Transparent Reputation Management for Composite Web Services

Domenico Bianculli Walter Binder
University of Lugano
Faculty of Informatics
Lugano, Switzerland
domenico.bianculli@lu.unisi.ch
walter.binder@unisi.ch

Luigi Drago Carlo Ghezzi
Politecnico di Milano
DEEP-SE group - DEI
Milano, Italy
drago.luigi@gmail.com
carlo.ghezzi@polimi.it

Abstract

The dependability of composite services is largely affected by their constituent Web services. Composite services have to operate in an open and dynamically changing environment in order to leverage the best performing services available at the moment. Hence, there is the need for an efficient mechanism to provide reliable service rankings.

In this paper we present a novel, generic, and customizable reputation infrastructure to automatically and transparently monitor the execution of composite services, taking both functional and non-functional properties into account. The experienced Web service Quality-of-Service is communicated to a configurable reputation mechanism that publishes service rankings. Our reputation infrastructure supports notifications upon changes in service reputation, enabling self-tuning and self-healing properties in the execution of composite services.

We implemented our architecture using standard technologies, such as BPEL and JavaEE. Performance measurements show that our infrastructure causes only moderate overhead.

Keywords: *Reputation mechanisms, composite web service execution, web service monitoring, middleware, BPEL.*

1. Introduction

The dependability of composite services, such as workflows or business processes, is largely affected by their constituent Web services[1]. Web service-based software architectures represent *open-world software*, as described in [4]. Composite services have to adapt to the open, dynamically changing environment where remote services may fail or new services may be offered at any moment. The ability to bind to required Web services at run time is a key mechanism to cope with the challenges of open-world software. As the market of available services for a given functionality changes over the time, composite services that depend on that functionality need to evolve, adapting their service bindings so as to leverage the best performing services currently available.

Selecting the best service for a required functionality presupposes a reliable and efficient mechanism to provide the service rankings. For this purpose, *reputation mechanisms* have been proposed [19]. They collect clients' ratings on experienced service behavior to compute the actual Quality-of-Service (QoS) delivered to clients and to rank functionally-equivalent services accordingly. Reputation mechanisms therefore promote the sharing of service monitoring information amongst clients. Researchers have shown that reputation mechanisms can be designed to provide incentives that make honest reporting rational for the clients [11].

However, current standard environments for the execution of composite services, such as BPEL [2] engines, do not integrate any reputation mechanisms. Although it is possible to program composite services that explicitly interact with a reputation mechanism so as to report feedback on service interactions and to dynamically choose the most efficient services, the needed development effort is prohibitive in practice. Moreover, feedback reporting on both experienced service functionality and QoS presumes an appropriate monitoring infrastructure.

To overcome these issues, we designed a reputation infrastructure that serves the following goals:

1. to provide a mechanism for assessing service behavior and ranking functionally-equivalent services based on past interactions with these services by other clients;

2. to support user notifications when particular reputation-related events occur, allowing for an

[1]In this paper, we use the terms "service" and "Web service" interchangeably.

early discovery of possible failure situations;

3. to ensure reputation-enabled execution of composite services in a way that is completely transparent to the programmer, who can concentrate exclusively on the functional aspects of the composite service;

4. to allow for an open and extensible platform supporting a high degree of customization of the way services' reputation is computed.

The scientific contributions of this paper are the proposed architecture of our reputation infrastructure, enabling the transparent integration of reputation mechanisms in standard execution environments for composite services, as well as our reference implementation built from state-of-the-art middleware technologies.

Our architecture includes a customizable reputation mechanism that is integrated with a customizable UDDI service repository, thus enabling reputation-aware service selection. The execution environment running the composite service (a BPEL engine) is instrumented for monitoring service invocations and reporting feedback to a reputation mechanism that computes services' reputation.

The reputation mechanism supports subscriptions for service functionalities, resulting in notifications upon changes in service reputation and upon the availability of better performing services for a given functionality, respectively. These notifications enable the automated update of service bindings, ensuring the automated evolution of composite service in response to a dynamically changing service market. Although, for example, upon receiving the notification that the reputation of a service has dropped below a given threshold, a client could replace the affected service and hence avoid possible problems before they actually occur, in this paper we focus on the generic reputation infrastructure itself and do not address the concrete actions taken upon reputation-related events, since these actions are specific to client policies.

To validate and evaluate our approach, we measured the overhead caused by our infrastructure both for deployment and execution of composite services. We explore separately the different aspects of our instrumentation, service execution monitoring and communication with an external reputation mechanism, to assess how much each of them contributes to the overall observed overhead.

