x, u_x, a_x)$. Its components represent the degree of belief (evidence supporting $x$), disbelief (evidence supporting $\neg x$) and uncertainty about the truth of $x$. By definition, they must sum up to 1. The atomicity $a_x$ is a measure of the *prior* probability of the truth value of $x$. Within this subsection, we will focus on a universe of discourse comprising only binomial, mutually exclusive sentences ($x$ and $\neg x$ respectively), thus $a_x$ defaults to 0.5. In the next subsection, we will briefly describe the multinomial case, where there are more than 2 possible, mutually exclusive values for a sentence.

For example, one can assign a value of 0.7 to the belief corresponding to the sentence "WS will succeed", 0.2 to the disbelief corresponding to the sentence "WS will succeed" (i.e. the belief corresponding to the sentence "WS will fail") and the remaining 0.1 to the uncertainty about the behavior of the WS. This one is due to the lack of perfect knowledge over the behavior of the given WS and can be seen as second order probability (modeling uncertainty over a first-order probabilities).

Given $\omega_x = (b_x, d_x, u_x, a_x)$, the corresponding probability expectation value (a generalization of a classical probability expectation), is defined by the formula below:

$$E(\omega_x) = b_x + a_x u_x \tag{2.1}$$

This definition corresponds to that of pignistic probability in [23] and is consistent with the principle

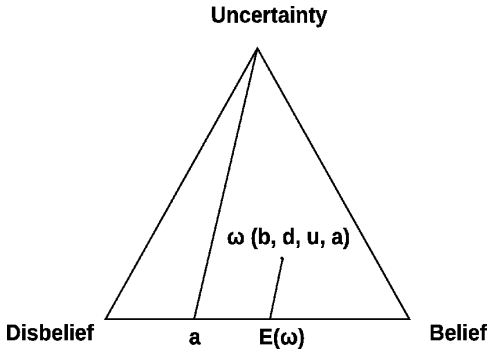



FIG. 2.1. *A Subjective Logic opinion inside the opinion triangle*

of equally dividing frame belief among its singletons. The interpretation of $a_x$ is the relative proportion of singletons of $x$.

As shown in [10], an opinion coresponds to a point inside a triangle. Figure 2.1 illustrates this, displaying an opinion $\omega(b, d, u, a)$, the base rate $a$, the vertexes of maximum belief, disbelief and uncertainty and the probability expectation value $E(\omega)$.

We regard such an opinion over a sentence as the subjective trust hold by the issuing agent towards the object of that sentence. As an example, we may consider $x$ to be the sentence above, assumed to be believed by the architect agent: "Web service $WS_1$ offered by provider $P_1$ will perform according to its parameters". In this case, $\omega_x = (0.7, 0.2, 0.1, 0.5)$ means the agent issuing sentence $x$ (i.e. the system architect) beliefs the sentence to a degree of 0.7; its negation $\neg x$ to a degree of 0.2 and has an uncertainty degree of 0.1 about it, possibly because of the lack of complete evidence. The value of $a$ is 0.5 as, so far, we have considered only a binary universe of discourse for each sentence. In this example, $\omega_x$ models the trust of the entity issuing sentence $x$ towards $WS_1$. Trust could also be concerned with sentences describing generic solutions like "In order to obtain a WS of type $T$, one should combine a WS of type $T_a$ and one of type $T_b$", or "Solution $S_1$ for the problem $P$ is preferred by 55% of the developers".

**2.2. Mapping Experiences into Opinions.** Let us suppose one has already had a number of experiences with a binary event, e.g. a WS behaving with the advertised latency or not. Let us assume there were $r$ positive (i.e. expected behavior from the WS side) and $s$ negative (i.e. poor) experiences. Posterior probabilities of binary events can be represented by a family of probability density functions, namely the Beta distribution:

$$\begin{aligned} Beta(p|\alpha, \beta) &= \frac{\Gamma(\alpha+\beta)}{\Gamma(\alpha)\Gamma(\beta)} p^{\alpha-1}(1-p)^{\beta-1} \\ \alpha &= r + 2a \\ \beta &= s + 2(1-a) \\ \Gamma(z) &= \int_0^\infty t^{z-1} e^{-t} dt \end{aligned} \tag{2.2}$$

with the restrictions $0 < a < 1$, $p \neq 0$ if $\alpha < 1$ and $p \neq 0$ if $\beta < 1$.

One possible interpretation for this is how likely the outcome of a future experience will have a given value (e.g. positive) and how much uncertainty exists within this prediction, given that $r$ positive and $s$ negative experiences have been recorded so far.

As showed in [10], there exists a function mapping the evidence space for a sentence (i.e. prior observations over its truth) into opinion space. Based on this correspondence, sentence opinions can be synthesized out of sequences of observations over those sentences. Opinions are further combined according to specific operators to be defined later. The resulting algebra of opinions is equivalent, both from the point of view of semantics and expressiveness, with the distributions, but it has the advantage of being much more efficient from the computational perspective.

If for the sentence $x$, we have $r$ experiences supporting $x$ and $s$ supporting $\neg x$, then the opinion's compents $b$, $d$ and $u$ are computed in the following way:

$$b = \frac{r}{r+s+2} \tag{2.3}$$

$$d = \frac{s}{r+s+2} \tag{2.4}$$

$$u = \frac{2}{r+s+2} \tag{2.5}$$

Some limit cases can be considered: $b = 1$ means logical $TRUE$ (probability 1), $d = 1$ means logical $FALSE$ (probability 0), $u = 1$ means vacuous opinion (total uncertainty, absolute lack of experiences) and $b + d = 1$ gives the classical probability (no uncertainty). One should notice the latter case happens if the number of experiences is infinite. This property will be uses further in this paper, when our extended formalism reduces to the original one when the number of experiences approaches $\infty$.

One should notice that, from this perspective, a set of 3 positive and 1 negative outcomes for an experience is different from a set of 30 positive and 10 negative experiences. Though in both cases the probability of a positive experience is 75%, the opinions are $(0.500, 0.166, 0.334, 0.5)$ and $(0.714, 0.238, 0.05, 0.5)$, so the uncertainty over the expected experiment value decreases.

**2.3. Subjective Logic Operators.** In order to combine opinions on propositions, SL introduces a set of operators over opinions, whose semantics relies on probability distributions as well. The following two SL operators are used in this work:

1. conjunction $\wedge$: given $\omega_x = (b_x, d_x, u_x, a_x)$ and $\omega_y = (b_y, d_y, u_y, a_y)$, their conjunction $\omega_{x\wedge y}$, written $\omega_x \sqcap \omega_y$, has the following components:

$$b_{x\wedge y} = b_x b_y + \frac{(1-a_x)a_y b_x u_y + a_x(1-a_y)u_x b_y}{1-a_x a_y} \tag{2.6}$$

$$d_{x\wedge y} = d_x + d_y - d_x d_y \tag{2.7}$$

$$u_{x\wedge y} = u_x u_y + \frac{(1-a_y)b_x u_y + (1-a_x)u_x b_y}{1-a_x a_y} \tag{2.8}$$

$$a_{x\wedge y} = a_x a_y \tag{2.9}$$

E.g. if the solution for designing a composite WS is expressed in BPEL4WS as $sequence(RG, WF)$, and we have opinions concerning reliability $\omega_{RG} = (0.7, 0.2, 0.1, 0.5)$ and $\omega_{WF} = (0.6, 0.3, 0.1, 0.5)$ respectively, then the sentence describing the success likelihood of the composite WS would be $\omega_{RGWF} = (0.4, 0.51, 0.09, 0.25)$.

2. disjunction $\vee$: given $\omega_x = (b_x, d_x, u_x, a_x)$ and $\omega_y = (b_y, d_y, u_y, a_y)$, their disjunction $\omega_x \vee \omega_y$, written $\omega_x \sqcup \omega_y$, has the following components:

$$b_{x\vee y} = b_x + b_y - b_x b_y \tag{2.10}$$

$$d_{x\vee y} = d_x d_y + \frac{a_x(1-a_y)d_x u_y + (1-a_x)a_y u_x d_y}{a_x + a_y - a_x a_y} \tag{2.11}$$

$$u_{x\vee y} = u_x u_y + \frac{a_y d_x u_y + a_x u_x d_y}{a_x + a_y - a_x a_y} \tag{2.12}$$

$$a_{x\vee y} = a_x + a_y - a_x a_y \tag{2.13}$$

E.g. if the solution for designing a WS called $RG$ requires choosing either service $RG_1$ or service $RG_2$, with reliability $\omega_{RG_1} = (0.7, 0.2, 0.1, 0.5)$ and $\omega_{RG_2} = (0.8, 0.1, 0.1, 0.5)$ respectively, then the sentence describing the success likelihood of WS would be $\omega_{RG} = (0.92, 0.043, 0.037, 0.75)$.

The definitions of the conjunction and disjunction operators we have just presented were first introduced in the paper [11]. Here, they are called normal binomial multiplication and comultiplication and denoted by the operators $\cdot$ and $\sqcup$ respectively. The two operations are shown to be very good approximations of the analytically correct operators applied to the Beta probability density functions. Whilst the exact operations quickly become unmanageable, their SL counterparts preserve the simplicity thus allowing analyzing complex models [10]. The outcome, albeit approximate, will approach the exact results as the uncertainty decreases.

**2.4. Multinomial Opinions.** If we consider $k$ possible, mutually exclusive values for a sentence, e.g. the reliability level could be $X = poor, average, good$ ($k = 3$), we need to shift to multinomial opinions. Such an opinion would be $\omega_X = (\vec{b_X}, u_X, \vec{a_X})$, with $u + \sum_{i=1}^{k} \vec{b_{x_i}} = 1$ and $\sum_{i=1}^{k} \vec{a_{x_i}} = 1$. The vector contains $k$ values for the belief levels corresponding to each possible value of the sentence, one value for the level of uncertainty and $k$ values for atomicity. We will write $b(x_i)$ to denote $\vec{b_{x_i}}$.

The components of the corresponding probability expectation vector are defined by:

$$E(x_i) = b(x_i) + a(x_i)u \tag{2.14}$$

To illustrate, let us suppose we have the sentences "Performance of $WS_1$ is bad", "Performance of $WS_1$ is average", "Performance of $WS_1$ is good", the opinion might be, for example, $\omega_x = (0.2, 0.2, 0.5, 0.1, 0.3, 0.4.0.3)$ meaning the the system architect beliefs the performance to be *bad* to a degree of 0.2, *average* to a degree of 0.2, *good* to a degree 0.5 and has an uncertainty degree of 0.1 The probability expectation vector would then be $(0.23, 0.24, 0.53)$.

Posterior probabilities of this type of events can be represented by another family of probability density functions, namely the Dirichlet distribution instead of the Beta distribution, but the approach is similar. For the sentence $x$, if we have the experience vector $(\vec{r}, \vec{a})$, mapping to opinions can be made by:

$$b(x_i) = \frac{r(x_i)}{2 + \sum_{i=1}^{k} r(x_i)} \tag{2.15}$$

$$u = \frac{2}{2 + \sum_{i=1}^{k} r(x_i)} \tag{2.16}$$

Next, we introduce the definition of multinomial multiplication, which extends the conjunction operation to the multinomial case. This definition is introduced in [12] as the Assumed Uncertainty Mass method. Given two sets of multinomial opinions $X$ and $Y$, the method for computing their conjunction relies on generating all the elements of the Cartesian product $X \times Y$ and redistributing some of the belief mass on the rows and columns to singleton elements and to the global set $X \times Y$. This is performed such that the expected value of each singleton in the Cartesian product will be equal to the product of the expected values of its components.

Let us assume we have two multinomial opinions $\omega_X(\vec{b_X}, u_X, \vec{a_X})$ having possible values $x_1, \ldots, x_k$ and $\omega_Y = (\vec{b_y}, u, \vec{a_y})$ with possible values $y_1, \ldots, y_l$. Computing the conjunction $\omega_x \sqcap \omega_y$ is based on the formulas below.

First, we compute the expected utilities of the singletons in $X \times Y$:

$$E((x_i, y_j)) = (b(x_i) + a(x_i)u_X)(b(y_j) + a(y_j)u_Y) \tag{2.17}$$

Then we estimate an intermediate uncertainty:

$$u^I_{X\times Y} = u^{Rows}_{X\times Y} + u^{Columns}_{X\times Y} + u^{Frame}_{X\times Y} \tag{2.18}$$

$$u^{Rows}_{X\times Y} = 1 - \sum b^{Rows}_{X\times Y} \tag{2.19}$$

$$b^{Rows}_{X\times Y} = \begin{pmatrix} u_X b(y_1) & \ldots & u_X b(y_l) \end{pmatrix} \tag{2.20}$$

TABLE 2.1
*Multinomial opinion multiplication*

| | belief | | | atomicity | | | Expected value | | |
|---|---|---|---|---|---|---|---|---|---|
| | *poor* | *avg* | *good* | *poor* | *avg* | *good* | *poor* | *avg* | *good* |
| *success* | 0.026 | 0.038 | 0.099 | 0.060 | 0.150 | 0.090 | 0.059 | 0.119 | 0.147 |
| $failure$ | 0.046 | 0.058 | 0.192 | 0.140 | 0.350 | 0.210 | 0.122 | 0.248 | 0.306 |

$$u_{X\times Y}^{Columns} = 1 - \sum b_{X\times Y}^{Columns} \tag{2.21}$$

$$b_{X\times Y}^{Columns} = \begin{pmatrix} b(x_1)u_Y & \dots & b(x_k)u_Y \end{pmatrix} \tag{2.22}$$

$$u_{X\times Y}^{Frame} = u_X u_Y \tag{2.23}$$

Based on this, we find the product uncertainty:

$$u_{X\times Y} = min\left\{u_{X\times Y}^{(i,j)}, (x_i, y_j) \in X \times Y\right\} \tag{2.24}$$

$$u_{X\times Y}^{(i,j)} = \frac{u_{X\times Y}^{I} E((x_i, y_j))}{b_{X\times Y}^{I}((x_i, y_j)) + a(x_i)a(y_j)u_{X\times Y}^{I}} \tag{2.25}$$

Beliefs and atomicity levels are then computed:

$$b_{X\times Y}((x_i, y_j)) = E((x_i, y_j)) - a_X(x_i)a_Y(y_j)u_{X\times Y} \tag{2.26}$$

$$a_{X\times Y}((x_i, y_j)) = \frac{E((x_i, y_j)) - b((x_i, y_j))}{u_{X\times Y}((x_i, y_j))} \tag{2.27}$$

Given the multinomial opinion multiplication, its disjunctive counterpart is computed in a similar way, taking into consideration that:

$$E((x_i, y_j)) = E(x_i) + E(y_j) - E(x_i)E(y_j) \tag{2.28}$$

As an example, we might consider the following situation. The consultant considers using a WS of type $T$ for a complex WS. There were 4 prior experiences labeled as *success* and 2 labeled as $failure$ for this design solution. For the moment, the a priori probabilities are assumed known; section 6 discusses how they can be obtained. Thus, for this type of WS, we may have the opinion $\omega_X = (0.250, 0.500, 0.250, 0.300, 0.700)$. On the other hand, we have one WS provider which, in the past, has offered a latency considered *poor* in 5, *average* in 10 and *good* in 13 cases, hence the opinion $\omega_Y = (0.167, 0.333, 0.433, 0.067, 0.200, 0.500, 0.300)$. We would like to know the belief level for the sentence "The design solution based on a WS of type $T$ is *success* and the latency is *good*". Table 2.1 summarizes the belief and atomicity levels for this situations. As we can see, the belief and atomicity for the sentence above are 0.192 and 0.210 respectively, for an uncertainty level of 0.541 and an expected value of 0.306.

**3. Belief Recipe Trees.** This section presents the Belief Recipe Tree (BRT), an extension of the Probabilistic Recipe Trees (PRT) introduced in [14]. Most of this section, as well as the next one, are borrowed from [15]. Some examples have been added and the multinomial case has been investigated and compared with the previous solution.

The central idea we advocate is that building a complex web service could be regarded as building up a plan, so we may use this structure for web service architecture elaboration. As a general approach, an architect is in charge with picking services and providers, while a consultant helps him, when possible, by suggesting alternative approaches, provided such a decision is rational from the point of view of the total utility.

**3.1. Probabilistic Recipe Trees.** This subsection briefly describes the PRT for self-containment purposes. For details and formal definitions, we refer to [14].

A PRT for an action $\alpha$ is basically a tree which incorporates a probability distribution over the recipes for accomplishing $\alpha$. A node in a PRT represents an action together with some associated properties: a leaf node represents a basic (i.e. atomic) action, while an intermediate node represents a complex one. Intermediate nodes are either AND or OR nodes. Each child of an AND node represents an action component of a recipe for completing the AND node action. Each child of an OR node represents a non-deterministic choice of a recipe for competing the OR node action. Each branch which descends from an OR node has associated a probability (which is assumed known a priori) for the corresponding child node to be selected as a recipe for completing the OR node action.

The context of an action, denoted $C_\alpha$, is the complex of information an agent bases her decision on at a specific moment of time. The predicate $Context(C_\alpha, G_1, \alpha, T)$ is used to express the idea that $C_\alpha$ is the context in which agent $G_1$ believes, at time $T$, that action $\alpha$ is done. Function $cba.basic(G_1, \beta, C_\beta)$ returns the probability that agent $G_1$ can bring about the basic level action $\beta$ within context $C_\beta$. Similarly, $cba.cost(G_1, G_2, \beta, C_\beta)$ returns the cost paid by agent $G_1$ when the basic level action $\beta$ is done by agent $G_2$ in context $C_\beta$. Function $V(G_1, \alpha, C_\alpha)$ returns the utility for $G_1$ if action $\alpha$ is performed in context $C_\alpha$ and includes, for complex actions, both the gain for the action itself and for its sub-actions. All *cba*, *cost* and $V$ are considered intrinsic properties of actions, known by all agents, although obtaining good estimations of them is not trivial for systems comprising more agents and little prior interactions.

Function $p\_CBA(PRT_\alpha, C_\alpha)$ returns the probability of action $\alpha$ to succeed, given context $C_\alpha$ and the tree $PRT_\alpha$. For leaf nodes, the returned value is *cba.basic*, for AND nodes, a product of children probabilities, while for an OR node, weighted average of children probabilities. Finally, function $Cost(G_i, PRT_\alpha, C_\alpha)$ returns the expected cost to be paid by agent $G_i$ when the group carries out the recipes in $PRT_\alpha$ in context $C_\alpha$. The returned value are as follows: *cost.basic* in case of leaf nodes, the sum of children costs for AND-type nodes and a weighted average of children costs for OR nodes.

An agent is assumed to be able to perform helpfully, conveying and asking actions. A helpful action requires the agent to be committed (believes $PRT_\alpha$ is the best way and all other agents intend to carry out $PRT_\alpha$) and to consider that performing $\gamma$ would increase the group utility. Conveying information action requires the agent to be committed and to believe that the conveyed information would increase the group utility, provided that there is some agent which may perform an action based on the received information. The asking information action can be done if there is another agent committed and he believes that he possesses information which would increase the group utility.

**3.2. Opinion based PRT: Belief Recipe Trees.** A BRT deals with opinions instead of mere probabilities. The rationale behind extending the PRT structure into BRT is the necessity of endowing the user with a mechanism of generating and adjusting the probability values needed by the tree. In the same time, the BRT "converges" towards the PRT when the number of experiences increases.

The BRT structure basically serves for representing in a compact manner the alternative recipes which might be used for achieving a goal (e.g. for building a composite web service). The size of such a structure is showed to be $\mathcal{O}(nm)^d$, where $n$ is the number of potential recipes for every action, $m$ is the average number of steps in one such action and $d$ is the number of levels of decomposition required to reach the atomic level of WS. This makes it exponentially smaller than the trivial $\mathcal{O}(n^{m^d})$ [14].

A BRT comprises terminal nodes, which model atomic tasks, and non-terminal nodes which represents composed tasks. Each node has attached the following information:

(i) an opinion describing the trust level the evaluator has in the success of the task represented by that node

(ii) a cost, which estimates how much the plan effector must pay in order to have the task corresponding to the node completed

(iii) an income modeling the benefit obtained if the task is accomplished.

The opinions attached to a terminal node estimate the success likelihood of that task, based on its own prior accomplishments; the opinions in the non-terminal nodes are computed based on those of their descendants. The costs for a non-terminal node are computed based on those of descendants and on the likelihood of a branch of being selected, while the income incorporates the benefits for achieving the tasks both in the descendants and in the node itself.

We consider a recipe as a sequence of steps to be performed in order to accomplish a goal. There exists a set of basic tasks which are the atomic constituents of every such recipe. They are described by the properties $cost_N$ and $income_N$, representing the cost and the reward respectively for completing the recipe step at the level of the node $N$. These are assumed common and known across the community. Unlike the original approach, the property "can bring about" (*cba*, the success likelihood for a specific service) is modeled as an SL *opinion* owned by the evaluator.

The BRT contains the following types of nodes:

1. leaf nodes: they are atomic tasks. For our case, such an task is selecting a specific atomic web service.
2. AND nodes: each child of an AND node must contain a constituent of the join task in the node. It is used to model the situation of a complex recipe consisting of two or more goals which must be achieved to fullfill the whole goal; for our case, it models a composed web service.
3. OR nodes represent possible alternatives for achieving a specific goal, e.g. making a nondeterministic choice of one alternative over the other in case one need a web service of type $T$ and two concrete web services of that type, $Tws_1$ and $Tws_2$ are available but just one needs to be chosen.

Opinion-based BRT functions are introduced and serve for modeling helpful behavior in the very same way their PRT counterpart do; each of these functions works on the goal in the BRT's root and evaluates the most appropriate recipe for achieving it.

Each node in the BRT has associated a specific opinion (as opposite to a mere probability), which represents its likelihood of being successful. For leaf nodes (corresponding to atomic WS) the *cba* property is estimated based on $r$ and $s$. Each time an agent, either architect or consultant, employs a specific WS, it logs the performance of that WS, labeling it as either *positive* or *negative*, thus increasing $r$ or $s$ respectively. We assume the existence of an underpinning WS taxonomy, which classifies each WS given by its URI into a class according to its functionality. The performance of a specific WS will be recorded in conjunction with its corresponding functionality class. For example, the *http://www.webservicex.net/globalweather.asmx?wsdl* web service will have its performances logged as a $WeatherForcastWS$ class member. Later on, if a WS in the $WeatherForcastWS$ class is needed, *http://www.webservicex.net/ globalweather.asmx?wsdl* will be considered for selection. The opinion for a node at this tree level is then updated based on $r$ and $s$:

$$\omega_N = (\frac{r}{r+s+2}, \frac{s}{r+s+2}, \frac{2}{r+s+2}, a) \tag{3.1}$$

For a non-terminal node $N$, the opinion $\omega_N$ is computed based on those of the node descendants:

1. AND node: if $D_N$ is the subset of all direct descendants of an AND node $N$, $\omega_s$ is the opinion in a direct descendant $s$ of $N$ (where $s \in D_N$), then the opinion $\omega_N$ in the node $N$ is:

$$\omega_N = \bigsqcap_{s \in D_N} \omega_s \tag{3.2}$$

It gives the likelihood of $N$ being successful based on the success of every subtree of it.

2. OR node: if $D_N$ is the subset of all direct descendants of an OR node $N$, $\omega_s$ is the opinion in a direct descendant $s$ of $N$ (where $s \in D_N$), $\omega_{b(s)}$ is the opinion associated to the tree branch going from $N$ to $s$, then the opinion $\omega_N$ is:

$$\omega_N = \bigsqcup_{s \in D_N} \omega_{b(s)} \sqcap \omega_s \tag{3.3}$$

This opinion describes the likelihood of $N$ being successful taking into consideration the potential of success of each subtree alone. One should note that, for an OR node, each descending branch (not just the nodes) is endowed with an opinion. These branch opinions are aimed to model information of the type "for the job $X$, 70% of the specialists would recommend solution $Y$", thus expressing knowledge about recipes rather than about particular executors. They are manipulated like any other BRT opinion.

