Towards Declarative Monitoring of Declarative Service Compositions

Kerry Taylor, Paul Brebner, Michael Kearney
CSIRO ICT Centre
Canberra, Australia
{FirstName.LastName}@csiro.au

Dana Zhang
Department of Computer Science & Software Engineering
The University of Melbourne
Melbourne, Australia
zhangd@csse.unimelb.edu.au

Kelly Lam
The University of Sydney
Sydney, Australia
kellylam85@gmail.com

Vladimir Tosic
National ICT Australia Limited (NICTA)
and Department of Computer Science
The University of Western Ontario
London, Canada
vladat@computer.org

Abstract

One of the great promises of Web Service technology is re-use through composition of basic services into new virtual services. To facilitate easier development of such compositions, recent approaches to declarative service composition ask a composition designer to provide a declarative specification of a composition goal, instead of an imperative workflow of composed services. Middleware services use this goal and expressive descriptions of available services to automatically generate an imperative workflow. However, if an error occurs during this process the source of the problem becomes difficult to trace.

In this paper, we discuss how an integration of two existing middleware infrastructures, the CSIRO’s semantic composition architecture and the Web Service Offerings Infrastructure (WSOI), can be used for monitoring of functional correctness and quality of service (QoS) in declarative Web service compositions. The combination supplements the generated workflows with checking of functional correctness, measurement and calculation of QoS metrics, and reporting of discovered problems. In future, it will be extended with correlation and analysis of monitored information to determine probable causes of problems and appropriate corrective actions. This will provide improved transparency and quality control that will be needed in practice.

1 Introduction

Many approaches for composing Web services are being developed. The most common in practice rely on traditional programmatic techniques or visual workflow design to specify a composition as a controlled chaining of grounded WSDL-defined services. These approaches lack the ability to scale with the potential availability of large numbers of Web services, as they rely on the composition designer to understand the function of each component, or each potential component of the composition. Recent declarative Web service composition approaches, such as [1, 8, 11], attempt to compose services based on expressive service descriptions coupled with declarative specifications of composition goals. These approaches rely on intelligent search or planning middleware to map a goal to a composition of grounded services that, when executed, achieves this goal. The composition designer does not need to understand the function of the components because the middleware does. One of the putative benefits of such an approach is that the task of solving problems by reuse of pre-
existing services can be given to problem domain experts rather than software engineers. If the composition seems to work for the composition designer, she/he may then publish the composition as a reusable virtual service resource for other users.

The intelligence in the middleware means that the details of what is occurring during translation and execution are hidden from the composition designer and composition users. The advantage is that users are shielded from potentially overwhelming details that are not essential to them. However, a major disadvantage is the lack of transparency for both the composition designer and subsequent users of the composition if there is a semantic error (e.g. a result message is wrong), a functional error (e.g., a fault message is returned or the composition goal unexpectedly fails) or a quality of service (QoS) problem (e.g., in performance). If the composition designer or the end user does not understand the purpose of a component in the composition, it is very difficult for them to understand or to rectify such problems, although possible causes of errors are numerous. During the design of a composition, there may be errors or misunderstandings in the machine-readable service descriptions, in the goal statement, or in the services themselves. The intermediary planning tools or registry tools may fail to terminate correctly even before a grounded orchestration plan is produced. During the execution of a grounded plan, basic services may fail to complete correctly, may be out of contact, or may not authorize necessary access. As a result, the observable effect of the execution of a composition may seem to be incorrect or incomplete. Similarly, there can be many causes of performance problems, such as overload of some of the composed services or network congestion between the services. The end result here is that the composition does not meet some QoS criteria expected of it, e.g., it could be too expensive or too slow for user needs. If other users use the virtual service recursively, as a component of another composition, and with an even weaker understanding of the underlying composition, then all the problems encountered by the first composition designer may propagate.