The rest of this paper is structured as follows. Section 2 gives an overview of our reputation infrastructure. Section 3 describes the specification languages we support within the monitoring architecture underlying the reputation infrastructure. Section 4 presents the technique we use to estimate service reputation. Section 5 explains the implementation of the main components. Section 6 presents the results of our experimental evaluation. Section 7 surveys related work. Final remarks and directions for future research are given in Section 8.

2. System overview

In the following, we describe the software architecture of our reputation infrastructure, both at the server- and the client-side. Afterwards, we explain the interactions among the system components.

2.1. Server-side software architecture

The architecture of the reputation infrastructure comprises three main components: the *enhanced registry*, the *reputation manager*, and the *subscription manager*.

Enhanced Registry. It is a UDDI-compliant registry extended with functionalities supporting the reputation infrastructure. As a UDDI registry, it provides standard UDDI interfaces, which supports service publishing and service discovery. The main extension this registry includes is the functionality to query for QoS estimations of registered services. The registry can be queried either by providing a specific service TModel or by providing the concrete service location and the WSDL interface it complies to.

Reputation Manager. It provides functionalities to manage the services registered for reputation and to estimate their QoS. It exposes a public message queue where service clients may post their feedback reports. Moreover, the *Reputation Manager* receives UDDI-related events from the *Enhanced Registry*. For instance, when a new service is registered into the service directory, this component creates the objects needed to represent those entities inside the infrastructure and initializes their reputations to a default value.

The *Reputation Manager* is in charge of managing *reputation policies*. The *reputation policy* is our abstraction for an algorithm that estimates service reputation. Instead of providing an on-line algorithm for immediately processing feedback reports as they arrive, we decided to introduce a scheduler, which invokes reputation policies periodically, to avoid minor fluctuations in the reputation estimates, which could trigger unnecessary notifications.

This component is also aware of the concept of a *reputation era*, which is a fixed length time interval during which the reputation estimate of resources does not change. All QoS reports received from clients during this period are stored and then processed at the end of the era. This implies that both updates and notifications of reputation-related events happen at the same

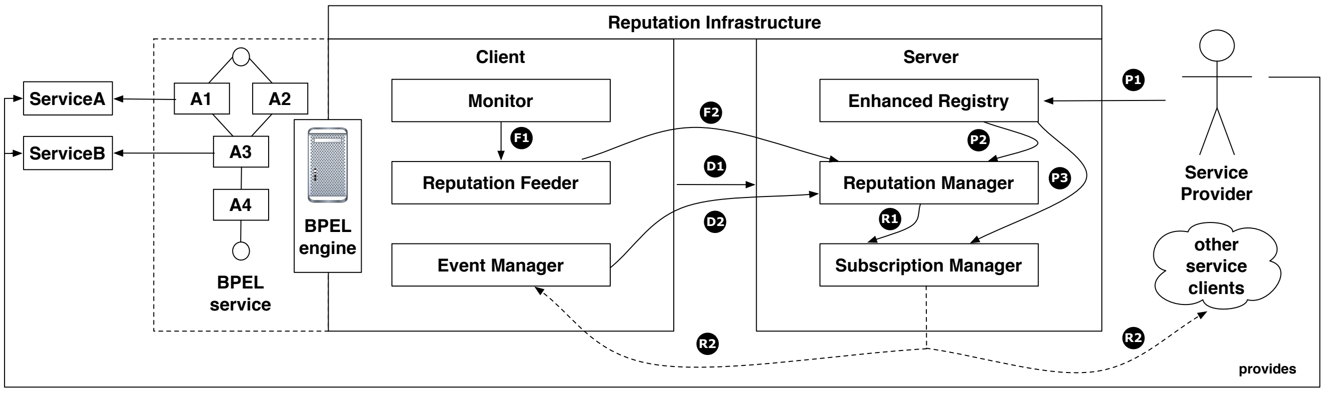

Figure 1. System architecture and interactions

time, right after the end of an era and before the beginning of the next one. This solution leads to steady QoS estimations through aggregation of feedbacks received within an era.

Subscription Manager. It provides functionalities to notify service consumers when reputation-related events occur and to manage the subscriptions to these events.

The reputation infrastructure supports two event types: *reputation decrease* and *availability of a service with better reputation*.

