Dependability in the Web Service Architecture

Ferda Tartanoglu[1], Valérie Issarny[2]
INRIA, UR Rocquencourt
Domaine de Voluceau - B.P. 105
78153 Le Chesnay France
[1]Galip-Ferda.Tartanoglu@inria.fr,
[2]Valerie.Issarny@inria.fr

Alexander Romanovsky
University of Newcastle upon Tyne
Department of Computing Science, NE1 7RU, UK
Alexander.Romanovsky@newcastle.ac.uk

Nicole Levy
Université de Versailles Saint-Quentin en Yvelines
45 avenue des Etats-Unis
78035 Versailles Cedex, France
Nicole.Levy@prism.uvsq.fr

ABSTRACT

In comparison with the state of the art in the field of Web Services architectures and their composition, we propose to exploit the concept of CA Actions to enable to dependable composition of Web Services. CA Actions introduce a mechanism for structuring fault tolerant concurrent systems through the generalization of the concepts of atomic actions and transactions, and are adapted to the composition of autonomous services.

1. INTRODUCTION

The Web service architecture targets the development of applications based on the XML standard [15], which eases the construction of distributed systems by enabling the dynamic integration of applications distributed over the Internet, independent of their underlying platforms. Currently, the main constituents of the Web service architecture are the following: (i) WSDL (Web Service Description Language) is a language based on XML that is proposed by the W3C for describing the interfaces of Web services [14]; (2) UDDI (Universal Description, Discovery and Lookup) is a specification of a registry for dynamically locating and advertising Web services [12]; (3) SOAP (Simple Object Access Protocol) defines a lightweight protocol for information exchange [13]. SOAP sets the rules of how to encode data in XML; it also includes conventions for partly pre-scribing the invocation semantics (either synchronous or asynchronous) as well as the SOAP mapping to HTTP.

There already exist platforms that are compliant with the Web service architecture, including .NET [6] and J2EE [10]. In addition, integration within CORBA is being addressed [9]. Even though the Web service architecture is quite recent and not fully mature, it is anticipated that it will play a prominent role in the development of next generation distributed systems mainly due to the strong support from industry and the huge effort in this area. However, there is clearly a number of research challenges in supporting the thorough development of distributed systems based on Web services. One such challenge relates to using Web services in developing business processes, which requires a support for composing Web services in a way that guarantees dependability of the resulting composed services. This calls for developing new architectural principles of building such composed systems, in general, and for studying specialized connectors "glueing" Web services, in particular, so that the resulting composition can deal with failures occurring at the level of the individual service components by allowing co-operative failure handling.

Solutions that are being investigated towards the above goal subdivide into (i) the definition of XML-based languages for the specification of Web services composition and (ii) revisiting classical transactional support so as to cope with the specifics of Web services (e.g., crossing administrative domains, Web latency), i.e., defining connectors offering transactional properties over the Internet. The two next sections respectively overview existing solutions to the two aforementioned points, and assess them with respect to Web service composition and its dependability. In particular, it is emphasized that while the transaction concept offers a powerful abstraction to deal with the occurrence of failures in closed systems, it imposes too strong constraints over component systems in open environment such as Web services. The main constraint relates to supporting backward error recovery that, firstly, requires isolating component systems for the duration of the embedded (nested) transaction in which they get involved and hence contradicts the intrinsic autonomy of Web services, and, secondly, relies on returning the service state back, which is not applicable in many real-life situations which involve documents, goods, money as well as humans (clients, operators, managers, etc.).

In the light of the above, the paper puts forward a solution based on forward error recovery, which enables dealing with dependability of composed Web services, and has no impact on the autonomy of the Web services, while exploiting their possible support for dependability (e.g., transaction support at the level of each service). Our solution, introduced in Section 4, lies in system structuring in terms of co-operative actions that have a well-defined behavior, both in the absence and in the presence of service failures. Finally, Section 5 discusses our current and future work aiming at enhancing the Web service architecture for the sake of dependability.

2. COMPOSING WEB SERVICES

Composing Web services relates to dealing with the assembly of autonomous components so as to deliver a new service out of the components' primitive services, given the corresponding published interfaces. In the current Web service architecture, interfaces are described in WSDL and published through UDDI. However, supporting composition requires further addressing: (i) the specification of the composition, (ii) ensuring that the services are composed in a way that guarantees the consistency of both the individual services and the overall composition. There are three main proposals in the area:

