Automated Dynamic Maintenance of Composite Services based on Service Reputation

Domenico Bianculli[1], Radu Jurca[2], Walter Binder[1],
Carlo Ghezzi[3], and Boi Faltings[2]

[1] Faculty of Informatics – University of Lugano
via G. Buffi 13 - CH-6900, Lugano, Switzerland
domenico.bianculli@lu.unisi.ch, walter.binder@unisi.ch
[2] Artificial Intelligence Lab – Ecole Polytechnique Fédérale de Lausanne
Station 14 - CH-1015, Lausanne, Switzerland
radu.jurca@epfl.ch, boi.faltings@epfl.ch
[3] Dipartimento di Elettronica e Informazione – Politecnico di Milano
Via Ponzio 34/5, I-20133, Milano, Italy
ghezzi@elet.polimi.it

Abstract. Service-oriented computing promotes the construction of applications by composing distributed services that are advertised in an open service market. In such an environment, individual services may change and evolve dynamically, requiring composite services to adapt to such changes. The prevailing strategy is to react on failures and replace the defective component of the composite service. However, this reactive approach does not fully exploit the opportunities of a dynamic market where older services may be replaced by better ones.
In this paper we promote a novel architecture for automated, dynamic, pro-active, and transparent maintenance and improvement of composite services. We leverage fine-grained client-side monitoring techniques to generate information regarding functional and non-functional properties of service behavior. A reputation manager is responsible for collecting and aggregating this information, and provides economical incentives for honest sharing of feedback. Composite services can thus use reliable reputation information to pro-actively improve their aggregate performance.

1 Introduction

The need for businesses to integrate corporate resources in a flexible and efficient way can be addressed by designing complex software solutions as collaboration of contractually defined services. Building applications by integrating standardized services promises to bring many benefits, such as reduced development effort and cost, ease of maintenance, extensibility, and reuse of services. Service-oriented architectures (SOAs) maximize decoupling between services and create well-defined interoperation semantics based on standard protocols.

In the following we consider service-oriented applications built from web services.[4] The composition of individual services into an added-value, composite

[4] In this paper, we use the terms *web service* and *service* interchangeably.

service is usually represented as a workflow. We assume that service compositions are described in BPEL [1], the *de-facto* standard for web service orchestrations.

Web services support a dynamic architectural style where the binding among components may change at runtime. New services may be developed and published in registries, and then discovered by possible clients. Previously available services may disappear or become unavailable. This situation has been characterized by the term *open-world software* [2], describing a situation where applications are composed out of parts that may change unpredictably and dynamically. It has been observed that open-world software introduces the requirement of continuous validation. Since a software architecture evolves dynamically, validation must extend from development time to runtime.

In order to ensure that composite services are executing as expected, it is necessary to monitor the interactions of individual services within a workflow. Monitoring involves both service functional behavior and non-functional properties, such as Quality-of-Service (QoS) parameters. If services are advertised by Service-Level Agreements (SLAs) that regulate service cost and QoS (e.g., maximum response time), monitoring delivered QoS allows clients to verify that they actually receive the QoS they are expecting and paying for.

When clients executing workflows observe failures or SLA violations of individual services, they have to replace the failing or badly behaving services. However, finding a replacement may take some time, resulting in reduced availability of the composite service. Moreover, there are no guarantees that the replacement will work better than the replaced service.

In this paper, we promote the sharing of service monitoring information amongst clients in order to enable the *pro-active replacement of misbehaving services* in workflows. The original contribution of the paper is an integrated infrastructure for service monitoring and maintenance of composite services. We promote novel techniques for monitoring composite services and introduce an incentive-compatible Reputation Manager (RM) to share reliable service quality information among clients. The RM is integrated with a UDDI service directory and employs a publish/subscribe mechanism to disseminate reputation information to clients.

RMs have emerged as efficient tools for service discovery and selection [3]. When electronic contracts cannot be enforced, users can learn to trust good providers by looking at their past behavior [4]. Maximilien and Singh [5] describe a conceptual model for reputation using which reputation information can be organized and shared and service selection can be facilitated and automated. Lie et al. [6] present a QoS-based selection model that takes into account the feedback from users as well as other business related criteria. Both [7] and [8] propose concrete frameworks for service selection based on the reputation of the service provider.

