Adaptive Web Transactions: An Approach for Achieving the Atomicity of Composed Web Services

L. Pajunen, J. Korhonen, J. Puustjärvi
Software Business and Engineering Institute, Helsinki University of Technology,
P.O. Box 9600, FIN-02015 HUT, Finland
{Lasse.Pajunen,Jarmo.Korhonen,Juha.Puustjarvi}@hut.fi

Abstract

The effective use of web services requires that new and more complex web services can be composed of other web services. With such composed web services, or web workflows as they are also called, the issues related to transactional properties are even more important. For example, execution correctness and reliability are needed. In this paper, we represent a meta ontology for transactional web workflows. It extends other related conceptualizations by incorporating transactional properties. Particularly, it supports a transaction model, called *adaptive web transaction model*. It introduces a variety of methods for a simple web services to be interconnected to a compose web services. Such interconnection methods provide an elegant way for constructing composed web services.

1. INTRODUCTION

Web services are self-describing applications that can be published and invoked across the Internet. Such a service can be anything from a simple request to a complicated business process. The effective use of the web services requires that new and more complex web services can be composed of simple web services. For example, the reservation of a business trip could be a composed web service. Its component services could be flight reservation, hotel reservation, and car renting.

Analogous to distributed transactions, also composed web services, or web workflows as they are also called, should be executed in an atomic way. For example, in the business trip case, the hotel reservation should not be done if the flight reservation failed.

There are different approaches (e.g. WSFL [12], and DAML-S [7]) for the specification of the execution structure of composed web services, i.e., the ordering and the conditional execution of the component web services. In addition, there are several proposals dealing with the atomicity of composed web services (e.g. XLANG [17], BTP [14], and TIP [13]). However, these atomicity protocols themselves are not very useful if their components, i.e., single web services, cannot be connected to the protocols.

In this paper, we introduce a new transaction model, called *adaptive web transaction model* or *AWT-model* for short, for ensuring the atomicity of composed web services. A salient feature of the model is that it provides different methods for single services to participate in the atomicity protocol. Even a single web service may support different methods for participating in the atomicity protocols.

The rest of the paper is organized as follows. First, in Section 2, we give an overview on the technology related to web services. Then, in Section 3, we extend the current technologies by presenting the AWT-model. A web workflow ontology including the AWT is presented in Section 4. After it, in Section 5, the usability of the proposed ontology is illustrated by the business trip example. Section 6 concludes the paper.

2. RELATED WORK AND STANDARDIZATION EFFORTS

Web Services research has provided many proposals. WSCI (Web Service Choreography Interface) [2], WSCL (Web Services Conversation Language) [3], WSFL (Web Services Flow Language) [12], and XLANG (Web Services for Business Process Design) [17] are specifications for describing workflows.

Semantic Web research has also studied service composition and interoperation. This work has been done in DAML-S project. DAML-S specification [7] describes their model. In addition, DAML-S work has been analyzed at [10] in respect to validation of composition of Web Services.

Workflow Management Coalition has analyzed workflows. XPDL (XML Process Description Language) [18] is their specification on how to model workflows.

Another view to describe workflows is coming from programming language direction [5, 9]. In these methodologies, a service composition is thought as a program that runs on the web and it is described using a programming language.

The Business Process Modeling Language (BPML) [1] defines a business process as an interaction between participants and the execution of activities according to a defined set of rules in order to achieve a common goal. BPML supports two transactional models: coordinated and extended. The coordinated model is also known as closed flat or ACID transactions, while the extended model is also known as open nested transaction, or Sagas.

The Business Transaction Protocol (BTP) [14] is a protocol, that is, a set of specific messages that are exchanged between computer systems supporting an application, with rules about the meaning and use of the messages. BTP is designed to allow coordination of application work between multiple participants owned or controlled by autonomous organizations. BTP uses a two-phase outcome coordination protocol to ensure the overall application achieves a consistent result.

