Enforcing Reputation Constraints on Business Process Workflows*

Benjamin Aziz[1†] and Geoff Hamilton[2]
[1] *University of Portsmouth*
Portsmouth, United Kingdom
benjamin.aziz@port.ac.uk
[2] *Dublin City University*
Dublin, Ireland
geoff.hamilton@computing.dcu.ie

Abstract

The problem of trust in determining the flow of execution of business processes has been in the centre of research interst in the last decade as business processes become a de facto model of Internet-based commerce, particularly with the increasing popularity in Cloud computing. One of the main measures of trust is reputation, where the quality of services as provided to their clients can be used as the main factor in calculating service and service provider reputation values. The work presented here contributes to the solving of this problem by defining a model for the calculation of service reputation levels in a BPEL-based business workflow. These levels of reputation are then used to control the execution of the workflow based on service-level agreement constraints provided by the users of the workflow. The main contribution of the paper is to first present a formal meaning for BPEL processes, which is constrained by reputation requirements from the users, and then we demonstrate that these requirements can be enforced using a reference architecture with a case scenario from the domain of distributed map processing. Finally, the paper discusses the possible threats that can be launched on such an architecture.

Keywords: Business Process Workflows, BPEL, Reputation, Trust Models

1 Introduction

Over the past decade, service-based computing has become the de facto model for defining online businesses and accessing online resources. As a result, this has given rise to a new computational paradigm known as service-oriented architectures (SOAs). The most facinating idea underlying SOAs is the ability to compose services in an automatic manner to yield more complex services. In recent times, this concepts has become a major part of the ever-rising popularity of Cloud computing services, where platforms such as the Amazon Simple Workflow Service (Amazon SWF) [2], Salesforce's Visual Workflow [3] and others have already deployed the concept in a well-established Cloud market. The composition of services is often identified in terms of two modes of composition; *service orchestration* and *service choreography*. *Service orchestration* refers to the kind of composition where a single peer service interacts with different sub-services at various times while preserving their compositionality and therefore providing a single layer of abstraction to the user of the service. On the other hand, in *service choreography*, the various services in a workflow act independently but in a pre-defined manner based on some established "flow" (similar to the idea of a script).

The issue of trust when composing and running services, whether orchestration or cchoreography-based, is important from the point of view of the robustness of the workflow. So, what do we mean by trust in this context? The main definition that this work considers in the context of computing systems is the definition of trust given by Grandison and Sloman in [4]. They elegantly defined computational trust

Journal of Wireless Mobile Networks, Ubiquitous Computing, and Dependable Applications, volume: 5, number: 1, pp. 120-140

*This paper is an extended version of the work originally presented at the 8th International Conference on Availability, Reliability and Security (ARES'13), Regensburg, Germany, September 2013 [1].

†Corresponding author: School of Computing, University of Portsmouth, Portsmouth PO1 3HE, Tel: +44-(0)23-9284-2265, Web: `http://azizb.myweb.port.ac.uk/`

as "*... the firm belief in the competence of an entity to act dependably, securely and reliably within a specific context.*". The essence of this definition is that it encapsulates desired properties of computational systems such as dependability, security and reliability and at the same time maintain the view that these can only be measured in some specific context where the system in question functions. As an example, a Web security solution may not be trusted in the context of the reliability of a defence fighter jet system, as the (reliability, robustness etc.) requirements of the latter are much stricter than the former. This viewpoint has also influenced recent research effort in trust and security for SOAs, in particular, at the point of composition of services. The main problem this paper considers is that services implement different security mechanisms and apply different trust and security constraints, therefore, this heterogeneity will necessarily (and should) impact the execution of any workflow composed from such services. In other words, the composition of services in a workflow must take into consideration also the trust and reputation requirements imposed by users of the workflow in order to achieve the minimum level of quality specified in the SLA with the client.

In summary, we present in this paper a model for controlling the execution of worflows defined in the Business Process Execution Language (BPEL). This control takes into consideration the reputation constraints at the level of the services, the service providers and the BPEL workflow as a whole. The reputation constraints are assumed specified as part of a SLA with the client and we provide a definition of how these constraints can be enforced on a BPEL workflow. The applicability of the model is demonstrated by means of a real world example from the domain of distributed map processes.

An earlier version of this work was presented in [1], however, this current paper improves on the earlier version by providing a more formal definition of the BPEL semantics under reputation constraints as well as a software architecture that defines how this semantics can be enforced in a BPEL workflow. The semantics expresses scenarios where a BPEL workflow will succeed or fail depending on the service reputation levels requried by the users of the workflow.

The main advantages of this approach are manyfold:

- to provide a platform for applications to be implemented as *trustworthy workflows*, with the measure of trust being the reputation of the services underlying those workflows.
- to provide a formal basis for the definition of the meaning of such trustworthy workflows in terms of a failure semantics that can reflect service-level reputation requirements defined by users of the workflow.
- to eventually be able to measure the trustworthiness of the business applications based on the reputation of the workflows by which they are implemented.
- to suggest a reference architecture that implements the model of trustworthy workflows proposed in this paper, and discuss at a high level, possible threats on such an architecture.

The rest of the paper is structured as follows. In Section 2, we give an overview of BPEL standard and its abstract syntax, and we show how this syntax can be used to model an example of BPEL workflows for distributed map processing. In Section 3, we define a model of utility-based reputation for the case of service-oriented workflows, which is capable of expressing the reputation of workflows, services and service providers. In Section 4, we show how reputation constraints can be specified, generated and enforced in workflows, and how these can be used to define a new semantics for BPEL. In Section 5, we propose an architecture that can be used to implement a reputation-controlled business workflow. In Section 6, we provide an indication of how the reverse problem of expressing and implementing users' reputation can be tackled in the future. Finally, in Section 7 we give an overview of related work and in Section 8, we conclude giving directions for future research.

2 Introduction to BPEL

The Web Services Business Process Execution Language (WS-BPEL or BPEL 2.0, simply called here BPEL) [5] is a standard specification language for expressing Web service workflows that was adopted by the Organization for the Advancement of Structured Information Standards (OASIS [6]). BPEL resulted from the merge of two earlier workflow languages: XLANG, which is a block structured language designed by Microsoft, and WSFL, which is a graph-based language designed by IBM, and in fact, it adopted their approach in using Web-based interfaces (WSDL, SOAP) as its external communication mechanism while using XML as its specification language. There are two methods that Web services may be combined: Orchestration and Choreography. Both these two cases are depicted in Figure 1.

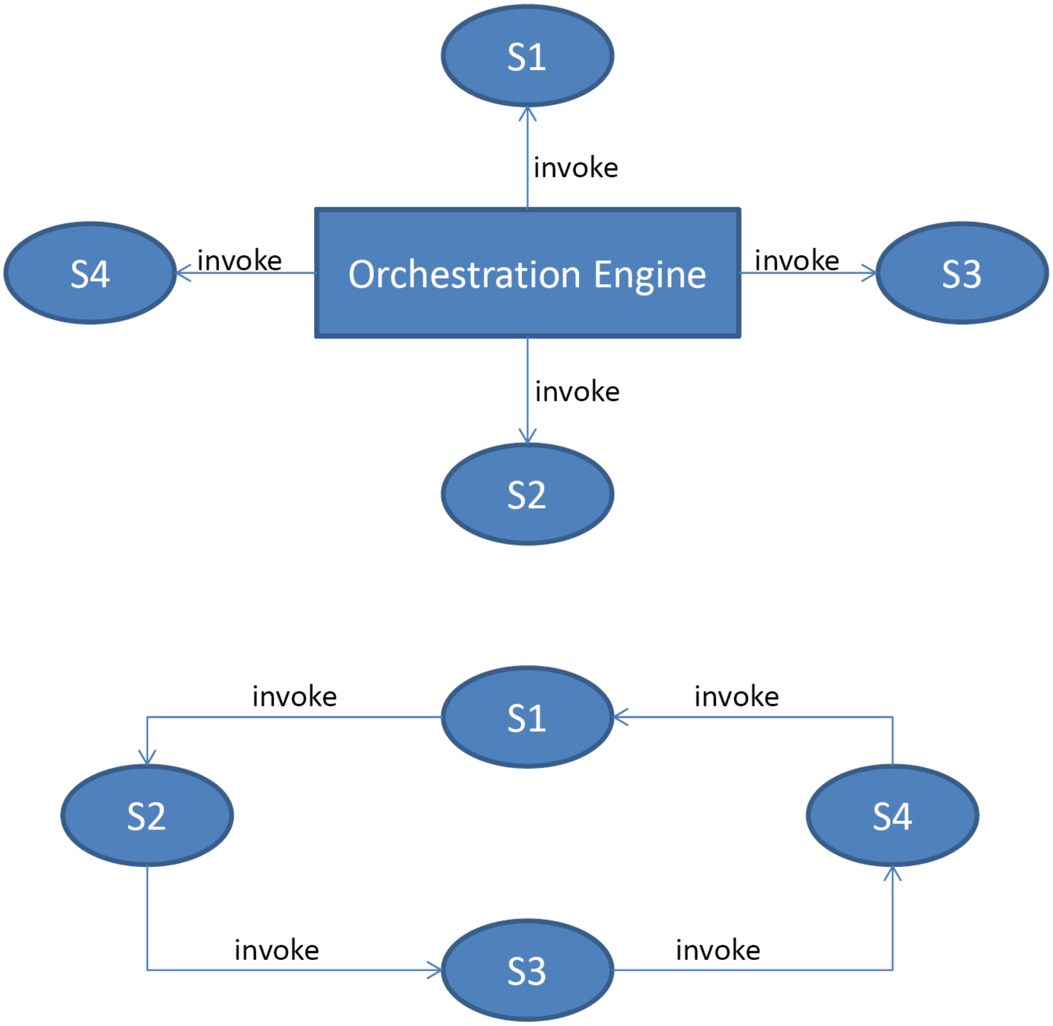

Figure 1: Orchestration versus Choreography Models.