Several works (see [9] for a detailed comparison of the approaches) have investigated monitoring of service compositions. However, to the best of our knowledge, this is the first attempt to use the result of observations deriving from

monitoring to build service reputations and make use of the latter to dynamically maintain service compositions.

2 Architecture

In this section we focus on the interaction between clients and services, on the collection of data about the behavior of services, and on the dissemination of information on service reputation from the registry to the clients. The architecture illustrated in Fig. 1 describes a client workflow which monitors the behavior of the invoked services and communicates the results of monitoring to the registry. The registry comprises the following components:

- *Reputation Manager (RM)*: its task is to collect feedback reports from the clients, to aggregate them, and to compute an estimate of the reputation of a service.
- *Subscription Manager*: this component handles dissemination of the information provided by the RM. We choose to design the communication infrastructure of our architecture using a publish/subscribe mechanism. Services may subscribe to two different kinds of events:
  - Notification by the RM when the reputation of a given service falls under a certain threshold;
  - Notification that a better service has become available, having either the same interface (exact-match) or a compatible interface (plugin-match) w.r.t. a given service.
- *Extended Service Directory*: with respect to its standard counterpart, this directory extends the registry by including information on the current estimated reputation of each registered service, as conveyed by the RM. Furthermore, the directory service is in charge to notify the *Subscription Manager* about the registration of new services.[5] We have explored techniques for efficient matchmaking in service directories in prior work [10].

[5] Note that for a newly registered service, the RM will not publish an associated reputation value before sufficient client feedback has been collected.

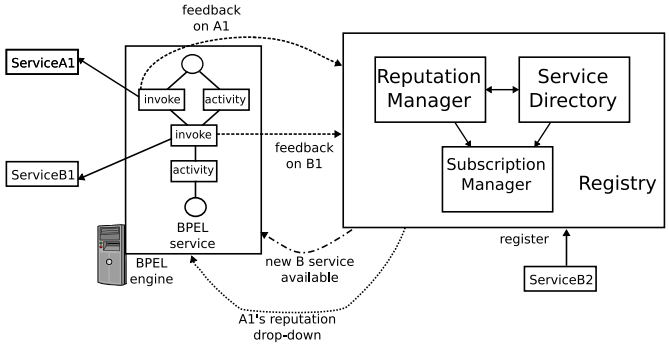

Fig. 1. System architecture

Figure 1 illustrates a BPEL service, sketched in the figure as a workflow containing two `invoke` activities, each one interacting with an external service, ServiceA1 (assumed to implement interface A) and ServiceB1 (assumed to implement interface B). The architecture also shows three kinds of message exchanged between the components:

- After each invocation of an external service, the BPEL service sends a feedback message back to the RM, which collects feedback from all the clients of a certain service. This message is labelled "feedback on `service-name`" and it is drawn in the figure by using a dashed line.
- Whenever the RM computes a new value of the reputation of a service, the Subscription Manager notifies all the subscribed clients if the reputation of the service dropped below the threshold set by each client. In our example, we assume that the BPEL service has subscribed to the drop-down of the reputation of the two used services and we show the case of A1's reputation drop-down. This message is labelled "`service-name` reputation drop-down" and it is depicted in the figure by using a dotted line.
- A third type of message, labelled "new `interface-name` service available", notifies all interested clients that a new service implementing a certain interface and with a better reputation became available. In the figure, a dash/dotted line depicts a message that notifies the client that a service implementing the interface B (in the example, ServiceB2), and having a better reputation, has been published in the directory.

The frequency of both monitoring and feedback reporting to the RM can be selected and tuned by the client. For simplicity, the architecture illustrated in Fig. 1 ignores how the workflow can dynamically adapt to the changes in service reputation through dynamic binding.

3 Monitoring

Some of the authors have previously explored the issue of monitoring web service compositions [9]. Under the assumption that the local workflow is correct, the *hot spots* where to place monitoring probes correspond to `receive`, `invoke`, and `pick` activities, i.e., to activities which represent interactions with external services.