Costs of each node are computed as below:

1. leaf node $N$: $cost_N$ is assumed known and given
2. AND node: if $D_N$ is the subset of all direct descendants of an AND node $N$:

$$cost_N = \sum_{s \in D_N} cost_s \tag{3.4}$$

3. OR node: if $D_N$ is the subset of all direct descendants of an OR node $N$, $\omega_{b(s)}$ is the opinion associated to the tree branch going from $N$ to $s$ and $EV(\omega_{b(s)})$ its expected value:

$$cost_N = \sum_{s \in D_N} cost_s * EV(\omega_{b(s)}) \tag{3.5}$$

Expected utilities of each node describe its profit ($income - cost$), weighted by the likelihood of this being achieved. They and are computed as follows:

1. leaf node $N$: if $income(N)$ is the price the customer agreed to pay for the delivery of $N$ (this is assumed to be known by agents), then the profit will be:

$$eval(N) = income(N) - cost(N) \tag{3.6}$$

2. AND node: if $D_N$ is the subset of all direct descendants of an AND node $N$, $income(N)$ is the price the customer agreed to pay for the delivery of $N$, $\omega_N$ is the opinion associated to node $N$ and $EV(\omega_N)$ its expected value:

$$eval(N) = EV(N) * income(N) + \sum_{s \in D_N} eval(s) \tag{3.7}$$

3. OR node: if $D_N$ is the subset of all direct descendants of an OR node $N$, $income(N)$ is the price the customer agreed to pay for the delivery of $N$, $\omega_N$ is the opinion associated to node $N$ and $EV(\omega_N)$ its corresponding expected value, $\omega_{b(s)}$ is the opinion associated to the tree branch going from $N$ to $s$ and $EV(\omega_{b(s)})$ its expected value:

$$eval(N) = EV(N) * income(N) + \sum_{s \in D_N} eval(s) * EV(\omega_{b(s)}) \tag{3.8}$$

**3.3. BRT for helpful behavior.** We used the BRT structure for making decisions in case of a cooperative activity where only partial team member involving is needed, though without risking to compromise the global team goal.

The kind of decision to be made is as follows: if a team comprising $A$ and $C$ aim at building a composite WS and $C$ has just gained some knowledge about new possible recipes for building a part of the system, is it rational, in team profit terms, for $C$ to inform $A$ about this or not? The following algorithm, similar to that in [14], is proposed for making such a decision.

The predicate $Committed(G_1, GR, \alpha)$ uses the belief recipe tree $BRT_\alpha$, which has $\alpha$ in its root, in order to select the design solution $\alpha$ over any other similar solution $\beta$ and commit to it. As an example, $\alpha$ might be a design goal like "Build a route generator WS". Formally, $G_1$ is committed to $\alpha$ iff $G_1$ believes that $BRT_\alpha$ maximizes the group's GR utility:

$$\exists BRT_\alpha \; BEL(G_1, \; \forall BRT_\beta \; BRT_\beta \neq BRT_\alpha \; \Rightarrow \\ Eval(BRT_\beta) \leq Eval(BRT_\alpha)) \; \wedge Int.Th(G_1, SelectedBRT(BRT_\alpha)) \tag{3.9}$$

If an agent is committed to $\alpha$ and believes that sending information $o$ to his/her partner will increase the group utility, he/she will do so (see Algorithm 3.3.1).

The next section presents an trace excerpt of this algorithm in order to decide over helpful behavior.

**Algorithm 3.3.1** Commitment decision

**if** $Committed(G_1, GR, \alpha)$ **then**
  $BRT_\alpha = PredictBRT(G_1, GR, \alpha, C_{GR})$
  $C_\beta = ContextUpdate(C_\beta, o)$
  $BRT_\beta = PredictBRT(G_1, G_2, \beta, C_\beta)$
  $BRT^o_\alpha = BRTReplace(BRT_\alpha, BRT_\beta)$
  $utility = Eval(BRT^o_\alpha) - Eval(BRT_\alpha)$
**end if**
**if** $utility \geq CommunicationCost(G_2)$ **then**
  $Int.To(G_1, Communicate(G_1, G_2, o))$
**end if**

**4. Running model.** The example in this section describes a simple scenario in which a team of agents make cooperative decisions about sharing knowledge using BRT for assessing the benefits of this.

Let us suppose we have two agents: a system architect $A$ and a consultant $C$ who intend to build a web service called *Navigator*. This web service should build routes between pairs of addresses for car drivers. When building such a route, the system takes into consideration information about the existing ways between start and destination points, as well as information concerning the traffic on various routes and about the weather forcast in that area. We will assume there exist simple web services offering each type of information above; they are respectively called $Mws_1, Mws_2, Mws_3$ (map web services), $Tws_1, Tws_2$ (traffic info web services) and $Wfws_1, Wfws_2$ (weather forcast services).

The whole system can be built as a combination of two web services: *Route Generator* and *Weather Forcast*. The former prepares routes taking into consideration both map and traffic information (thus having two service components: *Mapws* and *Trafficws*); the latter returns information concerning the weather in the region which will be visited. The customer agreed to pay money both for the partial components (e.g for the *Route Generator* part) as well as for the final delivery. Building the whole system will bring an income of \$40K; developing the *Route Generator* alone will bring \$30K, while the *Weather Forcast* part will get \$2K.

We assume both $A$ and $C$ know general recipes for building such a system, so their BRTs are similar; however, their knowledge about specific web services capable of carrying each atomic task might be different. We consider this assumption reasonable for software developers. A depiction of the common BRT is given in Figure 4.1. A square nodes represents an individual WS, while a circle corresponds to either an AND or an OR node.

Building the system can be seen as a collaborative task involving agents $A$ and $C$. Generating the solution recipe is equivalent to building a shared plan describing how each step is performed. Agent $A$ must therefore come up with a list specifying either a recipe or an atomic web service for each needed task (e.g. the *Route Generator* will be done by combining $Mws_2$ and $Tws_1$ into a *Simple Route Generator*, while $Wfws_1$ will do the job of *Weather Forcast*).

Consultant $C$ has just learned there might also exist a web service called *Complex Route Generator* which does the jobs of *Mapws* and *Trafficws* in one step; however, architect $A$ is not aware of this yet.

The problem consultant $C$ tries to solve is whether he/she should get involved into conveying this new piece of knowledge to $A$ or not. In order to achieve this, $C$ makes an assessment of the chances/costs for $A$ building the system, based on his own knowledge.

The agent which builds the plan has information about each atomic service above; this information is kept in form of the number of positive (success) and negative (failure) experiences the agent has recorded when exploring a specific service. Please note that, for the time being, every experience can have only 2 outputs: $success$ or $failure$. E.g. for the service $Mws_1$, the number of prior successes is 1 and of prior failures is also 1. Thus, according to Section 3, the opinion concerning $Mws_1$ is $(0.25, 0.25, 0.50, 0.50)$.

**4.1. Running sample.** This subsection traces step-by-step the above scenario; it presents the opinions at each level and also the decision to be made by the consulting agent $C$.

Table 4.1 summarizes the opinions of $C$ concerning different individual web services. For example, in the first line, one can see the information concerning $Mws_1$. The number of positive prior experiences $r$ was 1, the number of prior negative experiences $s$ was also 1, thus the opinion concerning $Mws_1$ is $(0.250, 0.250, 0.500, 0.500)$.

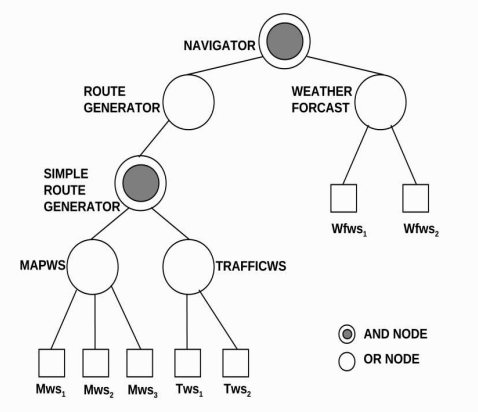



FIG. 4.1. *Web Service selection example*

TABLE 4.1
*Atomic WS evidence and opinions*

| **WS** | **r** | **s** | **b** | **d** | **u** | **a** |
|---|---|---|---|---|---|---|
| $Mws_1$ | 1 | 1 | 0.250 | 0.250 | 0.500 | 0.500 |
| $Mws_2$ | 3 | 1 | 0.500 | 0.167 | 0.333 | 0.500 |
| $Mws_3$ | 1 | 2 | 0.200 | 0.400 | 0.400 | 0.500 |
| $Tws_1$ | 5 | 1 | 0.625 | 0.125 | 0.250 | 0.500 |
| $Tws_2$ | 3 | 2 | 0.428 | 0.286 | 0.286 | 0.500 |
| $Wfws_1$ | 78 | 22 | 0.765 | 0.216 | 0.019 | 0.500 |
| $Wfws_2$ | 67 | 33 | 0.657 | 0.222 | 0.111 | 0.500 |

$C$ should also take into consideration the opinions for each branch. As presented in Section 3 the opinions concerning these options are used to model knowledge of the type "for the job $X$, 70% of the specialists would recommend solution $Y$", without specifying who should actually perform $Y$.

Considering the OR-type node *Mapws*, let us assume that the opinions for its descending branches $Mws_1$, $Mws_2$ and $Mws_3$ have the values $(0.200, 0.400, 0.400, 0.50)$, $(0.250, 0.250, 0.500, 0.50)$ and $(0.400, 0.200, 0.400, 0.50)$ respectively. Then, the opinion in the *Mapws* node is obtained by performing a logical OR among the logical ANDs from each branch, which leads to the result $(0.433, 0.288, 0.279, 0.578)$.

For the (AND-type) node *Simplerg*, the opinion in the node is computed as a simple logical AND between its descendants; the result would be $(0.345, 0.465, 0.190, 0.253)$.

Following the algorithm described above we get that the opinion for the *Navigator* task is $(0.227, 0.635, 0.138, 0.553)$.

Table 4.2 presents the opinions assumed for each branch and the opinions computed on this ground for every node in the BRT.

For the given costs and profits, we get $Eval(Navigator) = \$5{,}202.40$. Excerpts of this computation process are presented in Table 4.3, which contains the probability expectation $E$, cost, income and expected value $Eval$ for the nodes in the example tree.

Now agent $C$ might take into consideration the fact that he/she has just discovered another way of building the *Route Generator* part: there exists a new service called *Complex Route Generator* which does this job, having cost \$10,000. For the individual and branch opinions of $(0.666, 0.167, 0.167, 0.500)$ and $(0.333, 0.0, 0.667, 0.500)$ respectively, the new value for $Eval(Navigator)$ will be \$7,919.38, thus it worth telling $A$ about this, as long as the cost of communication is lower than the additional benefit. Figure 4.2 presents the new situation.

Let us suppose that the web service fails because the selected $CRG_1$ has failed twice. Then, the evidence and beliefs are adjusted to reflect this experience; their new values are those in Table 4.4.

In this case, the corresponding evaluations are \$5,202.40 (no composite web service disclosure) and \$4,180.70 respectively. In this case, the rational choice for $C$ would be not to suggest an alternative for the route generator part. One should notice this is not feasible in the classical approach when the values of probabilities remain the

TABLE 4.2
*Plan alternatives opinions*

| **WS** | **b** | **d** | **u** | **a** |
|---|---|---|---|---|
| *Mapws* to $Mws_1$ | 0.200 | 0.400 | 0.400 | 0.500 |
| *Mapws* to $Mws_2$ | 0.250 | 0.250 | 0.500 | 0.500 |
| *Mapws* to $Mws_3$ | 0.400 | 0.200 | 0.400 | 0.500 |
| Node *Mapws* | 0.433 | 0.288 | 0.279 | 0.578 |
| TRAFFICWS to $Tws_1$ | 0.600 | 0.200 | 0.200 | 0.500 |
| TRAFFICWS to $Tws_2$ | 0.375 | 0.375 | 0.250 | 0.500 |
| Node TRAFFICWS | 0.590 | 0.249 | 0.161 | 0.437 |
| Node *Simplerg* | 0.345 | 0.465 | 0.190 | 0.253 |
| RG to *Simplerg* | 0.666 | 0.167 | 0.167 | 0.500 |
| Node RG | 0.272 | 0.555 | 0.173 | 0.126 |
| $Wfws_1$ | 0.714 | 0.143 | 0.143 | 0.500 |
| $Wfws_2$ | 0.667 | 0.222 | 0.111 | 0.500 |
| Node *Wf* | 0.780 | 0.181 | 0.390 | 0.437 |
| Node NAV | 0.227 | 0.635 | 0.138 | 0.553 |

TABLE 4.3
*Costs and profits*

| WS | **E** | **Cost** | **Income** | **Eval** |
|---|---|---|---|---|
| $Mws_1$ | 0.500 | 1,000.00 | 0.00 | |
| $Mws_2$ | 0.666 | 2,000.00 | 0.00 | |
| $Mws_3$ | 0.400 | 3,000.00 | 0.00 | |
| *Mapws* | 0.594 | 3,033 | 0.00 | |
| $Tws_1$ | 0.500 | 4,000.00 | 0.00 | |
| $Tws_2$ | 0.666 | 5,000.00 | 0.00 | |
| *Route* | 0.523 | 0.00 | 30,000.00 | 2,170.80 |
| *Weather* | 0.608 | 0.00 | 2,000.00 | 950.86 |
| *Navigator* | 0.545 | 0.00 | 40,000.00 | 5,202.40 |

same and the selection could change due to $Selected\_PRT$ behavior only.

This example illustrates the advantages of a BRT over a PRT. A BRT allows building trust from prior mutual experiences and also updating it when further evidence becomes available, rather than keeping it unchanged. It also provides more flexibility as it takes knowledge uncertainty, due to the lack of enough evidence, into account when making decisions.

**4.2. PRT versus BRT and multinomial BRT.** In a scenario similar with the above one, we take into consideration 2 situations: in the first one, the first weather forcast WS has a performance record of $r = 3000, s = 1000$; in the second situation, the same WS has a performance record of $r = 3, s = 1$. The values for all other WS are unchanged. Since the ratio $r/s$ is the same, the corresponding PRT would be identical in the two situations. The BRT version would led to the results in Table 4.5.

If an uncertainty threshold is taken into consideration, e.g. 0.01, the BRT will lead to different decisions: in situation 1, the best is to commit to the weather forcast service, while in situation 2, this decision will be blocked and probably more evidence will be required. Considering uncertainty in a BRT offers the opportunity of making this type of decisions.

Table 4.6 shows the results in case more than 2 levels of quality are considered for the QoS. Here, we have a simple situation where only the *Route* and $Weather$, together with $Navigator$, are considered. The possible values for the $Weather$ WS performance are *good* and *bad*; *Route* can be *blue*, *silver* and *gold*. Let us consider the vector values: $(0.780, 0.181, 0.039, 0.500, 0.500)$ for $Weather$ and $(0.272, 0.025, 0.530, 0.173, 0.500, 0.400, 0.300)$ for *Route*. In this case, the expected value for the system to offer a *good* weather forcast AND a *gold* level route generator is 0.110. This offers the designer the opportunity to assess the system performance in a more meaningful manner, as opposite to a mere "does/does not work" in the previous version of the BRT.

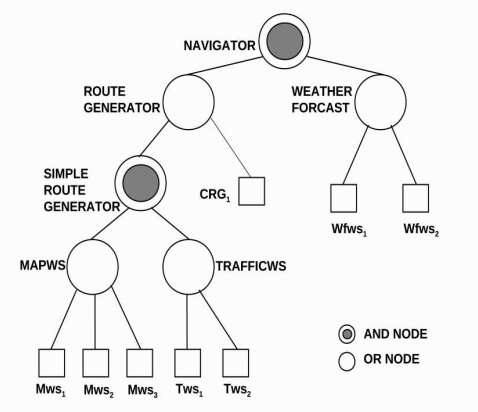



FIG. 4.2. *Adding an alternative plan*

TABLE 4.4
*Alternative atomic WS evidence and opinions*

| WS | **r** | **s** | **b** | **d** | **u** | **a** |
|---|---|---|---|---|---|---|
| $Mws_1$ | 1 | 1 | 0.250 | 0.250 | 0.500 | 0.500 |
| $Mws_2$ | 3 | 1 | 0.500 | 0.167 | 0.333 | 0.500 |
| $Mws_3$ | 1 | 2 | 0.200 | 0.400 | 0.400 | 0.500 |
| $Tws_1$ | 5 | 1 | 0.625 | 0.125 | 0.250 | 0.500 |
| $Tws_2$ | 3 | 8 | 0.428 | 0.286 | 0.286 | 0.500 |
| $Wfws_1$ | 78 | 22 | 0.765 | 0.216 | 0.019 | 0.500 |
| $Wfws_2$ | 67 | 33 | 0.657 | 0.222 | 0.111 | 0.500 |
| $CRG_1$ | 1 | 2 | 0.200 | 0.400 | 0.400 | 0.500 |

**5. Related work.** Within the context of knowledge potential uncertainty, combined with differences in goals and/or commitments, an agent can employ trust, seen as a "reduction of complexity" mechanism [17], inside the decision making process. Pressure for building realistic trust models comes from the area of delegation in multi-agent systems (selecting appropriate partners for tasks which require cooperation), as well as from the problem of security and quality of the knowledge available in large scale open environments (e.g. in the field of Semantic Web [1]). Thus, trust as an estimation of the likelihood of success of a specific task, was included among the ingredients involved in decision making and is present in many papers and research lines. Among them, we count the multi-agent systems and service-oriented architectures. Our work borrows ideas from both fields in order to tackle the problem of building web services.

In the area of multi-agent systems, the approach presented in [19, 20] aims to incorporate the well known Repage reputation model [21] into a BDI architecture and to model beliefs, desires and intentions as contexts connected by bridge rules, like in [6]. A Repage context is added, whose mission is to aggregate the information obtained from third party sources into sentence credibility level. A particular probabilistic logic is further employed in order to allow context manipulation; intentions are generated by an inference process conducted within this logic. However, the mechanism focuses on the perspective of a specific agent rather than committing to a shared goal as a part of a team.

The improved SharedPlan formalism in [8, 7] introduces a model for the dynamics of agent intentions in collaborative activity which integrates group decision making and group intentions updating. Agents can commit to joint activities and also can have only partial knowledge about how to perform an action, but the formalism does not address uncertainty aspects. In order to solve this issue, paper [14] introduced the Probabilistic Recipe Tree, which defined a probability distribution over the potential recipes which can solve the goal in its root node. The PRT structure is arguably more compact and works faster than the prior solutions (e.g. the mechanism described in [24]). However, the way the probabilities involved in reasoning are estimated and adjusted is not given. The solution our paper proposes for this is to use subjective probabilities. This will offer the opportunity of both estimating the probabilities those and modeling the uncertainty about these

TABLE 4.5
*BRT versus PRT decisions*

| Sit. | **r** | **s** | **b** | **d** | **u** |
|---|---|---|---|---|---|
| 1 | 3000 | 1000 | 0.749 | 0.250 | 0.001 |
| 2 | 3 | 1 | 0.5 | 0.167 | 0.333 |

TABLE 4.6
*Multinomial Navigator WS expected values and opinions*

| *Navigator* | belief | | | Expected value | | |
|---|---|---|---|---|---|---|
| | *blue* | *silver* | *gold* | *blue* | *silver* | *gold* |
| *bad* | 0.263 | 0.057 | 0.433 | 0.287 | 0.075 | 0.438 |
| *good* | 0.049 | 0.001 | 0.105 | 0.072 | 0.019 | 0.110 |

estimations as no large number of mutual prior interactions is usually available. Extending the PRT into the Belief Recipe Tree structure and adjusting all operations accordingly represent the main contributions of this paper.

Trust has also gained attention to the researchers in the service-oriented field. For example, paper [25] employs a novel trust model in order to solve the service collaboration problem. The model enjoys a set of desirable properties, such as temporal dynamics, context-awareness and the possibility of exchanging opinions concerning trustees. The service providers make simple decisions like granting and revoking resource access based on the trust level computed this way. The approach we presented allows reasoning about complex actins, i.e. actions whose completion involves more than one actions and one goal. Trust is involved at least at two levels: at the atomic level, by measuring one entity's own capability, and at the composed level, by assessing the global likelihood of success for the combined goal. The model fullfills the requirements of context-awareness, rule-orientation, non-symmetry, and temporal dynamics.

The paper [16] proposes a method for finding a possible reconfiguration region containing replaceable services which allow the system to meet the original QoS specification if a service fails and affects a complex of processes. However, the need of reasoning in advance over the likelihood of such an event lead to a more general approach including trust as a measure of service reliability. We took this latter path in this paper.

Paper [26] deals with incremental trust evaluation of WS providers, based on the feedback from the WS clients side. A fuzzy logic based method for connecting service period length and reputation is introduces. Our solution is based on incorporating knowledge on particular WS provider and on generic solutions into opinions along tree branches and leaves. The operations defined on BRT allow one to accommodate both types of knowledge.

A similar, but slightly different approach is introduced in [4]. Experiences are recorded objectively, based on a shared ontology aimed to allow describing past interactions in detail, filter the unreliable ones and then integrate them into global opinions. This offers the trustors a standard common language for recording their experiences, but makes no steps for supporting decisions which involves complex goals. We advocate the idea of grounding trust for complex actions on trust levels for basic actions because we consider rather unlikely to have a reasonable amount of experiences of each type of complex tasks in order to make a similarity-based judgment, so trustors will have to rely to a large extent on their own experiences, but being able to decide when to ask for more information.

The work presented in [27] addresses the problem of automatically composing web services within the context of uncertainty about their successfull invocations. Their solution is based on combining probabilistic situation calculus and hierarchical, symbolic planning and can deal with uncertainty and scale efficiently to large compositions. Our solution assumes a number of prior known recipes for hierarchically decomposing a task into subtasks up to the level of basic actions; this might be a weak point when compared to [27] as one cannot guarantee the completeness of such a collection of recipes. On the other hand, our approach offers flexibility on assessing the uncertainty involved in decisions and also on deciding whether the trustor does or does not know enough about a specific trustor in order to start reasoning over the interaction.

**6. Conclusions and future work.** Many times, the process of developing a complex WS implies collaborative design, hence the importance of deciding to which extent one should assist her peer. This paper

has presented a solution to this problem based on the novel BRT structure. A BRT is an extention of a PRT, endowed with trust values in form of subjective opinions describing both service providers and design recipes. Operations on the tree nodes are extended accordingly. The key advantage of it is that it permits estimating the likelihood of success and make rational decisions even if a small number of prior interactions are at hand, adjusting beliefs when further evidence is available, and deciding whether to seek for more information versus to rely on the available one. Agents are offered the possibility to deal with multinomial instead of binomial opinions, for an even richer (i.e. finer grained) set of labels for performance description. Belief Recipe Trees, together with the corresponding versions of algorithms, offer a more realistic approach for the uncertainty present in designing complex SOA, as well as an improved flexibility in making sound decisions.

The first goal for future work is to conduct extensive experiments in scenarios involving an increased number of team members, tasks and resources needed for accomplishing the tasks, in order to investigate the performance and scalability of the approach. This should be combined with different methods for distributing uncertainty (here, we used only the Assumed Uncertainty Mass for the conjunction and a De Morgan based difference for disjunction, but some other solutions should also be explored).

Addressing the problem of accurate estimation of the base rates also deserves attention. Following the line in [13], the base rate can be estimated by the general frequency. The main shortcoming would be that a great amount of observations is still required for high quality estimation. Another line to pursue originates in [2]. Machine learning algorithms can be used for clustering, then learning stereotypes about classes of agents. The learned stereotypes can be further used to estimate the base rates, alone or after aggregating them into stereotypical reputations. This line is more appropriate in agent groups whose life-span is much shorter than that of the whole system and the agent pool is large enough to render the frequency estimation unfeasible. Hopefully these improvements will lead to a more realistic world representation and better agent decisions.