3. THE ADAPTIVE WEB TRANSACTION MODEL

3.1 Atomicity Protocols

All proposed atomicity protocols are based on some form of voting. They differ from the traditional two-phase commit protocol in that the commitment does not necessary require each participant to vote commit. That is, some component services may be non-vital and in addition, there may be alternative component services. For example, in the case of the business trip, renting a car may be a non-vital task, and reserving a flight ticket and reserving a bus ticket may have alternative component services.

The main problem of atomicity protocols in loosely coupled environments is the implementation of the *uncertainty period*. It starts when the participant has voted Yes and it ends when the participant receives sufficient information to know what the coordinator's decision will be. The actual problem is that how to ensure that the coordinator's decision can be realized. According to the 2PC-protocol, data items are locked during the uncertainty period. However, such an approach is not possible without violating the autonomy of local systems. Therefore, the approaches based on compensating transactions, called *semantic atomicity*, are widely used in loosely coupled autonomous environments.

Semantic atomicity means that after the commitment of a subtransaction, or component web service, other transactions are allowed to access the updated data (dirty data), though the composed web service may still be aborted. Therefore, each compensating transaction undoes, according to the semantics of the application, the effects of the component service being compensated.

Compensations as an atomicity criterion, though well suitable for many component services, are not appropriate for all component web services. Compensations in some cases may not even be possible. For example, assuming that withdrawal is the compensation of the deposit, but the deposit fails if there is no more enough money in the account. In addition, the implementation of the compensations may be problematic, e.g., if the cancellation of a room reservation is not free of charge.

3.2 Supporting Adaptive Web Transactions

Analogous to the 2PC-protocol the communication in the protocol supporting AWTs is comprised of two phases: a voting phase and a decision phase. In the voting phase, the coordinator sends a request message to component web services, which then vote by sending either the failure or the commit message. The decision phase starts when the coordinator has received the responses from all the vital services. If all the vital services have voted commit, the coordinator decides to commit, and otherwise to abort. However, from alternative components only one needs to vote commit for successful transaction. If more than one alternative component votes commit, the coordinator aborts those services that are not needed. The coordinator sends the decision to all component services.

We make the distinction between three methods when a component service may vote commit.

- First, analogous to the 2PC-protocol the local transaction performing the component web service is in the prepared state, i.e., all its updates are logged in the stable storage.
- Second, the component transaction has successfully executed the service and in the case of the abortion the compensating actions can be executed.
- Third, the component service has executed the reservation, i.e. is in the *reservation state* [15]. It is a certification of the success of the potential later execution of the service.

3.3 Workflows and Transaction Models as Message Exchange Protocols

A workflow can be used to describe messages exchanged between services. A message can be in three states: coming from a sending service, going through some intermediaries, and going in to a receiving service. A workflow can be modelled from all these three views. A model based on view of sending services describes the output from the services. A model based on view of intermediaries describes routing and operations needed to transfer messages from the senders to the receivers. Then, the model based on receiving services describes input capabilities of the services. The input and output messages are typically described in service descriptions with each service. The service collaboration requires the aforementioned second model that is independent from individual services.

An interaction with more complex services may require more than one message exchange. In this case, a workflow can describe a usage of one individual service. The workflow describes then the order and other possible relations between exchanged messages. In addition, one individual service can operate on using many different message exchange protocols. In this case, a service composition includes the selection of an appropriate protocol for that case.

To support transactions within services, a two-phase protocol is needed as described in Section 3.1. This is a complex service as described previously. To model a complete workflow, the message exchanges between separate services and message exchange protocols with each service needs to be combined.

For a service to support different transaction methods, different message exchange patterns are needed. In Section 3.2, three different methods are identified. Here, we describe briefly the message exchange patterns and the role of messages.

In the first case, there needs to be three possible messages available to be sent to the service. The first message is sent to the service for preparing a coming operation. After the preparation, either of next two messages is sent: a commit or a rollback message. The second message terminates transaction successfully (commit) or unsuccessfully (rollback).

In the second case, one or two different kinds of messages are needed. If the message content can be changed to undo the operation, only one kind of message is needed. If opposing operations need different kinds of messages, two types of them are needed. For example, if there is a change value function with positive and negative parameters, only one function is needed. If there is a subtraction and addition functions separately with only positive parameters, two methods are needed.