- WFSL (Web Services Flow Language) addresses the former issue. It enables describing the composition of Web services through two complementary models [4]: (i) a flow model that serves specifying a sequence of actions over services in a way similar to workflow schema, and (ii) a global model that further describes the interactions between service providers and requesters and hence details the realization of each action of the flow model.
- XLANG deals with the latter issue by enriching the description of Web services' interfaces with behavioral specification. It aims at allowing the formal specification of business process as stateful long-running interactions [11]. Business processes always involve more than one participant. Hence, the full description of a process must not only show the behavior of each participant but also the way these behaviors match to produce the overall process. The focus is on the publicly visible behavior in the form of exchanged messages. More precisely, the interface of a Web service is enriched with the specification of how to consistently use the Web service, stating the necessary sequence of interactions. This is quite similar to the work done in the area of Architecture Description Language [5], when concerned with the formal specification of port and role behavior for checking the consistency of the architecture.
- XL is a language targeting the specification of Web service composition. It is fully based on XML for the specification and composition of Web services [3] and is built upon concepts of imperative programming languages, the CSP process algebra and workflow management.

There are other efforts towards supporting the composition of Web services: similar to the aforementioned solutions, these proposals rely on a new language and supporting environment, which are still under definition. While there is not yet a consensus about how Web services composition should be supported, existing work allows us to identify two major trends: (i) composition based on workflow management, (ii) using transactions to enforce dependability. The former trend justifies from the concern of supporting business processes but also by the fact that the composition process applies to autonomous services belonging to distinct administrative domains. However, the needed extension to WSDL still requires investigation. For instance, the behavioral specification for individual services introduced by XLANG complements the composition specification introduced by WFSL for checking composition consistency and also to possibly automate the generation of interactions. The next section shows why, from our standpoint, transactions do not offer solutions to the dependable composition of Web services.

3. TRANSACTIONS FOR THE DEPENDABLE COMPOSITION OF WEB SERVICES

Transactions have been proven successful in enforcing dependability in closed distributed systems. The base transactional model that is the most used guarantees ACID (atomicity, consistency, isolation, durability) properties over computations. Enforcing ACID properties typically requires introducing protocols for: (i) locking resources (i.e., two-phase locking) that are accessed for the duration of the embedding transaction, and (ii) committing transactions (i.e., two or three phases validation protocols). However, such a model is not suited for making the composition of Web services transactional for at least two reasons:

- The management of transactions that are distributed over Web services requires cooperation among the transactional support of individual Web services –if any-, which may not be compliant with each other and may not be willing to do so given their intrinsic autonomy and the fact that they span different administrative domains.
- Locking accessed resources (i.e., the Web service itself in the most general case) until the termination of the embedding transaction is not applicable to Web services, still due to their autonomy, and also the fact that they potentially have a large number of concurrent clients that will not stand extensive delays.

Enhanced transactional models may be considered to alleviate the latter shortcoming. In particular, the split model where transactions may split into a number of concurrent sub-transactions that can commit independently allows reducing the latency due to locking. Typically, sub-transactions are matched to the transactions already supported by Web services (e.g., transactional booking offered by a service) and hence transactions over composed services do not alter the access latency as offered by the individual services. Enforcing the atomicity property over a transaction that has been split into a number of sub-transactions then requires using compensation over committed sub-transactions in the case of sub-transaction abortion. Using compensation comes along with the specification of compensating operations supported by Web services for all the operations they offer. Such an issue is in particular addressed by XLANG [11]. However, it should be further accounted that using compensation for aborting distributed transactions must extend to all the participating Web services (i.e., cascading compensation by analogy with cascading abort), which is not addressed by XLANG due to its focus on the behavioral specification of individual Web services for assisting their composition.

Developing transactional supports for dependable Web service composition is an active area of research that is still in its infancy. Ongoing work includes BTP (Business Transaction Protocol) [8], TIP (Transaction Internet Protocol) [2] and extension to the OMG/J2EE Activity Service [7]. However, proposed solutions do not cope with all the specifics of Web services. From our standpoint, a major source of difficulty lies in the use of backward error recovery in an open system such as the Internet, which is mainly oriented towards tolerating hardware faults but poorly suited to the deployment of cooperation-based mechanisms over autonomous component systems that often require cooperative application-level exception handling among component systems. An alternative then lies in relying on the existing support of Web services for managing internal concurrency control so as to guarantee keeping the consistency of services, while relying on forward error recovery for ensuring the dependability of service composition. The next section introduces such a solution, which builds upon the concept of Coordinated Atomic (CA) Actions [16].

4. USING CA ACTIONS FOR THE DEPENDABLE COMPOSITION OF WEB SERVICES

The CA Actions [16] are a structuring mechanism for developing dependable concurrent systems through the generalization of the concepts of atomic actions and transactions. Basically, atomic actions are used for controlling cooperative concurrency among a set of participating processes and for realizing coordinated forward error recovery using exception handling, and transactions are used for maintaining the coherency of shared external resources that are competitively accessed by concurrent actions (either CA Actions or not). Then, a CA Action realizes an atomic state transition where: (i) the initial state is defined by the initial state SP_i of the participants Pi and the states SR_j of the external resources R_j at the time they were accessed by the CA Action, (ii) the final state is defined by the state of the participants (SP_i‘) at the action's termination (either standard or exceptional) and the state of the accessed external resources (SR_j‘ in the case of either standard termination or exceptional termination without abortion, SR_j in the case of exceptional termination with abortion).

