Reliability of Composed Web Services
From Object Transactions to Web Transactions

Thomas Mikalsen, Isabelle Rouvellou, Stefan Tai
IBM T.J. Watson Research Center, New York, USA
{tommi, rouvellou, stai}@us.ibm.com

Abstract

Multiple Web Services often need to be composed within some business process. Existing Web Service standards do not address reliability of such compositions. In this position paper, we argue that reliability can be achieved by adopting and extending existing advanced object transaction processing technology, like the OMG/J2EE Activity Service. We identify and discuss some important problems and research issues related to this approach.

1 Introduction

A Web Service is a program that can be invoked in a distributed web environment. Web Service technologies like SOAP, WSDL, and UDDI are standards to describe, publish, discover, and use Web Services; they define a development model for integrating applications over the web.

SOAP and WSDL are fundamental technologies for an application to issue a request to a Web Service. However, they do not provide support for the application to compose multiple Web Services. With SOAP and WSDL, requests to multiple Web Services are issued individually, but a set of requests cannot be grouped into a single process flow across the web. Because of this shortcoming, additional technologies for web business process specification and management are currently being developed. These include new languages extending WSDL, for example, IBM's WSFL/WSEL and Microsoft's XLANG.

WSFL and XLANG propose language constructs for defining web processes that can involve requests to multiple Web Services. However, the composition of Web Services as achieved with WSFL and XLANG observes no or only limited forms of reliability. We believe that this restricts their applicability unnecessarily. In many web systems for electronic commerce, for example, a Web Service composition must be reliable. Qualities of composition, such as atomicity of processing of a defined set of Web Services requests, or consistency of data transformations applied to the set of composed Web Services, should be well-defined, observable, and guaranteed. For this purpose, both language (contractual specification) as well as system infrastructure support is needed.

In this paper, we identify research issues in the development of system infrastructure support for reliability of composed Web Services (deferring language issues at this time). We investigate the suitability and applicability of object transactions, a common and proven approach to reliability in object systems, to web environments. Further, we outline ideas for a model of web transactions that is based on the OMG/J2EE Activity Service specification, an advanced object transaction service. Overall, we aim at exploring a practical solution to web transactions that is compliant to Web Services standards.

2 Object Transactions

Transactions are fundamental to reliable distributed computing, and most commercial distributed object systems provide support for traditional transaction models (e.g., ACID transactions, two-phase commit). Such systems typically

support standard architectures, such as the CORBA Object Transaction Service (OTS) and the J2EE Java Transaction Service (JTS). These systems offer a mature, interoperable, and widely used, solution to address reliability.

Increasingly, however, these systems are being used to build advanced applications, where traditional transaction semantics are not appropriate. For example, in applications where transactions execute over long periods of time, the concurrency control mechanisms employed to support ACID semantics are unacceptable. In other cases, dependencies between application activities are complex, requiring flexibility in determining transaction outcome. The transaction models required by these applications (so called “extended transaction models”) are rarely directly supported by the system, and are often implemented at the application level; this is typically done in an ad hoc manner, making it difficult to construct, maintain and integrate these applications.

The OMG Activity Service, and related J2EE Activity Service, represent the state-of-the-art in distributed object transactions. The service provides a framework for the construction of new middleware services used to support specific extended Unit of Work (UOW) models. One such UOW model might be used to attain strict ACID properties for a group of related activities; for example, a traditional distributed commit protocol, such as two-phase commit (2PC). In such a model, a unit of work represents a unit of failure: that is, if any single activity fails, the unit of work as a whole is implicitly undone. These semantics, however, are not always desirable (or acceptable). A different extended UOW model might be used to support long-lived computations, where a unit of work does not implicitly represent a unit of failure; rather, individual activities within the UOW can fail without necessarily abrogating the entire unit of work. Such a model might use compensating actions to repair actions damaged by a failure, and then continue processing within the same unit of work. Another extended UOW model might be appropriate for inter-business transactions, where business partners are less willing to submit to external commit coordination. Many other models, used to support different types of extended units of work, can be considered.

3 Towards Web Transactions

The flexibility offered by the Activity Service is particularly desirable in the Web Services world, where the sphere of distribution extends beyond individual enterprises. This change in scope places additional requirements on systems used to support these services. In particular, while Web Services demand reliability, they also require flexibility in how it is attained. We expect the need to support this flexibility to stimulate new extended UOW models – models which could be supported by a standard framework such as the OMG/J2EE Activity Service. Further, we believe that individual Web Services, and the infrastructure software used to support them, will often rely heavily on distributed object technology, including the OMG/J2EE Activity Service. (Many vendors have already extended their existing distributed object platforms to include support for both Web Services and the Activity Service).

The communication between Web Services (or between an application and the Web Services it uses), however, will not be based on the distributed object communication model of CORBA or J2EE. Rather, as existing Web Services standards define, interactions across the web will be SOAP/XML-message-based, and instead of a remote CORBA IDL or Java interface, a WSDL specification is required. Message communication is different from (distributed) object communication in that a different binding model, a different timing model (in particular, synchronous versus asynchronous communication), a different life-cycle dependency model (required availability versus tolerance of non-availability of an object/service at connection- and/or run-time), and a different arity in communication partners (one-to-one versus one-to-many) is possible.

The object transaction models, including the Activity Service, are based on a standard object communication model, but not message communication. In order to allow an intra-Web Service object transaction to include requests to other Web Services (thus, to become an inter-Web Service transaction), the transaction model and system infrastructure must be capable to bridge between the two communication models and technologies and support both styles in one integrated fashion.

