Reputation-Controlled Business Process Workflows

Benjamin Aziz
School of Computing
University of Portsmouth
Portsmouth, United Kingdom
benjamin.aziz@port.ac.uk

Geoff Hamilton
School of Computing
Dublin City University
Dublin, Ireland
Geoff.Hamilton@computing.dcu.ie

***Abstract*—This paper presents a model solution for controlling the execution of BPEL business processes based on reputation constraints at the level of the services, the service providers and the BPEL workflow. The reputation constraints are expressed as part of an SLA and are then enforced at runtime by a reputation monitoring system. We use our model to demonstrate how trust requirements based on such reputation constraints can be upheld in a real world example of a distributed map processing BPEL workflow.**

***Keywords*-BPEL; business process workflows; trust and reputation management**

I. Introduction

In recent years, service-oriented architectures have established themselves as a major programming and computing systems paradigm. The automated composition of basic *Web Services* is one of the most promising ideas. Recently, this idea has also emerged in the world of Cloud computing, with platforms such as Amazon Simple Workflow Service (Amazon SWF)[1], Salesforce's Visual Workflow[2] and others already well-established in the Cloud market.

Services composition can be made by a single peer service, which could interact with different services at different times, preserving their compositionality and providing a single transparent view to the customers. This model is often referred to as *service orchestration* in which a single service orchestrates several others. On the other hand, it is also possible to have the various services in a workflow acting independently but in a pre-defined manner. This mode of operation is often reffered to as *service choreography*.

Trust and reputation have been the centre of attention in distributed systems in general. Trust, as a concept in computer science has been elegently defined by Grandison and Sloman [1, p.3] as "*... the firm belief in the competence of an entity to act dependably, securely and reliably within a specific context.*". This definition encapsulates desired general properties of computing systems such as dependability, security and reliability while maintaining that these are only measurable within the specific context in which the system functions. For example, a Web security solution will not be trusted in the context of the reliability of aerodynamics systems. Trust and security have emerged as very important issues in the composition of Web/Cloud services. Services are provided by different entities in the network that could implement different security mechanisms and apply different trust and security constraints. The overall interaction of these constraints may not allow a service workflow execution to complete due to unexpected conflicts among such constraints. Indeed, services are composed for adhering to a business workflow and any trust and security constraints must adhere to this view.

This paper presents a model for controlling the execution of BPEL business processes based on reputation constraints at the level of the services, the service providers and the BPEL workflow. The reputation constraints are expressed as part of an SLA and are then enforced at runtime by a reputation monitoring system. We use our model to demonstrate how trust requirements based on such reputation constraints can be upheld in a real world example of a distributed map processing BPEL workflow.

The rest of the paper is structured as follows. In Section II, 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 III, 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 IV, we show how reputation constraints can be specified, generated and enforced in workflows. Finally, in Section V we give an overview of related work and in Section VI, we conclude giving directions for future research.

II. BPEL Overview

The Business Execution Language for Web Services (BPEL4WS, simply called here BPEL) [2] is a standard specification language for expressing Web service workflows that was adopted by the Organization for the Advancement of Structured Information Standards (OASIS[3]). 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. 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.

[1] aws.amazon.com/swf

[2] www.salesforce.com/platform/cloud-platform/workflow.jsp

[3] www.oasis-open.org

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 [3], that promote the use of BPEL as a choreography language.

BPEL has also been recently proposed for Clound computations. Figure 1 depicts the BPEL metamodel. Based on this model, the lifecycle of a BPEL-based executable workflow process is 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 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.

A. BPEL Abstract Syntax

We adopt here an abstract syntax for the BPEL language as defined by [4] and shown in Figure 2. The syntax defines a BPEL business process as a pair, $\{\!|\ B, F\ |\!\}$, 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.