**Acknowledgments.** We are grateful to the anonymous reviewers for the very useful comments.

REFERENCES

[1] G. Antoniou and F. van Harmelen, *A Semantic Web Primer (Second Edition)*, MIT Press, Cambridge, MA, 2008.
[2] C. Burnett, T. J. Norman, and K. Sycara, *Bootstrapping trust evaluations through stereotypes*, in Proceedings of The 9th International Conference on Autonomous Agents and Multiagent Systems, Toronto, Canada, 2010, pp. 241–248.
[3] J. Cardoso, J. Miller, A. Sheth, and J. Arnold, *Quality of service for workflows and web service processes*, Journal of Web Semantics, 1 (2004), pp. 281–308.
[4] M. Şensoy, J. Zhang, P. Yolum, and R. Cohen, *Poyraz: Context-aware service selection under deception*, Computational Intelligence, 25 (2009), pp. 335–366.
[5] A. P. Dempster, *Upper and lower probabilities induced by a multivalued mapping*, The Annals of Mathematical Statistics, 38 (1967), pp. 325–339.
[6] F. Giunchiglia and C. Ghidini, *A local models semantics for propositional attitudes*, in Formal Aspects of Context, P. Bonzon, M. Cavalcanti, and R. Nossum, eds., Kluwer Academic Publishers, Dordrecht, 2000, pp. 161–174.
[7] B. J. Grosz and L. Hunsberger, *The dynamics of intention in collaborative activity*, Cognitive Systems Research, 7 (2006), pp. 259 – 272.
[8] B. J. Grosz and S. Kraus, *Collaborative plans for complex group action*, Artificial Intelligence, 86 (1996), pp. 269–357.
[9] A. Jøsang, *A logic for uncertain probabilities*, International Journal of Uncertainty, Fuzziness and Knowledge-Based Systems, 9 (2001), pp. 279–311.
[10] A. Jøsang, *Subjective Logic*, Draft book, 2010.
[11] A. Jøsang and D. McAnally, *Multiplication and comultiplication of beliefs*, International Journal of Approximate Reasoning, 38 (2004), pp. 19–51.
[12] A. Jøsang and S. O'Hara, *Multiplication of multinomial subjective opinions*, in Proceedings of the 13th International Conference on Information Processing and Management of Uncertainty (IPMU2010), Dortmund, Germany, 2010, pp. 248–257.
[13] A. Jøsang, S. O'Hara, and K. O'Grady, *Base rates for belief functions*, in Proceedings of the Workshop on the Theory of Belief Functions, 2010.
[14] E. Kamar, Y. Gal, and B. J. Grosz, *Incorporating helpful behavior into collaborative planning*, in Proceedings of The 8th International Conference on Autonomous Agents and Multiagent Systems, Budapest, Hungary, 2009, pp. 875–882.
[15] I. A. Letia and R. R. Slavescu, *Helpful behavior based on trust for web services*, in Proceedings of the 12th International Symposium on Symbolic and Numeric Algorithms for Scientific Computing, Timisoara, Romania, 2010, pp. 526–533.
[16] K.-J. Lin, J. Zhang, Y. Zhai, and B. Xu, *The design and implementation of service process reconfiguration with end-to-end qos constraints in soa*, Service Oriented Computing and Applications, (2010), pp. 1–12.
[17] N. Luhmann, *Trust and Power*, John Wiley & Sons Ltd., 1979.
[18] T. J. Norman and C. Reed, *A model of delegation for multi-agent systems*, in Foundations and applications of Multi-Agent Systems, LNAI 2403, Springer-Verlag, 2002.
[19] I. Pinyol and J. Sabater-Mir, *Pragmatic-strategic reputation-based decisions in BDI agents*, in Proceedings of The 8th International Conference on Autonomous Agents and Multiagent Systems, Budapest, Hungary, 2009, pp. 1001–1008.

[20] I. Pinyol, J. Sabater-Mir, P. Dellunde, and M. Paolucci, *Reputation-based decisions for logic-based cognitive agents*, 2010, pp. 1–42.
[21] J. Sabater-Mir, M. Paolucci, and R. Conte, *Repage: Reputation and image among limited autonomous partners*, Journal of Artificial Societies and Social Simulation, 9 (2006), p. 3.
[22] G. Shafer, *A Mathematical Theory of Evidence*, Princeton University Press, Princeton, 1976.
[23] P. Smets and R. Kennes, *The transferable belief model*, Artificial Intelligence, (1994), pp. 191–234.
[24] M. Tambe and W. Zhang, *Towards flexible teamwork in persistent teams: Extended report*, Autonomous Agents and Multi-Agent Systems, 3 (2000), pp. 159–183.
[25] M. Uddin, M. Zulkernine, and S. Ahamed, *Collaboration through computation: incorporating trust model into service-based software systems*, Service Oriented Computing and Applications, 3 (2009), pp. 47–63.
[26] Y. Wang, K.-J. Lin, D. S. Wong, and V. Varadharajan, *Trust management towards service-oriented applications*, Service Oriented Computing and Applications, 3 (2009), pp. 129–146.
[27] H. Zhao and P. Doshi, *A hierarchical framework for logical composition of web services*, Service Oriented Computing and Applications, 3 (2009), pp. 285–306.

*Edited by:* Dana Petcu and Alex Galis
*Received:* March 1, 2011
*Accepted:* March 31, 2011

Scalable Computing: Practice and Experience
Volume 12, Number 1, pp. 121–135. http://www.scpe.org
ISSN 1895-1767


# SERVICE COMPONENT ARCHITECTURE EXTENSION FOR SENSOR NETWORKS

PAWEL BACHARA *AND KRZYSZTOF ZIELINSKI†

**Abstract.**
The role of small electronic devices has been growing over the past years. Many types of devices which allow for interaction with the environment can now be found in the market and there is large set of solutions for integration of mobile devices with enterprise systems. However, most such solutions focus only on providing base services for other SOA system elements. The paper presents an extension of the SCA programming model to include intelligent sensors and a concept of uniform programming model shared between sensor networks and standard computers. The architecture of such an extension is proposed and an initial prototype implementation is presented. The solution utilizes a proxy pattern, validated by a simple case study using Sun SPOTs and Sentilla. The advantages and limitations of the proposed extension are also discussed.

**Key words:** SOA, Sensor networks, Service Component Architecture, Sun SPOT, Sentilla, Proxy, Multiproxy

**1. Introduction.** The role of small electronic devices is constantly growing and they have already become an integral part of our lives. Their computational power is also on the rise despite their progressive miniaturization. This work focuses on intelligent sensors and sensor networks. Current generations of smart sensor devices can work either separately or collectively, providing important information about their environment. They may collect simple data such as temperature, acceleration or humidity, or measure more sophisticated data, e.g. concentration of different substances. Intelligent sensors can be embedded in everyday devices and applied to controlling business processes. Additional examples can be found in medicine, where sensors are being applied in patient care, to constantly collect data about various vital parameters. Another promising area is the industry domain where RFID sensors are used to monitor warehouses and production lines.

This paper extends the concept presented in [1], namely that of building SCA applications with use of intelligent sensors with inbuilt communication capabilities. Intensive work on emerging technologies, e.g. Sun SPOT [13] and Sentilla [15] points to the need for meeting business needs. The number of business scenarios which require intelligent sensor devices and their networks is constantly on the rise. While programming embedded devices used to require dedicated platforms, nowadays the computational and communication capabilities of these devices have increased and they can be fully integrated with enterprise applications as extensions of their respective programming models.

SOA appears to the most commonly accepted enterprise software programming paradigm. The goal of this paper is show how the SOA-related software platform called the Service Component Architecture (SCA) [9] can be used to provide a uniform way to access components deployed on standard computers and on intelligent sensor devices. The architecture of such an extension is described and a prototype implementation is validated in the context of a simple case study. The advantages and limitation of the proposed approach are discussed. The paper also compares the proposed solution to the Service Oriented Device Architecture (SODA) [2] and Mobile SOA (MSOA) [16]

The structure of the paper is as follows. Section 2 presents the problem of integrating sensor networks with systems built according to the SOA paradigm and discusses the most important assumptions. Related work is presented in Section 3. The SCA technology and the programming model it involves are briefly characterized in Section 4. In Section 5 two intelligent sensor network solutions are described. Section 6 introduces an architecture of the proposed extension of the SCA programming model. Details of the proposed prototype implementation are presented in Section 7. Section 8 presents the simplicity of using the proposed solution and the benefits it offers. A practical usage example is described in Section 9. The paper ends with conclusions.

**2. Device integration problem in SOA.** The SOA Solution Stack (S3) [4] proposed by IBM is often cited as a reference for classification of software platforms supporting service-oriented enterprise software [5]. It comprises the following nine layers: Operational System, Service Component, Services, Business Process, Consumer, Integration, QoS, Information Architecture, and Governance and Policies.

*University of Science and Technology, Department of Computer Science, Krakow Poland, Tel.: +48-12-6173902 (bachara@agh.edu.pl).
† University of Science and Technology, Department of Computer Science, Krakow Poland, Tel.: +48-12-6339406 (kz@agh.edu.pl).

Support for programming sensor network devices should be located in one of the presented layers. This decision determines the required level of software abstraction as well as the set of nonfunctional requirements that should be satisfied. Choosing different layers has been considered. Following detailed analysis of the SCA specification we have decided to represent an intelligent sensor as a software component [11]. This choice is justified by the fact that software components represent well-defined functionality exposed by services and also directly specify what is required from other components, enabling seamless integration.

Apart from representing devices as software components the following functional requirements are important [1]:

- Support for bidirectional communication between devices and the enterprise system.
- Dynamic discovery of devices and connection setup.
- Support for concurrent ad-hoc communication using many devices.

Other requirements, mostly nonfunctional in nature, even more strongly support the representation of small mobile devices as software components. They are as follows:

- Genericity - the representation should be usable by different applications.
- Ease of use - this is a key issue. The presented solution should effectively support most use cases rather than support all of them in a very complicated way.
- Scalability - The presented solution should be designed for use by a large set of devices.

There are also some nonfunctional requirements which are common for distributed applications, but should be mentioned for the record:

- Security - this issue affect all distributed systems.
- Reliability - this is a key aspect of in distributed systems where single points of failure must be avoided.

Taking these considerations into account the most suitable solution is to integrate small devices within the Service Component layer of the S3 model. Further analysis should therefore concentrate on the Service Component Architecture as the programming platform most commonly used in this layer [1].

**3. Related work .** The concept of integrating mobile devices with SOA systems is not new and many works directly address this topic. The most popular approach is SODA (Service Oriented Device Architecture) [2]. This architecture focuses on the layer separating the physical and digital realms. Integration of devices with the SOA runtime infrastructure is realized by the Enterprise Service Bus (ESB), exploiting the Integration Layer of the S3 model. Typically, external devices are connected to ESB via the Web Service Binding Component and their functionality is exposed as Web Services. SODA services allow programmers to use sensors and actuators as standard business services.

The details of integration with physical devices in SODA are hidden behind higher-level abstractions. Service interfaces separate enterprise developers from internal hardware-specific issues and allow for easy modifications to service implementations when migrating to a new generation of sensors or actuators. Another advantage of the SODA approach are the well-defined interfaces that can be tested independently of any application and may be reused in various applications, reducing the overall cost of system development.

The three main components of SODA are:

- Device adapters talk to device interfaces, protocols and connections on one side, and present an abstract service model of the device on the other side.
- The bus adapter moves device data over network protocols by mapping a device service's abstract model to the specific SOA binding mechanism used by the enterprise.
- The device service registry provides for the discovery and access to SODA services.

SODA has become most popular solution in the industry. It is also important that SODA implementations are not limited to specific technologies and any new devices and SOA standards may be used. However, in contrast to the solution proposed in this paper, SODA focuses on integration support and does not expose any uniform programming model. SODA exposes device functionality to SOA systems but treats them as external system elements, as shown in Figure 3.1.

Another alternative to SODA is proposed by the iMobile framework [3]. The iMobile idea appears somewhat similar to the solution presented in this paper, compared to other existing solutions (such as e.g. Mobile SOA [16]). iMobile is a proxy-based platform that addresses the research issues in building mobile services. The key element of this solution is a message gateway which allows integration of mobile devices using various protocols on different access networks. The iMobile specification consists of three main elements:

- A devlet, i.e. a driver attached to a proxy. It receives and sends messages through a particular protocol.

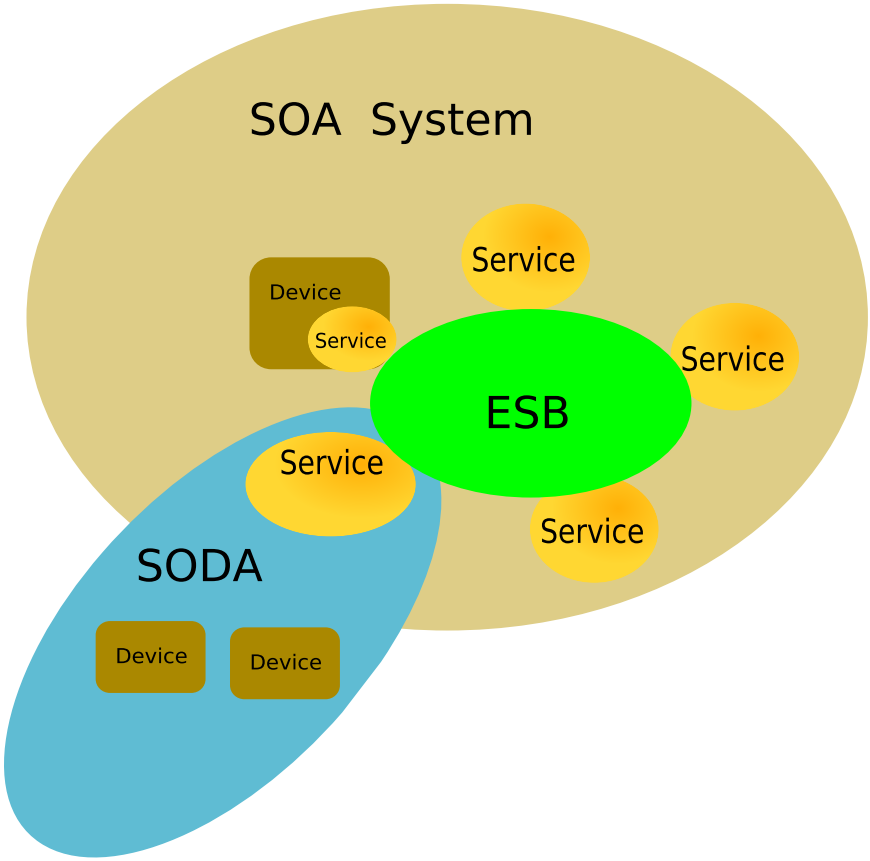



FIG. 3.1. *Device integration using SODA*

- An infolet hosted on iMobile, using an access method to provide an abstract view of the information space.
- An applet, implementing a service or application logic by processing information from various infolets.

The core of iMobile, the let engine, implements the basic framework for maintaining applets, devlets and infolets. It supports user and device profiles for personalization and transcoding, and invokes proper applets and infolets to serve requests from a devlet [3].

In summary, there is a marked difference between the presented solutions and the approach discussed in this paper. SODA aims to expose mobile devices as services and integrate them with external components, in compliance with the SOA paradigm, but it also limits devices to functions provided by Web Services. iMobile operates on a lower level and can provide better device utilization but it remains an experimental system with a relatively rigid architecture and it is not easily integrated with existing technologies [1].

**4. Service Component Architecture.** The Service Component Architecture (SCA) is a highly promising emerging standard for enterprise software development and integration of heterogeneous components. The specification of this technology involves a model for building applications and systems composed from services. In this sense, it remains very much in line with the Service-Oriented Architecture (SOA) [5]. The SCA programming model for applications and systems is based on the notion that business functions are provided as a series of services, assembled together to create solutions that serve a particular business need [9]. Applications can be constructed as composites which may contain new services along with business functions derived from existing systems and applications. This programming model aims to encompass a wide range of technologies for implementation of service components and to support communication and operation invocations between them [9] [10]. The SCA specification defines components as atoms used to build applications. Different program-

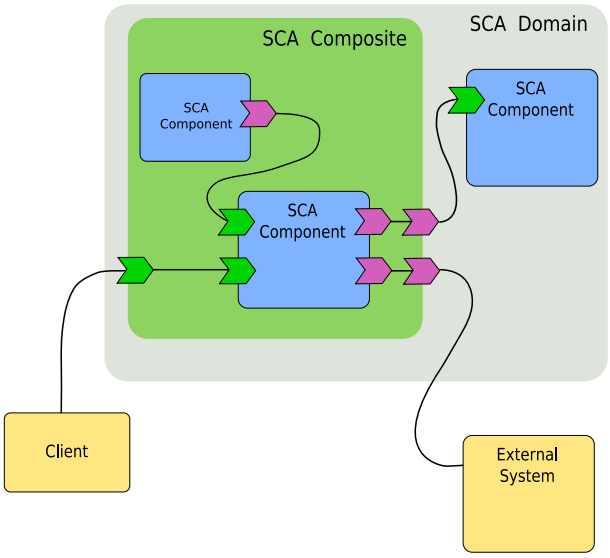



FIG. 4.1. *SCA application example*

ming languages can be used to create components. What is more, many popular frameworks and development environments associated with those languages are supported. Each component is an instance of an implementation configured using properties. The implementation of a component can be provided as a simple Java class but it may also assume the form of a Ruby script or even a BPEL process. In another paper [1] we demonstrate that it can also be represented as a mobile device, focusing on the properties of a sensor network. Important elements of SCA architecture include services and references. They allow components to communicate with one another and with external systems. References are responsible for defining the set of interfaces required by a component. The exposed functionality is described by services. Communication protocols used to access services or references are independent of their definitions.

An SCA application is built as a set of interconnected components. The basic unit of composition within an SCA system is the composite. It is an assembly of components, services, references and other interconnecting composites. All these elements can run on a single host or be distributed over a network; moreover, various technologies can be used in the scope of a single composite. The domain is the topmost element of SCA and aggregates all the above mentioned elements. It can span many computers and – thanks to proposed solution – involve the sensor network domain. An example of a simple SCA application illustrating the presented concepts is shown in Figure 4.1 .

An important advantage of the SCA architecture lies in its extensibility mechanisms defined in the specification. They describe how new bindings and implementations can be introduced. Developers who implement support for new technologies need to provide certain standard interfaces by which new components can cooperate with any other binding or implementation. For our research we chose Tuscany – the most widely used Java implementation of SCA, supported by the Apache Software Foundation [7] [12]. Its main benefits are as follows:

- open-source project,
- very good implementation of Java,
- wide range of component implementation technologies and binding types,
- large and active community.

Before we present our extension of SCA for sensor networks it is necessary to briefly discuss intelligent sensor devices.

**5. Intelligent Sensors.** Two relatively new technologies: Sun SPOTs [13] and Sentilla Perk [15] have been chosen as intelligent sensor samples. They have been selected because of their innovative potential and the large number of new applications they can support. They both represent a group of devices able to run a


FIG. 5.1. *Sun SPOT device [1]*

virtual machine but they differ on the conceptual level. Sentilla exposes a set of cooperating sensors as a unit (a sensor network), while Sun SPOT devices can work as a sensor networks, but also as standalone components.

**5.1. Sun SPOT.** A Sun SPOT (Small Portable Object Technology) device is a small, wireless, battery-powered experimental platform. It is programmed almost entirely in Java to allow regular programmers to create projects which used to require specialized embedded system development skills. The Sun SPOT includes a range of built-in sensors as well as the ability to easily interface external devices [13].

A free-range Sun SPOT device contains three main elements: a SPOT processor board, a sensor board and a battery. It is packaged in a plastic housing. The smaller basestation Sun SPOT consists of just the processor board in a plastic housing. Sun SPOT has a 180MHz 32-bit ARM920T core processor with 512K RAM and 4M Flash. The processor board features a 2.4GHz radio (IEEE 802.15.4 compliant) with an integrated antenna on board. Each processor board also provides a USB interface to connect to a PC. [13]. The basic Sensor Board includes the following components: a 3-axis accelerometer (with two range settings: 2G or 6G), a temperature sensor, a light sensor, 8 tri-color LEDs, 6 analog inputs readable by an ADC, 2 momentary switches, 5 general-purpose I/O pins and 4 high-current output pins. It is possible to replace the built-in sensor boards with custom devices. To provide communication with standard computers a Sun SPOT device can be connected to a USB port and work in a "basestation" mode. Owing to dedicated drivers, any Java application may access the functionality of a connected device and communicate with other nodes of the sensor network.

The Squawk Java Virtual Machine (JVM) works on Sun SPOTs. It is an open-source virtual machine for the Java language. Squawk is meant to be used in small, resource-constrained devices and has a small memory footprint. Reduction of memory usage on mobile devices has been achieved by performing many tasks (such as verification, optimization and conversion to an internal suite format) on the host machine during deployment. Suites are deployed onto Sun SPOT devices and are interpreted by the on-device VM. Squawk bases on the CLDC specification and introduces some important restrictions such as:

- java.io.Serializable cannot be used,
- Java Reflection API is not available,
- Remote Method Invocation (RMI)is not supported,
- User-defined class loaders have to be used.

Due to these restrictions using Sun SPOT devices with enterprise frameworks is much more complicated. For example, Tuscany uses the Reflection API to wrap managed components.

**5.2. Sentilla Perk.** The second type of device which has been selected for our work is Sentilla Perk. A standalone device and gateway used to communication with a PC are presented in Figure 5.2. Sentilla refers to elements of pervasive networks as motes or Jcreate pervasive computers. A single Sentilla mote is structurally


FIG. 5.2. *Sentilla devices and gateway [1]*

similar to SPOT devices but it is designed to reduce power consumption and operate at limited power. That's why it only provides an 8 MHz 16-bit Texas Instruments MSP430 microprocessor with 10 KB RAM. Flash memory capacity is limited to 1072 KB. Similar to Sun SPOT, wireless communication is built over IEEE 802.15.4. Each device also contains the following built-in sensor/effectors:

- a 3-axis accelerometer,
- a temperature sensor,
- 8 LEDs,
- two standard connectors (analog inputs),
- an expansion connector.

A USB gateway device provides communication between mobile devices and the host PC. It is also possible to install new applications thorough this wireless connection. Sentilla VM runs on Jcreate pervasive computers. Similarly to SPOTs, it is also based on CLDC 1.1 (with additional restrictions). RAM is limited to 2KB for user applications.

An important difference between SPOTs and Sentilla lies in the application deployment process. In Sun SPOT sensor networks each device requires separate deployment and different nodes can run different programs. In a Sentilla sensor network the application is deployed to the network and all participating devices must share the same code.

**6. Architecture - small device Proxy pattern.** To satisfy the requirements presented in section 2, extensions of the SCA platform have been designed and validated. This section presents the architecture and describes two different variants of its realization.

**6.1. Mobile Proxy.** The SCA specification provides two ways to introduce new elements (i) a specific binding which defines the protocol of communication with interface implementations, or (ii) creating a custom component implementation. The former solution requires low-level programming which is device-specific. The latter option is more general as it allows developers to introduce components implemented in different programming languages and frameworks, hiding all hardware dependencies [1].

This paper presents the concept of extending SCA over the sensor network domain by creating components which represent a device in the SCA application. Users can access the functionality provided by sensors or effectors and do not have to build code which would be deployed to a mobile device: they can simply refer to a new component called a mobile proxy instead. Implementations for SpotProxy and SentillaProxy have been created. The concept of our solution is presented in Figure 6.1 and a more detailed architecture can be seen in Figure 6.2.

