Advanced Fault Analysis in Web Service Composition

L. Ardissono, L. Console, A. Goy, G. Petrone, C. Picardi, M. Segnan
Dipartimento di Informatica
Università di Torino
Torino, Italy
authorFamilyName@di.unito.it

D. Theseider Dupré
Dipartimento di Informatica
Università del Piemonte Orientale
Alessandria, Italy
dtd@mfn.unipmn.it

ABSTRACT

Currently, fault management in Web Services orchestrating multiple suppliers relies on a local analysis, that does not span across individual services, thus limiting the effectiveness of recovery strategies. We propose to address this limitation by employing Model-Based Diagnosis to enhance fault analysis. In our approach, a Diagnostic Web Service is added to the set of Web Services providing the overall service, and acts as a supervisor of their execution, by identifying anomalies and explaining them in terms of faults to be repaired.

Categories and Subject Descriptors

D.2.11 [**Software Engineering**]: Software Architectures

General Terms

Languages, Standardization

Keywords

Web Service composition, fault management, diagnosis

1. INTRODUCTION

The emerging standards for Web Service composition, such as BPEL4WS [1], are key elements for the integration of heterogeneous software in open environments. However, they offer limited support to the development of robust services based on complex workflows. For instance, fault handlers are introduced to specify the actions to be taken when a service execution fails, but they are defined in *ad hoc* ways, similar to the exception handling techniques exploited in programming languages. Moreover, the handlers try to recover from the effects of a problem, but they do not attempt to identify its real causes and to address them before other service executions fail in a similar way.

We aim at extending Web Service composition with support for the specification, management, monitoring and recovery of complex workflows including the execution of internal activities and the invocation of external services. As a first step in this direction, we propose to integrate in a Web Service composition framework a diagnostic Web Service which identifies the causes of the problems occurring during the execution of a complex service and therefore supports a fine grained selection of the recovery strategy to be applied. The diagnostic Web Service relies on Model-Based Diagnosis [3] to identify the possible causes of problems.

2. A FRAMEWORK SUPPORTING ADVANCED FAULT MANAGEMENT

2.1 The Diagnoser Role

The occurrence of a failure in the execution of a Web Service triggers the generation of a fault to be locally handled, or of a fault message directed to the service consumer. The simplest failures, associated to local execution problems, are usually solved in a straightforward way; in contrast, subtler types of failures can only be recognized at later stages of the service execution, when a direct treatment of the problem is not possible. In order to enhance fault management in complex services, we propose to add a *diagnoser* Web Service that receives information about the execution of all the orchestrated services, as well as from the Web Service integrator, carrying out a global analysis of their activities. The diagnoser has the following responsibilities:

- During service execution, it monitors the activities carried out by the Web Services, logging the messages they exchange.
- When the diagnoser receives a fault message denoting the presence of a failure to be handled at the global level, it reasons about the problem to identify its causes. This can be done by exploiting logged messages, which help to track the propagation of data and errors between Web Services. The faults to be handled at the global level can be recognized by the diagnoser if the set of fault messages that can circulate is suitably partitioned in local and global ones. We call *alarm messages* the second type of messages.
- Finally, the diagnoser informs the Web Service integrator about the final diagnosis, so that a suitable recovery strategy can be applied.

The diagnoser (Diagnoser WS) interacts with the other Web Services via WSDL [4] messages. Specifically:

- The Diagnoser WS must offer a WSDL operation (`logMessage(String wsdlMsg)`) for the reception of the normal and fault messages to be logged. The Web Service exploits an internal "observer" component collecting the messages for later inspection. Moreover, it exploits a diagnostic engine supporting reasoning about faults.

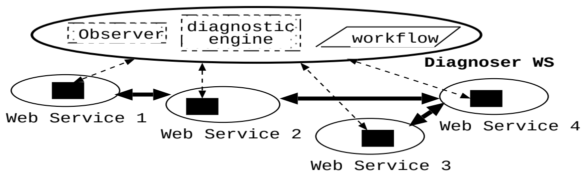

Figure 1: Architecture of a complex service exploiting an observer for advanced fault management

- The Web Service integrator and the individual Web Services must send copies of the outbound messages to the Diagnoser WS. To this purpose, each Web Service must be equipped with a "logging service" which intercepts the outbound WSDL messages and sends a copy of each message to the Diagnoser WS through the "logMessage" port.

Figure 1 depicts the architecture of a complex service exploiting a Diagnoser WS for advanced fault management; "logging services" are depicted as black rectangles; ordinary messages are represented as thick plain arrows and log messages are shown as dashed arrows.