BPEL supports both cases of service composition: service orchestration and service choreography. In the former case, a central process coordinates the execution of a workflow by calling individual services. The services themselves are agnostic to the existence of the workflow. Therefore, the central process acts as the orchestrator of the workflow. In the latter, there is no central coordinator and each service knows its own share of the workflow, in other words, it knows the exact operations it is meant to execute and which other services it should invoke. In this sense, services here are locked in a choreography.

The concept of *executable process* allows for services to be orchestrated in BPEL. On the other hand, the concept of *abstract business protocol* allows for the description of public communication messages without describing the internal details of the workflow process and hence facilitating the choreography of services. In the rest of the paper, we concentrate on the orchestration paradigm since it is more natural to BPEL. There are extensions of BPEL, such as BPEL4Chor [7], that promote the use of BPEL as a choreography language. BPEL has also been recently proposed for Cloud computations [8].

The lifecycle of a BPEL-based executable workflow process can be described intuitively as follows. The *process* representing the workflow is invoked by another external process (usually called the client) in which case the workflow process is started within its execution environment, typically a BPEL execution engine. The workflow process itself contains a description of *activities* that it must perform during the workflow. These activities may be either basic, such as the invocation of Web services, receiving invocations from other services/processes, replying to invocations etc., or structured, which describe the flow of control of basic activities, for example, the sequential composition, parallel composition or the conditional composition of activities. In each basic activity, the name of the *port type*, the name of the *partner link* offering that port type and the name of the *operation* on the port type are specified. Additionally, *parter links* may be grouped as one *partner* and they may have *partner roles*.

A process may also have a *correlation set*, which is a set of properties shared by all *messages* in a group of operations offered by a service. A process is divided into *scopes*, each of which contains an activity, a fault handler, a compensation handler and an event handler (we shall ignore event handlers from now on). Fault handlers catch faults and may sometimes re-throw them, whereas compensation handlers of successfully completed activities are used to reverse the effect of those activities (rollback) whenever a fault is caught in the workflow later on.

2.1 Abstract Syntax

We adopt here an abstract syntax for the BPEL language as defined by [9] and shown in Figure 2. The

B	$::=$		activity
		A	basic activity
		$\mid$ $skip$	do nothing
		$\mid$ $throw$	fault
		$\mid$ $sequence(B_1,B_2)$	sequential composition
		$\mid$ $flow(B_1,B_2)$	parallel composition
		$\mid$ $switch(\langle case\ b_1 : B_1\rangle,\dots,\langle case\ b_n : B_n\rangle,\langle otherwise\ B\rangle)$	conditional composition
		$\mid$ $scope\ n : (B,C,F)$	named scope
C,F	$::=$		compensation, fault handler
		$compensate$	compensate-all
		$\mid$ B	activity
P	$::=$	$\{\mid B,F \mid\}$	business process

Figure 2: Abstract Syntax of the BPEL Language.

syntax defines a BPEL business process as a pair, $\{\mid B,F \mid\}$, consisting of an activity, B, and a fault handler, F. The activity may be composed of several other activities. These could be either a basic activity, A, a do-nothing activity, *skip* or a fault throw activity, *throw*. Examples of basic activities are the communication activities, such as:

- Service invocations in the form of $invoke(ptlink,op,ptype)$, in which the operation, op, is invoked belonging to a partner link, $ptlink$, and the operation is invoked on a port type, $ptype$.
- Receiving a request in the form of $receive : (ptlink,op,ptype)$, where a service receives a request for an operation op on some port type $ptype$ by some client $ptlink$.
- Replying to a request, $reply : (ptlink,op,ptype)$, which generates a reply by calling an operation op over a port type $ptype$ belonging to a partner link $ptlink$.

For simplicity, in the abstract syntax of Figure 2 we have abstracted away all these basic activities and represented them by a simple activity, A, without loss of generality.

An activity may also be a *structured* activity. We consider the following structured activities:

- $sequence(B_1,B_2)$: this is a structured activity and it represents the sequential composition of two activities, B_1 and B_2. For B_2 to start executing, B_1 must have already terminated.

- $flow(B_1,B_2)$: this is a structured activity and it represents the parallel composition of two activities, B_1 and B_2. We do not assume anything here about the concurrency mode of these two activities (whether it is interleaving or non-interleaving).

- $switch(\langle case\ b_1 : B_1\rangle,\ldots,\langle case\ b_n : B_n\rangle,\langle otherwise\ B\rangle)$: this activity represents the conditional case-based statement, where an activity B_i is chosen if its logical condition, b_i, is true. If there are more than one logical conditions that are true, then one of these is chosen non-deterministically. Otherwise, the default B is executed if none of the logical conditions is satisfied. Conditions b are assumed to be expressed in some form of first order logic.

- $scope\ n : (B,C,F)$: this is a scope named n, which has a default activity, B, a compensation handler, C and a fault handler F. The scope usually runs as the default activity, B. If this executes successfully, the compensation handler, C, is installed in the context. Otherwise, the fault handler, F, is executed.

Fault and compensation handlers have the same definition as activities except that they can perform compensation-all calls. For simplicity, we do not consider named compensations, since these are a special case of compensation-all that require special operations to search for the name of the compensation scope belonging to past finished activities.

2.2 Example: Distributed Map Processing

We consider here a simple example of a distributed map processing application inspired by one of the application scenarios of project GridTrust [10]. The application could also be thought of as a Cloud-based workflow. The workflow representing interactions among the different components of the application are illustrated in Figure 3.

The application consists of a main orchestrator process, which is the *server farm*, that interacts with a couple of services, the *processing centre* and the *storage resources* services, whenever the server farm receives a request from the *client*. The workflow proceeds as follows:

1. A client cartographer submits a request to the server farm process, which advertises a map processing service that can create new maps. The request contains any relevant information related to the old and new maps requested by the client. As an example, we consider that the compensation for receiving the client's request is to request back to the client to send the map job again.

2. The server farm process invokes a local or a network-based resource storage service and stores on that service data related to the job submitted by the client. We consider that this invocation will be compensated by deleting the job data from the storage service.

3. The server farm process next submits a map processing request to a processing centre service requesting, which then retrieves information relevant to the new map and then sends the results back to the server farm.

4. Once the processing centre has ensured that the server farm is authorized to modify the map, the processing centre processes the job request and sends the results back to the server farm. These results contain the new map. We consider here that if the server farm is unable to receive the results of the map processing, then it will ask for a compensation of the finished previous activities.

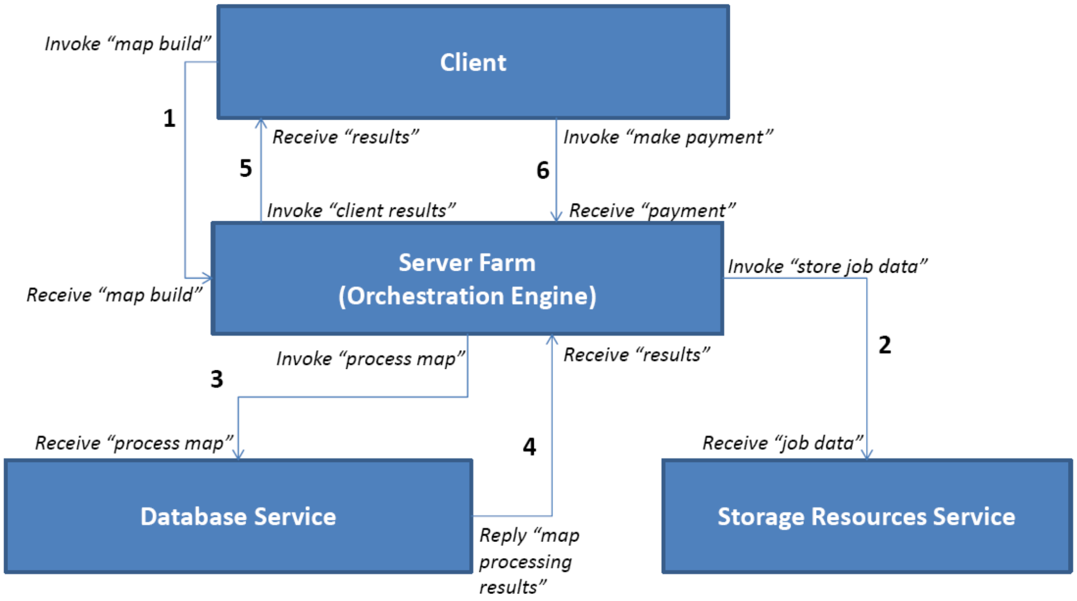

Figure 3: The Workflow for the Distributed Map Processing Application.

5. After having received the results from the processing centre, the server farm carries on final customisation processing on the new map and once finished, sends back the result to the client cartographer.

6. The client cartographer now is expected to make a payment to (possibly as a result of an off-line invoice it received) the server farm process. This payment is received and checked by the server farm process. If ok, the client is acknowledged.

The basic BPEL definition of the main server farm process is shown in Figure 4, where we have used the syntactic sugar $sequence(B_1,\ldots,B_n)$ instead of $sequence(B_1, sequence(\ldots,B_n))$.

3 A Reputation Model for BPEL Workflows

In this section, we provide an adaptation of the model presented in [11] for the case of BPEL-based workflows. Though initially designed for collaborative systems, the defined in [11] is general enough to be applicable to any distributed model, and furthermore we consider that workflows are a form of collaborative systems.