In the third case, two or three messages are needed. The first one prepares a future operation. The second message is then used to confirm the operation. If an operation cancellation needs to be explicitly implemented, the third message is used for that purpose.

In each message exchange patterns, the operation states are similar. The first message changes the service to a prepared or a reservation state. The second message then commits or rollbacks the actual operation. The differences are on how the states are implemented.

A workflow management system needs to maintain separate workflows concurrently [16]. Then, the actual implementation of workflows may require exchanging some workflow identifiers or some other means for implementing many concurrent workflows. These issues are not covered in this paper.

4. WEB WORKFLOW ONTOLOGY WITH AWT SUPPORT

4.1 Semantic Web Service Framework and Protocol Stack

Section 3.2 discusses AWT from abstract perspective. In this section, AWT is discussed on technical and implementation perspective. At first, we describe Web Services framework. The framework can be modeled as a protocol stack. The protocol stack consists of many individual specifications. The Figure 1 shows a model with most relevant named technologies.

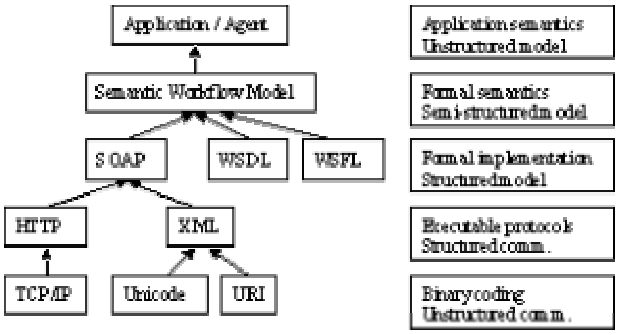

FIGURE 1: Semantic Web Service Framework

At the top most level, there is an actual application or agent. A programmer codes the application or the agent and therefore, they represent knowledge coded by a programmer. The semantics are included in the code and it cannot be separated.

At the second level, semantic workflow model is a formal methodology to represent workflows. It contains formal statements about the system. Compared to upper level, at this level semantics is independent from the current application and can be therefore separated. The other applications can read and understand semantics to be able enhance their interaction. In this paper, we focus on this level and provide a meta ontology describing this level. On the other hand, the topmost level application can be thought as an instance of specification corresponding to this meta ontology.

At the third level, SOAP [4], WSDL [6], WSFL [12] etc. provide an actual execution environment for higher-level models. This level describes individual services to be combined.

The fourth level describes how information is serialized and transmitted. HTTP and XML are concrete protocols to be used. Lower level specifications are only mentioned as background information.

Workflow is a real-life concept that has also a special meaning in computer science. Here, we use it in both meanings. Therefore, semantic workflow model can be further divided into two sublevels: a semantic and

implementation sublevels. The semantic sublevel has concepts that are relevant to real-life tasks. The concepts are for example a company, a payment, and an agreement. The semantic sublevel is an interface to an application description. The implementation sublevel has concepts that automate and help with these tasks. These are for example a service, a transaction, and an agent. The implementation sublevel is also a binding to actual implementations. The workflow ontology needs to have a combination of both of these sublevels and their concepts. After all, the implementation sublevel is used to implement the semantic sublevel. Figure 2 shows how these two sublevels are used in integrated architecture.

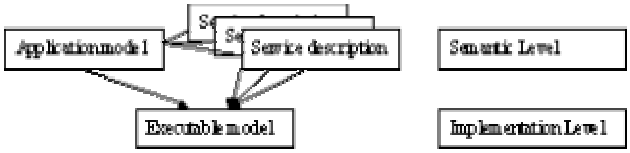

FIGURE 2: Division of Semantic Workflow Model

The application model is a white box model. This means that a developer knows the model and is able to edit it. On the other hand, service descriptions from other services are black box models. As other service providers provide them, the developer is not able to change them. There is usually enough information to use them (e.g. WSDL service descriptions), but implementation details and semantic meanings are not known.