To expose a service (green arrow) and provide access to device functions a dedicated component has been created. All communication details are hidden by this element. The out component has two main parts: the first one responsible for integration with SCA and providing communication on the host side, while the second one operating on mobile devices and responsible for precessing requests. When a device function is called using the exposed interface all request parameters are marshaled and sent to the assigned device (or devices). The remote device unpacks these parameters and method descriptions, and then invokes the selected action. Any obtained result is marshaled and sent back to the server. Finally, the component method returns the result

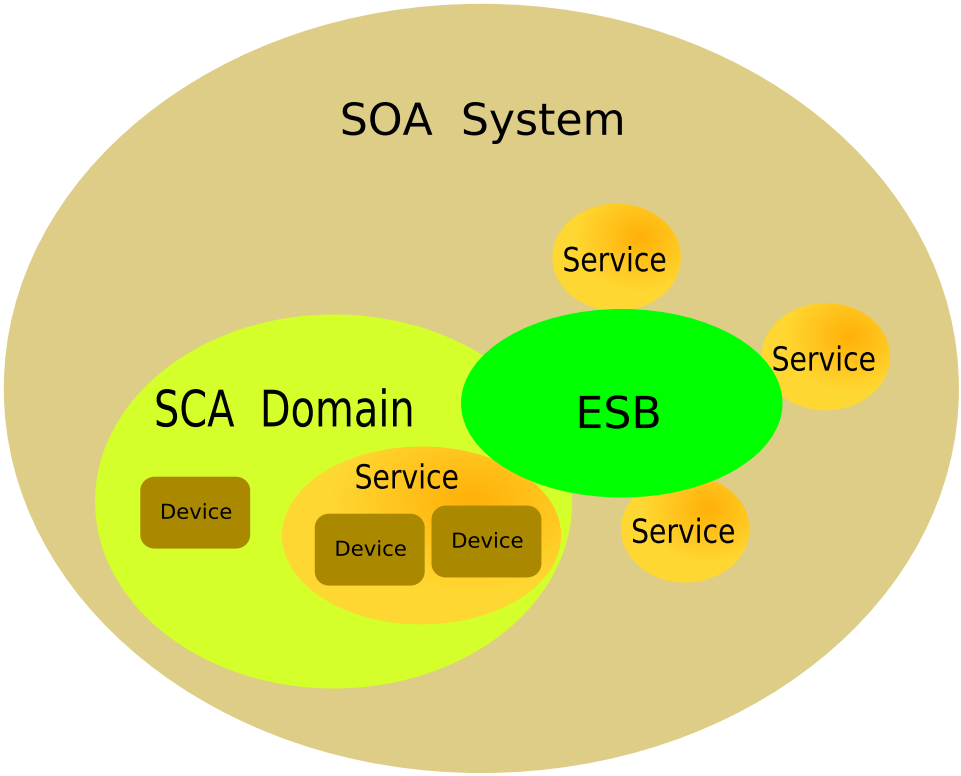



FIG. 6.1. *Device integration using SCA* .

value [1]. The presented solution simplifies access to sensor network devices. Its operation can be described in several simple steps:

1. Deploy the mobile part of the proxy component to sensor network devices (Our implementation provides an option to do it automatically).
2. Connect the basestation or gateway to a host computer.
3. Add a new component (built as a SpotProxy or SentillaProxy implementation) to SCA configuration files.
4. Configure and implement components which will use our sensor network base service.

A detailed description of how the application is constructed can be found in Section 8.

Based on the presented architecture, a mobile proxy implementation has been developed. In its simplest version the proxy represents one specific device. In our case it is identified by a MAC address provided by the end user. This version can be used for static sensor networks where we know which device will provide data of interest to the user. Sometimes this solution could prove sufficient, but in most scenarios we will lose most of the benefits of using sensor networks. Moreover, it is hard to utilize this version for a large set of devices. This is why a more complex alternative – the Mobile Multiproxy – has been designed.

**6.2. Mobile Multiproxy.** Due to the limitations of the simple version of Mobile Proxy, another solution has been proposed, utilizing the so-called Multiproxy which represents more than one small device. This component can represent a dynamic set of devices, reflecting the changes in the sensor network. The architecture of a Multiproxy is depicted in Figure 6.3. The concept of a context is used to define which devices should be represented by the Multiproxy. User are able to configure a set of properties applied to each sensor. Different options for defining sets of devices are provided:

- Address base:
  - simple device address
  - list of device addresses

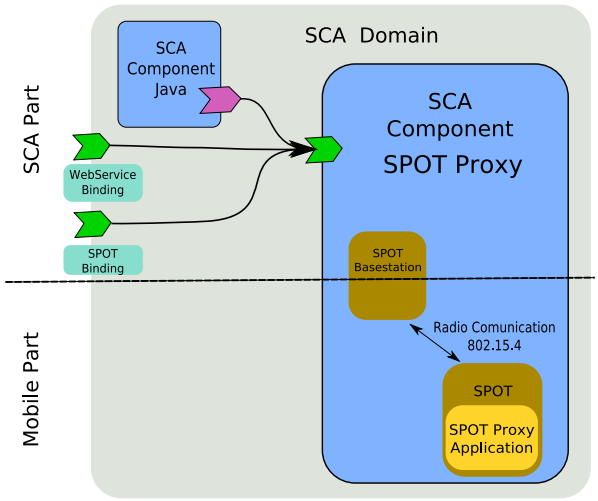



FIG. 6.2. *Spot proxy Architecture*

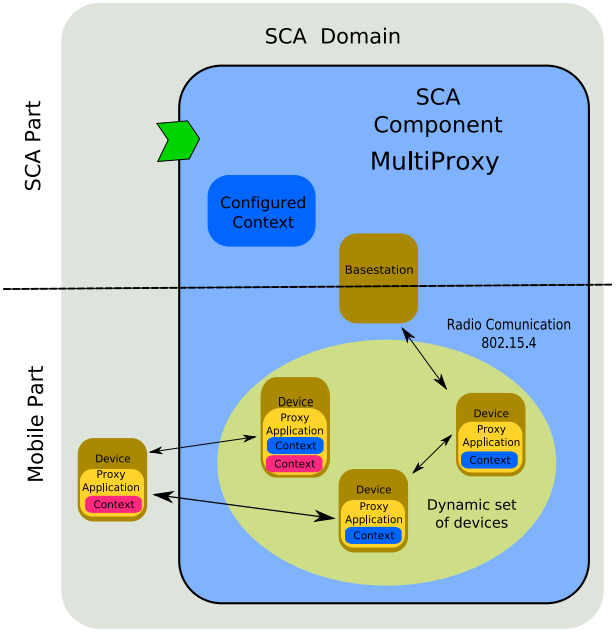



FIG. 6.3. *Multiproxy Architecture*

  - regular expression to match device addresses
- Parameter base:
  - simple text
  - list of texts
  - regular expression to match configured texts
- Dynamic:
  - hop number in the network
  - radio signal strength
  - sensor status

It is important to note that a single physical sensor may be utilized by different components at the same time. To make both implementations of the mobile proxy interchangeable the interface exposed by a multiproxy component is identical to the one exposed by the standard version. Unfortunately, this approach requires the Multiproxy to aggregate results of operations invoked on many devices. Following analysis of typical scenarios, several policies can be proposed:

- callAny - Choose any device and invoke an operation on it. The operation can be invoked on many devices but only one result is received (the first one). Another option is to send a request only to the device which (usually) returns the result in the least amount of time.
- callAnyRR - Similar to the previous one. The difference is that the system may try to use a different device for each invocation.
- callAll - Invoke the operation on all devices assigned to a component. In this case, if the user wishes to receive results, an aggregation should be chosen:
  - Maximum value
  - Minimum value
  - Logical AND or OR
- callMost - Similar to the previous one, but to provide results faster it only waits until a certain percentage of results is obtained.
  - Median
  - Average

The Multiproxy component enables better exploitation of the integration of dynamic sensor networks with SOA applications. Key advantages of this concept are as follows:

- support for dynamic sets of devices;
- components utilize available resources in an automatic way;
- communication with mobile devices is completely transparent to the user;
- users do not need to provide any code for the mobile devices;
- usage of mobile device functions is very easy;
- more effective management of SCA configuration due to limited number of components.

The existing implementation of the Multiproxy component constantly monitors the sensor network and maintains a list devices assigned to it. As part of our future work we want to implement and validate a more dynamic solution where devices execute operations only upon making an autonomous decision on being assigned to the component which has sent the request.

**7. SPOT proxy implementation.** The implementation of the proposed solution is presented on the example of a Mobile SpotProxy. The most important element of the implementation (from the developer's perspective) is its interface, exposed by the SpotProxy component and presented in Figure 7.1. In this example the user has access to all built-in SPOT sensors and can use LEDs and input/output connectors [1].

Another important element is the configuration of the SCA composite, defined as an XML file:

```
...
<element name="implementation.spot" type="e:SpotProxyImplementation"/>
<complexType name="SpotProxyImplementation">
   <complexContent>
     <extension base="sca:Implementation">
      <sequence>
        <element name="device" type="e:Device"
                  minOccurs="1" maxOccurs="1"/>
        <element name="spotProperties" type="e:spotProperties"
                  minOccurs="0" maxOccurs="1"/>
      </sequence>
      <attribute name="lazy" type="boolean" default="true" />
     </extension>
   </complexContent>
</complexType>

<complexType name="Device">
```

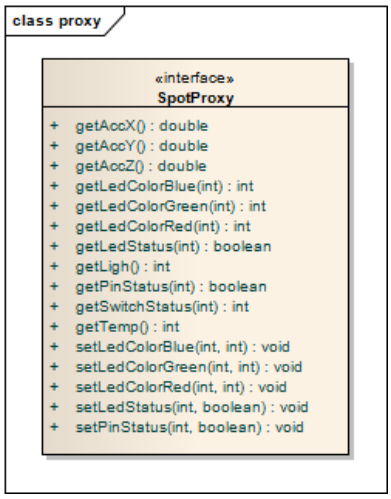



FIG. 7.1. *SpotProxy Interface [1]*

```
   <attribute name="spotPort" type="int" />
   <attribute name="spotAddress" type="anyURI" />
   <attribute name="spotName" type="QName" />
</complexType>

<complexType name="spotProperties">
   <attribute name="timeToLive" type="int" default="10"/>
   <attribute name="maxBroadcastHops" type="int" />
   <attribute name="chanelNumber" type="int"/>
   <attribute name="outputPower" type="int"/>
   <anyAttribute />
</complexType>
...
```

The implementation.spot element has the following attributes and subnodes:

- **implementation.spot** - This is the generic SPOT implementation type. The type is extensible so it is possible to add additional domain-specific attributes and elements. It extends the sca:Implementation type.
- **implementation.spot/Device** - Identifies the assigned device.
- **implementation.spot/Device/@name** - The name of the destination to which the proxy is connected.
- **implementation.spot/Device/@spotPort** - The port which will be used to establish the connection.
- **implementation.spot/Device/@spotUrl** - The MAC address of the assigned device.
- **implementation.spot/spotProperties** - The element which facilitates connection configuration.
- **implementation.spot/spotProperties/ @timeToLive** - Timeout value when waiting for a response. Default is 10s.
- **implementation.spot/spotProperties/ @maxBroadcastHops** - Used for datagram broadcast connections; limits broadcast range.
- **implementation.spot/spotProperties/ @channelNumber** - Forces a specific channel number.
- **implementation.spot/spotProperties/ @outputPower** - Sets the output power of a radio device. Default is maximum power.

The presented configuration is used to create an instance of the Mobile Proxy component in the SCA application. Key classes are presented in Figure 7.2. Instances of the SpotProxyImplementationImpl class store parameters configured in the domain definition file and they are created by SpotProxyImplementationFactoryImpl. SpotProxyProcessor is used to parse the XML configuration. Based on its configuration, the component can send and receive messages using SpotProxyExecutor whenever SpotProxyImplementationInvoker receives a

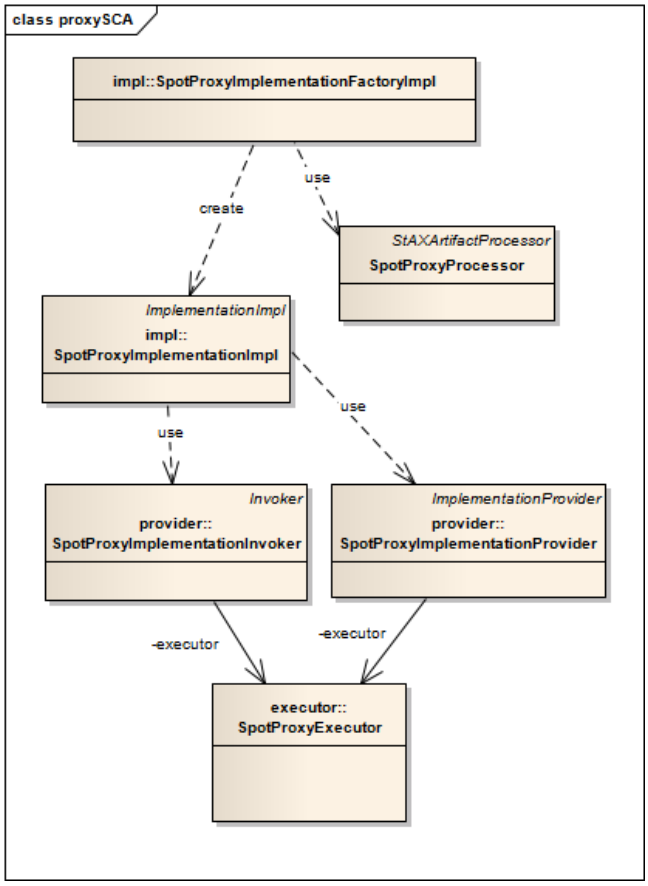



FIG. 7.2. *Main classes of Mobile SpotProxy [1]*

request to call some operation.

**8. Application development workflow.** Creating an application which utilizes sensor networks to provide data for large-scale enterprise SOA systems is a challenging task which requires in-depth knowledge of both domains. The proposed extension of the SCA architecture is designed to simplify it as much as possible. This section describe the steps necessary to create a simple SOA service. The application can turn a light on and off using Web Service access. It also monitors system temperature and disables the light when the temperature exceeds a specific safety threshold. For manual control, another device connected by the spot binding can be used. The entire system is depicted in Figure 8.1.

1. The first step is to define external interfaces of the exposed service.
2. The second step is to implement a class which provides the required functionality and implements the prepared interface. Access to the physical device is hidden by the SpotProxy interface presented in the previous section. For example, to turn the light off or on, the status of one of the output pins needs to be changed:
   ```
   public void setLightStatus(boolean status) {
   spotProxy.setPinStatus(PIN_NUMBER, !status);
   }
   ```
   To read the current temperature, another method has to be called:
   ```
   public int getCurrentTemp() {
   int temp = spotProxy.getTemp();
   return temp;
   }
   ```
   A reference to the proxy object is set by the SCA framework based on the domain configuration files.

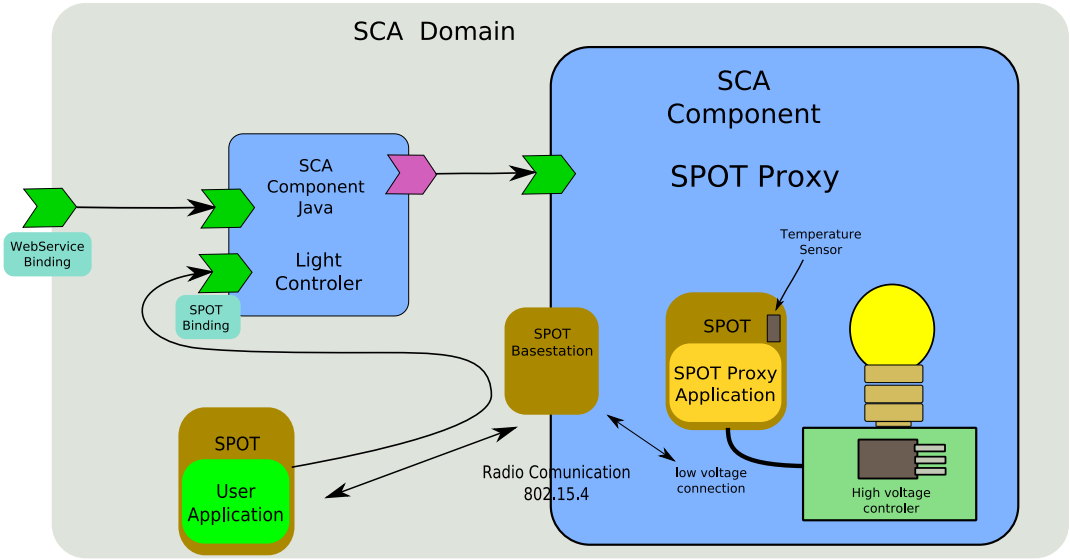



FIG. 8.1. *Light Controller application built using the Spot Proxy component.*

3. The next step is to prepare the physical device. Selected devices should have the SpotProxy application installed. This can be done by using a single **ant** command. In our solution a power module should be connected to the device. This module provides control over higher voltages than the internal SPOT components allow. In our case, a 230 Volt circuit can be controlled.
4. To integrate all of the described software components and sensor network devices, an SCA configuration file need to be created. A complete file is quoted here to show the simplicity of this step:

```
<?xml version="1.0" encoding="UTF-8"?>

<composite  xmlns="http://www.osoa.org/xmlns/sca/1.0"
  targetNamespace="http://spotsca" xmlns:cb="http://spotsca"
  xmlns:e="http://galaxy.agh.edu.pl/bachara" name="spotsca">

 <component name="SpotServiceComponent">
  <e:implementation.spot lazy="true">
   <e:device spotPort="114" spotAddress="0014.4F01.0000.13E4" spotName="testSpot">
   </e:device>
  </e:implementation.spot>
 </component>

 <service name="SpotProxyWeb" promote="SpotServiceComponent">
   <interface.java interface="pl.edu.agh.bachara.spotsca.proxy.SpotProxy"/>
   <binding.ws uri="http://localhost:8086/SpotProxy"/>
 </service>

 <component name="LightManagerComponent">
  <implementation.java
    class="pl.edu.agh.bachara.spotsca.proxy.demo.LightManagerImpl"/>

  <service name="LightManager">
   <interface.java interface="pl.edu.agh.bachara.spotsca.proxy.demo.LightManager" />
   <binding.ws uri="http://localhost:8086/LightManager"/>
   <e:binding.spot >
```

TABLE 9.1
*Comparison of startup time and memory allocation.*

| Application | Startup time [s] | Memory allocated [MB] | Average request time[ms] |
|---|---|---|---|
| SPOT simple | 1 | 9 | 59 |
| SPOT proxy | 6 | 40 | 68 |
| Sentilla simple | 0.5 | 7 | 1259 |
| Sentilla proxy | 5 | 37 | 1276 |

```
    <e:destination name="serDest" spotPort="103"
          spotUrl="radiogram://0014.4F01.0000.14D7" spotName="serDestSpot"/>
    <e:response>
      <e:destination name="serReqDest" spotPort="104"
          spotUrl="radiogram://0014.4F01.0000.14D7" spotName="serReqDestSpot"/>
    </e:response>
   </e:binding.spot>
  </service>
  <reference name="spotProxy" target="SpotServiceComponent" />
 </component>

</composite>
```

Let us briefly describe this configuration file. The first component, called SpotServiceComponent, is an instance of the mobile proxy. The assigned device address and communication channel is defined in the configuration. The next step involves exposing the service by using a Web Service binding which allows direct access to SPOT functions thorough the WS protocol. The LightManager component is subsequently defined using a SpotProxy reference and exposes a LightManager interface thorough Web Service and SPOT bindings.

5. To obtain a working application, runtime configuration is necessary. A class with a main method has to be created to spawn the SCA Domain using the prepared configuration:
`scaDomain = SCADomain.newInstance("proxy.composite");`
Required libraries should be added to the classpath and, finally, options required by SPOT libraries should be defined.

**9. Performance study .** The performance of the presented implementations is validated in this section. To measure the overhead generated by our solution an application devoid of SCA extensions has been prepared. Test results are summarized in Table 9.1. The first column presents a comparison of startup time, measured between application startup and reception of the result of the first remote call. The second column contains the allocated memory volume (measured one minute after application startup). The last column presents the average operation execution time. All tests are performed for a simple network where the device operates in a basestation (or gateway, in the case of Sentilla) radio range. Time is measured between sending a temperature measurement request and receiving a response.

The presented results show that the SCA base programming model for sensor network integration introduces approximately 15% overhead for a single operation call compared to raw implementation. This could be explained by execution of additional framework code in the course of communication and the relative complexity of the protocol used to send requests and receive responses. In some rare cases, the enlarged memory footprint and longer startup time may also become important [1].

To validate the Multiproxy implementation a more complicated sensor network has been created. This network is presented in Figure 9.1. The first test was conducted using a simple SpotProxy configured to use device 4. Because sensors number 3 and 4 are outside the basestation range, all communications have to be routed through devices 1 or 2. The second test validated support for more then one device. Here, Multiproxy was configured to use devices 3 and 4. In the final test, simulating a dynamic sensor network, all mobile devices were configured to be disabled 25% of the time at random intervals (the average duration of an active

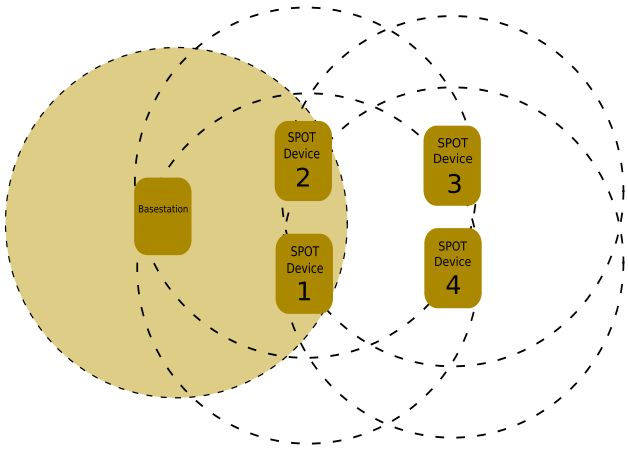



FIG. 9.1. *SpotProxy test configuration.*

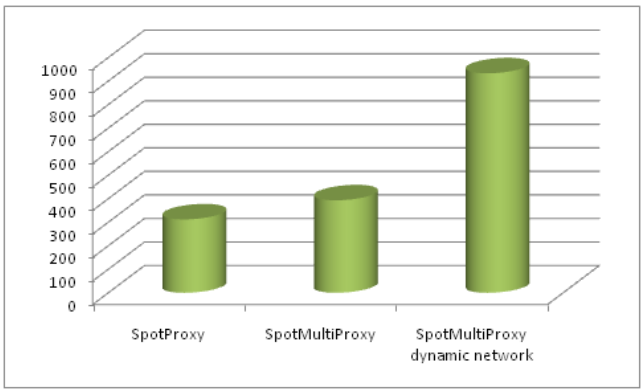



FIG. 9.2. *Average request time in ms.*

session is approximately 30s). Results are presented in Figure 9.2. Multiproxies exhibit longer average request times compared to a standalone implementation. This could be explained by the need to transmit additional monitoring messages concurrent with user requests. Due to the dynamic nature of the network used in the final test, request processing is slowed down by a factor of approximately 3. More detailed dynamic Multiproxy tests have been conducted in relation to the use of ad-hoc protocols but this issue falls beyond the scope of the presented research.

**10. Conclusions.** Extending the SCA architecture with new components which represent sensor network devices seems to be a very effective solution. Abstracting mobile devices as components enables access to their features consistent with the platform used for system implementation. It also provides a suitable level of decomposition of complex functionality into reusable elements. Thus, the SCA domain can be seamlessly expanded to cover the mobile domain, in addition to standalone computers. The SCA technology seems to be a suitable development platform for integration of both worlds. This work presents a practical verification of this approach on the Service Components Layer but it may also be applied to other layers of the S3 model such as the Services Layer used in SODA.

The proposed mobile proxy pattern used as an internal mechanism of the presented extension is flexible enough to provide the requested level of transparency for single device. The Multiproxy performs the same function for an entire set of devices and simplifies management of the sensor network. The proposed solution requires further evaluation for larger sets of devices and real sensor networks, but preliminary studies look very promising.

**11. Acknowledgment.** The research presented in this paper was partially supported by the European Union in the scope of the European Regional Development Fund program no. POIG.01.03.01-00-008/08.