Central to our reputation model is the notion of a *service provider*. A service provider is any entity (e.g. organisation, company, administrator) that provides a service in a workflow. The set of all service providers is denoted by *Sps*. We keep track of all service providers that have existed and use the set *WId* to denote the set of all workflow identifiers. These are unique identifiers that identify each workflow. The *services* we want to keep reputation values for are defined as elements of the set *Srv*. These services belong to service providers. We are interested in some particular *issues of interest* associated to an entity; the set of all issues of interest is represented by *Issue*. The following function defines the set of services offered by a service provider:

$$| \; sSP : Sps \to \mathbb{P}\, Srv$$

$$
\begin{array}{ll}
\mathit{ServerFarm} = & \{\!|\ \mathit{sequence}(\\
& \mathit{scope\ req} : (\mathit{receive}(\mathit{Client}, \mathit{mapBuild}, \mathit{Map\ Build\ Port}), C_{req}, \mathit{throw}), \\
& \mathit{scope\ str} : (\mathit{invoke}(\mathit{Storage\ Resources}, \mathit{storeJobData}, \mathit{Resource\ Port}), C_{str}, \mathit{throw}), \\
& \mathit{scope\ prcinv} : (\mathit{invoke}(\mathit{Processing\ centre}, \mathit{processMap}, \mathit{Process\ Port}), \mathit{skip}, \mathit{throw}), \\
& \mathit{scope\ prcrec} : (\mathit{receive}(\mathit{Processing\ centre}, \mathit{inputProcessingResults}, \mathit{Process\ Results\ Port}), \\
& \hfill \mathit{skip}, \mathit{compensate}), \\
& \mathit{scope\ res} : (\mathit{reply}(\mathit{Client}, \mathit{mapResults}, \mathit{Map\ Results\ Port}), \mathit{skip}, \mathit{throw}), \\
& \mathit{scope\ pay} : (\mathit{receive}(\mathit{Client}, \mathit{makePayment}, \mathit{Payment\ Port}), \mathit{skip}, \mathit{throw}), \\
& \mathit{scope\ ack} : (\mathit{invoke}(\mathit{Client}, \mathit{allOK}, \mathit{Payment}), \mathit{skip}, \mathit{throw})), \\
& \mathit{compensate}\ |\!\}
\end{array}
$$

where,

$$
\begin{array}{ll}
C_{req} = & \mathit{sequence}(\mathit{invoke}(\mathit{Client}, \mathit{resendMap}, \\
& \hfill \mathit{Map\ Build\ Port}), \mathit{receive}(\mathit{Client}, \mathit{mapBuild}, \mathit{Map\ Build\ Port}))
\end{array}
$$

and,

$$
C_{str} = \quad \mathit{invoke}(\mathit{Storage\ Resources}, \mathit{deleteJobData}, \mathit{Resource\ Port})
$$

Figure 4: The Server Farm Process.

On the other hand, the following function defines the set of service providers involved in a particular workflow:

$$
\mid \quad wS : WId \mathrel{+\!\!\!\rightarrow} \mathbb{P}\, Srv
$$

In our model, we assume the existence of *monitors* that deliver *events* indicating the current value (result) produced by a service invocation in relation to a particular *issue of interest* within a workflow, at an observed moment in time (local to the monitor). We represent an event as a tuple that contains the following elements: a *timestamp* of the event, a *service*, an *issue*, a workflow id *WId* and a real number indicating the value of the specific element of issue captured by the event:

$$
\mid \quad Event : TimeStamp \times Srv \times Iss \times WId \times
$$

For example, the following tuple generated by the monitor represents an event at 12:09:52 local time, indicating that the result of invoking service *map_processor* has produced a Quality of Service (QoS) value of 0.65 for the workflow whose identity is *my_workflow*:

$$
ev_{ex} = (12\!:\!09\!:\!52, map\_processor, QoS, my\_workflow, 0.65)
$$

The value of 0.65 could be either associated to a specific element of QoS (e.g. performance, bandwidth, failure rate etc.) which is being monitored, or it could reflect an aggregated value of all these elements.

The model in [11] also introduces another fundamental concept in the modelling of reputation, i.e. that of a *utility function*. A utility function is a fitness criterion, which represents the satisfaction of the user (in this case, the service invocator or orchestrator). We focus here on one definition of such utility functions, which incorporates events and Service Level Agreements (SLAs):

$$
\begin{array}{|l}
utility : Event \rightarrow [0,1] \\
\hline
\forall (t,s,i,w,r) \in Event \bullet \\
utility((t,s,i,w,r)) = \\
\qquad\qquad \begin{cases} 1 & \text{if } r \geq SLA(s,i,w) \\ \frac{r}{SLA(s,i,w)} & \text{if } r < SLA(s,i,w) \end{cases}
\end{array}
$$

where a SLA is defined as the following function, returning the *expected* value for the issue of interest:

$$SLA : Srv \times Iss \times WId \rightarrow$$

Hence, for ev_{ex} above, if $SLA(map\_processor, QoS, my\_workflow) < 0.65$ then $utility(ev_{ex}) = 1$. Otherwise, $utility(ev_{ex}) < 1$ reflecting the ratio between the actual and agreed values for the QoS.

3.1 Service Reputation Models

After introducing the main notions of an event and a utility function, we can now define three models of the reputation of services in workflows in the context of issues of interest. We start with the definition of the reputation of a specific service in a specific workflow with respect to a specific issue of interest. Given that $Event_w \subseteq Event$ is the set of events captured by the workflow monitor for the workflow w, then we can define our first reputation function as follows:

$$\begin{array}{l}
[WId, Srv, Iss] \\
\hline
srv\_rep\_wsi : TimeStamp \times Srv \times Iss \times WId \rightarrow [0,1] \\
\hline
\forall\, t : TimeStamp, s : Srv, i : Iss, w : WId \bullet \\
srv\_rep\_wsi(t, s, i, w) = \\
\frac{\sum_{ev \in \{(ts,s,i,w,r) \in Event_w\}} \varphi(t,ts) utility(ev)}{\#\{(ts,s,i,w,r) \in Event_w\}}
\end{array}$$

where $\#s$ denotes the cardinality of a set s and $\varphi(t,ts)$ is a time discount function that puts more importance (emphasis) on events registered closer in time to the moment of computing the reputation. Reputation, *srv_rep_wsi*, is defined as the weighted average of the utilities obtained from all generated events so far.

Based on the definition of *srv_rep_wsi*, we can next define the more general reputation of a specific service in a specific workflow in relation to *all* issues of interest, as follows:

$$\begin{array}{l}
[WId, Srv] \\
\hline
srv\_rep\_ws : TimeStamp \times Srv \times WId \rightarrow [0,1] \\
\hline
\forall\, t : TimeStamp, s : Srv, w : WId \bullet \\
srv\_rep\_ws(t, s, w) = \frac{\sum_{i \in Iss} srv\_rep\_wsi(t,s,i,w)}{\#Iss}
\end{array}$$

Which aggregates over the total number of issues of interest, *#Iss*, which the service is being monitored against.

The next level of reputation defines the reputation of a whole workflow, aggregating over the *srv_rep_ws* reputation values of all of its member services, as follows:

$$\begin{array}{l}
[WId] \\
\hline
srv\_rep\_w : TimeStamp \times WId \rightarrow [0,1] \\
\hline
\forall\, t : TimeStamp, w : WId \bullet \\
srv\_rep\_w(t, w) = \frac{\sum_{s \in wS(w)} srv\_rep\_ws(t,s,w)}{\#wS(w)}
\end{array}$$

Based on *srv_rep_ws*, in fact we can also define the reputation value of a specific service with respect to all the workflows it has participated in:

$$[Srv]$$

$$srv\_rep\_s : TimeStamp \times Srv \rightarrow [0,1]$$

$$\forall\, t : TimeStamp, s : Srv \bullet$$
$$srv\_rep\_s(t,s) = \frac{\sum\limits_{w \in \{w : s \in wS(w)\}} srv\_rep\_ws(t,s,w)}{\#\{w : s \in wS(w)\}}$$

Where $\{w : s \in wS(w)\}$ is the set of all those workflows that have the service s as a member. Finally, we are now able to define the reputation of a service provider based on the last model:

$$[Sps]$$

$$srv\_rep\_p : TimeStamp \times Sps \rightarrow [0,1]$$

$$\forall\, t : TimeStamp, p : Sps \bullet$$
$$srv\_rep\_p(t,p) = \frac{\sum\limits_{s \in sSP(p)} srv\_rep\_s(t,s)}{\#sSP(s)}$$

4 Reputation Constraints

Having defined the machinery for modeling reputation of processes (or services) in a workflow in the previous section, we now proceed to define a method by which reputation constraints can be enforced in a business workflow.

We define a *reputation constraint* to indicate that a specific reputation level must remain within the boundary of two real values *Min* and *Max*. For each reputation constraint, we assign a corresponding constraint identifier as follows:

$$\begin{aligned}
Cons_{wsi} &= Min \leq srv\_rep\_wsi(t,s,i,w) \leq Max \\
Cons_{ws} &= Min \leq srv\_rep\_ws(t,s,w) \leq Max \\
Cons_{w} &= Min \leq srv\_rep\_w(t,w) \leq Max \\
Cons_{s} &= Min \leq srv\_rep\_s(t,s) \leq Max \\
Cons_{p} &= Min \leq srv\_rep\_p(t,p) \leq Max
\end{aligned}$$

For the sake of simplicity, we shall use the general notation $Cons_x$ to refer to any of the above constraints, where $x \in \{wsi, ws, w, s, p\}$, and assume the set *Cons* to include all the above constraint function identifiers (hence treating all constraints as of the same type). A set of reputation constraints can be obtained using the following function:

$$repCons : User \nrightarrow \mathbb{P}\, Cons$$

for a specific user *u User*, who is a client of the business workflow or the orchestrator process. Therefore, one can imagine $repCons(u)$ as being a form of a SLA agreed with the user *u* on the quality of protection of their business requirements.

4.1 Constraints Generation

We define next a method for generating reputation constraints, $Cons_x$, of the previous section in a top-to-bottom approach. We write $Cons_x.Min$ to refer to the minimum value of the constraint, and $Cons_x.Max$ to the maximum value. We also define the relation $Cons_x \preceq Cons_y$ to mean that the definition of $Cons_y$ includes that of $Cons_x$ (i.e. the definition of the reputation function within $Cons_y$ is dependant on that within $Cons_x$). For example, we have that $Cons_{ws} \preceq Cons_{wsi}$ since *srv_rep_ws* is dependant in its definition on the definition of *srv_rep_wsi*.