The former event is fired when the *Reputation Manager* communicates that the reputation of a service has dropped below a certain threshold. Service users are thus notified of a possible failure condition by means of these messages, so that countermeasures can be taken. Upon subscription, each service client specifies the services for which it should receive notifications on reputation decrease and the reputation threshold for each service.

The latter event is used to notify service clients when the set of the "best" services compliant with a particular specification (i.e., the services with the currently best reputation) changes. By means of these notifications, we let service clients always know which are the best services available on the service market such that when a possible failure occurs they may rebind to another service, which exhibits a better behavior. Service clients subscribe to these events by specifying the WSDL interface they are interested in.

Furthermore, our reputation infrastructure includes some components implementing side facilities, such as security-related operations (log-in procedures and management of access credentials).

2.2. Client-side architecture

At the client-side, the architecture comprises three components:

Monitor. It monitors the behavior of external services used by the BPEL service client, by checking some functional and non-functional assertions.

Reputation Feeder. It provides methods to collect feedback reports and to send them to the server component of our reputation infrastructure.

Event Manager. It provides functionalities to subscribe to reputation-related events and to react to such notifications.

Figure 1 illustrates the components of the architecture, both at the server-side and at the client-side. It also depicts the messages exchanged when interacting with the reputation infrastructure, which will be described in the next subsection.

2.3. System interactions

A typical usage scenario of our reputation infrastructure is the following one:

1. Service providers publish their services (e.g., services A and B) using the UDDI-compliant interface offered by the *Enhanced Registry* (message P1 in Fig. 1). Internally, the *Enhanced Registry* notifies the *Reputation Manager* that a new service has been registered, and thus that a default reputation should be assigned to it (message P2). The *Enhanced Registry* also notifies the *Subscription Manager* such that it can notify interested service clients of the availability of a new service (message P3).

2. When service clients deploy their business processes into the BPEL engine, the client part of the reputation infrastructure logs into the server part (message D1), in order to get access credentials for subsequent communications. Service clients communicate the selected service bindings to the server using the *Event Manager* (message D2); in this way, clients subscribe to events related to (the type of) services they use. For example, the BPEL service depicted in Fig. 1 will communicate to the *Reputation Manager* its bindings to services A and B, used within the business process by the activities A1 and A3.

3. During execution, each time a client uses an external service, the built-in monitor evaluates a rule associated with the interaction. The result of the evaluation is sent to the *Reputation Feeder* (message F1), which generates a feedback report on the behavior of the external service, to be sent (message F2) to the *Reputation Manager*, on the server component of the reputation infrastructure.

4. After collecting reputation feedback reports, the *Reputation Manager* updates the reputation estimation of the services registered in the system. Whenever the *Reputation Manager* computes a new value of the reputation of a service, it notifies the *Subscription Manager* (message R1). The latter can then either communicate (message R2) to all subscribed clients that the reputation of a service dropped below a certain threshold, or it can notify them that a new service implementing a certain WSDL interface and with a better reputation became available.

3. Monitoring languages

Within our reputation infrastructure, feedback messages are defined as behavioral reports about a single service. Since feedbacks originate from the evaluation of monitoring rules, we require these rules to assert properties whose evaluation depend on the interaction with a single service. This requirement is enforced in the two languages for specifying monitoring rules, WSCoL and ALBERT, supported by our reputation infrastructure.

WSCoL [5] (Web Service Constraint Language) allows for asserting functional properties on the BPEL activities interacting with external services, i.e., the *invoke*, *receive*, and *pick* activities. Properties are expressed in the form of pre- and post-conditions, and can refer only to input and output variables of the activity or to historical variables stored in previous rule evaluations: this guarantees that each WSCoL rule can only express properties about an interaction with a single external service. Therefore, this language can be used in our reputation infrastructure without any restrictions.

ALBERT [3] (Assertion Language for BPEL Process Interactions) is a temporal logic language that can assert both functional and non-functional properties of BPEL processes. The high expressiveness of the language requires some restrictions on the language for using it within our reputation infrastructure.

First of all, feedback generation should be enabled only for a specific type of formula, called *Assumed Assertion* in ALBERT, which asserts properties on the external services a BPEL process interacts with. Furthermore, the events used in ALBERT predicates should be related only to the activities that exchange messages with external services, such as *invoke*, *receive*, *pick*, and *onMessage* event handlers. Finally, each formula may contain references of at most one external service.