2.2 Model-Based Diagnosis of Web Services

Model-Based Reasoning and, in particular, Model-Based Diagnosis (MBD), have been proposed in the Artificial Intelligence community for reasoning on possibly faulty physical systems, but they have also been applied in other domains, such as software diagnosis. Most MBD approaches rely on a component-oriented model of the system to be diagnosed. Component-based MBD assumes that:

- The system is modeled as a set of components.
- The behavior of each component is modeled as a relation on component variables. The model is provided for the correct and/or faulty behavior of the component; in some domains, the behavior under alternative *fault modes* is provided.
- Component variables include *interface variables* used to define interconnections in the system by equating interface variables of different components; e.g., an output variable of a component with an input variable of another component.

The system model is able to predict, or at least constrain, the effect of the incorrect behavior of a component also on variables that are not directly related to the component.

Diagnostic reasoning should identify diagnoses, as **assignments of behavior modes** to components, for a given set of observations (values for observable variables). A **diagnostic engine** explores the space of behavior mode assignments, finding those that explain an initial set of observations, and performs discrimination among alternative candidates, possibly suggesting additional pieces of information to be acquired to this purpose.

There are several formalizations of MBD; see [3]. Here, we use *consistency-based diagnosis*: a **diagnosis** is an **assignment of behavior modes** to components that is **consistent with observations**.

In order to apply the MBD paradigm to a complex Web Service, we base the inferences of the diagnoser on a model M of the service which is derived from the workflow specification as follows:

- The workflow is represented as a set of **activities** (which may correspond to sending a WSDL message to a WS) with **input** and **output** variables. Moreover, dependencies between input and output variables are specified to model the data flow, in order to distinguish whether an output variable is a *copy* of an input variable or it is *created* by an activity a, or it is *computed* by a depending on some of its input variables.
- A diagnostic model is derived, with a binary variable for each input/output variable v of activities, representing whether, in a given execution of the service, v has the expected value or not. The model then relates incorrectness of an output variable to errors in input values or faults in the activities.
- The fault messages denoting problems which would benefit of diagnostic reasoning are tagged as *alarms* and related to the model: a typical triggering alarm is a mismatch between service variables, and the model can relate the alarm to the (in)correctness of such variables. Moreover, *test conditions* on the parameters of (a subset of) the messages exchanged by the Web Services are defined in order to enable the diagnoser to acquire further evidence to confirm or disconfirm hypotheses. During diagnosis, the diagnoser will not go through the whole bunch of logged messages, but it will look at the outcome of test conditions to perform the diagnostic inferences.

The approach was successfully tested on the model of a Web-based catalog service, supporting customers in the purchase of books. The model was run in the SALVO [2] Model-Based Reasoning tool.

3. CONCLUSIONS

We proposed a framework supporting the exploitation of Model-Based Diagnosis to enhance fault analysis in complex Web Services exploiting multiple suppliers. In our approach, a Diagnoser Web Service is added to the set of composed Web Services and acts as a supervisor of the execution of the underlying workflow, by identifying anomalies in the execution of the composed Web Services, and by explaining such anomalies in terms of faults to be repaired.

The notification about (normal and faulty) messages sent to other Web Services can be added to the invoked Web Services without changing their internal structure. Thus, it represents a reasonable price to pay in order to enhance the robustness of services, at least in Enterprise Application Integration, where the pool of exploited services is well determined and fixed.

Recent developments of our work focus on the design of a framework where the diagnostic task is carried out in a distributed way within a service. When defining a complex service, we propose to add to each service S a local diagnoser which relates hypotheses about incorrect outputs of S to a misbehavior of S itself, or to incorrect inputs from other services. A global diagnostic service is then associated with the complex service. It coordinates the local diagnosers, exchanging messages with them and, without relying on any information about the internal structure of the sub-services, it can in turn compute diagnoses at the level of the global service.

4. REFERENCES

[1] T. Andrews, et al. Business Process Execution Language for Web Services version 1.1. http://www-106.ibm.com/developerworks/webservices/library/ws-bpel/, 2003.

[2] R. Bray, A. Buffo, F. Cascio, L. Console, C. Picardi, M. Segnan, and D. Theseider Dupré. SALVO: Model-based systems - applications in automotive industry. *Intelligenza Artificiale*, 1(3):13–20, 2004.

[3] L. Console and O. Dressler. Model-based diagnosis in the real world: lessons learned and challenges remaining. In *Proc. 16th IJCAI*, pages 1393–1400, Stockholm, 1999.

[4] W3C. Web Services Definition Language. http://www.w3.org/TR/wsdl, 2002.