The generation of the constraint $Cons_y$ based on the definition of $Cons_x$ can in fact be considered as a solution to a *constraint satisfaction problem* [12]. Assuming that $Cons_x.Min$ and $Cons_x.Max$ are available, and that $Cons_x$ will result in a k number of constraints at the next level, $Cons_{y1} \dots Cons_{yk}$ then one can generate the values for $Cons_{yi}.Min$ and $Cons_{yi}.Max$ for each $i \in \{1,\dots,k\}$ provided that the following two constraints on the generated values (solutions) are met:

$$\frac{\sum\limits_{i\in\{1,\dots,k\}} Cons_{yi}.Min}{k} = Cons_x.Min \tag{1}$$

$$\frac{\sum\limits_{i\in\{1,\dots,k\}} Cons_{yi}.Max}{k} = Cons_x.Max \tag{2}$$

These constraints say that the average of the generated values for the next level reputation constriant must be equal to the value of the higher level reputation constraint, both in the case of the minima (1) and the maxima (2). Despite the fact that this approach so far has been discussed in the case of top-to-bottom constraint generation, it is also valid for the case of bottom-to-top generation (i.e. from $Cons_{wsi}$ to $Cons_w$).

To demonstrate how this solution works, let's revisit our example of Section 2.2 of distributed map processing in this case to demonstrate how workflow-level reputation constraints can be used to enforce a SLA with the clients of the workflow by means of propagating this reputation down to the reputation of individual services and service providers. The distributed map processing workflow consisted of three main services; the Server Farm (which is also the orchestrator process), the Processing Centre service and the Storage Resources service.

In one such SLA, the Client and the Owner of the workflow process/orchestrator can agree on a workflow-level of reputation stating that this reputation must fall within the range of 0.5 and 0.75, for any time t:

$$0.5 \leq srv\_rep\_w(t, distributed\_map\_processing) \leq 0.75$$

In other SLAs, it is also possible to start from top reputation constraints on *srv_rep_p* or *srv_rep_s*.

Starting from this constraint and using the definition of *srv_rep_ws*, we can next solve the reputation constraints for each of the three services involved in the workflow. One such solution could be the following set of constraints:

$$\begin{aligned}
0.4 &\leq srv\_rep\_ws(t, SF, distributed\_map\_processing) \leq 0.8\\
0.2 &\leq srv\_rep\_ws(t, PC, distributed\_map\_processing) \leq 0.5\\
0.9 &\leq srv\_rep\_ws(t, SR, distributed\_map\_processing) \leq 0.95
\end{aligned}$$

for each of the three services (SF=Server Farm, PC=Processing Centre, SR=Storage Resources). Assuming we consider only two issues of interest for each of these services, namely performance effeciency (i.e. the service's response time and throughput) and availability (i.e. percentage of time the service is running), then we can deduce the following six reputation constraints at the level of each service and for each of the above two issues of interest (PE=Performance Effeciency, A=Availability):

$$\begin{aligned}
0.5 &\leq srv\_rep\_wsi(t, SF, PE, distributed\_map\_processing) \leq 1.0\\
0.3 &\leq srv\_rep\_wsi(t, SF, A, distributed\_map\_processing) \leq 0.6\\
0.2 &\leq srv\_rep\_wsi(t, PC, PE, distributed\_map\_processing) \leq 0.7\\
0.2 &\leq srv\_rep\_wsi(t, PC, A, distributed\_map\_processing) \leq 0.3\\
0.95 &\leq srv\_rep\_wsi(t, SR, PE, distributed\_map\_processing) \leq 1.0\\
0.85 &\leq srv\_rep\_wsi(t, SR, A, distributed\_map\_processing) \leq 0.9
\end{aligned}$$

The reputation monitoring service will issue events from time to time in and frequently calculate the value of *srv_rep_wsi* for each service and issue of interest in the distributed map processing workflow. Each time such calculation is made, the above constraints are checked and enforced, in order to enforce the top-level SLA agreement with the Client containing the workflow constraint of $0.5 \leq srv\_rep\_w(t, distributed\_map\_processing) \leq 0.75$.

4.2 Constraints Enforcement

Next, we discuss how the reputation constraint generated in the previous section can be enforced on the semantics of the BPEL abstract syntax. In [9], the authors define a big-step semantics for the same subset of the BPEL syntax of Section 2.1 based on a *transition system*, $\longrightarrow$:

$$\Gamma \vdash P \longrightarrow \Box, F$$

where $\Box$ is defined as being one of the following three termination states:

$\boxdot$: successful process termination.

$\boxtimes$: unsuccessful process termination with an error.

$\overline{\boxtimes}$: premature forced termination.

Sometimes we use $\underline{\Box}$ to imply $\{\boxtimes, \overline{\boxtimes}\}$ and $\boxminus$ to imply $\{\boxdot, \boxtimes\}$. Additionally, we define the operator, $\otimes$, as follows:

$\otimes$	$\boxdot$	$\overline{\boxtimes}$	$\boxtimes$
$\boxdot$	$\boxdot$	$\overline{\boxtimes}$	$\boxtimes$
$\overline{\boxtimes}$	$\overline{\boxtimes}$	$\overline{\boxtimes}$	$\boxtimes$
$\boxtimes$	$\boxtimes$	$\boxtimes$	$\boxtimes$

This last operator shows that the success of permitting an activity is empowered by the forced termination decision and the latter is empowered by the complete denial for executing the activity. The $\otimes$ operator is needed to express the constraint evaluation decisions regarding activities composed in parallel, where the denial of one activity forces the termination of all the other activities in parallel with it, as will be explained later in the semantics.

The environment of the BPEL orchestration engine, Γ, will determine for each transition performed by the process whether the transition will terminate according to one of the above three semantic outcomes. Here, we shall extend the transition system of [9] to be able to enforce our reputation constraints. First, we need to define a new type of events, which are emitted by the transition relation $\longrightarrow$ and which are captured by the reputation monitoring system. We call these events *monitoring hooks* and we write them as $\omega_1, \ldots, \omega_n \in \Omega$. A monitoring hook may contain any information about a transition step. Therefore, we leave the definition of a monitoring hook general, however, one possible such definition that we adopt as an example would be $\omega = (s, w, a)$, where $s \in Srv$ is the name of the service (BPEL process) involved in the transition step, $w \in WId$ the id of the workflow and $a \in B$ being a BPEL basic activity.

4.2.1 Reputation Constraints-based BPEL Semantics

We modify the transition system of [9] as follows:

$$\Gamma_{repCons(u)} \vdash P \overset{\{\omega_1 \ldots \omega_n\}}{\longrightarrow} \Box, F$$

This new system replaces the generic BPEL runtime environment Γ with $\Gamma_{repCons(u)}$ that incorporates the reputation constraints $repCons(u)$ of a specific user u of the BPEL workflow or business process. The set $\{\omega_1 \ldots \omega_n\}$ represents the monitoring hooks that have been captured by the reputation monitoring system during the course of transitions performed by the process P. Note that since this semantics is a big-step semantics leading from an initial state (i.e. P) to a final one (i.e. $\Box, F$), the captured hooks must be a set to reflect all the small-step transitions not visible in this semantics. Based on this, we can now