If declarative Web service composition approaches are to be applied in practice, then these issues must be addressed. In particular, both functional and QoS aspects of compositions must be monitored during run-time to collect and report information about the execution, detect when functional errors and performance problems occur, determine probable causes of problems and appropriate corrective actions, and (in an ideal system) predict and prevent occurrence of similar problems in the future. Unfortunately, these aspects are not addressed by the existing middleware for declarative Web service composition, such as CSIRO’s semantic composition framework [1]. The existing Semantic Web descriptions of Web services QoS are designed to facilitate selection of Web services but are not sufficient for run-time QoS monitoring, because they lack details about when, by which party, and how the monitoring or calculation of QoS metrics should be performed [14]. More importantly, the declarative service composition infrastructures do not contain adequate run-time monitoring modules.

On the other hand, the topic of run-time monitoring of functional and QoS characteristics of traditional Web services (i.e., not supporting Semantic Web technologies) and their imperative compositions is certainly not new. Many relevant papers have been published over the last few years and there are several commercial products. A partial bibliography is available in [15]. The most important work on QoS monitoring of Web services is the WSLA (Web Service Level Agreement) framework from IBM [7]. The Web Service Offerings Infrastructure (WSOI) [14, 13] was developed for a similar purpose, but it also provides monitoring of functional characteristics. While these and similar Web service QoS middleware systems are relatively powerful for QoS monitoring of individual Web services, we find that they do not completely address QoS monitoring of declarative Web service compositions. A particular challenge is correlation of information monitored at different places in a composition which is not known beforehand (but is generated from a declarative goal). For similar reasons, the recent works on monitoring of functional characteristics of imperatively-defined Web service compositions, such as [3] are also not enough. Existing works on QoS composition [2, 6, 16] have not created a complete solution covering all aspects of the theoretical model, QoS description language, composition algorithm and infrastructure.

To provide monitoring of functional correctness and QoS in declarative Web service compositions, we propose and discuss an integration of two existing Web service middleware architectures. To perform declarative composition, we use and extend the CSIRO’s semantic service composition framework. To perform run-time monitoring of functional and QoS characteristics, we use and extend the WSOI middleware. These two tools have certain technical advantages over other tools in their areas [1, 13], but in particular the declarative approach of each is mutually complementary. Both have prototype implementations that have been demonstrated on practical case studies.

The paper is structured as follows. We first describe CSIROs semantic composition framework as an exemplar declarative service composition method and then the WSOI as exemplar monitoring middleware. We then consider methods for diagnosing and reporting faults in the functional correctness of a composed service, by integrating the WSOI infrastructure with the composition middleware, and by appealing to declarative debugging techniques. We then consider monitoring and analysis of quality of service within the composition framework, again using the WSOI
middleware. We conclude with a summary of the paper’s contribution.

2 Declarative service composition

To enable declarative service composition, we can identify three representations of a composition over which tools may operate. The first is the most abstract descriptive representation of web service resources and the composition designer’s goal; the second may be a procedural representation which is a machine-executable representation containing control flow constructs to orchestrate grounded Web services; and the third may be an execution trace which comprises the selection and timestamps of control arcs traversed and the values of run-time dependent control variables.

CSIRO’s semantic service composition toolkit [1] provides an example that follows this general model. Figure 1 depicts the architecture as the processing chain moves downward from a descriptive representation during composition and deployment of a virtual service, to a procedural representation, a workflow after processing by the composition compiler, and to an execution trace as the underlying services are orchestrated.

In the first phase, the composition designer aims to define a goal for service composition, through interaction with a formal domain ontology. Mapping rules in a first order language are used to connect grounded services to the domain ontology [5]. The result of the first phase is a highly abstract rule based representation of a service composition. Services are selected down to their functional signatures, but a great deal of non-determinism remains. At this point the service may be deployed as a virtual service, automatically described in WSDL and annotated in OWL (Web Ontology Language), responsive to SOAP messages, and available for use by others.

At invocation time the service is invoked together with its run-time data parameters. At this time a workflow is generated by the composition compiler. Our workflow language is based on the graph language of service nodes and synchronous dependency arcs expressed in GraphML [4], but it may be simply translated to WSBPEL. Figure 2 shows a visualization of an actual workflow. The workflow is depicted with service invocation nodes annotated by SOAP RPC style control messages.