REFERENCES

[1] Pawel Bachara; Krzysztof Zieliñski, SOA-compliant programming model for intelligent sensor networks SCA-based solution, SYNASC 2010 : 12th international symposium on Symbolic and Numeric Algorithms for Scientific Computing : 23–26 September 2010 Timisoara Romania, ISBN 978-0-7695-4324-6, pp. 471–478.
[2] Scott de Deugd; Randy Carroll; Kevin Kelly; Bill Millett; Jeffrey Ricker, SODA: Service Oriented Device Architecture,IEEE Pervasive Computing, vol. 5, no. 3, pp. 94-96, July-Sept. 2006
[3] Chung-Hwa Herman Rao, Yih-Fam Robin Chen, Di-Fa Chang, Ming-Feng Chen, iMobile: A Proxy-Based Platform for Mobile Services, Workshop on Wireless Mobile Intemet 7/01 Rome, Italy 2001
[4] Arsanjani, A.; Liang-Jie Zhang; Ellis, M.; Allam, A.; Channabasavaiah, K., S3: A Service-Oriented Reference Architecture IT Professional Volume 9, Issue 3, May-June 2007 Page(s):10–17
[5] Thomas Erl, "Service-oriented Architecture: Concepts, Technology, and Design" Upper Saddle River: Prentice Hall PTR. ISBN 0-13-185858-0.
[6] Dave Chappell, "Enterprise Service Bus" , O'Reilly: June 2004, ISBN 0-596-00675-6
[7] Apache Tuscany Project. http://tuscany.apache.org/
[8] JSR-000139 Connected Limited Device Configuration 1.1
[9] OSOA - Service Component Architecture Home. http://www.osoa.org/display/Main/Service+Component+Architecture+Home
[10] IBM Service Component Architecture. http://www.ibm.com/developerworks/ library/specification/ws-sca/
[11] SCA Service Component Architecture, Assembly Model Specification. http://www.osoa.org/download/attachments/35/SCA_AssemblyModel_V100.pdf
[12] SCA Java Component Implementation V1.0, Assembly Model Specification http://www.osoa.org/download/attachments/35/ SCA_JavaComponentImplementation_V100.pdf
[13] Project SUN SPOT. http://www.sunspotworld.com/
[14] The Squawk Project. http://research.sun.com/projects/squawk/
[15] Project Sentilla Perk: Pervasive Computing Kit. http://www.sentilla.com/developer.html
[16] Ari Shapiro, Andreas Frank , Mobility Service Oriented Architecture Extending SOA to Mobile Devices http://developers.sun.com/learning/ javaoneonline/ 2007/pdf/TS-5639.pdf

*Edited by:* Dana Petcu and Alex Galis
*Received:* March 1, 2011
*Accepted:* March 31, 2011

Scalable Computing: Practice and Experience
Volume 12, Number 1, pp. 137–150. http://www.scpe.org
ISSN 1895-1767


# ANT-INSPIRED FRAMEWORK FOR AUTOMATIC WEB SERVICE COMPOSITION

CRISTINA BIANCA POP, VIORICA ROZINA CHIFU, IOAN SALOMIE, MIHAELA DINSOREANU, TUDOR DAVID, VLAD ACRETOAIE*

**Abstract.** This paper presents a framework for automatic service composition which combines a composition graph model with an Ant Colony Optimization metaheuristic to find the optimal composition solution. The composition graph model encodes all the possible composition solutions that satisfy a user request. The graph will be further used as the search space for the ant-inspired selection method targeting the identification of the optimal composition solution. To identify the optimal composition solution we define a fitness function which uses the *QoS* attributes and the semantic quality as selection criteria. The proposed composition framework has been tested and evaluated on an extended version of the SAWSDL-TC benchmark.

**Key words:** Web service composition, semantic Web services, graph model, ant colony optimization

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

**1. Introduction.** The growing number of Web services available in public and private repositories requires fast, precise and scalable algorithms that can process them so as to present users with the most suitable functionality. Such algorithms include the automatic Web service discovery and composition as well as the selection of the optimal composition solution.

Automatic Web service discovery algorithms approach the problem of providing users with the Web services that most closely match their request, usually expressed as a list of inputs and a list of desired outputs.

Automatic Web service composition algorithms address the scenario in which the above mentioned discovery algorithms fail to return a Web service relevant to the user request because such a service is not registered in the available repositories. By Web service composition, a set of services are combined into a more complex service which satisfies the user request.

Automatic Web service discovery and composition can be achieved by combining the Web service technology with Semantic Web, thus enhancing the representation of data and processes available on the Internet with machine-interpretable information. In semantic Web service composition it is important to ensure that the results satisfy both the functional and non-functional requirements specified by the user. However, because the total number of possible composition solutions is often extremely high, an exhaustive evaluation of all the solutions is unpractical. Consequently, techniques that identify the optimal or a near-optimal composition solution without processing the entire search space are required.

In this context, our paper proposes a framework for automatic service composition which combines a composition graph model with an ant-inspired method to find the optimal composition solution. The composition graph model stores all the composition solutions that satisfy a user request. In our approach, a user request is described in terms of: functional requirements - ontological concepts that semantically describe the inputs and outputs of the requested composed service and non-functional requirements - weights associated to user preferences regarding the relevance of a composition solution's semantic quality and its *QoS* attributes. The proposed ant-inspired method, previously introduced in [3], adapts the Ant Colony Optimization (ACO) metaheuristic [1] to identify the optimal composition solution encoded in the composition graph model. We employ the following methodology for adapting the ACO metaheuristic: first, we study and analyze the biological source of inspiration of the meta-heuristic; second, we model the biological entities, relationships and processes so that they fit into our problem of selecting the optimal service composition; finally we adapt the algorithm proposed by the metaheuristic to the problem of service selection. To identify the optimal solution a fitness function which uses the *QoS* attributes and the semantic quality as selection criteria is defined. The ant-inspired composition framework has been tested and evaluated on an extended version of the SAWSDL-TC benchmark [12].

The rest of the paper is structured as follows. In Section 2 we introduce related work. The proposed framework architecture is presented in Section 3, while the associated workflows are presented in Sections 4 and 5. Section 6 discusses the adjustable parameters of the Ant-inspired composition framework and evaluates experimental results. The paper ends with our conclusions and future work proposals.

*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).

**2. Related Work.** This section reviews some related works in the field of Web service composition.

A broker-based framework for composing Web services is described in [5]. The framework starts from a composition plan which contains a set of abstract activities. Each abstract activity is further mapped to a set of concrete services providing the same functionality but having different *QoS* attributes values. The framework identifies the optimal composition solution which provides the best global *QoS* score by using an adapted version of the HEU heuristic introduced in [11]. The main idea behind the HEU heuristic is to start from an initial feasible solution and to replace the solution elements which negatively affect the quality of the solution with other better elements. For efficiency reasons, the concept of dominance is introduced to skip the solution elements which have been previously replaced by other better elements.

In [4] an autonomous semantic Web service composition and management system inspired by the neuroendocrine-immune network is presented. The framework functionality relies on the cooperation of a set of entities to compose Web services rather than on a central coordinator. Each Web service is associated to a biological entity defined by a set of attributes, a body containing functional information and a biological behavior such as create, sleep, mutate and execute. A composed Web service satisfying a user request is represented as a network of biological entities obtained through negotiation. The negotiation between biological entities is performed using partial deduction and Linear Logic theorem proving.

An integer programming-based framework for Web service composition is proposed in [6]. The composition method used in this framework takes into consideration functional and non-functional *QoS* parameters. The framework first generates the integer-linear programming (ILP) description of the services available in a repository and then based on the user request it processes these descriptions using an ILP solver. The Web service composition solutions returned by the solver are encoded in WSBPEL.

In [8] and [9] an event calculus-based framework has been proposed as a solution for the Web service composition problem. The authors demonstrate that when a goal situation is given, the event calculus can find suitable service compositions by using the abductive planning technique. The main contributions of these works are a translation algorithm from OWL-S semantic descriptions of Web services into the event calculus and a formal framework that shows how generic composition procedures are described in the event calculus. The event calculus-based framework is presented as an alternative approach to the agent-based composition.

In the Meteor-S project [10], a Web Service Composition framework for dynamic composition of Web services has been developed which takes into consideration business constraints. The idea behind this framework is to describe the composed service as an abstract process in BPEL, and then to discover the services whose Profile matches with the defined abstract process. Once the requested service is discovered, the candidate services are selected based on the business and process constraints. The disadvantage of this approach is that the proposed technique is not totally automated.

**3. Framework Architecture.** This section presents the proposed Ant-inspired framework for automatic Web service composition. The aim of the framework is to: (*i*) publish semantic Web services, (*ii*) discover semantic Web services, (*iii*) compose semantic Web services, and (*iv*) select the optimal or a near-optimal composition solution according to user constraints. The conceptual architecture of the Ant-inspired framework is organized on the following layers (see Figure 3.1): the Web Service Publication layer, the Web Service Composition layer and the Composition Selection layer. Each layer supports user interaction and has an associated workflow.

*Service Publication Layer.* Service providers interact with the Service Publication layer through an associated graphical user interface to publish their semantic Web services in the Semantic UDDI registry. Based on the functional and non-functional information provided by the service provider, the Publication Module (*i*) generates the XML structure containing the semantic descriptions of a Web service, (*ii*) generates a *tModel* containing a reference to the XML file, (*iii*) generates the standard structures (for example *BusinessEntity* and *BusinessService*) required for storing the information about a Web service in a UDDI repository and (*iv*) groups services into clusters according to their semantic similarity. To evaluate the semantic similarity between two services, the Publication module interacts with the Service Matching module. The Publication module stores the generated structures in a Semantic UDDI registry. We chose to group services into clusters to make the discovery process more efficient.

*Service Discovery Layer.* Service requestors interact with the Service Discovery layer to find semantic Web services that satisfy some functional and non-functional requirements expressed using ontological concepts. In the Service Discovery layer, an important role is played by the Service Matching Module which is responsible

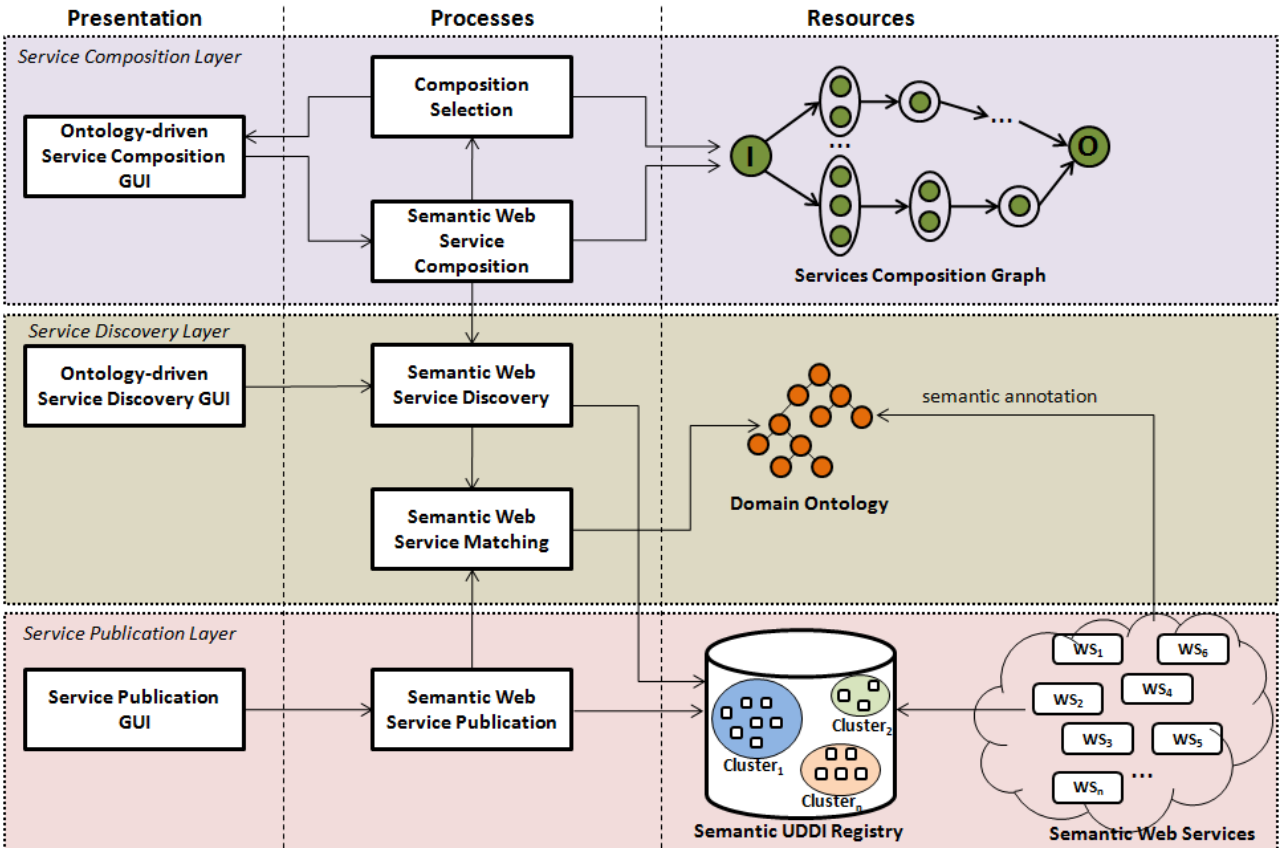



FIG. 3.1. *Framework architecture*

for evaluating the semantic similarity between two services. This module interacts directly with the ontology repository.

*Service Composition Layer.* The Service Composition Module interacts with the Service Discovery Module to build the composition graph which stores all the possible solutions for a specified user request. To find the optimal composition solution according to *QoS* attributes and semantic quality, the Selection Module interacts with the Service Composition Module. The Selection Module returns a ranked set of composition solutions, the first one being the optimal solution. The user is allowed to choose the solution he prefers.

**4. The Web Service Composition Workflow.** The main objective of the composition workflow is to obtain a graph-based representation of all the possible compositions that satisfy a user request. The composition graph will be further used as the search space in the selection process targeting the identification of the optimal composition solution according to the constraints specified in the user request.

**4.1. The Composition Graph.** The composition graph (see the UML representation in Figure 4.1) is automatically built for each user request.

In our approach, a *graph node* represents a cluster of similar services. A cluster contains a set of semantic similar services. We consider that two services are semantically similar if there is a degree of semantic matching ($DoM$), higher than a specified threshold, between them. The method for evaluating the degree of match between two services has been presented in a previous work [2]. Besides the graph nodes containing clusters of similar services we also define the input and output nodes as two special types of nodes. The input node contains a cluster with a single service which only has outputs representing a set of ontology concepts describing the user provided inputs. The output node on the other hand, contains a cluster with a single service having just inputs representing the concepts describing the user requested outputs.

A *directed edge* links a pair of cluster nodes if there is a degree of semantic match between the outputs of one of the clusters and the inputs of the other cluster.

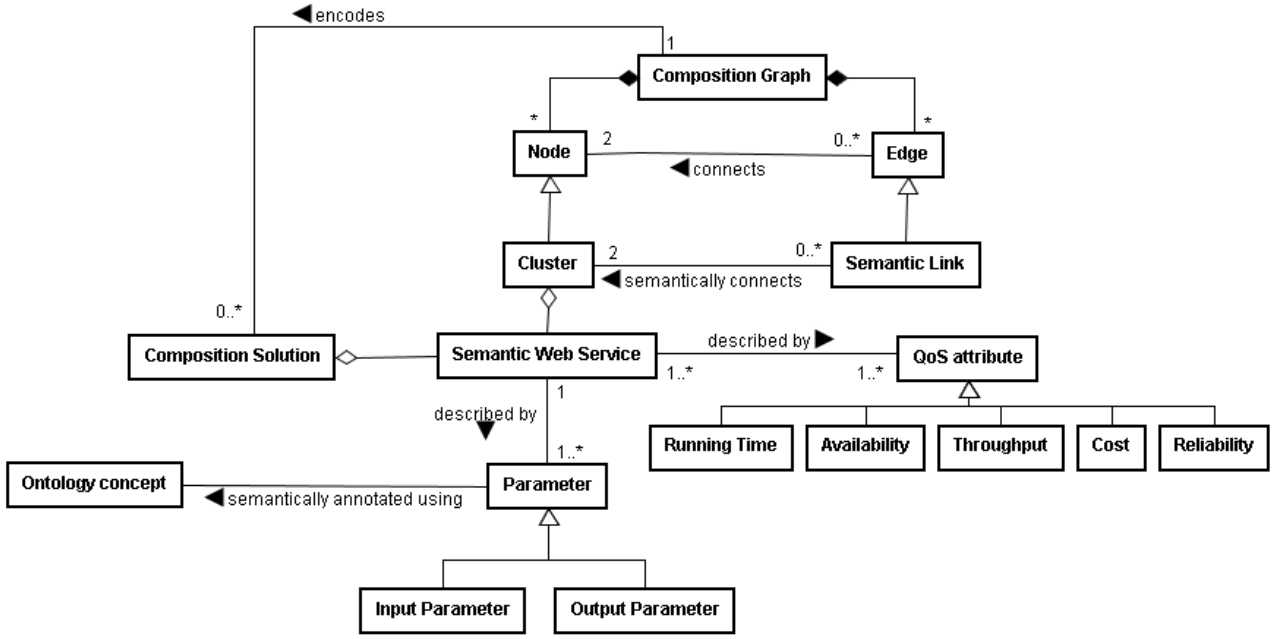



FIG. 4.1. *UML representation of the composition graph model*

The construction of the composition graph starts with a single node representing the user provided inputs. Then, the graph is expanded with new clusters of services which have a degree of match with the clusters already added in the graph. The clusters are obtained by applying a clustering method we have introduced in [2]. This way, the expansion of the graph is performed until all the inputs of the output node are satisfied. A service cluster should be added only if the clusters already present in the graph provide all the outputs required for its execution.

A *composition solution* is a directed acyclic sub-graph which contains the service of the input node, a set of services (one from each considered cluster node) and the service of the output node.

**4.2. The Composition Algorithm.** The composition algorithm (see Algorithm 4.2.1) takes as inputs (*i*) the user request, (*ii*) the set of services available in a repository and (*iii*) two threshold values. The threshold values are required for evaluating the semantic matching between two clusters and between two concepts.

First the composition graph is initialized (see *Initialize_Graph* in Algorithm 4.2.1) based on the user request so that it contains only the input node. Then, the algorithm iteratively performs the following two operations: (1) a set *CL* of service clusters are discovered (see *Discovery* in Algorithm 4.2.1) [2] based on the semantic matching between the concepts describing the inputs of these services and the ones describing the outputs of the services that are already in the graph, (2) for each discovered cluster *c* a corresponding node is created and added to the composition graph and as a result the set of graph edges is updated (see *Update_Edges* in Algorithm 4.2.1). A new edge is added between two nodes if there is a semantic matching between the clusters associated to these nodes.

The algorithm ends either when the user requested output parameters are provided by the services added to the composition graph or when the graph reaches a fixed point. Reaching the fixed point means that no new service clusters have been added to the composition graph for a predefined number of iterations. If the graph does not reach a fixed point means that there is at least one composition solution. In this former case, we employ a pruning process (see *Prune* in Algorithm 4.2.1), detailed in Algorithm 4.2.2, that starts from the output node and eliminates the service clusters which do not provide either (*i*) an output parameter to other services in the graph or (*ii*) an output parameter requested by the user.

The pruning algorithm (see Algorithm 4.2.2) takes a composition graph as input. The algorithm performs a breadth-first search starting from the output node in order to remove all nodes that are not on a path connecting the output node with the input node.

Regarding performance, out of the operations used in Algorithm 4.2.1, the *Discovery* procedure has the

**Algorithm 4.2.1** Build_Composition_Graph

**Input:**
$req = (in, out)$ - the user request containing concepts that describe the provided inputs ($in$) and requested outputs ($out$);
$S$ - set of available services;
$clTh$ - threshold for clusters matching;
$cTh$ - threshold for concepts matching;
**Output:** $G = (V, E)$ - the composition graph containing a set of vertices $V$ and a set of edges $E$;
**begin**
  $v_{in}$ = **Generate_Input_Node**($req.in$)
  $v_{out}$ = **Generate_Output_Node**($req.out$)
  $G$ = **Initialize_Graph**($v_{in}$)
  **while** (($v_{out} \notin G.V$) **or** (!**Fixed_Point**($G$))) **do**
    $CL$ = **Discovery**($G.V, cTh$)
    **foreach** $c \in CL$ **do**
      $G.V = G.V \cup \{c\}$
      $G.E$ = **Update_Edges**($G.V, clTh$)
    **end for**
  **end while**
  **if** ($v_{out} \notin G.V$) **then return** $null$
  $G$ = **Prune**($G$)
  **return** $G$
**end**

**Algorithm 4.2.2** Prune

**Input:** $G = (V, E)$ - the composition graph containing a set of vertices $V$ and a set of edges $E$;
**Output:** $G$ - the pruned composition graph;
**Comments:** $O$ - the output node;
**begin**
  $E' = \emptyset$
  $V' = \emptyset$
  $L = \{O\}$
  **while**($L \neq \emptyset$)**do**
    $v$ = **Remove_First**($L$)
    $V' = V' \cup \{v\}$
    **foreach** $w \notin V', (wv) \in E$ **do Append**($L, w$)
    **foreach**$(wv) \in E, w \in V', v \in V'$ **do** $E' = E' \cup (wv)$
  **end while**
  **return** $(V', E')$
**end**

highest running time: $O(N^2)$, where $N$ is the number of clusters. In our algorithm, we call this procedure at most $N$ times. The code in the while statements has $O(N)$ complexity in the worst case. The complexity of the pruning algorithm is $O(|V| + |E|)$. Taking all this into consideration, the complexity of Algorithm 1 is $O(N^3)$.

**4.3. An Example of Building a Composition Graph.** The step-by-step construction of a composition graph is illustrated in Figure 4.2. In step 1, only the input node is present; in step 2, the service clusters discovered based on the user inputs have been added; in step 3 the service clusters whose inputs are satisfied by the previously discovered clusters are added to the graph; by step 4, the entire composition graph has been built; in step 5, pruning is applied in order to remove the unnecessary parts (notice that the services $S_4$ and $S_7$ are elliminated from the composition graph). Finally, the set of possible solutions encoded in the composition graph consists of {$S_1$, $S_2$, $S_5$}, {$S_1$, $S_2$, $S_6$}, {$S_1$, $S_3$, $S_5$} and {$S_1$, $S_3$, $S_6$}.

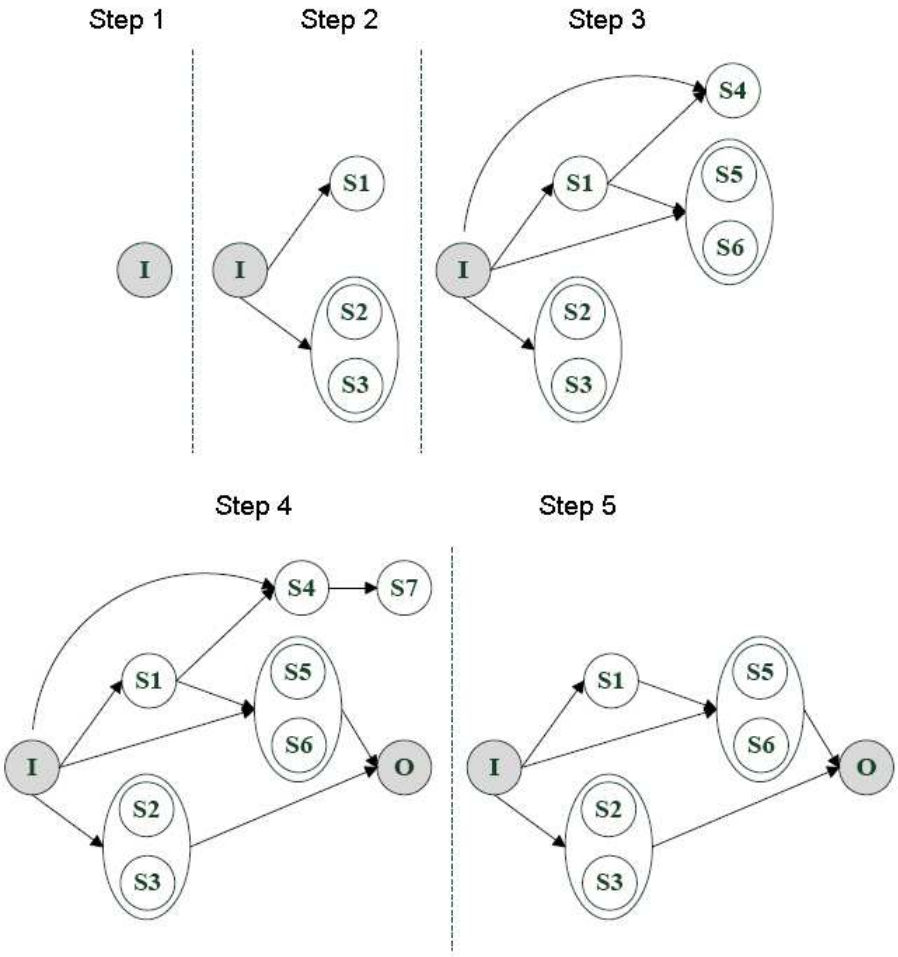



FIG. 4.2. *Composition Graph Construction*

**5. The Ant-inspired Selection Workflow.** For finding the optimal composition we adapted the Ant Colony Optimization (ACO) metaheuristic [1] which was proposed for solving combinatorial optimization problems. Our Ant-inspired selection workflow uses the previously described composition graph and a multi-criteria function to identify the optimal composition solution according to *QoS* user preferences and semantic quality.