Our specification language for monitoring, called Timed WSCoL, supports both functional and non-functional properties. In this particular context, we require each property to refer to only one service. This constraint guarantees that a violation of the property can be immediately mapped to a violation/failure of the service involved in the property.

Our monitoring infrastructure is based on an open-source BPEL engine, ActiveBPEL[6]. We have extended the engine using an aspect-oriented programming (AOP) approach [11], by implementing all the monitoring logic using AspectJ.

[6] http://www.activebpel.org

The architecture of the monitoring infrastructure includes (1) a data collector and aggregator, which gathers sequences of timestamped messages from the interactions with external services, and (2) a Timed WSCoL analyzer, which is actually in charge of checking the validity of a property. The output of this analyzer is binary, stating if the property has been violated or not. This output is then sent to the RM, together with the identifier of the service being "evaluated".

AOP is also used to instrument the engine to perform subscription to messages delivered by the Subscription Manager, each time an instance of a BPEL process is started. For each service x the BPEL service interacts with, it can make two subscriptions: (1) notification upon drop-down of the reputation of service x below a certain threshold τ_x, and (2) notification of the availability of new services with an interface equal or compatible to the one of service x.

A third instrumentation is required to make the BPEL workflow respond to reputation notifications. We achieve this by modifying how the engine behaves when a new BPEL process is deployed into it. The basic requirement is that the BPEL process should be able to bind dynamically a *partner link* to a new service, either because of a misbehavior of the service it is currently bound to or because a new service with a better reputation became available. Dynamic binding is achieved by updating the end-point reference of a partner link. Our AOP-based instrumentation of the engine modifies the BPEL process by inserting an event handler in the global scope (or by modifying the handler, if it already exists) for the two kinds of message that the Subscription Manager can send. Message "new service available" triggers the handler to update the end-point reference of the partner link representing the service to be replaced, whereas message "reputation drop-down" triggers a query to the registry to retrieve a substitute service with better reputation.

4 The Reputation Manager

The Reputation Manager is an important component of our framework, that collects feedback from the clients, and output quality measurements for the web services. Clients are required to submit a binary report: positive if the service met the quality constraints set in the SLA, negative otherwise. The reports submitted by the clients also contain the timestamp of the interaction with the service.

We model the behavior of web services by a Hidden Markov Model with two states: the *good* state describing the normal functioning mode when client requests are successfully satisfied with the unknown probability p_G, and the *bad* state describing a failure mode where the quality of the service is very low. The probability of transition from the good state into the bad state is assumed fixed and known, characterizing the different hazards the service is subject to.

Given the sequence of N binary feedback reports, $(r_i)_{i=1\ldots N}$, about the same web service, the RM can (a) estimate the parameter p_G of the web service, and (b) output the probability $Pr[B|(r_i)]$ that the web service is in the bad state. p_G is computed by likelihood maximization, while the probability that the service

is in the bad state can be computed using standard HMM tools like the Viterbi algorithm [12]. As the quality can change in time, the RM only uses the most recent N feedback reports. The estimates published by the RM can be used by future clients to optimize their workflows, or to dynamically replace defective services.

An important requirement for the reputation manager is to ensure honest feedback. Since clients may tamper with the default monitoring code in order to manipulate reputation information, special incentives must guarantee that lying, even if technically feasible, is economically uninteresting. The RM pays submitted reports an amount that depends on the feedback provided by other clients about the same web service. The payments can be designed such that truthful reporting maximizes the expected revenue (due to feedback payments) of a client, and honesty thus becomes an equilibrium of the mechanism. The budget for these payments can be raised by the RM from fixed participation fees that service providers and/or clients have to pay.

For example, a report is being paid only if it has the same value as another randomly chosen report. Intuitively, this simple mechanism encourages honest reporting because the private experience of a client changes her belief regarding the reputation of the service, and consequently, her expectation for the value of the report used to compute her payment. If the experience is positive, the client expects with slightly higher probability to be rewarded for a matching positive report. Likewise, if the experience is negative, the client expects with slightly higher probability to be rewarded for a matching negative report. This asymmetry in beliefs can be used to scale the payments for matching positive or negative reports so that honesty becomes optimal. The formal details and algorithms for computing these reward mechanisms are given in [13]. In the same time, measures can be taken to discourage collusion.