4. Reputation estimation

The reputation infrastructure has been designed in an open and extensible way, so as to support different methods for computing service reputation and to enable reputation estimation for new kinds of entities (e.g., the reputation of a service provider could be defined by aggregating the feedback reports received for all the services offered by the same provider). The *Reputation Manager* ensures extensibility through the installation of new *reputation policies* provided as plugins.

The default reputation policy plugin in our reference implementation estimates the reputation of each single service published in the infrastructure by using a *binary-based rating approach with reputation propagation*. *Reputation propagation* captures the concept that the *Reputation Manager* builds reputations by using indirect knowledge of services. *Binary-based* means that service clients can rate services by using only boolean values. These values correspond to the evaluation of logical formulae associated with monitoring rules, from which feedback reports are created.

This *binary-based rating approach* is based on the *endorsements-refusals ratio* algorithm. It generates the reputation by computing the ratio of the number of positive feedbacks and the total number of feedbacks received until the computation of the estimation is triggered by the system.

Let S be the set of services published in the reputation infrastructure; F be the set of feedbacks f, where each f is a tuple $\langle s, v, t \rangle$, with $s \in S$ being the service that is the object of the feedback, $v \in \{0, 1\}$ the value of the feedback and $t \in \mathbb{R}^+$ the time-stamp at which the feedback is received at the server. Let $F_{s,t} = \{ f \in F \mid f.s = s \wedge f.t <= t \}$, $s \in S$, $t \in \mathbb{R}^+$, be the feedback set, i.e, the set of the feedbacks received for a service s until time t; let $P(s,t) = \sum_{f \in F_{s,t}} f.v$ be the amount of endorsement received for service s until time t, and $N(s,t) = |F_{s,t}|$ be the number of total feedbacks received for a service s until time t. The reputation $\rho(s,t)$ for a service s at instant t is then computed using the endorsement-refusals ratio as:

$$\rho(s,t) = \frac{P(s,t)}{N(s,t)}$$

5. Implementation

The reputation infrastructure has been entirely implemented as a JavaEE compliant application; the reason for this choice is that the JavaEE platform is the *de facto* standard for the development of back-end and distributed applications.

The *Reputation Manager* and the *Subscription Manager* have been implemented by means of both stateless session beans and message driven ones.

Most of the functionalities of the *Enhanced Registry* have been implemented by means of stateless session beans; standard UDDI services are instead provided by means of Web service beans and by the Grimoires UDDI registry [29]. Grimoires is a registry enhanced with functionalities to add metadata information to UDDI concepts. We adopted it both because it is open-source and because it allows for storing reputation of services directly as a metadata in the registry. Actually, we used a modified version of Grimoires, extended to support notifications about changes in the database of UDDI entities.

Some operations, such as finding the best services compatible with a certain interface, require a notion of service equivalence. We implemented a component that performs the analysis of WSDL documents —associated with UDDI TModels— based on syntactic checking[2], and generates sets of compatible services.

At the client-side, service monitoring is performed within the ActiveBPEL engine [1]. The version we have used has been already instrumented with the Dynamo monitoring facility [6]. We extended this version with a functionality to send feedback when a monitoring rule is evaluated, by using an aspect-oriented approach [13] so as to minimize the impact on pre-existing code.

In terms of security, every message exchanged in the system is secured against tampering; access to the system is granted by means of a public key mutual authentication algorithm.

[2]The use of TModels (as pointers to WSDL documents) for checking service compatibility is recommended in the UDDI specifications. However, as pointed out by the semantic web research community, a syntactic comparison of WSDL service specifications could not be sufficient for determining service equivalence. Several efforts are dealing with this issue [22, 20]; however they are out of the scope of this work. Therefore, we designed the service equivalence checker as a replaceable component to support different models of service equivalence.

6. Experimental evaluation

We evaluated the performance impact of our reputation infrastructure with two BPEL processes, the *LoanApproval* process defined in the BPEL specification [2] and the *Radiology* process available in the WSCoL monitoring distribution[6]. In both cases, we implemented the external services required by the BPEL process.

For each process, we measured both the BPEL process deployment time and the process execution time in three different configurations: *(original)* vanilla BPEL engine without any instrumentation; *(monitor instrumentation)* BPEL engine instrumented to support the Dynamo monitoring facility; *(complete instrumentation)* BPEL engine instrumented to support both the monitor and the reputation infrastructure.