In the execution phase the workflow is interpreted by a workflow engine that traverses the arcs and invokes services asynchronously when their inputs are available. The workflow relies on a controlled amount of peer-to-peer interaction among services without referencing back to the workflow engine: data is transferred directly between some services according to control messages orchestrated by the workflow engine. These peer-to-peer messages are represented in Figure 1 where the numeric annotations refer to the message ordering and parallel invocations use the same number.
A concrete example is the development of a procurement service based on purchasing writing instruments from independent stationery suppliers. The composition designer specifies the goal - “Find the price for all chisel tip permanent markers” - by selecting and linking concepts (e.g. “chisel tip”, “marker”, “permanent”) from the domain ontology. Figure 3 illustrates part of such an ontology as displayed by the ontology editor Protege. Once complete the composition is deployed as a virtual service. Upon invocation, the composition compiler translates the goal into a workflow that orchestrates the necessary grounded services in order to satisfy the goal. The grounded services in this case include services to access the suppliers’ catalogues and helper services to map between alternative data encodings. Figure 2 shows the resulting workflow. Each node of the workflow represents a call to a Web service. The overlayed window shows the WSDL, operation and operation parameters for one node. The execution of the workflow provides the desired results.

In this paper, we focus principally on the execution phase. We are interested in how middleware can measure and report run-time information about functional errors or QoS problems that occur in the automatically generated service composition (e.g., in the grounded services). This can be achieved by integrating the CSIRO’s semantic service composition middleware with WSOI.

3 WSOI and WSOL

In order to be able to perform monitoring for Web services in a declarative approach, it is first necessary to formally describe the monitoring needs. Needs may identify which QoS metrics are monitored (measured or calculated), which conditions (requirements and guarantees related to functionality, QoS, security) are checked, monetary implications for meeting or not meeting these guarantees, which party is responsible for the monitoring, checking and billing activities, and when and how frequently these activities are performed. Several languages for description of QoS-related aspects of Web services have appeared recently, but only a few are detailed and precise enough for the run-time monitoring of QoS. One of these languages is the Web Service Offerings Language (WSOL) [14]. It is compatible with WSDL and enables additional formal specification of service offerings (SO), which are classes of service provided by a Web service. A WSOL service offering can contain various constraints (functional, QoS, access rights), management statements (prices, monetary penalties) and reusability constructs (extension, inclusion and template instantiation). Compared to the other languages in this domain [7], WSOL has several advantages [14]. One of the advantages is that it enables specification not only of QoS characteristics, but also functional constraints such as preconditions, post-conditions, and invariants.

The Web Service Offerings Infrastructure (WSOI) [13] is a Web service management infrastructure that enables monitoring of Web services described in WSOL and run-time adaptation of service compositions using manipulation of service offerings. It is an extension of Apache Axis, an open-source SOAP engine. Axis has an extensible architecture based on SOAP message processing components called handlers. Among main building blocks of WSOI are new WSOI-specific handlers, such as, QoS-measurement (or calculation) WSOI handlers, constraint-evaluation WSOI handlers and accounting WSOI handlers. These handlers can be written for different QoS metrics and constraint checks and then used when needed. Currently, the WSOI-specific handlers have to be coded manually, but the Web Service Offerings Compiler (WSOC), now in the final stages of implementation, will automatically generate
WSOI-specific handlers from WSOL files. The monitored values can be piggybacked into headers of regular SOAP messages or exchanged through special push and pull operations provided by WSOI. The WSOI management information model stores run-time values from monitoring activities. The dynamic adaptation algorithms and protocols address selection, switching, deactivation, reactivation, deletion and creation of service offerings. WSOI can be used at the Web service provider side, requester side, and one or more independent parties that act as SOAP intermediaries or probes. Further details about the architecture of the WSOI middleware and its performance in experimental case studies is available in [13, 14] and other publications by the same authors.

In our procurement service case, a simple application of WSOI would monitor the response time of every stationary supplier participating in the composition. WSOI would be used at the requester side to retain independence from suppliers, and the monitoring information would be used to provide an estimated response time for the composed service, or to avoid querying suppliers that fail to respond quickly enough. If the requester has a contractual arrangement of this form with the suppliers, then monitoring through an independent intermediary would be preferable.