B. 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 [5]. 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:

- 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

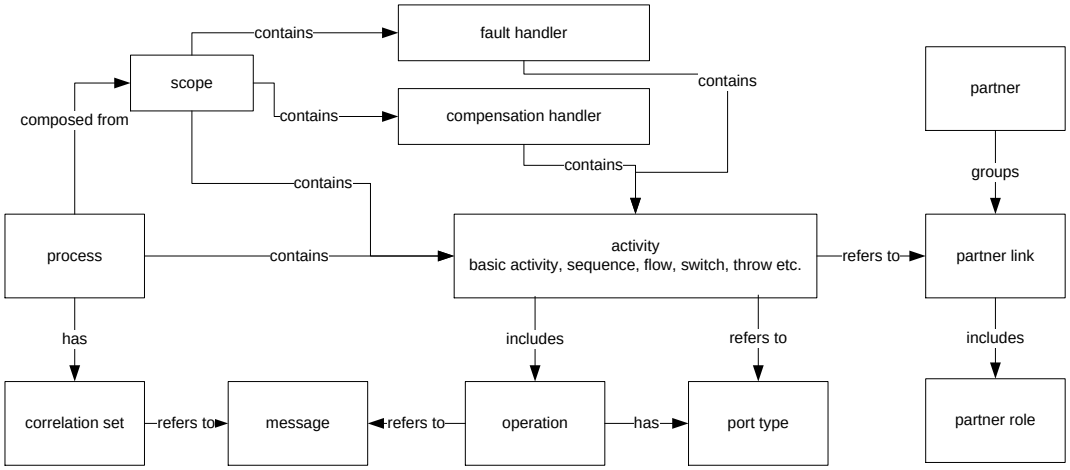

Figure 1. The BPEL Metamodel.

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
	$\mid$	$switch(\langle case\ b_1 : B_1\rangle, \ldots, \langle case\ b_n : B_n\rangle, \langle otherwise\ B\rangle)$	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.

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.

- 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.
- 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.
- 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.
- 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.
- 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 $sequence(B_1, sequence(\ldots, B_n))$.

III. A Utility-based Reputation Model for Workflows

In this section, we provide an adaptation of the model presented in [6] for the case of BPEL-based workflows. 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

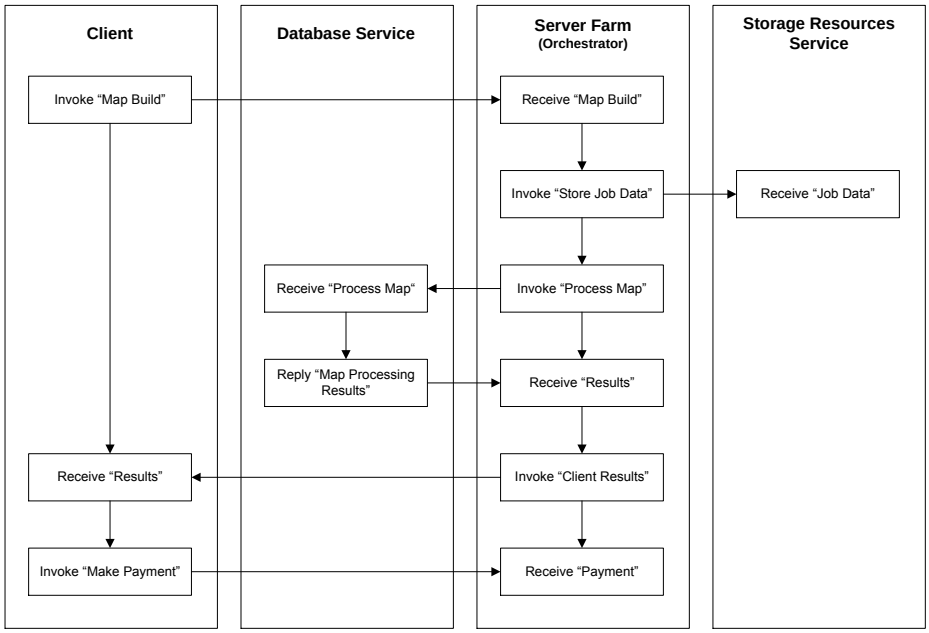

Figure 3. Workflow for the Distributed Map Processing Application.

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

where,

$$
\begin{aligned}
C_{req} = \quad & sequence(invoke(Client, resendMap, \\
& \qquad\qquad\qquad\qquad Map\ Build\ Port), receive(Client, mapBuild, Map\ Build\ Port))
\end{aligned}
$$

and,

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

Figure 4. The Server Farm Process.

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:

$$| \quad sSP : Sps \rightarrow \mathbb{P}\, Srv$$

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

$$| \quad wS : WId \nrightarrow \mathbb{P}\, Srv$$

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:

$$| \quad Event : TimeStamp \times Srv \times Iss \times WId \times \mathbb{R}$$

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 [6] 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):

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

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

$$SLA : Srv \times Iss \times WId \to \mathbb{R}$$

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.