**5.1. Problem Formalization.** The ACO metaheuristic relies on a set of artificial ants which communicate with each other to solve combinatorial optimization problems. The behavior of artificial ants is modeled according to the behavior of real ants in nature, which search for the shortest route to a food source and communicate indirectly with each other by means of the pheromone they lay on their route. In this section we present how we adapted the ACO metaheuristic to the problem of Web service composition and selection.

Just as real ants search the shortest route to a food source in a natural environment, we define a set of artificial ants that traverse the composition graph in order to find the optimal composition solution. In its search, an artificial ant can explore the composition graph and for each step it has to choose an edge that links the service where the ant is currently positioned with a new service. The choice is stochastically determined with the probability $p$ taking into account the level of pheromone $\tau_{ij}$ (see Formula 5.2) on the edge leading to a candidate service and some other heuristic information. The heuristic information refers to the *QoS* of the candidate service at the end of the edge and the degree of semantic matching between the two services connected by the edge. The probability [1] to choose an edge $(i, j)$ from the current service and leading to a candidate service is defined as follows:

$$p_{i,j}^{k} = \begin{cases} \frac{\tau_{ij}^{\alpha} \times \eta_{ij}^{\beta}}{\sum_{c_{pq} \in N(s^p)} \tau_{pq}^{\alpha} \times \eta_{pq}^{\beta}} & \text{if } c_{pq} \in N(s^p), \\ 0 & \text{otherwise.} \end{cases} \tag{5.1}$$

where $N(s^p)$ is the set of candidate edges (which may be added to the partial solution), and $\eta_{ij}$ is the heuristic information associated with an edge. The heuristic information is evaluated with the following formula:

$$\eta_{ij} = \frac{w_{QoS} \times QoS(S_2) + w_{Match} \times DoM(S_1, S_2)}{w_{QoS} + w_{Match}} \tag{5.2}$$

where $S_1$ is the current service where the ant is positioned, $S_2$ is the candidate service, $QoS$ represents the $QoS$ score of $S_2$, $DoM$ is the degree of semantic matching between $S_1$ and $S_2$. The heuristic information also considers the weights $w_{QoS}$ and $w_{Match}$ which represent the relevance of the $QoS$ score and of the degree of semantic matching, respectively. The $QoS$ score of a service is computed as:

$$QoS(s) = \frac{\sum_{i=1}^{n} w_i \times Attr_i(s)}{\sum w_i} \tag{5.3}$$

where $Attr_i(s)$ represents the value of the i-th quality of service attribute, $w_i$ represents the associated weight of the quality of service attribute and $n$ is the number of $QoS$ attributes considered. We use a $QoS$ model defined in a previous work [7]. The current model considers the running time, availability, reliability, throughput and cost but it may be extended with other $QoS$ attributes. Each of these $QoS$ attributes needs a separate calculation method, because they have different interpretations: for some of them a higher value is better, meanwhile for others smaller values are better.

The quality of a composition solution built by an artificial ant is evaluated with the following formula:

$$Score(sol) = \frac{w_{QoS} \times QoS(sol) + w_{Match} \times Sem(sol)}{(w_{QoS} + w_{Match}) * |sol|} \tag{5.4}$$

In Formula 5.4, *QoS(sol)* represents the overall $QoS$ of the composition solution *sol* and *Sem(sol)* represents the semantic quality score of *sol*.

*QoS(sol)* is computed as:

$$QoS(sol) = \frac{\sum_{i=1}^{|sol|} QoS(s_i)}{|sol|} \tag{5.5}$$

where $|sol|$ represents the number of services involved in the composition solution *sol* and *QoS(s)* is the $QoS$ score of a service $s$ (see formula 5.3).

The semantic quality of the composition solution *sol* is determined as:

$$Sem(sol) = \frac{\sum_{i=1}^{m} DoM(s_j, s_k)}{m} \tag{5.6}$$

where $m$ is the number of edges contained in the composition solution *sol* and $DoM$ evaluates the degree of semantic matching [2] between two services.

There are two types of pheromone updates, performed as suggested by the ACO metaheuristic: local update and offline update. The local update is performed by each ant at each step it makes on the composition graph according to the following formula [1]:

$$\tau_{ij} = (1 - \varphi) \times \tau_{ij} + \varphi \times \tau_0 \tag{5.7}$$

where $\tau_{ij}$ is the current pheromone level, $\varphi$ is the pheromone decay coefficient, and $\tau_0$ is the initial value of the pheromone.

The offline pheromone update is applied at the end of an ant's solution construction process as follows [1]:

$$\tau_{i,j} = (1 - \rho) \times \tau_{i,j} + \rho \times \Delta\tau_{ij} \tag{5.8}$$

where $\Delta\tau_{ij} = 1/S_{best}$, $S_{best}$ is the best score of a composition solution found so far and $\rho$ is the pheromone evaporation rate.

**5.2. The Selection Algorithm.** The selection algorithm (see Algorithm 5.2.1) takes as input a composition graph, the number of artificial ants used in the search process, the number of algorithm iterations and returns a ranked set containing the best composition solutions identified in each iteration. The algorithm adapts the Ant Colony Optimization (ACO) metaheuristic [1]. Like most ACO algorithms, the selection algorithm iteratively searches for the best composition solutions.

**Algorithm 5.2.1** Ant-based_Selection

**Input:**
$noIt$ - the number of iterations;
$noAnts$ - the number of artificial ants;
$G = (V, E)$ - the composition graph;
**Output:**
$SOL$ - the set of the best composition solutions;
**begin**
  $SOL = \emptyset$
  $noItModif = 0$
  $Ants =$ **Initialize_Ants**$(noAnts)$
  **while** $(noItModif < noIt)$ **do**
    $noItModif = noItModif + 1$
    $max = 0$
    **foreach** $a \in Ants$ **do**
      $result =$ **Find_Solution**$(G, a)$
      $minScore =$ **Find_Min_Score**$(SOL)$
      $maxScore =$ **Find_Max_Score**$(SOL)$
      **if** $(minScore <$ **Score**$(result))$ **then**
        $SOL =$ **Remove_Worse**$(SOL)$
        $SOL =$ **Add**$(SOL, result)$
        $noItModif = 0$
      **end if**
      **if** $(maxScore <$ **Score**$(result))$ **then**
        $max =$ **Score**$(result)$
        $maxSol = result$
      **end if**
    **end for**
    $G =$ **Apply_Offline_Pheromone**$(G, maxSol)$
  **end while**
  **return** $SOL$
**end**

Within an iteration, a set of artificial ants traverse the composition graph searching for composition solutions. In the first step, an ant is randomly placed on a graph node. Then, the ant will try to find input concepts for the node it is currently on. To do this, it will go backwards on an edge. When positioned on another node, the ant will try again to find appropriate inputs for that node. This operation will be performed until there are no more services whose inputs have no correspondents to outputs of other services. Once this is done, the ant will pick an edge in a forward direction from its current position node trying to move towards the output

node. From then on, the ant will pick edges outgoing from nodes that are not providing inputs to any node which is already present in the partial solution. Once a new node is picked, the ant will again go on back edges searching to provide all its inputs. This process is repeated until the ant reaches the output node and all the inputs of all the nodes are satisfied. The algorithm avoids adding a new edge if a cycle is about to be created in the composition graph. After all the ants manage to obtain a composition solution, the offline pheromone update is applied (see *Apply_Offline_Pheromone_Update* in Algorithm 5.2.1) on the edges of the best solution found so far. The best solution is identified based on Formula 5.4 (see *Score* in Algorithm 5.2.1). Finally, the algorithm returns the list of the best composition solutions found in each iteration.

Algorithm 5.2.2 illustrates how an artificial ant builds a solution. First, an empty solution is initialized. Then a random node is chosen followed by choosing an appropriate service from the corresponding cluster. The selected service is added to the partial solution. Then, all the services providing the required inputs for the partial solution are backward searched (see *Find_Inputs* in Algorithm 5.2.2). If the output node is not yet present in the partial solution, the algorithm performs a forward search to find another service (see *Find_Next_Service* in Algorithm 5.2.2) for which the current partial solution can provide inputs. The algorithm finally returns the found solution.

**Algorithm 5.2.2** Find_Solution

**Input:** $G$ - the composition graph
**Output:** *sol* - the found solution
**Comments:**
$Choose\_Random\_Node$ - picks a random graph node
$Pick\_Service$ - chooses a service from a cluster
$Is\_Solution$ - evaluates the partial solution
**begin**
  $sol = \emptyset$
  $N =$ **Choose_Random_Node**$(G)$
  $N.chosenService =$ **Pick_Service**$(N)$
  $sol =$ **Add_Node**$(sol, N)$
  **while** (**Is_Solution**$(sol) == false$) **do**
    $sol =$ **Find_Inputs**$(sol, N)$
      **if** (**Contains**$(sol, O) == false$) **then**
        $N =$ **Find_Next_Service**$(sol)$
      **end if**
  **end while**
  **return** *sol*
**end**

Algorithm 5.2.3 finds the services that provide inputs for the services that are contained in a partial solution. The algorithm is used recursively until all the inputs are provided. For a service in the partial solution, the algorithm first iterates through all the inputs which are not yet provided. It then finds a providing service, such that by adding the corresponding edge the graph remains acyclic. This service is chosen stochastically (see Formula 5.1), with the probabilities given by the pheromone level and heuristic information of the edges connecting the current service and the considered one. Then, the algorithm adds the service and the edge to the partial solution if they are not already present. These steps are repeated recursively until a partial composition solution is found.

The *Find_Next_Service* procedure in Algorithm 5.2.2 which is detailed in Algorithm 5.2.4 picks a new service to add to a partial solution with all the inputs provided. First, it determines the services which should provide inputs for the newly added one (see *Get_Nodes* in Algorithm 5.2.4). Since we want to go towards the output node, we will pick those services in the partial solution which are not already providing inputs for other services in it. We then determine the set of edges to consider: all the outgoing edges with the start service in the set previously determined, and the end service not in our partial solution (see *Get_Edges* in Algorithm 5.2.4). We then call a procedure that will choose an edge stochastically based on the pheromone levels and the heuristic information (see *Pick_Edge_Stochastically* in Algorithm 5.2.4). Next we add the edge and the end service to our solution and apply the local update procedure (see *Local_Update* in Algorithm 5.2.4).

**Algorithm 5.2.3** Find_Inputs

**Input:**
$sol_{part}$ - a partial composition solution
$s$ - the service where the ant is currently positioned
**Output:** $sol_{part}$ - the updated partial composition solution
**begin**
  **foreach** $a \in Ants$ **do**
    **repeat**
      $s' =$ **Find_Providing_Service**$(s.in)$
    **until**(**Edge**$(s, s') \notin Edges(sol_{part})$**or**
      **Acyclic**$(sol_{part} \cup$ **Edge**$(s, s')) == true)$
    **if** $(Edge(s, s') \notin Edges(sol_{part}))$ **then**
      $sol_{part} =$ **Add_Service**$(sol_{part}, s')$
      $a =$ **Pick_Service**$(s')$
    **end if**
    **Local_Update**(**Edge**$(s, s')$)
  **end for**
  **if** (**Unprovided_Inputs**$(s') \neq \emptyset$) **then**
    $sol_{part} =$ **Find_Inputs**$(sol_{part}, s')$
  **end if**
  **return** $sol_{part}$
**end**

**Algorithm 5.2.4** Find_Next_Service

**Input:** $sol_{part}$ - the partial solution; $G$ - the composition graph;
**Output:** $sol_{part}$ - the updated partial solution;
**begin**
  $candidateNodes =$ **Get_Nodes**$(G, sol_{part})$
  $candidateEdges =$ **Get_Edges**$(G, CandidateNodes, sol_{part})$
  $edge =$ **Pick_Edge_Stochastically**$(candidateEdges)$
  $sol_{part} = sol_{part} \cup \{$**Get_End_Node**$(edge), edge\}$
  $edge =$ **Local_Update**$(edge)$
  **return** $edge$
**end**

In the worst case scenario, when constructing a solution the entire composition graph is traversed, so the complexity of Algorithm 5.2.2 is $O(V + E)$. Based on this, the complexity of Algorihtm 5.2.1 may be estimated. Its complexity is $O(N \times A \times (V + E))$, where $N$ is the number of iterations and $A$ is the number of ants. $A$ will generally be a small constant. However, in the worst case scenario, $N$ can be as high as the number of possible solutions. In practice, as it will be shown in Section 6, this number is much lower, and $N \times A$ is often about 5% of the number of possible solutions. An alternative to this implementation (should the worst-case scenario be unacceptable) is to run for a fixed number of iterations. The algorithm is expected to perform well in such a case too, but care should be taken when fixing the number of iterations, as too few may result in bad solutions and too many will result in extra work. While in theory this will lead to a better complexity, in practice the currently chosen method works better.

**5.3. An Example of Selecting a Composition Solution.** Figure 5.1 presents the way in which an ant selects a composition solution. For simplicity, we suppose that there is only one service in each cluster and that at the beginning of the iteration the pheromone levels are the same on all the edges. The composition graph can be seen in subfigure 5.1-(a). In the first step, the initial random node is chosen - suppose it is $S_3$ (subfigure 5.1 - (b)). $S_3$ needs providers for its inputs, so it calls one of its providers in the composition graph. Such a node is $S_2$, which can actually provide all of the required inputs (see subfigure 5.1 - (c)).

The ant selects it and searches for a provider for the newly added node. The node $I$ is the only possibility because it provides all the inputs for $S_2$, so it is added to the solution (subfigure 5.1 - (d)). The ant can now go

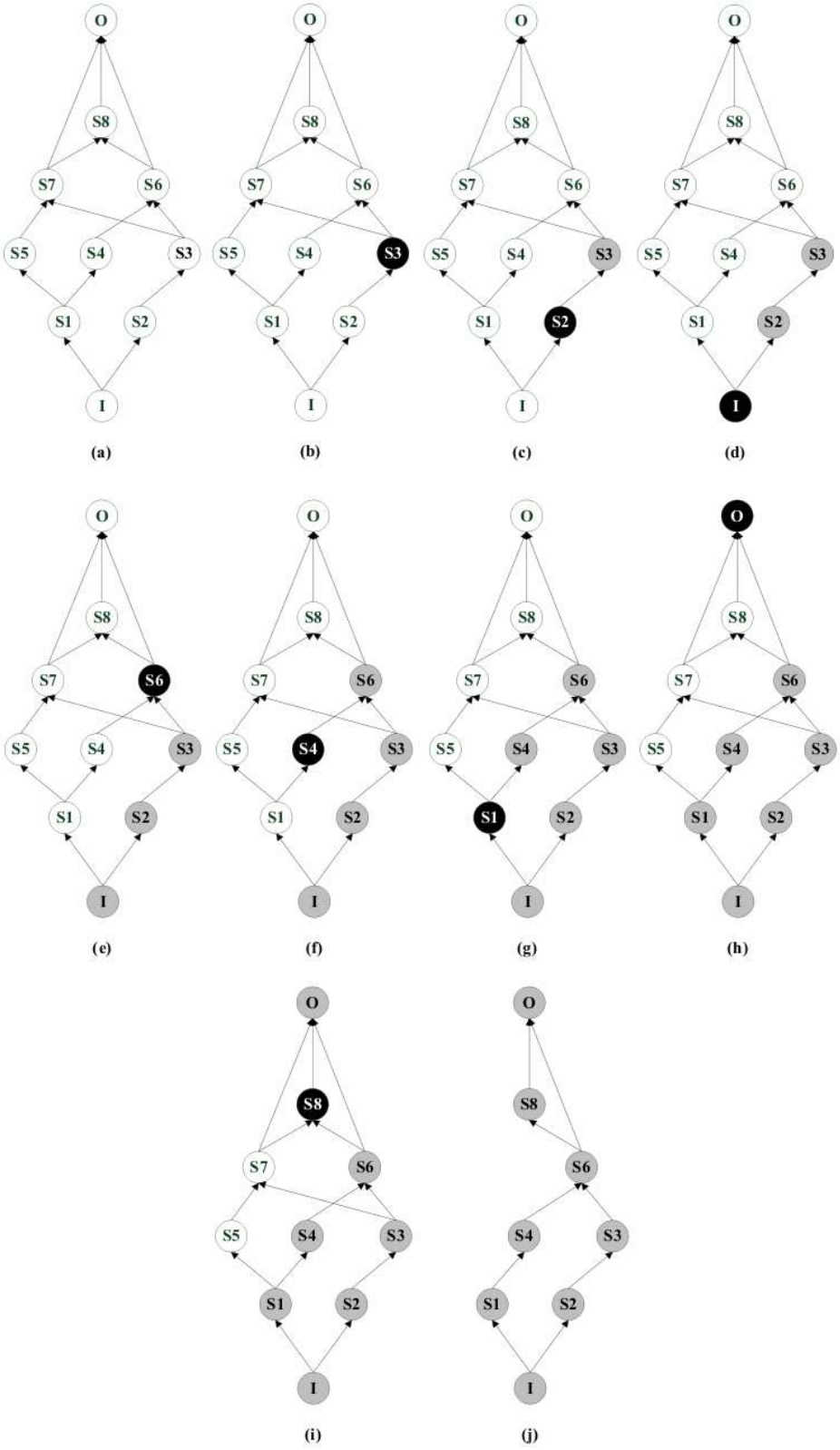



FIG. 5.1. *Example of Selecting a Composition Solution*

further, by looking for a node which requires the inputs that the current partial solution can provide. One such node is $S_6$ (subfigure 5.1 - (e)). Another possible choice would have been $S_7$, but the ant notices that $S_6$ has a better semantic matching. Also, the ant prefers $S_6$ to $S_5$ because its providers are not already supplying for nodes currently in the partial solution. The ant notices that $S_4$ can provide all the required inputs and therefore the ant adds the service to the partial solution (subfigure 5.1 - (f)). The ant looks again for nodes which can provide inputs for $S_4$, and it picks $S_1$ (subfigure 5.1 - (g)). The ant searches for inputs for $S_1$ and finds $I$, which is already in the partial solution. Then the ant goes forward again, and it can choose from $S_8$ and $O$. Suppose it picks $O$ (subfigure 5.1 - (h)) - because only part of its inputs are provided, the ant needs to search for nodes which can provide the others. $S_8$ is such a node, so the ant picks it (sub-figure 5.1 - (i)). Now the ant needs to search for inputs for $S_8$, and $S_6$ already provides them. $S_6$ can provide all the inputs necessary and is already in the solution, so no providers for its inputs need to be searched. Therefore, since both $I$ and $O$ are in the sub-graph and there are no free inputs, the ant can consider this to be a solution and stops searching (subfigure 5.1 - (j)). The score of the identified solution is computed, and, if necessary, the pheromones are updated.

**6. Evaluation and Testing.** This section describes the evaluation and testing methodology applied on the Ant-inspired composition framework as well as the way some adjustable parameters have been tuned.

**6.1. Evaluation Methods.** The main criteria according to which the Ant-inspired composition framework has been evaluated are its correctness and its efficiency. These framework properties will be expressed numerically in order to have an accurate estimation. To evaluate how well the framework works, we have used the following criteria: the quality of the provided composition solutions and the framework efficiency.

The quality of the provided composition solution is measured by counting how many solutions in the list of $k$ best found solutions should actually be there, i.e. how many of the best $k$ possible composition solutions were found by the selection algorithm. The ratio between the number of such solutions found and $k$ will show how correct the selection algorithm is.

To evaluate the efficiency of the composition framework, we counted the number of composition solutions encoded in the composition graph and compared this number to the total number of solutions returned by the selection algorithm for a given user request. The number of iterations has been set so that at least 50% of the best possibile $k$ solutions were found by the selection algorithm. The smaller the ratio between the selected number of solutions and the total number of solutions, the more performant our approach is.

**6.2. Test Service Collection.** We have tested our ant-based service composition technique on SAWSDL-TC [12], a benchmark containing 894 services, proposed for evaluating the performances of service matchmaking algorithms. This benchmark contains 894 Web services which are semantically annotated using concepts from several domain ontologies according to the SAWSDL specification. The services belong to the following domains: education, medical care, food, travel, communication, economy and weapon. In order to obtain a more complex composition scenario, a few extra services were added to the original set. Also, some services in the initial set were removed, as they referenced ontologies that were no longer available on the Web. In total, the set we worked with had around 700 services in it.

Because the semantic Web services from the SAWSDL-TC benchmark lack *QoS* attributes, we associated to each service a *QoS* description. When choosing the range (see Table 6.1) of the *QoS* values we were inspired by [13]. In computing the final score of a composition solution, each *QoS* parameter was normalized and received a score between 0 and 1, so that none of them would be given more importance than others by default.

TABLE 6.1
*QoS Value Ranges*

| **QoS Parameter** | **Range** | **Evaluation** |
|---|---|---|
| Response Time | 50 - 2000 | (2000 - value)/2000 |
| Availability | 50 - 98 | value/100 |
| Throughput | 1 - 12 | value/12 |
| Reliability | 60 - 75 | value/100 |
| Cost | 0 - 100 | value/100 |

**6.3. Test Scenario.** To evaluate our Ant-inspired service composition framework we have chosen a scenario from the medical domain. This scenario refers to a typical request from a person (identified by name,

address, country), which has certain symptoms and wants to go to a particular doctor. As a result, the person will be assigned to a hospital room (indicated by city, hospital name and room number) at a certain date. For the considered scenario the user request is presented in Table 6.2.

TABLE 6.2
*User Provided Parameters*

| Inputs | Outputs |
|---|---|
| PatientOntology.owl#PersonName | PatientOntology.owl#City |
| PatientOntology.owl#Address | PatientOntology.owl#Hospital |
| PatientOntology.owl#Country | PatientOntology.owl#Room |
| PatientOntology.owl#Symptom | PatientOntology.owl#TransportNumber |
| PatientOntology.owl#Physician | PatientOntology.owl#DateTime |

**6.4. Parameter Tuning.** By analyzing the experimental results we noticed that there are two adjustable parameters which strongly influence the composition method, namely the *cluster threshold* and the *concept threshold*. Based on our experimental results, the most appropriate value for the cluster threshold *clTh* proved to be 0.4. If this threshold value is lower, there will be too many edges, and if it is too high, there will be too few edges in the graph. The concept threshold *cTh* is used to determine if two concepts are similar enough to consider them as representing the same thing. This threshold is used both in the discovery method and in the edge generation procedure. Taking into account the values returned by the matching module, 0.25 was chosen as an appropriate value for the concept threshold parameter.

In the case of the selection method, the most important are the parameters related to pheromone evolution. A first parameter, the *initial evaporation*, is used to initialize the pheromone levels on the edges of the graph. It is currently set to 1, but as long as its value is strictly positive it does not influence the outcome in a significant way. Another parameter is the *pheromone evaporation rate*. This is the parameter used in the offline update, and determines the evolution of the pheromone from one iteration to the next one. We found that a value which allows the pheromones to vary in a satisfactory manner is 0.15. A third parameter is the *pheromone decay coefficient* used in the local pheromone update, which is applied by each ant as its constructs a solution on each edge it traverses. Therefore, its values must be smaller than that of the previous parameter, otherwise it would have a too large weight in pheromone evolution. We found that 0.05 is an appropriate value. Besides the parameters related to pheromone evolution, the number of ants and the number of iterations in which the global optimal solution has not changed should be tuned. Based on our experimental results we have set the number of ants to 4 and the number of iterations to 3.

**6.5. Results Analysis.** An example of services involved in the composition and selection processes for the considered scenario (see Table 6.2) is provided in Tables 6.3 and 6.4. All the concepts describing these services are part of a *patient ontology* from the SAWSDL-TC benchmark.