$$\begin{array}{ll}
(BPEL1) & \Gamma_{repCons(u)} \vdash skip, F \xrightarrow{\{\omega_1 \ldots \omega_n\}} \boxdot, F \\
(BPEL2) & \Gamma_{repCons(u)} \vdash throw, F \xrightarrow{\{\omega_1 \ldots \omega_n\}} \boxtimes, F \\
(BPEL3) & \Gamma_{repCons(u)} \vdash A, F \xrightarrow{\{\omega_1 \ldots \omega_n\}} \boxdot, F \;\Rightarrow\; (\bigwedge_{c \in repCons(u)} c) \\
(BPEL4) & \Gamma_{repCons(u)} \vdash A, F \xrightarrow{\{\omega_1 \ldots \omega_n\}} \boxtimes, F \;\Rightarrow\; \neg(\bigwedge_{c \in repCons(u)} c) \\
(BPEL5) & \Gamma_{repCons(u)} \vdash B_1, \alpha \xrightarrow{\{\omega_1 \ldots \omega_i\}} \boxdot, \gamma \wedge \Gamma_{repCons(u)} \vdash B_2, \gamma \xrightarrow{\{\omega_{i+1} \ldots \omega_n\}} \Box, \beta \;\Rightarrow \\
 & \Gamma_{repCons(u)} \vdash sequence(B_1, B_2), \alpha \xrightarrow{\{\omega_1 \ldots \omega_i, \omega_{i+1} \ldots \omega_n\}} \Box, \beta \\
(BPEL6) & \Gamma_{repCons(u)} \vdash B_1, \alpha \xrightarrow{\{\omega_1 \ldots \omega_i\}} \underline{\Box}, \gamma \;\Rightarrow \\
 & \Gamma_{repCons(u)} \vdash sequence(B_1, B_2), \alpha \xrightarrow{\{\omega_1 \ldots \omega_i, \omega_{i+1} \ldots \omega_n\}} \underline{\Box}, \gamma \\
(BPEL7) & \exists i \in \{1, \ldots, n\} : b_i = \texttt{True} \wedge \Gamma_{repCons(u)} \vdash B_i, \alpha \xrightarrow{\{\omega_1 \ldots \omega_n\}} \Box, \gamma \;\Rightarrow \\
 & \Gamma_{repCons(u)} \vdash switch(\langle case\ b_1 : B_1 \rangle, \ldots, \langle case\ b_n : B_n \rangle, \langle otherwise\ B \rangle), \alpha \longrightarrow \Box, \gamma \\
(BPEL8) & \forall i \in \{1, \ldots, n\} : b_i = \texttt{False} \wedge \Gamma_{repCons(u)} \vdash B, \alpha \xrightarrow{\{\omega_1 \ldots \omega_n\}} \Box, \gamma \;\Rightarrow \\
 & \Gamma_{repCons(u)} \vdash switch(\langle case\ b_1 : B_1 \rangle, \ldots, \langle case\ b_n : B_n \rangle, \langle otherwise\ B \rangle), \alpha \longrightarrow \Box, \gamma \\
(BPEL9) & \Gamma_{repCons(u)} \vdash B, \langle\rangle \xrightarrow{\{\omega_1 \ldots \omega_n\}} \boxdot, \gamma \;\Rightarrow \\
 & \Gamma_{repCons(u)} \vdash scope\ n : (B, C, F), \alpha \xrightarrow{\{\omega_1 \ldots \omega_n\}} \boxdot, (n : C : \gamma).\alpha \\
(BPEL10) & \Gamma_{repCons(u)} \vdash B, \langle\rangle \xrightarrow{\{\omega_1 \ldots \omega_n\}} \underline{\Box}, \gamma \wedge \Gamma_{repCons(u)} \vdash F, \gamma \xrightarrow{\{\omega_m \ldots \omega_k\}} \boxminus, \beta \;\Rightarrow \\
 & \Gamma_{repCons(u)} \vdash scope\ n : (B, C, F), \alpha \xrightarrow{\{\omega_1 \ldots \omega_n\}} \boxminus, \alpha \\
(BPEL11) & \Gamma_{repCons(u)} \vdash compensate, \langle\rangle \xrightarrow{\{\omega_1 \ldots \omega_n\}} \boxdot, \langle\rangle \\
(BPEL12) & \Gamma_{repCons(u)} \vdash C, \beta \xrightarrow{\{\omega_1 \ldots \omega_n\}} \boxdot, \gamma \wedge \Gamma_{repCons(u)} \vdash compensate, \alpha \xrightarrow{\{\omega_1 \ldots \omega_n\}} \boxminus, \langle\rangle \;\Rightarrow \\
 & \Gamma_{repCons(u)} \vdash compensate, (n : C : \beta).\alpha \xrightarrow{\{\omega_1 \ldots \omega_n\}} \boxminus, \langle\rangle \\
(BPEL13) & \Gamma_{repCons(u)} \vdash C, \beta \xrightarrow{\{\omega_1 \ldots \omega_n\}} \boxtimes, \gamma \;\Rightarrow\; \Gamma_{repCons(u)} \vdash compensate, (n : C : \beta).\alpha \xrightarrow{\{\omega_1 \ldots \omega_n\}} \boxtimes, \langle\rangle \\
(BPEL14) & \Gamma_{repCons(u_1)} \vdash compensate, \alpha_1 \xrightarrow{\{\omega_1 \ldots \omega_i\}} \boxdot, \beta_1 \wedge \\
 & \Gamma_{repCons(u_2)} \vdash compensate, \alpha_2 \xrightarrow{\{\omega_{i+1} \ldots \omega_n\}} \boxdot, \beta_2 \wedge \\
 & \Gamma_{repCons(u_1)} \cup \Gamma_{repCons(u_2)} \vdash compensate, \alpha \xrightarrow{\{\omega_1 \ldots \omega_i, \omega_{i+1} \ldots \omega_n\}} \boxminus, \beta \;\Rightarrow \\
 & \Gamma_{repCons(u_1)} \cup \Gamma_{repCons(u_2)} \vdash compensate, ((\alpha_1 \parallel_C \alpha_2).\alpha) \xrightarrow{\{\omega_1 \ldots \omega_i, \omega_{i+1} \ldots \omega_n\}} \boxminus, \langle\rangle \\
(BPEL15) & \Gamma_{repCons(u_1)} \vdash compensate, \alpha_1 \xrightarrow{\{\omega_1 \ldots \omega_i\}} \boxtimes, \beta_1 \vee \\
 & \Gamma_{repCons(u_2)} \vdash compensate, \alpha_2 \xrightarrow{\{\omega_{i+1} \ldots \omega_n\}} \boxtimes, \beta_2 \;\Rightarrow \\
 & \Gamma_{repCons(u_1)} \cup \Gamma_{repCons(u_2)} \vdash compensate, ((\alpha_1 \parallel_C \alpha_2).\alpha) \xrightarrow{\{\omega_1 \ldots \omega_i, \omega_{i+1} \ldots \omega_n\}} \boxtimes, \langle\rangle \\
(BPEL16) & \Gamma_{repCons(u_1)} \vdash B_1, \alpha \xrightarrow{\{\omega_1 \ldots \omega_i\}} \Box_1, (\gamma)\frown(\alpha) \wedge \\
 & \Gamma_{repCons(u_2)} \vdash B_2, \alpha \xrightarrow{\{\omega_{i+1} \ldots \omega_n\}} \Box_2, (\beta)\frown(\alpha) \;\Rightarrow \\
 & \Gamma_{repCons(u_1)} \cup \Gamma_{repCons(u_2)} \vdash flow(B_1, B_2), \alpha \xrightarrow{\{\omega_1 \ldots \omega_i, \omega_{i+1} \ldots \omega_n\}} (\Box_1 \otimes \Box_2), ((\gamma \parallel_C \beta).\alpha) \\
(BPEL17) & \Gamma_{repCons(u)} \vdash B, \langle\rangle \xrightarrow{\{\omega_1 \ldots \omega_n\}} \boxdot, \alpha \;\Rightarrow\; \Gamma_{repCons(u)} \vdash \{\!|\, B, F \,|\!\}, \langle\rangle \xrightarrow{\{\omega_1 \ldots \omega_n\}} \boxdot, \langle\rangle \\
(BPEL18) & \Gamma_{repCons(u)} \vdash B, \langle\rangle \xrightarrow{\{\omega_1 \ldots \omega_i\}} \boxtimes, \alpha \wedge \Gamma_{repCons(u)} \vdash F, \alpha \xrightarrow{\{\omega_{i+1} \ldots \omega_n\}} \boxminus, \beta \;\Rightarrow \\
 & \Gamma_{repCons(u)} \vdash \{\!|\, B, F \,|\!\}, \langle\rangle \xrightarrow{\{\omega_1 \ldots \omega_n\}} \boxminus, \langle\rangle
\end{array}$$

Figure 5: A Reputation-Constrained Labelled Transition Semantics for BPEL.

define the rules of the semantic operator $\overset{\{\omega_1 \ldots \omega_n\}}{\longrightarrow}$ under some orchestration engine, $\Gamma_{repCons(u)}$, containing the constraints from a specific user, u, as shown in Figure 5. Informally, Rule $(BPEL1)$ assumes that a *skip* activity is always permitted by any set of constraints. Rule $(BPEL2)$ assumes that a fault *throw* resembles the situation where the reputation constraints have denied the execution of the current activity encompassing *throw*. This is true for any set of such constraints. Rules $(BPEL3)$ and $(BPEL4)$ state that a basic activity is permitted (resp. denied) execution by the constraint enforcement mechanism that the orchestrator operates if the constraints evaluate to True (resp. False) for that basic activity. Rules $(BPEL5)$ and $(BPEL6)$ deal with the case of sequential composition of activities. Rule $(BPEL5)$ states that if the first activity in the composition is permitted by the set of reputation constraints to execute and succeed, then the outcome of the composition is the outcome of the second activity as decided by the overall set of constraints on the composition. Rule $(BPEL6)$ states that if the first activity is denied execution or is force-terminated, then regardless of what the status of the second activity is going to be, the sequential composition will also be denied execution or be force-terminated.

The next pair of rules, $(BPEL7)$–$(BPEL8)$, consider the case of the conditional composition where the final state of the *switch* activity will depend on the status of the selected activity and whether the latter is permitted, denied or is force-terminated. The selection of the particular activity is by case and depends on the truth value of its logical guard. Rules $(BPEL9)$ and $(BPEL10)$ deal with scopes. Let $\langle\ \rangle$ represent the empty compensation context, then Rule $(BPEL9)$ states that if the default activity in a scope is permitted to execute and succeed, then the compensation handler corresponding to it is installed in the compensation context $((n : C : \gamma).\alpha)$. In this case, the outcome of the scope is the same as that of the default activity under the reputation constraints. Rule $(BPEL10)$ states that if the main activity in the scope is denied execution or is force-terminated (by the enforcement of the set of reputation constraints) then the fault handler activity takes over execution and the outcome of the scope is that of the fault handler's activity. Note that we assume that the fault handler is never force-terminated, and so it is always either permitted or denied execution.

Rules $(BPEL11)$–$(BPEL15)$ deal with the case of compensations. Rule $(BPEL11)$ states that a *compensate* call in an empty compensation context will always succeed regardless of the reputation constraints (therefore its semantics resembles the semantics of *skip*). Rule $(BPEL12)$ states that if the execution of the head of a compensation context is allowed, then the outcome of a compensation call depends on the outcome of the execution of the tail of the context. Rule $(BPEL13)$ states that if the execution of the head of a compensation context is denied by the policy, then so will be the execution of the overall compensation context. The next two rules deal with the case of parallelism in compensation contexts resulting from parallelising BPEL activities. Rule $(BPEL14)$ states that if two compensation contexts are allowed to run under their respective sets of constraints (presumably supplied by possibly two different users), then the outcome of the overall compensation context will depend on the outcome of its tail. Conversely, rule $(BPEL15)$ states that if one of the two compensation contexts are denied execution, then both acting as the head of a larger compensation context will cause the latter to be denied as well. Both the last two rules use the parallel composition of compensation contexts operator, $\|_C$, which is described in the next rule.

Let $\hat{}$ be the symbol for the concatenation of compensation contexts, Rule $(BPEL16)$ deals with the parallel composition of activities using the *flow* activity. There are a couple of interesting notes on this rule. First, the special operation $\otimes$ is used to propagate the outcome (permission, denial or force-termination) of one activity to another, which is in parallel with it. This operator then determines the outcome of the overall composition. The second point is related to the fact that any new compensation contexts generated by the parallel activities must be treated as being in parallel as well. Therefore, these are composed using a special syntactic operator, $\|_C$, to indicate that these must be dealt with in parallel and each under its own set of reputation constraints generated by possibly different users.