Prior to the new work of this paper, WSOI addressed monitoring of individual Web services acting as providers, requesters, or management third parties (SOAP intermediaries or probes), but did not provide correlation of information monitored across a complex Web service composition. In particular, a WSOI-enabled service requester opens a session with a WSOI-enabled service provider to enable correlation of management information collected within the session. However, if this provider wants to act as a requester of another Web service, a new session must be opened. Earlier versions of WSOI did not reuse the same session ID or represent the relationship between the two open sessions. Our improved version of WSOI defines session IDs as globally unique and enables their reuse across multiple requester-provider pairs. Several additional minor technical improvements of WSOI were developed to facilitate its use for complex Web service compositions (including those that are automatically generated by the CSIRO’s semantic service composition middleware).

### 4 Monitoring functionality

If the execution of the workflow shown in Figure 2 encounters faults, we want to determine what sort of information should be provided (and how) to support visibility into the problem. An obvious approach is to use the workflow itself as a data structure to collect an execution trace. As the workflow engine executes each node in the workflow, additional monitored run-time information is added to the node. At the coarsest level, this simply marks nodes as successfully executed or not. Finer-grained information includes values of run-time dependent control variables and information whether functional constraints were satisfied. If QoS is monitored, then information from services about their claimed QoS guarantees and actual monitored values of QoS metrics and satisfaction of QoS constraints is added. Details provided by the called service are added to the node. The execution trace is reported dynamically or at the end of the execution. A declarative Web service composition tool could display the trace in addition to its workflow and enable navigation to display collected information for each node. The visualisation tool of Figure 2 could display the collected information through integration with WSOI. An end user of a composition may not have the skill or motivation to use the execution trace to understand runtime problems. They might not have access to some of this information as data about which grounded services were composed might be confidential. Therefore the execution trace should be saved as a file, providing a self contained description that an expert or an automated analysis tool can examine to locate the cause of execution faults.

The scenario in Figure 4 shows how WSOI can be used in conventional Web service compositions. A client application as a requester invokes a composite service (Level 1). The first-level composite service in turn invokes other services (Level 2) including a grounded service (leaf nodes, D) and other composite services which invoke further ser-
services recursively (Level 3 and below). A service can at the same time function as a requester in one interaction and a provider in another. In every interaction, a WSOL service offering is chosen and monitored. Under the current architecture of WSOI, a provider Web service uses the same WSOL instance for all its requester clients. However, since it is intended for use at one participant in one role (requester, provider, or management third party), if a Web service acts both as a requester and a provider, it has to have at least two WSOI instances - one on each side. The recent improvements to WSOI mentioned previously, enable correlation of information stored in these two WSOI instances, but these instances are quite autonomous. If a Web service is requester to multiple providers (as is the case with $B$ in Figure 3), it currently must also have multiple requester WSOI instances - one for each provider. A goal for a future version of WSOI is to enable one WSOI instance to be used for all interactions of a Web service.

Now, we return to Figure 1 and discuss the integration of WSOI into the CSIRO's middleware for declarative service composition. During the composition description and deployment, WSOL descriptions should be added. If a grounded Web service does not have pre-existing WSOL descriptions and QoS monitoring is desired, some default WSOL descriptions enabling monitoring of common QoS metrics (e.g., response time and availability) can be used. During the workflow creation, the composition compiler middleware would add calls to WSOI instances. During the execution phase, information about functional correctness and QoS characteristics would be collected and processed by the WSOI middleware. Communication between the workflow engine and the grounded services in the composition would go through requester-side WSOI instances at the workflow engine. This gives monitoring information from the perspective of the workflow engine, that is for the context of the composed service execution. For the monitoring information from the perspective of a grounded service, its provider-side (server) WSOI instance can be used if it pre-exists or calls to management third parties can be configured by the workflow engine. Using the WSOI mechanisms for exchange of monitored information, all this information can be gathered at the workflow engine for correlation, further processing and analysis. At this stage, we have provided means for data collection and, to some extent, correlation. This integration method for the declarative composition process from description through to execution trace, relies on the capability of the anticipated WSOC, so that WSOI instances can be configured declaratively and dynamically rather than by pre-written code.