5 Conclusions

In this paper we have presented an architecture supporting automated, dynamic, pro-active, and transparent maintenance and improvement of composite services. Our architecture leverages monitoring techniques in order to generate feedback on the quality of service (from both a functional and a non-functional point of view) perceived by clients. This feedback is collected and aggregated by a reputation manager which computes services reputation; information on the reputation is then transmitted to clients that can pro-actively maintain and improve their composite services. Our reputation manager provides economical incentives for honest sharing of feedback.

We are currently focusing on optimizing our registry for queries involving functional properties (to support selecting plugin matches) and non-functional properties (to support ranking according to a user-defined utility function involving QoS parameters, services cost, and service reputation). Future work includes 1) verification of the approach, in terms of measurements of improvement by deployment and simulation; 2) the development of a new model that considers

more precise reports on QoS observations (e.g., a response time violated within a 10% bound may concur in a minor way to a decrease of the service reputation); 3) investigation on behavioral reflection mechanisms for workflow languages so as to better support dynamic re-binding within BPEL processes.

Acknowledgements. Part of this work has been supported by the EU project "PLASTIC" (contract number IST 026995), and the EU project "Knowledge Web" (FP6-507482).

References

1. Andrews, T., Curbera, F., Dholakia, H., Goland, Y., Klein, J., Leymann, F., Liu, K., Roller, D., Smith, D., Thatte, S., Trickovic, I., Weerawarana, S.: Business Process Execution Language for Web Services, Version 1.1 (2003)
2. Baresi, L., Di Nitto, E., Ghezzi, C.: Towards Open-World Software. IEEE Computer **39** (2006) 36–43
3. Singh, M.P., Huhns, M.N.: Service-Oriented Computing. Wiley (2005)
4. Zacharia, G., Maes, P.: Trust management through reputation mechanisms. Applied Artificial Intelligence (14) (2000) 881–907
5. Maximilien, E.M., Singh, M.P.: Conceptual model of web service reputation. SIGMOD Rec. **31**(4) (2002) 36–41
6. Liu, Y., Ngu, A.H., Zeng, L.Z.: Qos computation and policing in dynamic web service selection. In: WWW Alt. '04: Proceedings of the 13th international World Wide Web conference on Alternate track papers & posters, New York, NY, USA, ACM Press (2004) 66–73
7. Maximilien, E.M., Singh, M.P.: Toward autonomic web services trust and selection. In: ICSOC '04: Proceedings of the 2nd international conference on Service oriented computing, New York, NY, USA, ACM Press (2004) 212–221
8. Alunkal, B., Veljkovic, I., Laszewski, G., Amin, K.: Reputation-Based Grid Resource Selection. In: Proceedings of AGridM. (2003)
9. Baresi, L., Bianculli, D., Ghezzi, C., Guinea, S., Spoletini, P.: A timed extension of WSCoL. In: Proceedings of the IEEE International Conference on Web Services (ICWS 2007), IEEE Computer Society Press (2007) 663–670
10. Constantinescu, I., Binder, W., Faltings, B.: Flexible and efficient matchmaking and ranking in service directories. In: Proceedings of the IEEE International Conference on Web Services (ICWS 2005), IEEE Computer Society Press (2005) 5–12
11. Kiczales, G., Lamping, J., Mendhekar, A., Maeda, C., Lopes, C.V., Loingtier, J., Irwin, J.: Aspect-oriented programming. In: ECOOP'97 - Object-Oriented Programming, 11th European Conference, Proceedings. Volume 1241 of Lecture Notes in Computer Science., Springer (1997) 220–242
12. Forney, G.: The Viterbi algorithm. Proceedings IEEE **61** (1973) 268–278
13. Jurca, R., Faltings, B., Binder, W.: Reliable QoS monitoring based on client feedback. In: WWW '07: Proceedings of the 16th international conference on World Wide Web, New York, NY, USA, ACM Press (2007) 1003–1012