Finally, we can define now the meaning of a business process, $\{\!| B, F |\!\}$, under the control of a set of constraints, $\Gamma_{repCons(u)}$. This meaning is defined in rules $(BPEL17)$ and $(BPEL18)$. Rule $(BPEL17)$ states that if the main activity B of the business process is permitted by the reputation constraints, then the business process is also permitted by the constraints to execute. Rule $(BPEL18)$, on the other hand, states that if activity is denied at any stage, then the fault handler of the business process takes over, under control from the residue of the policy at the point the business process was denied execution. The outcome of the business process will be the same as the fault handler's outcome. Again, there is an assumption here that a fault handler is never force-terminated.

4.2.2 A Reputation Monitoring System

A reputation monitoring system can be itself defined as $\mathscr{M} : \Omega \rightarrow \mathbb{P}\mathit{Event}$, which is a function taking a monitoring hook and produces a set of events each concerned with one issue of interest. For example, let's consider the Server Farm process defined in Figure 4. Running this process within the distributed map processing workflow could emit the following monitoring hooks, which are then captured by the workflow's monitor as shown in Figure 6 (assuming the workflow has a successful flow execution). Additionally, the monitoring system will also update its internal state reflecting the values of the various definitions of reputation, $\mathit{srv\_rep\_x}$ for $x \in \{wsi, ws, w, s, p\}$ based on the events generated from $\mathscr{M}(\omega)$. We write such internal function as $\mathit{update}(\mathscr{M}(\omega), \mathit{srv\_rep\_x}) = \mathit{srv\_rep\_x}'$, where $\mathit{srv\_rep\_x}'$ is an updated reputation relation calculated based on the model of Section 3.1. Hence, for a new $\mathit{srv\_rep\_x}'$, a service, workflow or service provider will have a new reputation value that can be obtained by applying $\mathit{srv\_rep\_x}'$ to the appropriate parameters.

$$\begin{aligned}
\mathscr{M}(\quad & (\mathit{server\_farm}, \mathit{distributed\_map\_processing}, \mathit{receive}(\mathit{Client}, \mathit{mapBuild}, \mathit{MapBuildPort}))) = \\
& \{(12{:}09{:}52, \mathit{server\_farm}, \mathit{PE}, \mathit{distributed\_map\_processing}, 0.6), \\
& (12{:}09{:}52, \mathit{server\_farm}, \mathit{A}, \mathit{distributed\_map\_processing}, 0.5)\}
\end{aligned}$$

$$\begin{aligned}
\mathscr{M}(\quad & (\mathit{server\_farm}, \mathit{distributed\_map\_processing}, \mathit{invoke}(\mathit{Storage\ Resources}, \mathit{storeJobData}, \mathit{ResourcePort}))) = \\
& \{(12{:}09{:}57, \mathit{server\_farm}, \mathit{PE}, \mathit{distributed\_map\_processing}, 0.51), \\
& (12{:}09{:}57, \mathit{server\_farm}, \mathit{A}, \mathit{distributed\_map\_processing}, 0.43)\}
\end{aligned}$$

$$\begin{aligned}
\mathscr{M}(\quad & (\mathit{server\_farm}, \mathit{distributed\_map\_processing}, \mathit{invoke}(\mathit{Processing\ Centre}, \mathit{processMap}, \mathit{ProcessPort}))) = \\
& \{(12{:}10{:}02, \mathit{server\_farm}, \mathit{PE}, \mathit{distributed\_map\_processing}, 0.46), \\
& (12{:}10{:}02, \mathit{server\_farm}, \mathit{A}, \mathit{distributed\_map\_processing}, 0.22)\}
\end{aligned}$$

$$\begin{aligned}
\mathscr{M}(\quad & (\mathit{server\_farm}, \mathit{distributed\_map\_processing}, \mathit{receive}(\mathit{Processing\ Centre}, \mathit{inputProcessingResults}, \\
& \mathit{ProcessResultsPort}))) = \\
& \{(12{:}10{:}04, \mathit{server\_farm}, \mathit{PE}, \mathit{distributed\_map\_processing}, 0.77), \\
& (12{:}10{:}04, \mathit{server\_farm}, \mathit{A}, \mathit{distributed\_map\_processing}, 0.54)\}
\end{aligned}$$

$$\begin{aligned}
\mathscr{M}(\quad & (\mathit{server\_farm}, \mathit{distributed\_map\_processing}, \mathit{reply}(\mathit{Client}, \mathit{mapResults}, \mathit{MapResultsPort}))) = \\
& \{(12{:}10{:}09, \mathit{server\_farm}, \mathit{PE}, \mathit{distributed\_map\_processing}, 0.81), \\
& (12{:}10{:}09, \mathit{server\_farm}, \mathit{A}, \mathit{distributed\_map\_processing}, 0.33)\}
\end{aligned}$$

$$\begin{aligned}
\mathscr{M}(\quad & (\mathit{server\_farm}, \mathit{distributed\_map\_processing}, \mathit{receive}(\mathit{Client}, \mathit{makePayment}, \mathit{PaymentPort}))) = \\
& \{(12{:}10{:}17, \mathit{server\_farm}, \mathit{PE}, \mathit{distributed\_map\_processing}, 0.71), \\
& (12{:}10{:}17, \mathit{server\_farm}, \mathit{A}, \mathit{distributed\_map\_processing}, 0.49)\}
\end{aligned}$$

$$\begin{aligned}
\mathscr{M}(\quad & (\mathit{server\_farm}, \mathit{distributed\_map\_processing}, \mathit{invoke}(\mathit{Client}, \mathit{allOK}, \mathit{Payment}))) = \\
& \{(12{:}10{:}22, \mathit{server\_farm}, \mathit{PE}, \mathit{distributed\_map\_processing}, 0.5), \\
& (12{:}10{:}22, \mathit{server\_farm}, \mathit{A}, \mathit{distributed\_map\_processing}, 0.29)\}
\end{aligned}$$

Figure 6: Events Transmitted by the Reputation Monitor of the Server Farm Process.

Hence, for the example of Server Farm process, and given the monitoring events of Figure 6, one can get the following intermediate average values for $\mathit{srv\_rep\_wsi}$ for the cases of PE and A respectively as shown in Figure 7 (where we assume that $\varphi(t, ts) = 1$). Both of these issues of interest are within the acceptable constraints for $\mathit{srv\_rep\_wsi}$ specified in Section 4.1 at all times during the execution of the Server Farm process. More formally, we can define the property of *reputation constraints enforcement* as follows.

Property 1 (Reputation Constraints Enforcement)**.** *We say that a transition system for a BPEL process,* $\Gamma_{\mathit{repCons}(u)} \vdash P \overset{\{\omega_1 \ldots \omega_n\}}{\longrightarrow} \boxdot, F$ *has enforced the reputation constraints specified in* $\mathit{repCons}(u)$ *for some user*

$$
\begin{aligned}
&\mathit{srv\_rep\_wsi}(12{:}09{:}52, \mathit{server\_farm}, \mathit{PE}, \mathit{distributed\_map\_processing}) = 0.6\\
&\mathit{srv\_rep\_wsi}(12{:}09{:}57, \mathit{server\_farm}, \mathit{PE}, \mathit{distributed\_map\_processing}) = 0.56\\
&\mathit{srv\_rep\_wsi}(12{:}10{:}02, \mathit{server\_farm}, \mathit{PE}, \mathit{distributed\_map\_processing}) = 0.52\\
&\mathit{srv\_rep\_wsi}(12{:}10{:}04, \mathit{server\_farm}, \mathit{PE}, \mathit{distributed\_map\_processing}) = 0.59\\
&\mathit{srv\_rep\_wsi}(12{:}10{:}09, \mathit{server\_farm}, \mathit{PE}, \mathit{distributed\_map\_processing}) = 0.63\\
&\mathit{srv\_rep\_wsi}(12{:}10{:}17, \mathit{server\_farm}, \mathit{PE}, \mathit{distributed\_map\_processing}) = 0.64\\
&\mathit{srv\_rep\_wsi}(12{:}10{:}22, \mathit{server\_farm}, \mathit{PE}, \mathit{distributed\_map\_processing}) = 0.62
\end{aligned}
$$

$$
\begin{aligned}
&\mathit{srv\_rep\_wsi}(12{:}09{:}52, \mathit{server\_farm}, A, \mathit{distributed\_map\_processing}) = 0.5\\
&\mathit{srv\_rep\_wsi}(12{:}09{:}57, \mathit{server\_farm}, A, \mathit{distributed\_map\_processing}) = 0.47\\
&\mathit{srv\_rep\_wsi}(12{:}10{:}02, \mathit{server\_farm}, A, \mathit{distributed\_map\_processing}) = 0.38\\
&\mathit{srv\_rep\_wsi}(12{:}10{:}04, \mathit{server\_farm}, A, \mathit{distributed\_map\_processing}) = 0.42\\
&\mathit{srv\_rep\_wsi}(12{:}10{:}09, \mathit{server\_farm}, A, \mathit{distributed\_map\_processing}) = 0.4\\
&\mathit{srv\_rep\_wsi}(12{:}10{:}17, \mathit{server\_farm}, A, \mathit{distributed\_map\_processing}) = 0.4\\
&\mathit{srv\_rep\_wsi}(12{:}10{:}22, \mathit{server\_farm}, A, \mathit{distributed\_map\_processing}) = 0.4
\end{aligned}
$$

Figure 7: Intermediate Reputation Values for *srv_rep_wsi* for the case of Performance Effeciency and Availability Issues of Interest.

u during the course of its transitions ending in the successful termination state $\boxdot, F$ *if and only if the following holds true:*

$$\forall x \in \{wsi, ws, w, s, p\}, i \in \{1 \ldots n\}, \mathit{srv\_rep\_x}' \in \{\mathit{update}(\omega_i, \mathscr{M}, \mathit{srv\_rep\_x})\} \;:\; \bigwedge_{\mathit{cons} \in \mathit{repCons}(u)[\mathit{srv\_rep\_x}'/\mathit{srv\_rep\_x}]} \mathit{cons}$$

Otherwise, if the reputation constraints are violated at any stage of the semantics, the system will terminate unsuccessfully in some bad state $\square, F_{bad}$.