Other problems in service composition may be caused by the mismatch between service implementations and published service descriptions. In our procurement scenario, this kind of error would result in, for example, an execution returning the prices of ball-points as well as the expected chisel-tip permanent markers. The services execute without an apparent fault, but the answers are incorrect or incomplete. This kind of semantic error cannot be tackled without reference to the mechanisms of the approach used to transform a goal statement to an executable service composition. In a composition environment layered with abstractions this is particularly difficult. There may be semantic errors in the domain ontology with respect to which the composers goal was framed. There may be semantic errors in the mappings that relate services to the domain ontology, missing or faulty data, or incorrect logic in the underlying services themselves.

One possible direction for tackling this problem is known as declarative debugging [9, 12], arising from logic programming research. This method has the advantage of being able to locate an error of missing answers, wrong answers, or wrong rules occurring anywhere in a path from top-level goal to final answers. This method remains true to the declarative model of a composition request and is independent of execution strategies. However, this basic mechanism would have to be extended. Firstly, it would have to take account of the different logical structures present in description logic-based languages such as OWL, perhaps by integration with description logic debugging techniques [10]. Secondly, it would have to be extended to deal with the expressiveness and presentation of our mapping language or other rule-based languages. Thirdly, it would have to be adapted from an interactive top-down execution environment to deal with a set-at-a-time multiphase execution environment. Finally, it relies on an oracle to answer questions about the intended interpretation of logical statements throughout the system. This amounts to saying that someone must be able to provide an opinion on whether the output of every relevant service invocation was correct and complete. An aid to the oracle would be WSOI monitoring of WSOL functional constraints and, possibly, also of analysis of information quality, data currency, and historical data about QoS performance. This may assist fault detection and location. We leave the development of middleware support for this approach to future work, but note that the method will require inspection of the run-time message content coordinated by the workflows, not just the control variables. An implementation would also require the source rules arising from the description phase to be linked to the execution trace by the composition compiler.

5 Monitoring quality of service

The described integration of WSOI into the CSIRO’s declarative service composition middleware can be also used for monitoring QoS. However, QoS monitoring is more complicated, due to the diversity of QoS metrics.
our project, QoS-related information is described in WSOL and all QoS monitoring is performed by WSOI. A difference can be made between external and internal QoS monitoring of a service composition. The former is monitoring QoS between a composite service (B in Figure 3) and its requester (A in Figure 3), such as between the writing instrument procurement service and a special purpose client. The latter is monitoring QoS between a composite service (B in Figure 3) acting as a requester of composed grounded services (C or D in Figure 3) or between two composed grounded services (C and D in Figure 3). In our scenario, this occurs between the writing instrument procurement service and each of the underlying suppliers’ catalogue services, for example.

To perform external monitoring, it is necessary to define a WSOL service offering for the whole composed service. We assume that this can be done by the composition designer, based on the expectations from the composed service and reasonable assumptions of what a composition could deliver. It has been proven by the software performance analysis community that the task of estimating performance of a composition of software components from performance of individual components is extremely difficult. The existing approaches require detailed knowledge and modelling of the composition and provide limited results. Performance models for different QoS metrics (e.g., response time vs. availability) can be quite different. The crucial complication with declarative Web service composition is that the required details of the composition are not known beforehand. Ideally, after grounded services are selected for composition, the Composition compiler would build and solve performance models for all required QoS metrics to estimate whether the resulting composition is viable from the QoS viewpoint. Due to the complexities, we have not yet built support for this into our middleware. In the future, we might build in support for some elementary performance models, which in spite of limitation could help prevent executing some inappropriate compositions. On the other hand, our middleware enables run-time monitoring. In principle, only provider-side WSOI at the composed service is necessary, while requester-side WSOI at the client application is optional (but highly desired). The Workflow engine calls this provider-side WSOI before the first and after the last grounded service. If WSOI discovers violation of a WSOL service offering, manipulation of service offerings [13] can be performed. Lower-level violations, such as timeouts, can be handled in different ways, such as throwing SOAP faults handled by WSOI as functional errors or invoking an ‘assassin’ thread terminating the composition thread.