For our measurements, all components of our infrastructure as well as the external services required by the two BPEL processes were started on a single machine, an Intel Centrino Duo T2300 CPU with 2GB RAM, running GNU/Linux. To ensure reliable measurements, we removed unnecessary processes as much as possible. The reputation infrastructure was deployed on JBoss AS 4.2.0-GA. We also used MySQL 5.0.45 as DBMS, ActiveBPEL 2.5 as BPEL engine, and the Grimoires Enhanced Registry 1.2.3 as UDDI registry. The BPEL engine and the Grimoires registry were deployed on Apache Tomcat 5.5.25.

Regarding deployment time, we measured the wallclock time taken by the BPEL engine to deploy the process. For each experiment, the application container was restarted. Concerning process execution time, we ran an external client that invoked the BPEL process 10 times with different parameters and we measured the overall wallclock time taken by that client. Due to the complexity of our middleware, measurements are not exactly reproducible; this is a well-known phenomenon, for example in Java-based environments, where measurement variances due to application-inherent non-determinism are often amplified by differences in thread scheduling, dynamic just-in-time compilation, or garbage collection [8]. In order to compensate for the measurement variances, we repeated each experiment 10 times (under the same settings) and reported the geometric mean of the 10 trials.

As for the test configuration of the reputation infrastructure, we set the reputation era interval to 10s, and we used the *endorsements-refusals ratio* as reputation policy.

Table 1 shows the measured deployment time and the execution time for each process. For each trial, in addition to the execution time, we also show the relative overhead factor ("ovh") with respect to the measurement in the original, vanilla setting.

At deployment time, performance degradation is mostly due to the reputation infrastructure instrumentation. In fact,

Table 1. Performance Analysis

Deployment Time

	original	monitor instrumentation		complete instrumentation	
LoanApproval	**[s]**	**[s]**	**ovh**	**[s]**	**ovh**
trial1	3.96	4.12	**4.04%**	6.91	**74.49%**
trial2	3.85	4.15	**7.79%**	6.72	**74.55%**
trial3	4.03	4.08	**1.24%**	6.82	**69.23%**
trial4	3.84	4.13	**7.55%**	6.74	**75.52%**
trial5	4.05	4.19	**3.46%**	6.83	**68.64%**
trial6	4.01	4.08	**1.75%**	6.94	**73.07%**
trial7	3.91	4.08	**4.35%**	6.72	**71.87%**
trial8	4.02	4.18	**3.98%**	6.92	**72.14%**
trial9	4.01	4.18	**4.24%**	6.91	**72.32%**
trial10	3.99	4.11	**3.01%**	6.87	**72.18%**
Geo. mean	3.97	4.13	**3.63%**	6.84	**72.37%**
Radiology	**[s]**	**[s]**	**ovh**	**[s]**	**ovh**
trial1	4.11	4.47	**8.76%**	11.40	**177.37%**
trial2	3.97	4.12	**3.78%**	10.62	**167.63%**
trial3	4.24	4.43	**4.48%**	10.75	**153.50%**
trial4	3.87	4.25	**9.82%**	10.57	**173.04%**
trial5	3.95	4.31	**9.11%**	10.94	**176.78%**
trial6	4.08	4.38	**7.35%**	10.91	**167.36%**
trial7	4.00	4.18	**4.50%**	10.13	**153.17%**
trial8	3.80	4.09	**7.63%**	9.37	**146.66%**
trial9	3.98	4.24	**6.53%**	11.22	**182.14%**
trial10	4.04	4.13	**2.23%**	9.83	**143.49%**
Geo. mean	4.00	4.26	**5.87%**	10.56	**163.58%**

Execution Time

	original	monitor instrumentation		complete instrumentati7on	
LoanApproval	**[s]**	**[s]**	**ovh**	**[s]**	**ovh**
trial1	0.39	0.48	**23.08%**	0.57	**46.15%**
trial2	0.35	0.45	**28.57%**	0.59	**68.57%**
trial3	0.37	0.52	**40.54%**	0.65	**75.68%**
trial4	0.30	0.41	**36.67%**	0.43	**43.33%**
trial5	0.45	0.54	**20.0%**	0.68	**51.11%**
trial6	0.31	0.38	**22.58%**	0.57	**83.87%**
trial7	0.36	0.51	**41.67%**	0.59	**63.89%**
trial8	0.36	0.42	**16.67%**	0.61	**69.44%**
trial9	0.38	0.44	**15.79%**	0.65	**71.05%**
trial10	0.41	0.51	**24.39%**	0.73	**78.05%**
Geo. mean	0.37	0.46	**25.55%**	0.60	**63.67%**
Radiology	**[s]**	**[s]**	**ovh**	**[s]**	**ovh**
trial1	24.78	33.52	**35.27%**	41.51	**67.51%**
trial2	21.35	29.75	**39.34%**	38.65	**81.03%**
trial3	23.99	28.92	**20.55%**	37.89	**57.94%**
trial4	27.63	31.50	**14.01%**	42.59	**54.14%**
trial5	22.10	29.73	**34.52%**	39.82	**80.18%**
trial6	23.50	34.97	**48.81%**	37.84	**61.02%**
trial7	22.84	29.65	**29.82%**	41.56	**81.96%**
trial8	27.35	33.24	**21.54%**	36.92	**34.99%**
trial9	24.89	28.63	**15.03%**	40.93	**64.44%**
trial10	23.21	29.73	**28.09%**	38.79	**67.13%**
Geo. mean	24.09	30.90	**26.68%**	39.61	**63.35%**