Examining the results of Figure 7, we can see that none of the two constraints is violated in its intermediate values according to the above definition of reputation constraints enforcement. If however, the monitor had captured an event that returned a value outside $(0.5-1.0)$ for PE or outside $(0.3-0.6)$ for A, then the above property would have been violated since the conjunction of the constraints on reputation would have become false.

5 A Reputation Management Service for BPEL

Here we present a reference architecture that implements our model of a reputation management service that can be integrated in BPEL-based workflows and that can facilitate the rating of BPEL services. This architecture is by no means the ”only” implementation of our model, and there could be other variations of this architecture based on the specific problem.

The architecture is shown in Figure 8, which shows the sequences of interactions among the various components of the system.

The architecture consists of four major components

- First a *reputation monitoring* system running at the level of interactions between BPEL services and users will trap any access attempts that the users will request from the services. In doing so, the system will be able to monitor special hooks related to the issue of interest on which the reputation values will later be calculated. This monitor will also generate the events consumed by the next component.

- Second, a *reputation management* service, which receives events from the reputation monitoring system and issues requests for the calculation and updating of service reputation using the next component.

- Third, the *reputation calculation* engine, is the main intelligence in the whole system, containing the formulae defined in the model in Sections 3 and 4. This engine will receive requests from

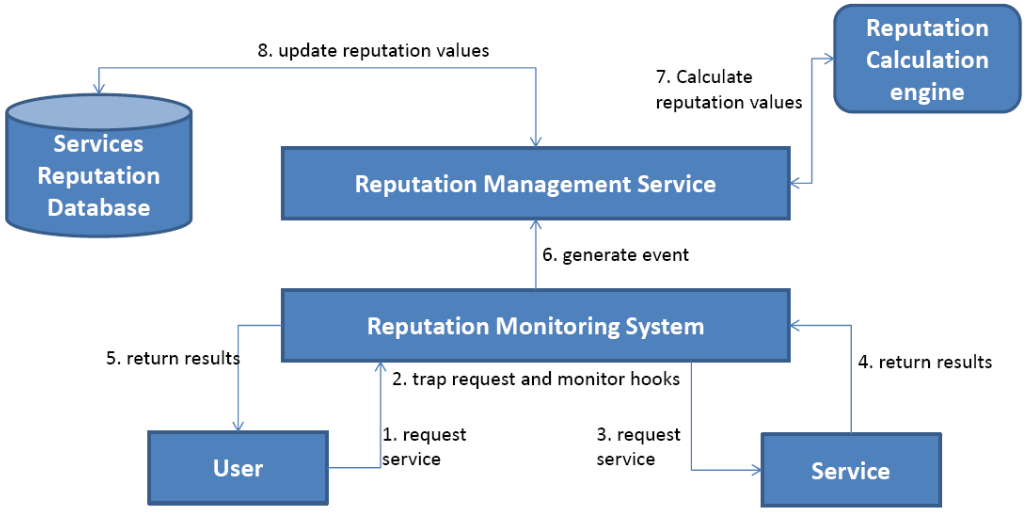
8. update reputation values
Reputation Calculation engine
7. Calculate reputation values
Services Reputation Database
Reputation Management Service
6. generate event
Reputation Monitoring System
5. return results
4. return results
2. trap request and monitor hooks
1. request service
3. request service
User
Service

Figure 8: An Architecture for Reputation Management in BPEL Workflows

the reputation management service to calculate new values of service reputation based on their behaviour as captured by the low-level events.

- Finally, the *services reputation* database is where the updated values for all the reputation levels are stored, by the reputation management service.

5.1 A Threat Model for the Architecture

The above architecure will need to consider security issues when deployed in a Grid environment. There are several issues that we highlight here, and that constitute a high-level threat model for the architecture.

- Trust in the architecture's components: this issue itself is a reflection of how trustworthy are the compoenents of the system itself. This will largely depend on the implementation and its quality. As a minimum, there also needs to be some kind of certification-based trust [4]. This implies that the different components will need to implement digital certificates-based authentication, depending on the level of distribution of these components.

- Securing communications: this issue is related to securing the communicated messages (e.g. requests, responses and reputation data values) whenever any interaction takes place among the different system components, and with the services being monitored and the users. A TLS-based connection [13] will be generally suffecient in authenticating components and establishing secure connections with them.

- Protection of the system components: The distribution of the system components will also imply the presence of possible malicious attackers and/or inadvertant tampering of the these components. As a result, the system as a whole should be run on well-secured and protected platforms where only authorised access is allowed. This is specifically important for the case of the service reputation database, which holds the important reputation information about the various Grid services. In certain extreme circumstances, it may even be desirable to run the database behind a Demilitarized Zone (DMZ).

6 Users' Reputation

The model presented above is generic enough for the reverse problem, i.e. the users reputation constraints, to be also expressed and implemented. Managing the reputation of users is not as popular as managing the reputation of business processes, mainly in open environments where it is not easy to track users and often (as in the case of Web services) no state is maintained for users. However, in stateful systems where some *profile* state for a user is kept (e.g. in Grid systems), the reputation of their past behaviour may be a useful factor in this profile that could determine decisions about the users' future rights.

For example, *users*, denoted by a set *User*, could execute some pre-defined actions on services following pre-established policies. *Services*, on the other hand, again denoted by set *Srv*, can qualify users in relation to their actions. If a user attempts to execute an action that is not allowed by the policy of a service (process) in a workflow, it will be given a bad qualification by the orchestrator of the workflow in relation to the usage of that service. This negative qualification is reflected in the user's reputation. In some sense, the model here reverses the role of services and users with respect to the model of the previous sections.

For such a new model, one can define a *policy* to indicate the set of actions (from the set of all possible actions *Action*) allowed on a particular service in a workflow.

$$policy : Srv \times WId \rightarrow \mathbb{P} Action$$

At the same time, one can define a *penalty* function that penalises a user with a value in the interval $[0,1]$ if the user executes non-permitted actions on some service in a specific workflow.

$$\begin{array}{l} penalty : User \times Srv \times WId \times Action \rightarrow [0,1] \\ \hline \forall u : User, s : Srv, w : WId, a : Action \bullet \\ \quad (u,s,w,a) \in \mathrm{dom}(penalty) \Rightarrow \\ \qquad\qquad a \notin policy(s,w) \end{array}$$

Now, events are re-defined from the previous model as follows:

$$Event == TimeStamp \times User \times Iss \times WId \times Action$$

where users' identity is now included in the event instead of the service, as well as the action that was performed by that user. The issue of interest, *Iss*, can be given the value *Usage* to indicate that we are interested in the user's usage of a workflow service. For example, the event:

$$ev_{ex} = (12{:}09{:}52, map\_processor, baziz, Usage, my\_workflow, \text{“}download\_file\text{”})$$

represents a tuple generated by the orchestration monitor indicating an event at time 12:09:52 local time on the invocation of the service *map_processor* part of the workflow called *my_workflow*. The event indicates that the usage action is for downloading a file by a user whose identity is *baziz*. At this point, we can define the utility that a service gets according to the actions performed by a user in a workflow. These functions are domain-specific, and based on their definition, one can further define the utility function as follows.

$$\begin{array}{l} utility : Event \rightarrow \\ \hline \forall (t,r,u,Usage,w,a) \in Event \bullet \\ utility((t,r,u,Usage,w,a)) = \\ \begin{cases} 1, \text{ if } a \in policy(u,r,w) \\ 1 - penalty(u,r,w,a), \text{ if } a \notin policy(u,r,w) \end{cases} \end{array}$$

This definition allows the workflow orchestration engine to assign a utility value to a user's action (i.e. behaviour) based on whether that action is or is not in the policy permitted by a particular service the user is interacting with. If the action is legitimate, the value is 1, otherwise, it is reduced by the penalty assigned to violating a policy. Currently, this utility function is limited in the fact that it does not consider the severity of a violating action, only the fact that it is outside the service policy. However, it can be used in the future as the basis for extending the reputation model defined in the paper to model user reputation and reputation constrains provided by business processes.

7 Related Work

Reputation is a general concept widely used in all aspects of knowledge ranging from humanities, arts and social sciences to digital sciences. It is a concept closely related to trust and it is defined by the Merriam-Webster dictionary [14] as the “overall quality or character as seen or judged by people in general”. In fact, reputation is often seen as one measure by which trust or distrust can be built based on good or bad past experiences and observations (direct trust) [15] or based on collected referral information (indirect trust) [16]. In recent years, the concept of reputation has shown itself to be useful in many areas of research in computer science, particularly in the context of distributed and collaborative systems, where interesting issues of trust and security manifest themselves. Therefore, one encounters several definitions, models and systems of reputation in distributed computing research [17].

There are many works in the literature that tackle the security and trust management of workflow-based systems. In [18], the authors are concerned with the modelling of access control policies for BPEL processes. In particular the authors presents an approach to integrate Role-Based Access Control and BPEL on the meta-model level. They describe a mapping of BPEL to RBAC elements and extracts them from BPEL. In particular they present a XSLT script, which transforms BPEL processes to RBAC models in an XML format.

On the contrary [19] presents two languages, RBAC-WS-BPEL and BPCL in order to be able to specify authorization information associating users with activities in the business process and authorization constraints on the execution of activities. Their RBAC-WS-BPEL architecture works on the orchestrator process. As a matter of fact through these languages, they rewrite the specification of the orchestrator by inserting authorization specification specified in BPCL. In our approach we consider also the satisfaction of possible local policies of each service by considering each of them as a subject that interacts with the orchestrator.

In [20], the author presents an analysis of the satisfiability of task-based workflows. The model of workflows adopted is enriched with entailment constraints that can be used for expressing cardinality and separation of duties requirements. Given such and other authorisation constraints, the analysis then answers questions related to whether the workflow can or cannot be achieved and whether an enforcement point can or cannot be designed based on all instances of the workflow. This is similar to our semantics, however we work closer to a standard language (i.e. BPEL) and we do not deal with the design of the enforcement point (i.e. the PDP). On the other hand, in [21], fine-grained access control policies were proposed for BPEL workflows based on process algebra. The current abstract syntax of BPEL adopted in this paper is based on the syntax defined in [21]. In [11], the authors proposed a general reputation model for collaborative computing systems as a measure for trust in such systems. This current paper builds directly on these two existing works by providing a special instance of the reputation model of [11] for the case of services/service workflows and then utilise this instance to control the semantics of the BPEL model defined in [21].