To perform internal monitoring, it is necessary to have WSOL service offerings for each of the grounded services in the composition. If a grounded service does not provide them, some default service offerings enabling monitoring of common QoS metrics can be defined and used. For example, these defaults could be defined to perform monitoring of QoS metrics presented in the service offering for the composite service, but without defining QoS constraints or prices and penalties. The task of determining requirements on performance of individual components based on requirements on performance of their composition is inverse to the task discussed in the previous paragraph, so it is also quite difficult and currently without a general solution. Therefore, our middleware currently does not enable selection of grounded Web services based on their WSOL service offerings or historical records of satisfaction of their QoS guarantees, but some simple support might be built into a future version. The support for run-time monitoring of QoS is analogous to the previously described support for monitoring functionality - the Composition compiler adds calls to WSOI instances and the Workflow engine gathers and correlates data. As mentioned above, one of the main challenges in our work has been correlation of monitoring information. Note that if desired, the QoS information monitored internally can be made available through WSOI mechanisms to the client application of the composite service.

Typically, there are trade-offs between the value of information gained from a monitoring infrastructure and the costs of the infrastructure itself. Monitoring requires time to collect and process data. Our approach requires specification of WSOL service offerings and hosting of the WSOI infrastructure. This all introduces extra cost and raises the potential for the monitoring infrastructure itself to introduce functional errors and impact performance. A Web service has to meet its QoS guarantees in WSOL service offerings. The WSOI mechanisms for actual control to meet these guarantees are quite limited and further research in this direction is needed. However, QoS will be a major differentiator between Web services providing similar functionality for many business applications. The technical solutions built into WSOI are based on solid concepts, with proven practical benefits. For example, the interceptor mechanism (supported in WSOI both internally though WSOL-specific SOAP message handlers and externally through management third parties) is increasingly used to address extra-functional cross-cutting concerns in distributed systems. It does not require changes to service implementations and provides flexibility in the design of the QoS monitoring infrastructure and reporting mechanisms. Taking these and other arguments [13, 14], the benefits of supporting WSOI outweigh the costs in many situations.

6 Conclusions and future work

The task of run-time monitoring of Web service compositions to discover and locate functional errors and QoS...
problems is very complex. In declarative service compositions, where the workflow is not known beforehand, this task is even more difficult. On the other hand, the information provided by such monitoring is crucial for many business uses of Web services, often as a differentiator between Web services with similar functionality and measure of their trustworthiness. The monitored information could be used by the composition middleware to dynamically and autonomously adapt to overcome detected problems and to optimize future composition activities. Therefore, our integration of CSIRO’s semantic composition framework with WSOI aims to provide required monitoring capabilities both at boundaries of and within composite services. It enables addition of necessary checking of functional correctness, monitoring of QoS metrics and constraints, and reporting of discovered problems to workflows automatically generated by the composition framework. Prototype implementations of both integrated software tools have been extended to support this. Our discussions and solutions for monitoring both functionality and QoS go beyond previous works on monitoring individual Web services and recent papers on monitoring some aspects of procedural Web service compositions.

Due to the complexity of the problem area, our current solutions have limitations, which are the objects of our ongoing and future work. First, the current middleware supports data gathering and limited data correlation, but more advanced data correlation and analysis to determine probable problem causes and appropriate corrective actions is needed for many practical applications. Second, further work is needed on middleware support for using WSOL service offerings and historical monitored information to support the choice of Web services at the time of declarative composition. Third (and probably most important), advanced support for dynamic control of Web service compositions, particularly their adaptation to overcome detected problems and prevent future ones, is necessary to increase autonomy and flexibility of Web service compositions. All this will offer a higher level of transparency and quality control needed for practical acceptance of declarative Web service compositions.

References