A. 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:

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

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:

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

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:

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

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:

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

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:

$$\begin{array}{|l} [Sps] \\ \hline srv\_rep\_p : TimeStamp \times Sps \to [0,1] \\ \hline \forall t : TimeStamp, p : Sps \bullet \\ srv\_rep\_p(t,p) = \frac{\sum_{s \in sSP(p)} srv\_rep\_s(t,s)}{\#sSP(s)} \end{array}$$

IV. Reputation Constraints in BPEL Workflows

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 functionidentifiers (hence treating all constraints as of the same type). A set of reputation constraints can be obtained using the following function:

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

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.

A. Generating the Reputation Constraints

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* [7]. 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, $Cons_{y1} \ldots Cons_{yk}$ then one can generate the values for $Cons_{yi}.Min$ and $Cons_{yi}.Max$ for each $i \in \{1, \ldots, k\}$ provided that the following two constraints on the generated values (solutions) are met:

$$\frac{\sum_{i\in\{1,\ldots,k\}} Cons_{yi}.Min}{k} = Cons_x.Min$$

$$\frac{\sum_{i\in\{1,\ldots,k\}} Cons_{yi}.Max}{k} = Cons_x.Max$$

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 maxima and the minima. 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 II-B 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 constriants 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 $0.5 \leq srv\_rep\_w(t, distributed\_map\_processing) \leq 0.75$.

B. Enforcing the Reputation Constraints

Next, we discuss how the reputation constraint generated in the previous section can be enforced on the semantics of the BPEL abstract syntax. In [4], the authors define a big-step semantics for the same subset of the BPEL syntax of Section II-A based on a *transition system*, $\longrightarrow$:

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

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

$\dot{\square}$: successful process termination.
$\boxtimes$: unsuccessful process termination with an error.
$\overline{\boxtimes}$: premature forced termination.

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. In [8], this semantics was extended to deal with fine-grained access control policies controlling the termination outcome of BPEL processes.

Here, we shall extend the transition system of [4] 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 transitio step, $w \in WId$ the id of the workflow and $a \in B$ being a BPEL basic activity.

We modify the transition system of [4] as follows:

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

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. $\square, F$), the captured hooks must be a set to reflect all the small-step transitions not visible in this semantics.

A reputation monitoring system can be itself defined as:

$$\mathcal{M} : \Omega \to \mathbb{P}\, Event$$

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 5 (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, *srv_rep_x* for $x \in \{wsi, ws, w, s, p\}$ based on the events generated from $\mathcal{M}(\omega)$. We write such internal function as $update(\mathcal{M}(\omega), srv\_rep\_x) = srv\_rep\_x'$, where srv_rep_x' is an updated reputation relation calculated based on the model of Section III-A. Hence, for a new srv_rep_x', a service, workflow or service provider will have a new reputation value that can be obtained by applying srv_rep_x' to the appropriate parameters.

Hence, for the example of Server Farm process, and given the monitoring events of Figure 5, one can get the following intermediate average values for *srv_rep_wsi* for the cases of PE and A respectively as shown in Figure 6 (where we assume that $\varphi(t, ts) = 1$). Both of these issues of interest are within the acceptable constraints for *srv_rep_wsi* specified in Section IV-A 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, $\Gamma_{repCons(u)} \vdash P \overset{\{\omega_1 \ldots \omega_n\}}{\longrightarrow} \square, F$ has enforced the reputation constraints specified in $repCons(u)$ for some user u during the course of its transitions ending in the terminated state $\square, F$ if and only if the following holds true:

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

Examining the results of Figure 6, we can see that none of the two constraints is violated in its intermediate values according to the above definition of reputation constraints enforcement.

V. 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 [4] 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) [9] or based on collected referral information (indirect trust) [10]. 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 [11].

There are many works in the literature that tackle the security and trust management of workflow-based systems. In [12], 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