CA Action naturally fits the specification of Web service composition:

- Each participant specifies the interactions with each composed Web service, stating the role of the specific Web service in the composition. In particular, the participant specifies actions to be undertaken when the Web service signals an exception, which may be either handled locally to the participant or be propagated to the level of the embedding CA Action. The latter then leads to co-operative exception handling according to the exceptional specification of the CA Action.
- Each Web service is viewed an external resource. However, unlike the base CA Action model, interactions are not enforced to be transactional. The interactions adhere to the semantics of the Web service operations that are invoked. An interaction may then be transactional if the given operation that is called is. However, transactions do not span multiple interactions.
- The standard specification of the CA Action gives the expected behavior of the composed Web service in either the absence of failures or in the presence of failures that are locally handled (i.e., either system-level exceptions or programmed exceptions signaled by Web services operations that do not need to be cooperatively handled at the CA Action level).
- The exceptional specification of the CA Action states the behavior of the composed Web service under the occurrence of failure at one or more of the participants, that need cooperative exception handling. The resulting forward recovery may then realize a relaxed form of atomicity (i.e., even when individual operations of the Web service are transactional, its intermediate states may be accessed by external actions between such operations executed within a given action) when Web services offer both transactional and compensating operations (to be used in cooperative handling of exceptions).

To apply the general concept of CA actions in the context of composing Web services, we introduce the concept of WSCA (Web Service Composition Action). WSCAs differ from CA Actions in (i) relaxing the transactional requirements over external interactions (which are not suitable for wide-area open systems) and (ii) introducing composition of WSCAs where each participant may actually be a WSCA, which is abstracted as a single unit of computation from the standpoint of peer participants.

In order to illustrate the use of WSCAs for specifying the composition of Web services, we take the classical example of a travel service. We consider joint booking of accommodation and flights using respective hotel and airline Web services. Then, the composed Web service is specified using nested WSCA as follows. The outermost WSCA TravelAgent comprises the User and the Travel participants. The Travel participant is a nested WSCA that composes the Airline and the Hotel participants. A diagrammatic specification of the WSCAs is shown in Figure 1.

In TravelAgent, the User participant requests the Travel participant to book a return ticket and a hotel room for the duration of the given stay. Then, the two Travel WSCA participants respectively request the Hotel Web service for a hotel room and the Airline Web service for a return ticket, given the departure and return dates provided by the user. Each participant request is subdivided into reservation for the given period and subsequent booking if the reservation succeeds. In the case where either the reservation or the booking fails, the participant raises the unavailable exception that is cooperatively handled at the level of the Travel WSCA. If both participants signal the unavailable exception, then Travel signals the abort exception so that the exception gets handled by TravelAgent in a cooperation with the User (e.g., by choosing a alternative date). If only one participant raises the unavailable exception, cooperative exception handling includes an attempt by the other participant to find an alternative booking. If this retry fails, the booking that has succeeded is cancelled and the abort exception is signaled to the embedding TravelAgent WSCA for recovery with user intervention.