TABLE 6.3
*Example of Web service 1*

| **Diagnosis_TaxedPriceCostAndHealingPlan_Service.wsdl** | | |
|---|---|---|
| **Service ID**: 3 | **Inputs** | **Outputs** |
| **Cluster ID**: 2 | Diagnosis | TaxedPrice<br>CostAndHealingPlan |
| **QoS Parameters** | | |
| Running Time: 1367 | Availability: 74 | Throughput: 6 |
| Cost: 93 | Reliability: 69 | |

In our experiments, we have set all the weights (see Formula 5.4) used to calculate the score of a solution to 1. This way, we give the same importance to the *QoS* score and the semantic matching score. Also, within the *QoS* score computation, all the attributes count the same. For these parameters, the calculated optimal composition solution has a score of 0.8459. The algorithm returned four (see Table 6.5) of the possible five best results. That means that we had a success rate of 0.8. Also, out of the 1200 possible compositions, only about

TABLE 6.4
*Example of Web service 2*

| **Patient_HealthInsuranceNumberCommercialOrganisation_Service.wsdl** | | |
|---|---|---|
| **Service ID**: 13 | **Inputs** | **Outputs** |
| **Cluster ID**: 3 | Patient | HealthInsuranceNumber |
| | | Commercial Organization |
| **QoS Parameters** | | |
| Running Time: 1169 | Availability: 65 | Throughput: 6 |
| Cost: 73 | Reliability: 68 | |

90 were generated. In order to have a success rate of 0.5, 60 generated solutions would have been enough. That means that only a fraction 0.05 of the total number of solutions need to be generated to have relevant results. For about 100 iterations, we had the best solution among the list of retrieved solutions in more than 80% of the time. This leads us to the conclusion that the method achieves its purposes: provides the user with good composition solutions with a small computational cost.

**7. Conclusion.** In this paper, we propose an Ant-inspired Web service composition framework which uses a composition graph model combined with an Ant-inspired method to find the optimal service composition solution. The composition graph model stores all the composition solutions that satisfy a user request. The Ant-inspired method identifies the optimal or a near-optimal composition solution encoded in the composition graph model. The composition solutions are evaluated with a fitness function which considers the *QoS* attributes and the semantic quality as selection criteria. The ant-inspired composition framework has been tested and evaluated on an extended version of the SAWSDL-TC benchmark [12].

As future work, we intend to speed-up the selection algorithm by parallelizing the ants' searching process.

REFERENCES

[1] M. DORIGO, M. BIRATTARI, AND T. STTZLE, *Ant Colony Optimization, Artificial Ants as a Computational Intelligence Technique*, The IEEE Computational Intelligence Magazine, (2006).
[2] C. B. POP, V. R. CHIFU, I. SALOMIE, M. DINSOREANU, T. DAVID, AND V. ACRETOAIE, *An Ant-inspired Approach for Semantic Web Service Clustering*, Proceedings of the 9th Roedunet IEEE International Conference, (2010), pp. 145-150.
[3] C. B. Pop, V. R. Chifu, I. Salomie, M. Dinsoreanu, T. David, and V. Acretoaie, *Ant-Inspired Technique for Automatic Web Service Composition and Selection*, Proceedings of the 12th International Symposium on Symbolic and Numeric Algorithms for Scientific Computing, (2010), pp. 449-455.
[4] Y. DING, H. SUN, AND K. HAO, *A bio-inspired emergent system for intelligent Web service composition and management*, Knowledge-Based Systems Journal, Volume 20, Issue 5, (2007)
[5] L. YUAN-SHENG, ET AL., *An Improved Heuristic for QoS-aware Service Composition Framework*, Proceedings of the 10th IEEE International Conference on High Performance Computing and Communications, (2008), pp. 360 - 367.
[6] J. JUNG-WOON YOO, S. KUMARA, AND D. LEE. *A Web Service Composition Framework Using Integer Programming with Non-Functional Objectives and Constraints*, Proceedings of the 10th Conference on E-Commerce Technology and the Fifth Conference on Enterprise Computing, E-Commerce and E-Services, (2008), pp. 347 - 350,
[7] V. R. CHIFU, C. B. POP, I. SALOMIE, M. DINSOREANU, A. E. KOVER, AND R. VACHTER, *Web Service Composition Technique Based on a Service Graph and Particle Swarm Optimization*, Proceedings of the 2010 IEEE International Conference on Intelligent Computer, Communication and Processing, Cluj-Napoca (Romania), (2010), pp. 265-272.
[8] O. AYDIN, N.K. CICEKLI, AND I. CICEKLI, *Towards Automated Web Service Composition with the Abductive Event Calculus*, Applications of Logic Programming in the Semantic Web and Semantic Web Services, Seattle, (2006).
[9] O. AYDIN, N. K. CICEKLI, I. CICEKLI, *Automated Web Services Composition with the Event Calculus*, International Workshop in Engineering Societies in the Agents World, Athens, (2007).
[10] Z. WU, ET. AL., *Automatic Composition of Semantic Web Services using Process and Data Mediation - Technical Report*, (2007).
[11] M.M.AKBAR, E.G.MANNING, G.C. SHOJA AND S.KHAN, *Heuristic Solutions for the Multiple-Choice Multi-Dimension Knapsack Problem*, International Conference on Computational Science, (2001) LNCS 2074.
[12] SAWSDL-TC, http://projects.semwebcentral.org/projects/sawsdl-tc/
[13] The QWS Dataset, http://www.uoguelph.ca/ qmahmoud/qws/index.html

*Edited by:* Dana Petcu and Alex Galis
*Received:* March 1, 2011
*Accepted:* March 31, 2011

TABLE 6.5
*Top 4 composition solutions for the considered scenario*

| **Solution number** | **The services in the solution** | **Solution score** |
|---|---|---|
| 1 | PersonNameAddressCountry_Patient_Service; SymptomPhysician_Diagnosis_Service; Patient_HealthInsuranceNumberOrganisation_Service; PatientDiagnosis_HealthInsuranceNumberInsuranceCompany TaxFreePriceCostAndHealingPlan_Service; Diagnosis_TaxedPriceCostAndHealingPlan_Service; TaxedPriceCostAndHealingPlanHealthInsuranceNumber InsuranceCompany_CityHospitalRoomDateTime; CityHospitalRoomDateTime_TransportNumber_Service | 0.8459 |
| 2 | PersonNameAddressCountry_Patient_Service; SymptomPhysician_Diagnosis_Service; PatientDiagnosis_HealthInsuranceNumber CommercialOrganization TaxFreePriceCostAndHealingPlan_Service; Patient_HealthInsuranceNumberInsuranceCompany_Service; Diagnosis_TaxedPriceCostAndHealingPlan_Service; TaxedPriceCostAndHealingPlanHealthInsuranceNumber InsuranceCompany_CityHospitalRoomDateTime; CityHospitalRoomDateTime_TransportNumber_Service | 0.8433 |
| 3 | PersonNameAddressCountry_Patient_Service; SymptomPhysician_Diagnosis_Service; Diagnosis_TaxedPriceCostAndHealingPlan_Service; PatientDiagnosis_HealthInsuranceNumber CommercialOrganization TaxFreePriceCostAndHealingPlan_Service; TaxedPriceCostAndHealingPlanHealthInsuranceNumber InsuranceCompany_CityHospitalRoomDateTime; CityHospitalRoomDateTime_TransportNumber_Service | 0.8422 |
| 4 | PersonNameAddressCountry_Patient_Service; SymptomPhysician_Diagnosis_Service; Patient_HealthInsuranceNumberOrganisation_Service; Diagnosis_TaxedPriceCostAndHealingPlan_Service; PatientDiagnosis_HealthInsuranceNumber CommercialOrganization TaxFreePriceCostAndHealingPlan_Service; TaxedPriceCostAndHealingPlanHealthInsuranceNumber InsuranceCompany_CityHospitalRoomDateTime; CityHospitalRoomDateTime_TransportNumber_Service | 0.8422 |

Scalable Computing: Practice and Experience
Volume 12, Number 1, pp. 153–163. http://www.scpe.org
ISSN 1895-1767


# DISTRIBUTED FAULT SIMULATION WITH COLLABORATIVE LOAD BALANCING FOR VLSI CIRCUITS *

EERO IVASK, SERGEI DEVADZE AND RAIMUND UBAR†

**Abstract.** In this paper we present a web-based distributed fault simulation framework built with standard Java Applet/Servlet technology and the popular platform independent open source database MySQL. Fault simulation plays an important role in digital electronics design flow by creating and evaluating test patterns, identifying and locating the causes of the circuit failures. Because of the rapid increase of the circuit sizes, there is ever growing need for memory and computational power. Our solution allows to seamlessly aggregate many remote computers for one application. We propose a novel collaboration centric computing approach using a participant credit based priority concept. Issues of task partitioning, task allocation, load balancing and model security are also handled. Our primary goal during circuit partitioning has been the decrease of the needed simulation memory to allow working with larger circuits for which conventional simulation methods are not always usable. Our novel method of partitioning produces model slices with no interdependences, suitable for Internet based use. Larger scalability is achieved by using model partitioning along the test pattern set partitioning.

**Key words:** distributed computing, collaborative computing, fault simulation, critical path tracing, digital test

**1. Introduction.** The complexity of today's very large scale integrated (VLSI) circuits is still increasing according to Moore's law, which states that transistor density on a chip doubles about every two years. This trend is predicted to continue at least for another decade. The International Technology Roadmap for Semiconductors (ITRS) [1], an interesting assessment of the future technology requirements, states that number of logic gates in consumer level System-On-Chips (SoC) is 28 million at present and it is expected to increase 17 times by the year 2024. On the other hand, the number of test patterns required for high fault coverage of stuck-at-faults only, is at the moment about 30 thousand and is estimated to be 16 times higher respectively at the end of the roadmap.

The pattern count will increase as logic gate count increases. This fact influences both test development and test application times, and will be a major driver of overall development costs in the future. Automatic test pattern generation (ATPG) for structural test, design-for-testability (DFT) methods like random pattern logic built in self-test (BIST) and test pattern compaction are important techniques to reduce large amount of test data for logic cores [1]. Widely used procedure in these computation intensive test engineering tasks is fault simulation, which evaluates quality of the test patterns in terms of fault coverage. Within applications this intermediate step of the fault simulation usually needs to be carried out thousands of times, hence making simulation speed one of the key issues for overall task performance.

One solution for faster simulation, besides of improving fault analysis algorithms, would be to cooperate and combine available computing resources. We can assume that participants' computers are not optimally loaded all the time. The work could be more effective when every participant had a chance to run his tasks using collective resources, especially when it is possible to parallelize the task execution. Parallelization technique is application specific, though.

In case of fault simulation, there are several possibilities: algorithm itself can be parallelized, circuit model can be partitioned into separate components and simulated concurrently, fault set and test pattern set can be divided and simulated in parallel. Fault parallelism [2] and pattern parallelism [3] are easy to implement (data set simply needs to be splited) and have shown relatively good performance. Combining both of them has been proved even more effective: easy-to-detect faults are identified first with fast preprocessing and simulated in parallel among processors, remaining faults are targeted by all processors, each using only subset of test vectors corresponding to its partition [4]. Circuit partitioning has got less attention, but it is useful when number of gates is as big as it is today– modern fault simulators require a lot of memory to be efficient. When simulation model does not fit into memory, performance is drastically reduced or simulation is not possible at all. Parallel fault simulation with circuit partitioning was used in [5,6] for vector-synchronous implementations on message passing multiprocessor systems. Circuit partitioning approach for shared memory systems was presented in [7]. The method presented in [8] distributes the component models of the circuit partitions to unique processors

*This work was supported by Estonian SF grants 7068, 7483, EC FP7 IST project DIAMOND, ELIKO Development Centre and European Union through the European Regional Development Fund (Research Centre CEBE).
†Tallinn University of Technology, Estonia ({ieero|serega|raiub}@pld.ttu.ee).

of a parallel processor system for concurrent and asynchronous execution. Partitioning issues are not handled here, however a manual partitioning was supported.

This paper presents a loosely coupled asynchronous Internet-based fault simulation approach, relying on model parallelism and test parallelism. Fault detectability computation model for the circuit and test pattern set are divided. Sub-sets of test are evaluated on partial computational models concurrently on different computers in wide or local area network. Our approach has no specific fault list as faults are already in the simulation model. During the model partitioning some overlapping is likely to occur as it is better to avoid interdependences, because of the communication penalties. These repetitive model parts are simulated several times. Results of fault simulation on partial models will be accumulated in overall fault coverage.

The concept of the current solution was initially inspired from MOSCITO framework [9,10] used for example in European VILAB cooperation project. The original system was implemented as client-server Java application. It allowed invoking single work tools remotely and organizing them into predefined automated workflows. Users in different locations cooperated via sharing their software tools in joint workflows. However, parallel execution was not supported. Moreover, the major limitation for widespread Internet based use was TCP/IP socket based communication schema that required opening dedicated communication ports in firewalls not only on server side, but also on the client side for each application and for end users.

A flexible web-based solution for remote tool usage following some key ideas of MOSCITO was proposed in [11]. The socket communication was replaced with HTTP, Java Servlets were used on server side and Applets along with Java applications were used on client side and also a database was introduced. In the current paper, this concept is revised and improved to support distributed computing and collaboration via effective resource sharing and load balancing.

There exist also several general purpose frameworks for distributed computing like BOINC [12], Globus [13], AliCE [14] and Condor [15] to name a few. Most popular seems BOINC (Berkeley Open Infrastructure for Network Computing), a non-commercial middleware system for volunteer computing, originally developed to support the SETI@home project, but intended to be useful for other applications in areas as diverse as mathematics, medicine, molecular biology, climatology, and astrophysics. The intent of BOINC is to make it possible for researchers to tap into the enormous processing power of personal computers around the world.

A major part of BOINC is the backend server. The server can be run on one or many machines to allow BOINC to be scalable for projects of any size. BOINC servers run on Linux based computers and use Apache, PHP, and MySQL as a basis for its web and database systems. Its framework uses cross-platform WxWidgets toolkit for building GUI-s. BOINC is a software that provides an infrastructure for distributed applications. It is capable to download of input data (work units), schedule of multiple BOINC projects on the same CPU, and provides a user interface to the integrated system. Scientific computations are run on participants' computers and results are analyzed after they are uploaded from the user PC to a science investigator's database and validated by the backend server. The validation process involves running all tasks on multiple contributor PCs and comparing the results.

The major drawback of the BOINC system is the use of remote procedure call (RPC) mechanisms which is often felt to be security risk, because they can be the route by which hackers can intrude upon targeted computers (even if it's configured for connections from the same computer). The use of PHP over Java cannot be considered as an advantage.

Globus is a collection of libraries and programs that address common problems that occur when building distributed services and applications. Issues related to security, resource access, management, discovery, data transfer, service deployment, system components monitoring and user control are handled. Globus toolkit makes extensive use of Web Services [16] to implement these infrastructure services. A Web service is a software system designed to support interoperable machine-to-machine interaction over a network. It has an interface described in a format that a machine can process (WSDL). Other systems interact with the Web service in a manner prescribed by its description using Simple Object Access Protocol (SOAP [17]) messages, typically conveyed using HTTP with an XML serialization in conjunction with other Web-related standards [16]. Initially, the work on Globus was motivated by the demand of virtual organizations in science, and then business applications became also important. Now, Globus is deployed in many large projects like TeraGrid [18], Open Science Grid [19], LHC Computing Grid [20], etc. Globus services are used to support different communities, each of them executing their own application specific code on top of those services. The disadvantage of Web service based solutions is their reliance on XML markup notation. XML is nicely readable to human being and easy to parse for computer programs, however it requires more processing power and network bandwidth than binary-coded

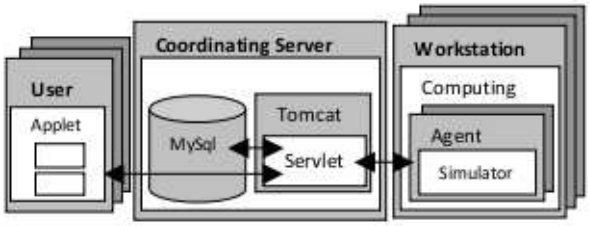



FIG. 2.1. *System components and communication*

formats and protocols.

AliCE, developed in National University of Singapore, attempted to become a grid development system instead of just being collection of grid tools. Similarly to Globus, AliCE core layer has components for resource management, discovery, and allocation, data management, monitoring and accounting, communication and security infrastructure. On top comes extensions layer consisting distributed shared memory programming templates, runtime support infra-structure, and advanced data services. The running system has consumer, producer and resource broker entities. The consumer submits the application code to the grid, the resource broker directs the application to appropriate task farm manager, which initiates the application and creates a pool of tasks. Task references are returned to the resource broker, which schedules the tasks for execution on producers. The results are returned to the consumer. Communication infrastructure supports the migration of codes, data and results via "Space"– a special form of shared memory. Communication is carried out with objects. Objects and code are serialized and packed into jar archive fail – obviously in order to reduce the amount of transferred data. AliCE is based on Java Jini technology [21] and JavaSpaces [22]. Java Native Interface (JNI) is used to invoke non-Java code. The authors have used the system in several projects like [23], but it seems that activity around AliCE has lost its momentum at present. Technology itself is still promising. Only drawback in our point of view is that Jini technology is based on Remote Method Invocation (RMI)– although elegant programming solution for distributed computing, were one program can remotely invoke methods physically residing in other machine, however, firewall traversal can be problematic as dedicated communication ports are needed. Strict security policy might not allow that.

Condor is another project providing support to high-throughput computing, allowing users to take advantage of idle machines that they would not otherwise have access to. Condor is suitable for executing long running jobs that require no user interaction. Application source code must be recompiled with special libraries. Condor is useful to do task allocation and load balancing; it does not provide task decomposition however.

The rest of the paper is organized as follows: the overall concept of web-based infrastructure is described in section 2. In section 3 the priorities concept is explained. Task partitioning and task allocation are handled in sections 4 and 5. Section 6 considers the circuit model security. Section 7 presents workflow with distributed computing. Experimental results are presented in section 8 and conclusions are given in section 9.

**2. Distributed environment.** Our Web-based infrastructure is built on Java Applet/Servlet technology [24] and popular platform independent open source relational database MySQL [25]. The communication flow between system components and implementation details can be seen in Fig. 2.1. Servlet is a Java application that runs in a Web server or special application server and provides server side processing like different calculations, database access, e-commerce transactions, etc. Servlets are designed to handle HTTP requests and are the standard Java replacement for a variety of other methods, including CGI scripts, Active Server Pages (ASPs) and proprietary C/C++ plug-ins for specific Web servers (ISAPI). Because servlets are written in Java, they are portable between servers and operating systems. The servlet programming interface (Java Servlet API) is a standard part of the Java EE (Enterprise Edition of Java), the industry standard for enterprise Java computing. Tomcat [26] is open source servlet container (application server software) which is one way to run Java Servlets. Tomcat is developed by the Apache Software Foundation (ASF) and implements the Java Servlet and the JavaServer Pages (JSP) specifications from Sun Microsystems, and provides a pure Java HTTP web server environment to run a Java code.

In conclusion, Tomcat and servlets running on it play important role in order to access our intranet resources on workstations and the MySQL database. It is simple and lightweight alternative to other full blown enterprise scale solutions.

**2.1. Data management.** The data handling takes place on coordinator node. The problem is that web-based HTTP communication is stateless and the session is valid for short time only, but the simulation process may run much longer. Therefore, the users must be identified; their tasks and results must be stored for later access.

The data module has three layers: presentation (user interface), business-logic (database queries, data processing) and physical database. First two layers are implemented in Java. The user is accessing database only via presentation layer, which consists of several functions to run middle layer queries. The database access is implemented according to Data Access Object (DAO) design practice. The data access objects manage access to relational databases. Each table in a relational database corresponds one Java class. The database table attributes map to Java class properties. For each property, there exist "set" and "get" methods. Additionally, DAO class has methods to insert, update and query the records in the database tables. For example, for table "Tasks", we will have at least properties like "taskId", "userId" and "status"; methods could be like "getTaskId", "setStatus", "insertTask", "getCompletedTask", etc. The standard Java mechanism for accessing databases is using Java Database Connectivity (JDBC) API. For convenience purposes, we have captured basic DB connection code into single DB access class and every specific DAO class, like "TasksDAO" class, extends that class i.e. basic connection methods are inhereted and used inside the class.

Alternatively, it could be possible to use the popular Hibernate [27] and Spring [28] frameworks to simplify objects to relational DB mapping (ORM). For large and mission critical projects also Java EE technology like Enterprise JavaBeans is available [29]. However, for simple data persistency in current situation, the proposed solution is adequate and was faster to implement. Setting up and closing DB connections is time consuming operation, therefore we have used Tomcat's native connection pooling to speed up DB transactions.

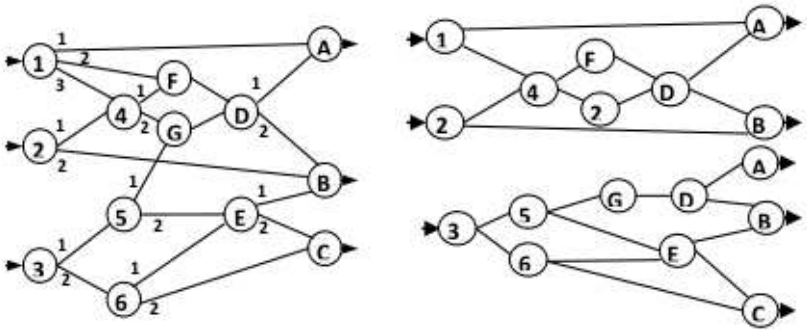
**2.2. Communication.** The use of applet/servlet approach implies that the general communication is based on HTTP protocol. The tools on different computers and on different computing platforms (UNIX, Linux, Windows) can easily exchange data as serialized Java objects. Data passing between components is implemented following Transfer Object (TO) design practice. A transfer object is a lightweight version of DAO object, it has only properties and "get" and "set" methods. Information is sent as data bundle as opposed to single strings.

HTTP protocol allows us also easy firewall traversal as we can use default web server port and Java servlet extensions on web servers as sort of proxies in order to reach intranet resources. There is no need for opening extra ports in the firewall on the user side as it is the case in TCP/IP socket based communication or when relying on Java RMI (which would be major restriction). Communication can be secured via SSL encryption by appropriate modifications in Tomcat configuration file, when necessary.

**2.3. Tool encapsulation.** Our simulator is implemented in C language, it has no graphical interface and network communication abilities. In order to integrate it into web based environment, it is necessary to implement additional software (Agent) layer. The simulator will be invoked from Java program (Agent), which allows to adapt the input data, convert the tool-specific data, retrieve the simulation results (log files, test vectors, etc.), map the control information to the embedded tool, transfer the status information (warning and error messages) to be submitted to the user, etc. The technically simplest way is to encapsulate the simulator as an entire program. Generally, integration of other work tools is possible similar way. Tool has to be able to run as a batch job. Also embedding of a library (e.g. C, C++ routines) via the Java Native Interface (JNI) is possible and also direct integration of Java-classes and applications (for Java software).

**2.4. Graphical user interface.** User interface (GUI) is based on Java Applet, which can be integrated into HTML page when needed. Java applets are very versatile in features and easy to develop. For rapid prototyping we have used NetBeans IDE [30], which supports visualized GUI development with drag and drop operations. Final tweaks to generated code still had to be done manually.

User GUI has fields to gather test tool's parameters, allows browsing for circuit model file, has button to start the tool, a console window to display all the messages from the running tool. When the task is complete, results download is enabled. The user can browse and select the folder where to save results. Since local hard drive access for usual Java applets is restricted for security reasons, therefore the GUI applet had to be signed digitally. We used so called self-signed certificate for simplicity. The certificate shows owner specific information. The only difference for end user is that when signed Applet is first time downloaded into user's computer, the informative dialog box is displayed. It is user's responsibility to verify the origin and content of




(a) Reconvergency graph $G$ of a circuit

(b) Partial reconvergency graphs $G_1$ and $G_2$

FIG. 4.1. *Reconvergency graphs*

the Applet. The user can contact Applet owner about authenticity of certificate, when neccessary. The Java Runtime Environment (JRE) is required to run the GUI Applet.

**3. Priorities.** One important prerequisite for collaborative computing is that performance of each participant is enhanced overall; nobody should experience unfair lack of the computing power, especially these participants who previously have donated the most of their resources. This implies we have to introduce a priority system based on each participant credit. The credit for host computer is calculated:

$$Credit = Credit + Mode \cdot HostPerf \cdot CompTime \cdot LoadFraction, \tag{3.1}$$

where Mode equals "1" when computer is donating and "-1" when consuming; *HostPerf* referes to performance index of the computer, it is calculated initially after executing sample calibration task; *CompTime* is the time host computer has devoted to the current task; *LoadFraction* is number between 0 and 1, inclusively. Credit rating of the host increases while other participants are consuming its processor time, at the same time ratings of the quests will decrease the same amount. Credit numbers can go negative, it shows who is contributing and who is not. Participant with higher credit rating will have higher priority.