In the context of composite services systems, [22], the authors propose an architecture for automated, dynamic, pro-active, and transparent maintenance and improvement of composite services, which in [23, 24] is extended to deal with the reputation of Web services based on QoS issues. In [25] a model of *probabilistic success* of BPEL-based systems is defined, and in [26], a method is defined for integrating human agents into BPEL process that permits the monitoring of service behaviour and the calculation of their reputation based on this behaviour. Finally, we should mention that this paper represents an extension of a previous paper [1], which also defined a solution for the problem of enforcing reputation constraints on BPEL business workflows. The current paper includes additional detail on the new reputation constraints-controlled semantics for BPEL.

8 Conclusion and Future Work

Reputation is a popular measure for trust in any distributed computing system, and therefore, we have considered it in this paper as a solution to the problem of controlling the execution of business process workflows based on trust requirements specified by the clients of the workflow. The current paper demonstrates the use of reputation as part of a formal definition of the semantics of BPEL workflows. The model presented is also capable of expressing and enforcing reputation constraints on such workflows as well as the individual sub-services from which the workflows are composed and the service providers providing the service. The semantics considers cases of failure or success of a BPEL workflow based on how well the reputation requirements from the users are adhered to. The paper also demonstrated the applicability of the model through a real world use case based on the domain of distributed map processing, and it proposes a reference architecture that can be implemented in the future to enforce the reputation-controlled semantics for BPEL. The suggested architecture is by no means the ”only“ architecture that can implement the model, however it is one example of such implementation.

This approach is not without drawbacks, and one such major drawback is that the underlying model does not incorporate the notion of time, therefore, it is not capable of adjusting the impact of events on the calculated reputation as time passes by and the event become “older”. Such older events may be considered to have less impact than the more recent ones, or vice versa.

In addition to the possibility of expressing and implementing constraints on users’ reputation as we discussed in Section 6, there are several other directions for future research and development that this work can be extended towards. We highlight some of these below:

- The first direction would focus on the enhancement of the robustness of the model by adding reliability measures to the events generated by the reputation monitor. In particular, such robustness measures will consider the threat model discussed for the architecture.

- The expressivity of the model can also be improved by adopting a well-defined language for expressing the issues of interest and SLAs to be able to better specify whether a service/workflow/service provider meets the expectations of the client.

- Future development will focus on implementing the architecture and provide evaluation of the model based on the such implementation. We would also like to extend the application of the model and its implementation to more complex examples.

- Grid computing is one of the backbones of Cloud computing, and as a consequence, we are currently investigating the possibility of adapting our model and architecture to the Cloud computing paradigm.

- Finally, another interesting future research direction is related to configuration analysis, where reputation constraints are used at the beginning of service composition to *configure* the right workflow satisfying those constraints. This would render reputation constraint a fundamental non-functional criterion when building a workflow, which would be added to the functional requirements underlying the workflow process.

References

[1] B. Aziz and G. Hamilton, “Reputation-controlled business process workflows,” in *Proc. of the 8th International Conference on Availability, Reliability and Security (ARES’13), Regensburg, Germany*. IEEE, September 2013, pp. 42–51.

[2] Amazon SWF, aws.amazon.com/swf.

[3] Salesforce Visual Workflow, www.salesforce.com/platform/cloud-platform/workflow.jsp.

[4] T. Grandison and M. Sloman, “A Survey of Trust in Internet Applications,” *IEEE Communications Surveys and Tutorials*, vol. 3, no. 4, September 2000. [Online]. Available: http://pubs.doc.ic.ac.uk/TrustSurvey/

[5] BEA, IBM, Microsoft, SAP, and Siebel, "Web Services Business Process Execution Language Version 2.0," OASIS Standard, April 2007, http://docs.oasis-open.org/wsbpel/2.0/OS/wsbpel-v2.0-OS.pdf.

[6] Oasis, www.oasis-open.org.

[7] G. Decker, O. Kopp, F. Leymann, and M. Weske, "BPEL4Chor: Extending BPEL for Modeling Choreographies," in *Proc. of the IEEE 2007 International Conference on Web Services (ICWS'07), Salt Lake City, Utah, USA*. IEEE, July 2007.

[8] T. Dornemann, E. Juhnke, and B. Freisleben, "On-demand resource provisioning for BPEL workflows using Amazon's elastic compute cloud," in *Proc. of the 9th IEEE/ACM International Symposium on Cluster Computing and the Grid (CCGRID'09), Shanghai, China*. IEEE, May 2009, pp. 140–147.

[9] Z. Qiu, S. Wang, G. Pu, and X. Zhao, "Semantics of BPEL4WS-Like Fault and Compensation Handling," in *Proc. of the International Symposium of Formal Methods Europe (FM'05), Newcastle, UK, LNCS*, vol. 3582. Springer-Verlag, July 2005, pp. 350–365.

[10] GridTrust, "Deliverable D5.1(M19) Specifications of Applications and Test Cases, 2007."

[11] A. E. Arenas, B. Aziz, and G. C. Silaghi, "Reputation management in collaborative computing systems," *Security and Communication Networks*, vol. 3, no. 6, pp. 546–564, 2010.

[12] B. Nadel, "Some applications of the constraint-satisfaction problem," Wayne State University, Tech. Rep. CSC-90-008, 1990.

[13] T. Dierks and E. Rescorla, "Rfc 5246: The transport layer security (tls) protocol version 1.2," Aug. 2008.

[14] Merriam-Webster Dictionary, www.merriam-webster.com.

[15] A. Jøsang, R. Ismail, and C. Boyd, "A Survey of Trust and Reputation Systems for Online Service Provision," *Decision Support Systems*, vol. 43, no. 2, pp. 618–644, March 2007.

[16] A. Abdul-Rahman and S. Hailes, "Supporting trust in virtual communities," in *Proc. of the 33rd Hawaii International Conference on System Sciences (HICSS'00), Maui, Hawaii, USA*, vol. 6. IEEE, January 2000.

[17] G. C. Silaghi, A. Arenas, and L. M. Silva, "Reputation-based trust management systems and their applicability to grids," Institutes on Knowledge and Data Management and System Architecture, CoreGRID - Network of Excellence, Tech. Rep. TR-0064, February 2007.

[18] J. Mendling, M. Strembeck, G. Stermsek, and G. Neumann, "An Approach to Extract RBAC Models from BPEL4WS Processes," in *Proc. of the 13th IEEE International Workshops on Enabling Technologies: Infrastructure for Collaborative Enterprises (WETICE'04), Modena, Italy*. IEEE, June 2004, pp. 81–86.

[19] E. Bertino, J. Crampton, and F. Paci, "Access Control and Authorization Constraints for WS-BPEL," in *Proc. of the 2006 IEEE International Conference on Web Services (ICWS'06), Chicago, Illinois, USA*. IEEE, September 2006, pp. 275–284.

[20] J. Crampton, "An Algebraic Approach to the Analysis of Constrained Workflow Systems," in *Proc. of the 3rd Workshop on Foundations of Computer Security (FCS'04), Turku, Finland*, July 2004, pp. 61–74.

[21] B. Aziz, A. Arenas, F. Martinelli, I. Matteucci, and P. Mori, "Controlling usage in business process workflows through fine-grained security policies," in *Proc. of the 5th International Conference on Trust, Privacy and Security in Digital Business (TrustBus'08), Turin, Italy, LNCS*, vol. 5185. Springer-Verlag, September 2008, pp. 100–117.

[22] D. Bianculli, R. Jurca, W. Binder, C. Ghezzi, and B. Faltings, "Automated dynamic maintenance of composite services based on service reputation," in *Proc. of the 5th international conference on Service-Oriented Computing (ICSOC'07), Vienna, Austria, LNCS*, vol. 4749. Springer-Verlag, September 2007, pp. 449–455.

[23] D. Bianculli, W. Binder, L. Drago, and C. Ghezzi, "Transparent reputation management for composite web services," in *Proc. of the 2008 IEEE International Conference on Web Services (ICWS'08), Beijing, China*. IEEE, September 2008, pp. 621–628.

[24] D. Bianculli, W. Binder, M. L. Drago, and C. Ghezzi, "Reman: A pro-active reputation management infrastructure for composite web services," in *Proc. of the 31st International Conference on Software Engineering (ICSE'09), Vancouver, Canada*. IEEE, May 2009, pp. 623–626.

[25] Y. Chen and X. Wu, "Success measurement of web services with BPEL," in *Proc. of the 5th IEEE International Symposium on Service Oriented System Engineering (SOSE'10), Nanjing, China*. IEEE, June 2010, pp. 86–90.

[26] B. Jennings and A. Finkelstein, "Flexible Workflows: Reputation-based Message Routing," in *Proc. of the 9th Workshop on Business Process Modeling, Development, and Support (BPMDS 08), Montpellier, France*, June 2008.

Author Biography

Benjamin Aziz is a Senior Lecturer in Computer Security at the School of Computing, University of Portsmouth. Benjamin holds PhD degree in formal verification of computer security from Dublin City University (2003) and has research experience in the field of computer and information security spanning 15 years where he worked in the past at Rutherford Appleton Laboratory and Imperial College London, and has published more than 70 articles and book chapters in areas related to the security of large-scale systems, formal security, requirements engineering and digital forensics. He is on board program committees for several conferences and working groups, such as ERCIM's FMICS, STM, Cloud Security Alliance and IFIP WG11.3.

Geoff Hamilton is a Senior Lecturer in the School of Computing, Dublin City University and a Senior Researcher in Lero, the Irish Software Engineering Research Centre. Geoff has a Ph.D. in the area of program transformation from Stirling University (1993) and has research experience in the fields of formal methods and security spanning 25 years, starting at Stirling University, then Keele University and currently at Dublin City University. Geoff has published more than 70 articles and book chapters in areas related to his research and is on board program committees for several conferences and workshops.