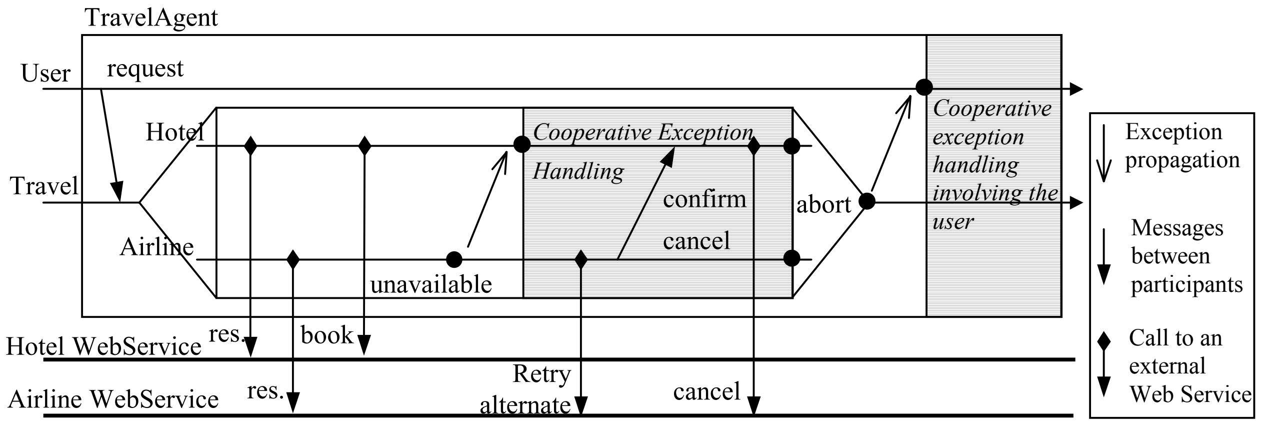

Figure 1. WSCA for composing Web Services

5. CONCLUSIONS

The Web service architecture is expected to play a major role in developing next generation distributed systems. However, the architecture needs to evolve to support all the requirements appertained to distributed systems. Addressing such requirements relates, in particular, in reusing solutions from the distributed system community. However, most solutions will not be reusable as is, mainly because of the openness of the Internet. Hence, making evolve the Web service architecture to support the thorough development of distributed systems raises a number of challenges.

This paper has addressed one of the issues raised in this context, which is the dependable composition of Web services, i.e., understanding how fault tolerance should be addressed in the Web service architecture. While dependability in closed distributed systems is conveniently addressed by transactions when concerned with both concurrency control and failure occurrences, it can hardly rely on such a mechanism in an open environment. Our solution to this concern lies in forward error recovery that enables accounting for the specific of Web services and that leads to structure Web services-based systems in terms of co-operative actions. In particular, we are able to address dependable service composition in a way that neither undermines the Web service's autonomy nor increases their individual access latency.

Further work is still needed towards offering a complete solution to dependability in the Web service architecture. Our next step is on the formal specification of WSCAs using the B formal notation [1] so as to precisely characterize the dependable behavior of WSCAs, and in particular the relaxed form of atomicity that is introduced. Our aim is to propose an architectural style for specifying architectures of systems based on WSCAs by defining associated connectors and components. We will then investigate the definition of an architecture description language and associated methods and tools for supporting the development of dependable systems based on the Web service architecture.

6. ACKNOWLEDGMENTS

This research is partially supported by the European IST DSoS project (IST-1999-11585) [1].

7. REFERENCES

[1] Abrial, J. R. *The B Book – Assigning Programs to Meanings*. Cambridge University Press. 1996.

[2] Evans, K. Transaction Internet Protocol: Facilitating Distributed Internet Applications. *Proceedings of the W3C Workshop on Web services*. 2001.

[3] Florescu, D., and Kossmann, D. An XML Programming Language for Web Services Specification and Composition. *Bulletin of the IEEE Computer Society Technical Committee on Data Engineering*. 2001.

[4] Leymann, F. Web Services Flow Language (WSFL 1.0). IBM Software Group. http://www-4.ibm.com/software/solutions/webservices/pdf/WSFL.pdf. 2001.

[5] Medvidovic, N. and Taylor, R. N. A Classification and Comparison Framework for Software Architecture Description Languages. *IEEE Transactions on Software Engineering*. 2000.

[6] Microsoft. .NET. http://msdn.microsoft.com/net/.

[7] Mikalsen, T., Rouvellou, I., and Tai, S. Reliability of Composed Web Services – From Object Transactions to Web Transactions. *Proceedings of the OOPSLA'01 Workshop on Object-Oriented Web Services*. 2001.

[1] http://www.newcastle.research.ec.org/dsos/

[8] Oasis Committee. Business Transaction Protocol. Draft Specification. January 2002. http://www.oasis-open.org/committees/business-transactions/

[9] OMG. Corba Web Services. OMG TC Document orbos/2001-06-07. http://www.omg.org. 2001.

[10] Sun Microsystems Inc. Java 2 Platform, Enterprise Edition (J2EE). http://java.sun.com/j2ee/

[11] Thatte, S. XLANG: Web Services for Business Process Design. Microsoft Corporation. http://www.gotdotnet.com/team/xml_wsspecs/xlang-c/default.htm. 2001.

[12] UDDI Specification. Version 2.0. http://www.uddi.org/specification.html. 2001.

[13] W3C. Simple Object Access Protocol (SOAP) 1.1. W3C Note. http://www.w3.org/TR/SOAP/. 2000.

[14] W3C. Web Services Description language (WSDL) 1.1. W3C Note. http://www.w3.org/TR/2001/NOTE-wsdl-20010315. 2001.

[15] W3C. Second Edition of the Extensible Markup Language (XML). 1.0 Specification. W3C Recommendation. http://www.w3.org/TR/2000/REC-xml-2001006. 2000.

[16] Xu, S., Randell, B., Romanovsky, A., Rubira, C. M. F., Stroud, R. J., and Wu, Z. Fault Tolerance in Concurrent Object-Oriented Software through Coordinated Error Recovery. *Proceedings of the IEEE Symposium on Fault Tolerant Computing*. 1995.