during this phase, the reputation infrastructure has to analyze the business process and the WSDL definitions of the external services it uses, to find the remote services within the *Reputation Manager*'s internal registry. The complexity of this phase is proportional to the number of external services the business process interacts with. Indeed, in the case of the *Radiology* process, the analysis takes more time than in the case of the *LoanApproval* process, because the former interacts with 8 remote services, whereas the latter interacts only with 2 remote services. This explains why the complete instrumentation of the *Radiology* process causes a deployment overhead of 164% on average, while it results only in 72% overhead for the *LoanApproval* process.

Concerning process execution, the relative overhead due to the instrumentation code is surprisingly uniform for both processes, albeit their execution time in seconds is significantly different. On average, the overhead caused by the monitoring is 26–27%, whereas the overhead for complete instrumentation is about 63%.

We conclude that neither the overhead upon process deployment nor the run-time overhead upon process execution is prohibitively high. Thus, our reputation-aware process execution infrastructure can be readily used in practice.

7. Related work

Several mechanisms to evaluate trust and reputation have been proposed in literature; see [9] for a complete survey and [19] for a classification of such mechanisms driven by the concepts of contextualization and personalization.

Although in this paper QoS refers to the experienced service behavior (including both functional and non-functional aspects), often in the context of service-oriented architectures, QoS typically denotes the performance metrics of services [21]. Approaches that select services based on QoS usually extend service registries to support this type of information. An example is UDDIe [23], which extends UDDI with support for the concept of *blue pages*, i.e., information on the QoS properties of a service, which are published by service providers and can be used in queries by service clients. Match-making between properties guaranteed by providers and properties required by clients relies upon the *find* operators defined in the standard UDDI specification. A similar extended registry is described in [28], where a service broker performs QoS-based selection by using an ontology reasoning mechanism for match-making. A common weakness of these approaches is that they rely on the assumption that providers are honest and only advertise QoS properties that they can guarantee.

In [17], the authors propose a conceptual model for Web service reputation; this model is at the basis of an agent-based trust framework for service selection, described in [18]. In contrast to our approach, the authors use a different architectural style, where a software agent is attached to each Web service; the agents are in charge of querying and reporting service reputation. Each service client builds its reputation of services based on the *local* information provided by its neighbours.

A QoS-based service selection model is presented in [14]. The model takes into account the feedback from users as well as other business-related criteria; moreover it is also extensible, in order to support multiple QoS selection criteria. In comparison to our approach, it is neither pro-active (because variations of service reputation are not disseminated to other service clients), nor transparent (since service requesters are required to support specific mechanisms for ad-hoc execution monitoring and feedback reporting).

A service recommendation system is proposed in [16]. In this system, clients rate services by using a comparative matrix containing the QoS values advertised by the provider, and the QoS values measured at run time. However, the system does not use Web service standards for service discovery and selection, but relies on ontology-based descriptions. Moreover, user feedback reporting is not automated.

In [27], the authors describe a method to collect monitoring data from clients and to use this information for service recommendations. However, the supported QoS metrics are limited: they support only metrics related to client side performance, such as throughput, response time, or latency.

A collaborative filtering approach to derive prediction of QoS of Web services that were not used yet, based on the experience of consumers of similar services, is proposed in [24]. However, the whole approach is poorly integrated in the execution environment and it is neither fully automated nor transparent. Moreover, it supports only the prediction based on the evaluation of timeliness-related QoS properties.

The approach described in [30] adopts a point of view that is complementary to ours. The reputation of a composite service is derived based on the reputation of the single services used within the composition. The reputation mechanism used to compute the reputation of the single services is similar to ours.