[4] http://www.merriam-webster.com/

$$\mathcal{M}(\quad (\textit{server\_farm}, \textit{distributed\_map\_processing}, \textit{receive}(\textit{Client}, \textit{mapBuild}, \textit{MapBuildPort}))) = \{(12:09:52, \textit{server\_farm}, \textit{PE}, \textit{distributed\_map\_processing}, 0.6), (12:09:52, \textit{server\_farm}, A, \textit{distributed\_map\_processing}, 0.5)\}$$

$$\mathcal{M}(\quad (\textit{server\_farm}, \textit{distributed\_map\_processing}, \textit{invoke}(\textit{Storage Resources}, \textit{storeJobData}, \textit{ResourcePort}))) = \{(12:09:57, \textit{server\_farm}, \textit{PE}, \textit{distributed\_map\_processing}, 0.51), (12:09:57, \textit{server\_farm}, A, \textit{distributed\_map\_processing}, 0.43)\}$$

$$\mathcal{M}(\quad (\textit{server\_farm}, \textit{distributed\_map\_processing}, \textit{invoke}(\textit{Processing Centre}, \textit{processMap}, \textit{ProcessPort}))) = \{(12:10:02, \textit{server\_farm}, \textit{PE}, \textit{distributed\_map\_processing}, 0.46), (12:10:02, \textit{server\_farm}, A, \textit{distributed\_map\_processing}, 0.22)\}$$

$$\mathcal{M}(\quad (\textit{server\_farm}, \textit{distributed\_map\_processing}, \textit{receive}(\textit{Processing Centre}, \textit{inputProcessingResults}, \textit{ProcessResultsPort}))) = \{(12:10:04, \textit{server\_farm}, \textit{PE}, \textit{distributed\_map\_processing}, 0.77), (12:10:04, \textit{server\_farm}, A, \textit{distributed\_map\_processing}, 0.54)\}$$

$$\mathcal{M}(\quad (\textit{server\_farm}, \textit{distributed\_map\_processing}, \textit{reply}(\textit{Client}, \textit{mapResults}, \textit{MapResultsPort}))) = \{(12:10:09, \textit{server\_farm}, \textit{PE}, \textit{distributed\_map\_processing}, 0.81), (12:10:09, \textit{server\_farm}, A, \textit{distributed\_map\_processing}, 0.33)\}$$

$$\mathcal{M}(\quad (\textit{server\_farm}, \textit{distributed\_map\_processing}, \textit{receive}(\textit{Client}, \textit{makePayment}, \textit{PaymentPort}))) = \{(12:10:17, \textit{server\_farm}, \textit{PE}, \textit{distributed\_map\_processing}, 0.71), (12:10:17, \textit{server\_farm}, A, \textit{distributed\_map\_processing}, 0.49)\}$$

$$\mathcal{M}(\quad (\textit{server\_farm}, \textit{distributed\_map\_processing}, \textit{invoke}(\textit{Client}, \textit{allOK}, \textit{Payment}))) = \{(12:10:22, \textit{server\_farm}, \textit{PE}, \textit{distributed\_map\_processing}, 0.5), (12:10:22, \textit{server\_farm}, A, \textit{distributed\_map\_processing}, 0.29)\}$$

Figure 5. Events Transmitted by the Reputation Monitor of the Server Farm Process.

$$\begin{aligned}
&\textit{srv_rep_wsi}(12:09:52, \textit{server_farm}, \textit{PE}, \textit{distributed_map_processing}) = 0.6\\
&\textit{srv_rep_wsi}(12:09:57, \textit{server_farm}, \textit{PE}, \textit{distributed_map_processing}) = 0.56\\
&\textit{srv_rep_wsi}(12:10:02, \textit{server_farm}, \textit{PE}, \textit{distributed_map_processing}) = 0.52\\
&\textit{srv_rep_wsi}(12:10:04, \textit{server_farm}, \textit{PE}, \textit{distributed_map_processing}) = 0.59\\
&\textit{srv_rep_wsi}(12:10:09, \textit{server_farm}, \textit{PE}, \textit{distributed_map_processing}) = 0.63\\
&\textit{srv_rep_wsi}(12:10:17, \textit{server_farm}, \textit{PE}, \textit{distributed_map_processing}) = 0.64\\
&\textit{srv_rep_wsi}(12:10:22, \textit{server_farm}, \textit{PE}, \textit{distributed_map_processing}) = 0.62
\end{aligned}$$

$$\begin{aligned}
&\textit{srv_rep_wsi}(12:09:52, \textit{server_farm}, A, \textit{distributed_map_processing}) = 0.5\\
&\textit{srv_rep_wsi}(12:09:57, \textit{server_farm}, A, \textit{distributed_map_processing}) = 0.47\\
&\textit{srv_rep_wsi}(12:10:02, \textit{server_farm}, A, \textit{distributed_map_processing}) = 0.38\\
&\textit{srv_rep_wsi}(12:10:04, \textit{server_farm}, A, \textit{distributed_map_processing}) = 0.42\\
&\textit{srv_rep_wsi}(12:10:09, \textit{server_farm}, A, \textit{distributed_map_processing}) = 0.4\\
&\textit{srv_rep_wsi}(12:10:17, \textit{server_farm}, A, \textit{distributed_map_processing}) = 0.4\\
&\textit{srv_rep_wsi}(12:10:22, \textit{server_farm}, A, \textit{distributed_map_processing}) = 0.4
\end{aligned}$$

Figure 6. Intermediate Reputation Values for *srv_rep_wsi* for the case of Performance Effeciency and Availability Issues of Interest.