More specifically, we propose that the Activity Service be extended to support the communication models available to Web Services. This will enable flexible “web transactions” that span multiple Web Services and the objects that implement those services.

According to the Activity Service model, a Web Service (one or more of its WSDL operations) to be used would then need to be registered as an `action` with the `activity coordinator` of the transaction originating Web Service/application. The registration describes an interest to receive transaction `signals` of a defined `signal set`, and to communicate back `action` outcomes. Such registration for signals therefore is one example of a necessary *contractual agreement* between Web Services that want to participate in the same transaction. The implementation of such a "web-enabled" Activity Service would then need to support both kinds of distribution and communication models as described above: the distribution (of signals and other notifications) within the Web Service ("*object distribution*" using CORBA or J2EE) and the distribution across the web ("*web distribution*" using SOAP/WSDL). The notifications themselves must be either object invocations or SOAP messages, but both must be semantically equivalent. Figure 1 illustrates the idea of a web-enabled object transaction service that supports both object and web communication models and technologies.

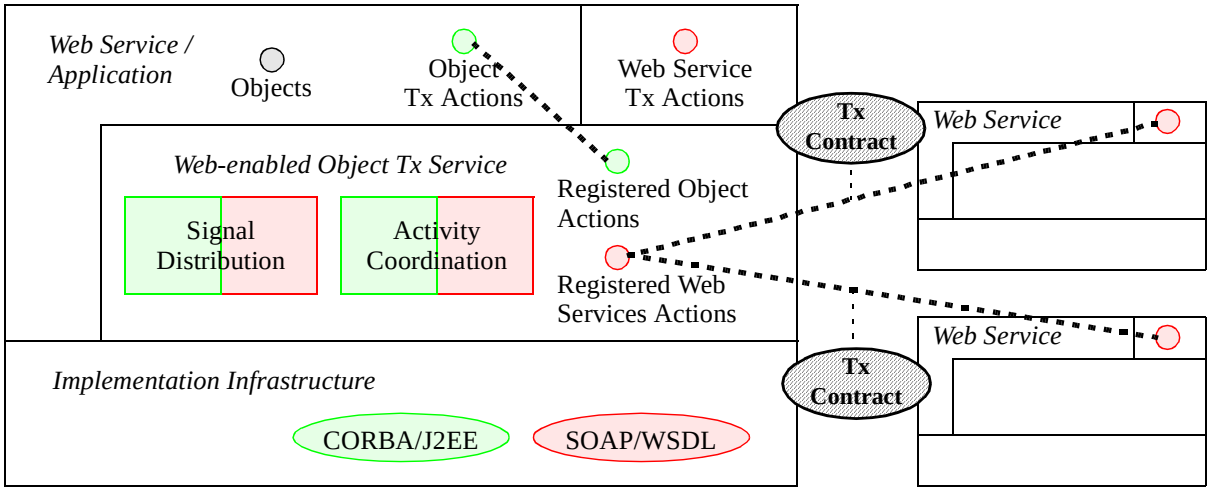

FIGURE 1. Web-enabled Object Transaction Service

4 Discussion

Reliability through some notion of transactional behavior is expected to be a relevant Quality of Service (QoS) for Web Services. At some level, one could argue that a Web transaction is a "legal" contract between two trading partners. Service A "tells" Service B that operation 1 can be included in an atomic transaction originated by Service B. Service B chooses to do so. If the transaction fails, Service A is informed of the failure, and Service B can safely assume that operation 1 is "cancelled". As far as Service B is concerned, it is as if operation 1 had never been invoked (e.g., B will not assume any cost of operation 1). Service B does not care how Service A implements the contract (an internal classical transaction is rolled back or compensated, Service A writes off the cost of operation 1 as a loss.....). In this scenario, atomicity has clear unambiguous semantics shared by both services. Web Service transactions clearly will not be possible if we do not standardize the agreement/description of the transactional semantics of a service business process (these semantics could be incorporated within WSFL/WSEL and/or XLANG, for example). It is also clear that the semantics of those web transactions will be much looser than the semantics associated with the traditional object transactions.

In this position paper we propose an approach to build practical infrastructure support for such loose transaction contracts. The goal of the infrastructure is to automate some of that support, making it cheaper to build, maintain, and integrate. Service B might not care how Service A implements the contract, but the enterprise providing Service A probably prefers to roll back a transaction rather than writing off a cost as a loss. Furthermore, this enterprise is also probably using its distributed object platform as the underpinning of its Web Service and Service A will typically delegate work to smaller-grained distributed objects in the distributed system that underlies the Web Service. The infrastructure should therefore extend as much as possible the existing object transaction technologies rather than ignoring them – eventually end to end integration from legacy integration to enterprise application integration (EAI) to B2B

integration will be needed. The infrastructure must support different levels of service granularity for what we expect to be highly flexible contracts between trading partners.

Concluding, we suggest three main areas of research relating to reliability of composed Web Services:

1. Application Interaction / Contractual Agreements

   For example, how does a Web Service expose its ability, willingness, certification, and so on, to participate in a web transaction?

2. Infrastructure

   For example, how are object transaction contexts represented and propagated using Web Services communication models, and how do non-object based Web Services participate in these transactions?

3. Relationship between application level and infrastructure level

   For example, how do application interactions and contractual agreements relate to infrastructure constructs?