The problem of trust and reputation management in open dynamic environments is discussed in [31]. The authors propose some guidelines to build self-organizing referral networks as a means for establishing trust in open environments. However, the technology-agnostic, simulation-based approach adopted in the paper does not allow for a concrete use in Web services-based architectures.

In an earlier paper, we proposed [7] an architecture to share reliable service quality information amongst clients, supported by a theoretical model of an incentive-compatible reputation mechanism [10]. However, the requirement of a bank paying for honest feedbacks, postulated by the theoretical model, made the implementation impractical.

Several approaches propose specification languages for defining (non-functional) service behavior and propose an associated monitoring architecture. Keller and Ludwig [12] propose WSLA and focus on monitoring QoS properties such as performance and costs. A model and an analysis technique for reasoning on the monitorability of systems are described in [26]. SLAs are expressed using the SLAng [25] language, whose semantics has been provided using a model-denotational approach.

8. Conclusion and future work

In this paper we presented a reputation-aware service execution infrastructure, which manages the reputation of Web services used by BPEL orchestrations in an automated and transparent manner.

We use monitoring techniques to collect information about functional and non-functional properties of Web service behavior. The resulting feedback data is sent to the server component of our infrastructure, which computes a reputation value for each service registered in the infrastructure. Reputation information is then propagated back to the affected service clients, which can use it in order to bind to the best available services in the market.

Regarding future work, we will explore methods to make the reputation estimation context-aware such that multiple reputation values can be associated with a service on the basis of the context in which it operates. Moreover, based on the work described in [32, 15], we will extend our system with techniques for identifying unfair ratings and thus evaluating raters' credibility. This will be also the basis for mechanisms to discourage clients from cheating when reporting feedback.

9. Acknowledgements

Part of this work has been supported by the EU project "PLASTIC" (contract number IST 026995), by the EU project "S-Cube" (funded within FP7/2007-2013 under Objective 1.2 "Services and Software Architectures, Infrastructures and Engineering") and by the Swiss National Science Foundation.

References

[1] Active Endpoints. ActiveBPEL engine. `http://www.activevos.com/`.

[2] T. Andrews, F. Curbera, H. Dholakia, Y. Goland, J. Klein, F. Leymann, K. Liu, D. Roller, D. Smith, S. Thatte, I. Trickovic, and S. Weerawarana. Business Process Execution Language for Web Services, Version 1.1, May 2003.

[3] L. Baresi, D. Bianculli, C. Ghezzi, S. Guinea, and P. Spoletini. Validation of web service compositions. *IET Softw.*, 1(6):219–232, December 2007.

[4] L. Baresi, E. Di Nitto, and C. Ghezzi. Towards Open-World Software. *IEEE Computer*, 39:36–43, October 2006.

[5] L. Baresi and S. Guinea. Towards dynamic monitoring of WS-BPEL processes. In *Proceedings of ICSOC'05*, volume 3826 of *Lecture Notes in Computer Science*, pages 269–282. Springer, December 2005.

[6] L. Baresi, S. Guinea, and L. Pasquale. Self-healing BPEL processes with Dynamo and the JBoss rule engine. In *Proceedings of ESSPE'07*, pages 11–20. ACM, 2007.

[7] D. Bianculli, R. Jurca, W. Binder, C. Ghezzi, and B. Faltings. Automated dynamic maintenance of composite services based on service reputation. In *Proceedings of ICSOC'07*, volume 4749 of *Lecture Notes in Computer Science*, pages 449–455. Springer-Verlag, September 2007.

[8] A. Georges, D. Buytaert, and L. Eeckhout. Statistically rigorous Java performance evaluation. In *Proceedings of OOPSLA'07*, pages 57–76. ACM, 2007.

[9] A. Josang, R. Ismail, and C. Boyd. A survey of trust and reputation systems for online service provision. *Decis. Support Syst.*, 43(2):618–644, 2007.

[10] R. Jurca, W. Binder, and B. Faltings. Reliable QoS Monitoring Based on Client Feedback. In *Proceedings of WWW'07*, pages 1003–1011, May 2007.

[11] R. Jurca and B. Faltings. Minimum Payments that Reward Honest Reputation Feedback. In *Proceedings of EC'06*, pages 190–199. ACM, June 2006.

[12] A. Keller and H. Ludwig. Defining and monitoring service-level agreements for dynamic e-business. In *Proceedings of the 16th Conference on Systems Administration*, 2002.