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 [13] 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 [14], 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).

In [8], 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 [8]. In [6], the authors proposed a general reputation model for collaborative computing systems as a measure for trust in such systems.

In the context of composite services systems, [15], the authors propose an architecture for automated, dynamic, pro-active, and transparent maintenance and improvement of composite services, which in [16], [17] is extended to deal with the reputation of Web services based on QoS issues. In [18] a model of *probabilistic success* of BPEL-based systems is defined, and in [19], 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.

VI. Conclusion

Reputation is one means for measuring trust among entities in a distributed system. In this paper, we have demonstrated how reputation expressed as a constraint in a mathematical model, can be used to control the execution of a BPEL-based workflow. Our model is capable of capturing and enforcing constraints on the reputation of workflows, individual services and service providers. We have shown one

real world example of the applicability of the model related to the domain of distributed map processing applications.

Future research could focus on enhancing the robustness of the model by adding reliability measures to the events generated by the reputation monitor. 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. 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] 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/

[2] BEA and IBM and Microsoft and SAP and Siebel, "Web Services Business Process Execution Language Version 2.0," OASIS Standard, 2007.

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

[4] Z. Qiu, S. Wang, G. Pu, and X. Zhao, "Semantics of BPEL4WS-Like Fault and Compensation Handling," in *Proceedings of the International Symposium of Formal Methods Europe (FM 2005)*, ser. Lecture Notes in Computer Science, vol. 3582. Newcastle, UK: Springer, 2005, pp. 350–365.

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

[6] 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.

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

[8] B. Aziz, A. Arenas, F. Martinelli, I. Matteucci, and P. Mori, "Controlling usage in business process workflows through fine-grained security policies," in *Proceedings of the 5th international conference on Trust, Privacy and Security in Digital Business*, ser. TrustBus '08. Springer-Verlag, 2008, pp. 100–117.

[9] 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.

[10] A. Abdul-Rahman and S. Hailes, "Supporting trust in virtual communities," in *HICSS '00: Proceedings of the 33rd Hawaii International Conference on System Sciences-Volume 6*. Washington, DC, USA: IEEE Computer Society, 2000.

[11] 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.

[12] J. Mendling, M. Strembeck, G. Stermsek, and G. Neumann, "An Approach to Extract RBAC Models from BPEL4WS Processes," in *Proceedings of the Thirteenth IEEE International Workshops on Enabling Technologies (WETICE 2004): Infrastructure for Collaborative Enterprises*. Modena, Italy: IEEE Computer Society, 2004, pp. 81–86.

[13] E. Bertino, J. Crampton, and F. Paci, "Access Control and Authorization Constraints for WS-BPEL," in *Proceedings of the 2006 IEEE International Conference on Web Services*. Chicago, Illinois, USA: IEEE Computer Society, 2006, pp. 275–284.

[14] J. Crampton, "An Algebraic Approach to the Analysis of Constrained Workflow Systems," in *Proceedings of the 3rd Workshop on Foundations of Computer Security*, 2004, pp. 61–74.

[15] D. Bianculli, R. Jurca, W. Binder, C. Ghezzi, and B. Faltings, "Automated dynamic maintenance of composite services based on service reputation," in *Proceedings of the 5th international conference on Service-Oriented Computing*, ser. ICSOC '07. Springer-Verlag, 2007, pp. 449–455.

[16] D. Bianculli, W. Binder, L. Drago, and C. Ghezzi, "Transparent reputation management for composite web services," in *Proceedings of the 2008 IEEE International Conference on Web Services*, ser. ICWS '08. IEEE Computer Society, 2008, pp. 621–628.

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

[18] Y. Chen and X. Wu, "Success measurement of web services with bpel," in *Proceedings of the 2010 Fifth IEEE International Symposium on Service Oriented System Engineering*, ser. SOSE '10. IEEE Computer Society, 2010, pp. 86–90.

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