**4. Task partitioning.** Generally, there are several ways to parallelize the fault simulation: the algorithm can be parallelized, the circuit model can be partitioned into separate components and simulated in parallel, the fault set can be partitioned and simulated in parallel or the test pattern data can be partitioned. In this paper, we rely on model parallelism and test set parallelism, faults being included already in our simulation model.

Our method of fault simulation based on the parallel exact critical path tracing, together with the detailed method of model partitioning can be found in [31]. The simulation model is created by a special topological analysis of the circuit. Each reconvergent subnetwork of the circuit produces a sequence of computation formulas, and the nested reconvergent subnetworks produce nested computation formulas. The whole set of formulas created in the process of the topological analysis of the circuit composes the simulation model. This model allows carrying out fast efficient fault simulation of the circuit where the number of repeated computations is minimized. In [31] it is shown how to divide the process of fault simulation driven by the simulation model into a number of parallel sub-processes. For each sub-process, a partial simulation model is constructed. The creation of such a set of independent simulation models gives us the opportunity to use a distributed environment to achieve higher speed of fault analysis.

As an example, in Fig. 4.1(a) the topology of a given circuit is shown where the nodes represent fan-out stems and/or reconverging fan-ins, and the edges represent signal paths of the circuit through other nodes of the gate-network of the given circuit. Such a graph is called a reconvergency graph of the given circuit that is the basis for constructing of the simulation model. Reconvergency refers to the case when two signal paths in the circuit have the same origin, and converge in the same node.

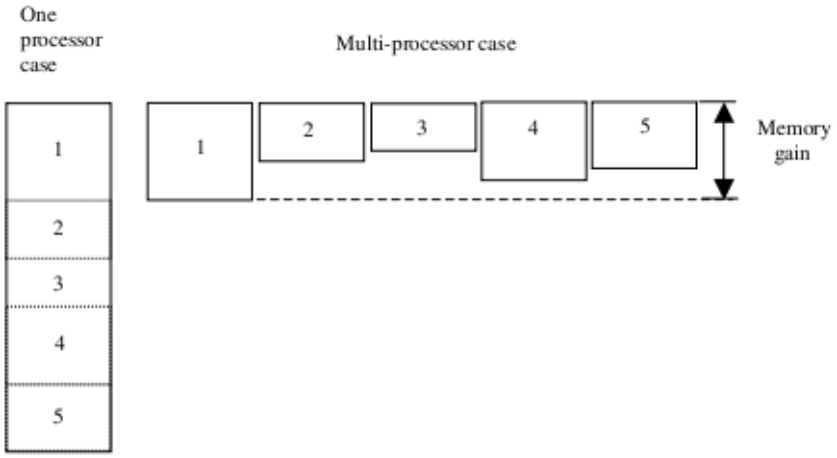



FIG. 4.2. *Memory distribution between processing units*

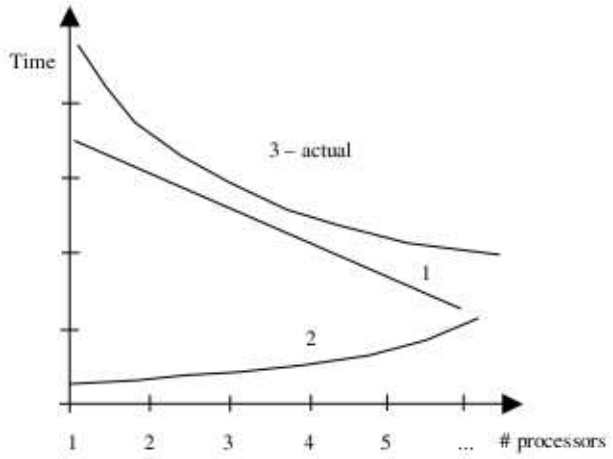



FIG. 4.3. *Components of the simulation time cost*

Fig. 4.1(b) shows a possible solution of dividing the reconvergency graph in Fig. 4.1(a) into two partially overlapped reconvergency graphs which allow creating two simulation models to be processed independently in parallel preserving all the advantages of parallel fault simulation by critical path tracing.

Fig. 4.2 illustrates the model partitioning. On the left, one processor case is depicted– all imaginary pieces one to five, are simulated subsequently. On the right, a multiprocessor case is depicted. We see the same model pieces distributed along processors. Largest model piece determines the actual simulation time, i.e. the time the user will have to wait. Also maximum reduction in memory requirement is determined by largest model piece ("1" in our example). Because of the overlapping, total amount of the memory used by simulation system will increase, but on a single computer required memory amount will decrease. It would be possible to find the optimal selection of partitions to minimize the sizes of overlapped areas. We have used randomized technique here. First we select nodes randomly into partitions, thereafter we count the number of the overlapping nodes. Then we generate a new partitioning and evaluate the number of overlapping nodes again. On each iteration, we keep the best solution. The number of iterations is a function of the number of primary inputs of the circuits. On the Fig. 4.3, there is abstractly shown how actual simulation time is influenced by nonlinearity of component 2– time increases due to calculation model pieces overlapping. Component 1 would be the ideal simulation time.

**5. Task allocation.** In real-life situations tasks are accumulated and then scheduled in batches [32]. Optimal task allocation ensures that all the computers stop computing at the same time. Several aspects have to be considered: very slow performance machines should get only a small fraction of total load; workstation,

which has fault, should get no load at all. Distribution of the tasks should happen within short time interval. The scheduler has to be fast not to delay the execution of the tasks. The scheduling objective is to minimize overall execution time, which is actually time of the longest executing subtask.

There are possibly two conflicting goals while allocating tasks to different computers: 1) maximize the speed of particular task execution for particular user 2) maximize system overall throughput from the perspective of all users. In the case of collaborative computing we assume that the latter is more important. Generally, task allocation can depend on several factors that may change in time: number of application users, number of computers, computers workload and speed of the communication links.

Let's assume we have the system depicted in Fig. 2.1. There is one central Coordinating Server, several users and several simulation Agents. The work of Agents is controlled by a Coordinator instance running on the Server as a Servlet. Initially, idle simulation Agents are polling the Coordinating Server. Each Agent announces its allowable load threshold, its current load and current performance index. Performance indexes are determined by host profiling i.e. Agents execute a small sample task and measure the completion time. Thereafter, Agents prepare their profiling messages for the Coordinator. These messages include also time stamps, so the Coordinator can measure the transfer time and determine network latency indexes for different Agents (it is assumed that clocks of the hosts are synchronized). The Coordinator collects the information initially for some predetermined time, obtained coefficients are applied later– faster machines will get larger tasks respectively.

Let's assume now that tasks are already submitted to the Coordinator. The scheduler takes the task with highest credit rating and selects an Agent with least load index in the list, given that agent's load is smaller than allowed threshold. Thereafter, the scheduler takes the task with next highest credit rating and searches for next Agent satisfying the threshold condition above. This repetitive process continues until there are no tasks in the list. The Agent selection idea in this intuitive scheduling strategy corresponds to Best-fit algorithm described in [33] and used for memory allocation problem.

When a higher-priority task is submitted to scheduler, and there is no idle Agent available in the system, then the scheduler has to suspend or reallocate some of the current tasks if expected completion time of the priority task is greater than migration time of the current task. The candidate task for migration is the one who has the lowest priority and longest execution time. Migration means that Agent notifies the Coordinator and sends intermediate result of simulator tool to the Coordinator and thereafter the Agent terminates the simulation task. The scheduler finds new target candidate for the task under migration– it will be the Agent with the lowest credit rating and with the best performance rating.

The task must be also migrated when the load on the host increases above allowable threshold, i.e. if the local computing activity increases, as our goal was that participants in collaboration may not suffer. Exception is the case when such task belongs to the host owner. The load threshold is adjustable by each participant. Further information on load balancing can be found for example in [34,35,15].

**6. Circuit model security.** In this paper, we assume that the server host is trusted– circuit models must be uploaded first. Other hosts of participants can be less trusted, because to a certain extent, we can overcome the model security problem automatically by partitioning the model itself– partial model is not very valuable for potential malicious agent. We may even enforce that model pieces are scheduled to the same processors during repetitive executions of the same circuit model, disabling the potential collection activity. However, identifying the right pieces of model must be handled. It is possible to compute the hash value of the model piece and save it to database. When later within limited time frame same hash value is computed for some model piece, then this task is scheduled to the same processing unit. Less limiting strategy is to ensure that not all the model pieces will be scheduled to the same processing unit. Malicious collector still will have no complete puzzle. Let us note that it is much harder to assemble the meaningful full model from pieces than just to recognize the same model piece. Computation of hash values could be performed in parallel with separate execution threads on multiprocessor computer.

**7. Simulation workflow.** First, the user specifies parameters and design file location for simulation tool (see Figure 2.1). In addition, the size of the simulation task can be predefined by user. Thereafter the user GUI contacts with the Coordinator Server and described parameters along the model are passed automatically. The task coordinator process (Java servlet) records all requests from user(s) to the database. The Java based test Agents poll constantly the Coordinator and if any subtask is scheduled by the Coordinator process, then the Agents receive the appropriate parameters and model file and will start actual native simulator tool executable.

TABLE 8.1
*Distributed simulation results.*

| Circuit | B17 | B18 | B21 | B22 |
|---|---|---|---|---|
| Max model partitions | 13 | 8 | 12 | 13 |
| Max model build, s | 0.24 | 1.83 | 0.32 | 0.37 |
| Max subtask simul,s | 214 | 1534 | 146 | 195 |
| Subtask simul dev, % | 21.0 | 24.4 | 15.7 | 5.5 |
| Model size reduction | 4.1 | 2.8 | 2.5 | 2.6 |
| Speedup by model part. | 3.2 | 6.4 | 2.5 | 2.9 |
| Speedup by test part. | 10.3 | 7.9 | 8.7 | 10 |

The simulator first constructs simulation model taking into account of the size limit of the subtask. While reaching the limit, it saves the breakpoint information into local file system. The simulation Agent then reads the breakpoint information and passes it to the Server where it will be stored for other simulation agents. When the next idle Agent is polling, then it will receive the circuit file along parameters and breakpoint information. The Agent starts a new copy of simulator tool on the other computer. The simulator first constructs the simulation model again, but this time it is not starting from beginning, but restores from the breakpoint up to the point when task size limit will be reached. New break point information will be again saved to local file system and later passed to the Server database by the Agent. The simulator Agents then wait until their subtasks will be completed and report results back to the Coordinator.

The process repeats until there are no simulation sub tasks left. Note that simulators have been started subsequently, but thereafter they run concurrently. Total starting delay is small compared to runtime. Finishing order of simulation sub-tasks is not pre-determined as simulation speed depends on the piece of the circuit model– some pieces are more complicated to simulate and need more time.

When all simulators are finished, then the Coordinator assembles sub results into final result and stores in the database. Results are passed to user when requested.

**8. Experimental results.** In experiments we measured communication overhead, memory reduction by circuit partitioning, simulation speedup and scalability with current task partitioning when number of processing units increases. Simulation was carried out on UltraSPARC IV+ 1500Mhz servers. Tomcat servlet engine and MySQL database were running on two core AMD Athlon 64 6000+ 3 GHz processor with 2 Gb memory. User applet was also running on the similar Athlon machine. Experiments show, that on the user computer circuit file loading takes about second for the over 3 Mb size circuit files depending on computer's performance. File transfer to the database and user notification takes about 6 seconds. Thereafter, simulation agent receives circuit file from the Coordinator 3-4 seconds later. Total communication delay was 12-15 seconds in case of distributed web-based solution. The total communication overhead was about 1% compared to single processor solution in the case of largest circuits. The overhead depends on the size of the circuit and the number of test vectors simulated.

Simulation results for sample circuits [36] are presented in Table 8.1. 100K test patterns were applied to each circuit. We can see that model build time for subtask (circuit slice) is very small (0,1% for b18 circuit) compared to simulation time.

Final simulation time is dominated by the longest subtask simulation time. We see that there is some deviation from ideal mean time. This implies that model partitioning could be still improved– model slices could be more balanced in size. This would lead to more equal and shorter simulation times and user would get final result faster. However, the possibilities of balancing the partitioning of the model depend essentially on the circuit structure.

The last rows of Table 8.1 present simulation speed-up for the simulation distributed on several processors compared to single processor local simulation. Scalability in case of model partitioning is degrading due to model pieces overlapping. For the purpose of fair comparison, the speedup results in rows 6 and 7 are calculated for the same number of partitions (first row in Table 8.1) for both types of partitioning.

Fig. 8.1 represents the simulation time dependence on model partitioning. The number of processors equals to the number of model pieces after partitioning. For better comparison, circuits only with similar model partition count were selected. Fig. 8.2 shows that the total model build time used for sub-task simulation is growing linearly with the increasing number of processors. Fig. 8.3 illustrates that the total memory used in order to simulate all sub-task models on processors, is also growing linearly depending on how many partitions

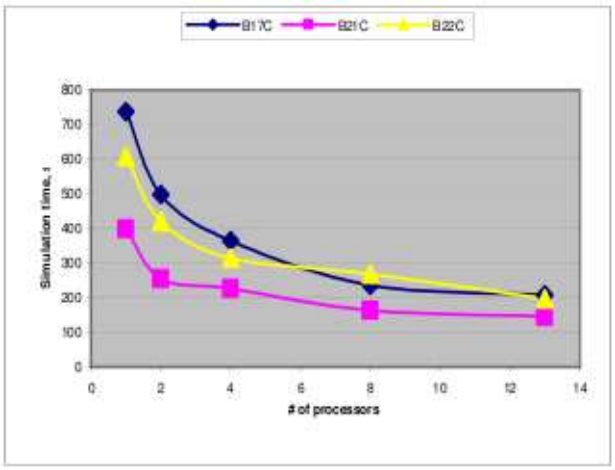



FIG. 8.1. *Simulation of 100K vectors*

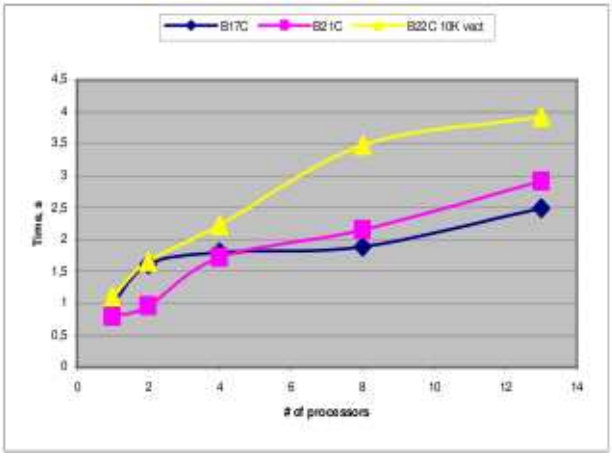



FIG. 8.2. *Total model build time*

are used. Figure 8.4 illustrates results for largest (104580 logic gates) B18 circuit. Model partitioning speedup is quite close to test partitioning speedup. Initially, up to 6 processors model partitioning has an advantage compared to test partitioning. On the same figure, also a rather linear reduction in memory requirements is depicted.

**9. Conclusion.** We have presented how distributed computing paradigm can be used collaboratively in the field of digital fault simulation. In contrast to existing solutions, we have developed a novel Internet based loosely coupled system, which allows seamlessly aggregate computers of dislocated working groups into one powerful simulation application. We have introduced a credit based priority management method, addressed task scheduling and task partitioning problems, handled model security. Our primary goal during task partitioning has been the decrease of circuit simulation memory requirement. Generally, there are three ways to achieve this goal: parallelizing circuit model, dividing test set and partitioning the fault set. In case of our solution, simulation model, derived from circuit model, contains already fault information. Therefore, we are partitioning only simulation model and test set. In contrast to previous parallel simulation solutions we have introduced a circuit partitioning method that is suitable to be used in systems with communication delays like the Internet based distributed simulation environment, as our partitioning method will produce model slices with no interdependences.

In experiments, circuit partitioning has been proved to be useful as it reduced required memory up to 4 times and at the same time the simulation was speed up 3 times compared to single processor simulation in

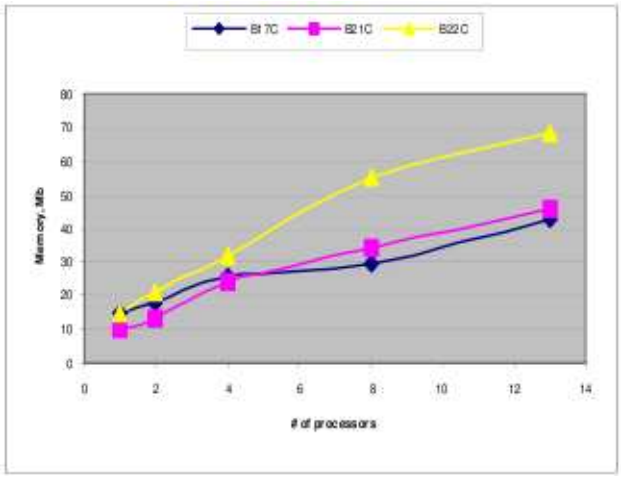



FIG. 8.3. *Total memory usage*

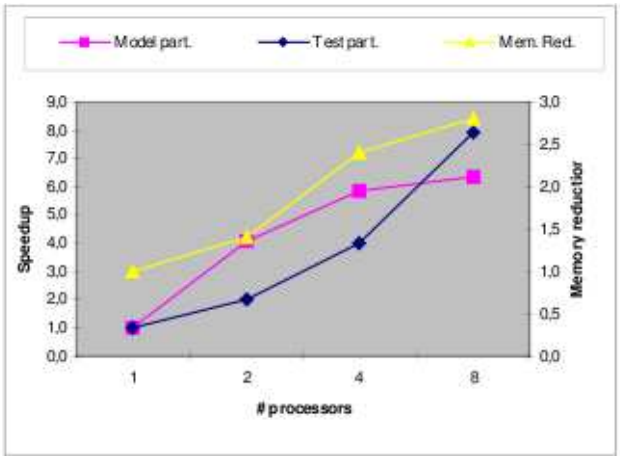



FIG. 8.4. *Circuit B18 experimental results*

case of the circuit B17, for example. Model partitioning is also useful in sense that it offers basic circuit model security since entire model needs to be handed over to central server only. The server however can be well secured.

In experiments, the total communication delay for end user was approximately 12-15 seconds in average compared to local single processor solution. The communication overhead depends on the size of the circuit and the number of test vectors simulated. In practice, circuits and vector sets are much larger - the overhead would be consequently smaller. The proposed collaborative computing approach allows to solve larger fault simulation problems which otherwise would be difficult to achieve due to memory and time constraints with conventional simulation approach. Combining model partitioning with test partitioning allows achieving a better scalability.

Model partitioning algorithm could be possibly improved in the future to support finer granularity. However, combined with test set partitioning, it is already sufficient for moderately sized collaborative consortium, since in reality there is usually more than single circuit model in use at given time, consequently total number of tasks to be scheduled may be larger and appropriate for total number of computers available. In the future also tradeoff estimation between model partitioning and test set partitioning could be introduced. Design pattern proposed in current paper can be easily used for other distributed applications, only task partitioning is specific.

## REFERENCES

[1] *ITRS 2010 technology roadmap*, http://www.itrs.net/Links/2010ITRS/Home2010.htm

[2] Han and S. Y. Lee, *A Parallel Implementation of Fault Simulation on a Cluster of Workstations*, in Proc. IEEE International Symposium Parallel and Distributed Processing IPDPS, 2008, pp. 1–8.
[3] M. B. Amin and B. Vinnakota, *Data Parallel-Fault Simulation*, IEEE Trans. VLSI Systems, vol. 7, no. 2, pp. 183–190, Jun. 1999.
[4] E. M. Rudnick and J. H. Patel, *Overcoming the serial logic simulation bottleneck in parallel fault simulation*, in Proc. 10th Int. Conf. VLSI Design, 1997, pp. 495–501.
[5] R. B. Mueller-Thuns, D. G. Saab, R. F. Damiano, and J. A. Abraham, *Portable parallel logic and fault simulation*, in Proc. Int. Conf. CAD, 1989, pp. 506–509.
[6] J. F. Nelson, *Deductive fault simulation on hypercube multiprocessors*, in Proc. 9th ATT Conf. Electronic Testing, 1987.
[7] S. Patil, P. Banerjee, and J. Patel, *Parallel test generation for sequential circuits on general purpose multiprocessors*, in Proc. 28th ACM/IEEE Design Automation Conf., San Fransisco, CA, 1991.
[8] S. Ghosh, *NODIFS: A noval, distributed circuit partitioning based algorithm for fault simulation of combinational and sequential digital designs on loosely coupled parallel processors*, LEMS, Division of Engineering, Brown University, Providence, RI, Tech. Rep., 1991.
[9] *MOSCITO*, http://www.eas.iis.fhg.de/solutions/moscito
[10] A. Schneider et. al. *Internet-based Collaborative Test Generation with MOSCITO*, in Proc. DATE02, Paris, France, March 4-8, 2002, pp.221–226.
[11] E. Ivask, J. Raik, R. Ubar, A. Schneider, *WEB-Based Environment: Remote Use of Digital Electronics Test Tools*, In Virtual Enterprises and Collaborative Networks, Kluwer Academic Publishers, 2004, pp. 435–442.
[12] *BOINC*, http://boinc.berkeley.edu/
[13] Ian Foster, *Globus Toolkit Version 4: Software for Service-Oriented Systems*, Journal of Computer Science and Technology,vol. 21, no.4, Jul. 2006, pp. 513–520.
[14] Y.M. Teo and X. B. Wang, *AliCE: A Scalable Runtime Infrastructure for High Performance Grid Computing*, in Proc. IFIP Int. Conf. Network and Parallel Computing, Springer-Verlag Lecture notes in Computer Science, Wuhan, China, October 2004.
[15] *Condor*, http://www.cs.wisc.edu/condor/
[16] D. Booth, H. Haas, F. McCabe et. al., *Web Services Architecture*, W3C Working Group Note, 2004. http://www.w3.org/TR/ws-arch/
[17] *Simple Object Access Protocol (SOAP)*, http://www.w3.org/TR/soap/
[18] *TeraGrid*, http://www.teragrid.org/about/
[19] *Open Science Grid*, http://www.opensciencegrid.org/
[20] *Large Hadron Collider (LHC) Computing Grid*, http://public.web.cern.ch/public/en/lhc/Computing-en.html
[21] *Java Jini Technology*, http://www.jini.org/wiki/
[22] *JavaSpaces*, http://www.jini.org/wiki/JavaSpaces_Specification
[23] Y.M. Teo, S.C. Low, S.C.Tay, J.P. Gozali, *Distributed Geo-rectification of Satellite Images using Grid Computing*, In International Parallel and Distributed Processing Symposium, IEEE Computer Society, Washington, 2003.
[24] *Java Servlet Technology*, http://java.sun.com/products/servlet/overview.html
[25] *Open source database MySQL*, http://www.mysql.com/why-mysql/
[26] *Apache Tomcat*, http://tomcat.apache.org/
[27] C. Bauer and G. King, *Hibernate in Action*, Manning Publications, 2004
[28] C. Walls, *Spring in Action*, Third Edition. Manning Publications, 2011.
[29] D. Panda, R. Rahman and D. Lane, *EJB 3 in Action*. Manning Publications, First Edition, 2007.
[30] *NetBeans IDE*, http://netbeans.org/features/
[31] Devadze, R. Ubar, J. Raik, A. Jutman, *Parallel Exact Critical Path Tracing Fault Simulation with Reduced Memory Requirements*, DTIS09
[32] K. Hwang and Z. Xu, *Scalable Parallel Computing: Technology, Architecture, Programming*, McGraw-Hill, San Francisco, 1998.
[33] D. E. Knuth. *Fundamental algorithms*. Vol 1. Second edition. Addison-Wesly, 1973
[34] D.L Eager, E. D. Lazowska, Zahorjan. *Adaptive load sharing in homogenious distributed systems*, IEEE Transactions on Software Engineering, SE-12(5:662–675), May 1986
[35] N. G. Shivaratri, P. Krueger, M. Singhal, *Load distributing for locally distributed systems*. IEEE Computer 25 (12), December 1992.
[36] *Example circuits*, http://www.cad.polito.it/tools/itc99.html

*Edited by:* Dana Petcu and Alex Galis
*Received:* March 1, 2011
*Accepted:* March 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.