[13] G. Kiczales, J. Lamping, A. Mendhekar, C. Maeda, C. V. Lopes, J. Loingtier, and J. Irwin. Aspect-oriented programming. In *Proceedings of ECOOP'97*, volume 1241 of *Lecture Notes in Computer Science*, pages 220–242. Springer, 1997.

[14] Y. Liu, A. H. Ngu, and L. Z. Zeng. QoS computation and policing in dynamic web service selection. In *Proceedings of WWW Alt. '04*, pages 66–73. ACM, 2004.

[15] Z. Malik and A. Bouguettaya. Evaluating rater credibility for reputation assessment of web services. In *Proceedings of WISE'07*. Springer, 2007.

[16] U. S. Manikrao and T. V. Prabhakar. Dynamic selection of web services with recommendation system. In *Proceedings of NWESP '05*, page 117. IEEE Computer Society, 2005.

[17] E. M. Maximilien and M. P. Singh. Conceptual model of web service reputation. *SIGMOD Rec.*, 31(4):36–41, 2002.

[18] E. M. Maximilien and M. P. Singh. Toward autonomic web services trust and selection. In *Proceedings of ICSOC '04*, pages 212–221. ACM, 2004.

[19] L. Mui. *Computational Models of Trust and Reputation: Agents, Evolutionary Games, and Social Networks*. PhD thesis, Massachusetts Institute of Technology, 2003.

[20] M. Nagarajan, K. Verma, A. P. Sheth, J. Miller, and J. Lathem. Semantic Interoperability of Web Services - Challenges and Experiences. In *Proceedings of ICWS'06*, pages 373–382. IEEE Computer Society, 2006.

[21] L. O'Brien, L. Bass, and P. Merson. Quality attributes and service-oriented architectures. Technical Report CMU/SEI-2005-TN-014, CMU - Software Engineering Institute, Pittsburgh, PA, September 2005.

[22] S. V. Pokraev, D. A. C. Quartel, M. W. A. Steen, and M. U. Reichert. Requirements and method for assessment of service interoperability. In *Proceedings of ICSOC'06*, volume 4294 of *Lecture Notes in Computer Science*, pages 1–14. Springer, 2006.

[23] A. ShaikhAli, O. F. Rana, R. Al-Ali, and D. W. Walker. UDDIe: An extended registry for web services. In *Proceedings of SAINT'03*, pages 85–89. IEEE Computer Society, 2003.

[24] L. Shao, J. Zhang, Y. Wei, J. Zhao, B. Xie, and H. Mei. Personalized QoS Prediction forWeb Services via Collaborative Filtering. In *Proceedings of ICWS 2007*, pages 439–446. IEEE Computer Society, 2007.

[25] J. Skene, D. D. Lamanna, and W. Emmerich. Precise service level agreements. In *Proceedings of ICSE '04*, pages 179–188. IEEE Computer Society, 2004.

[26] J. Skene, A. Skene, J. Crampton, and W. Emmerich. The monitorability of service-level agreements for application-service provision. In *Proceedings of WOSP '07*, pages 3–14. ACM, 2007.

[27] N. Thio and S. Karunasekera. Web service recommendation based on client-side performance estimation. In *Proceedings of ASWEC '07*, pages 81–89. IEEE Computer Society, 2007.

[28] M. Tian, A. Gramm, T. Naumowicz, H. Ritter, and J. Schiller. A concept for QoS integration in Web services. In *Proceedings of WQW 2003*. IEEE Computer Society, 2003.

[29] S. C. Wong, V. Tan, W. Fang, S. Miles, and L. Moreau. Cluster Computing and Grid 2005 Works in Progress: Grimoires: A Grid Registry with a Metadata-Oriented Interface. *IEEE Distributed Systems Online*, 6(10), October 2005.

[30] S. J. H. Yang, J. S. F. Hsieh, B. C. W. Lan, and J.-Y. Chung. Composition and evaluation of trustworthy web services. In *Proceedings of BSN '05*, page 5. IEEE Press, 2005.

[31] P. Yolum and M. Singh. Engineering self-organizing referral networks for trustworthy service selection. *Systems, Man and Cybernetics, Part A, IEEE Transactions on*, 35(3):396–407, 2005.

[32] Y. Zou, L. Gu, G. Li, B. Xie, and H. Mei. Rectifying prejudicial feedback ratings in reputation based trust management. In *Proceedings of SCC 2007*, pages 530–535. IEEE Computer Society, July 2007.