When a workflow is designed, a model describing the complete workflow is composed. The composition includes modeling a new service and linking other services together. A sample of this merging process is described in Figure 3. However, the figure shows a very simplified case. In practise, cases are more difficult.

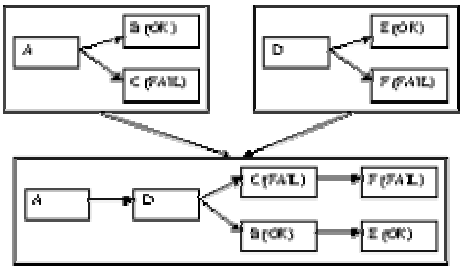

FIGURE 3: Merging Two Separate Workflows for a Composite Workflow

The complete workflow is called a *global model*. The linked services form separate *local models*. Each model can have different constraints that must be guaranteed during a process. During service composition, used workflow descriptions and constraints are merged to composite workflow with constraints. The service specific local constraints and composite service specific global constrains analyzed. The workflow specification tool validates that workflow and other constraints are not broken.

In addition to service validation, the complete execution order is specified. There may be different optional service execution orders and transactions that require more than one operation call per one action. These are specified in service composition.

4.2 Workflow Meta Ontology

Here, we describe our meta ontology. It has been designed by combining concepts from WSDL and WSFL and by extending them with transactional properties.

The workflow consists of activities and links between activities. The activities, links, and their relations are expressed using UML notation in Figure 4. The activities are send message, receive message, request-

response, solicit-response, start transaction, end transaction, and notify error. There are two kinds of links: control links and data links. The control links specify the execution order. There are several types of control links. The control link types are go, fork, join, start, end, loop, commit transaction, and abort transaction. Each control link has an optional Boolean expression to describe if link is currently enabled or is it disabled. The data links describe how information flows during workflow. Each data link has an optional data translation to describe how information is formatted during link transition.

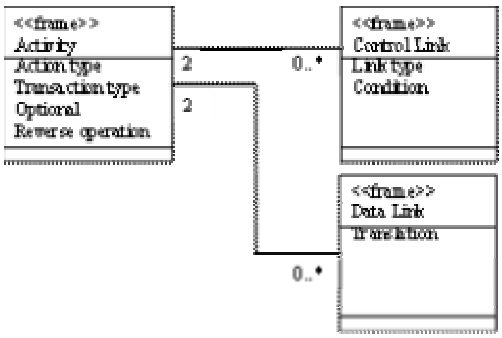

FIGURE 4: UML Diagram of Ontology Concepts

In our ontology, concepts can be mapped to class hierarchy and relationships between classes. The top classes are Activity, Control link, and Data link. Activity and Control link have derived classes.

Derived classes of Activity specify operations. Send message class is used when the wanted operation is to send a notification message to some other service. Receive message class is used to catch that notification message. Receive-response class models a synchronous method call from other service. Solicit-response class models a synchronous method call to some other service. Notify error class is used to send an error message to some other service.

Start and end transaction classes are used to model transactions. The start transaction describes the beginning of transaction, and makes a workflow engine to log operations. If error happens during the transaction, the workflow engine makes compensation actions for already successfully executed operations. After the compensation, the workflow continues from the end transaction activity. If the end transaction is reached through successful service calls, committing operations are executed and then the workflow continues.

Derived classes from Control links describe on how workflow execution goes after an Activity. Go class specifies a normal step. Fork class describes that next steps are done in parallel. Join class waits until all its inputs are executed. Fork and join are used to implement parallelism and synchronization during the workflow. Start and end classes mark the beginning and ending of workflow. These classes specify places where separate workflows are combined. Commit and Abort transaction classes are used at the end transaction activity to specify committing and compensation paths.

Activities (send message, receive message, request-response, and solicit-response) have three attributes related to transactions. The first one flags if the service is optional (non-vital) in transaction. The second one is the transaction model. The last one is functions to be called in the end transaction phase. The last parameter depends on the transaction model.

The workflow engine keeps track of successful service calls and actions needed in the case of commit or rollback. If non-vital services are failed, the complete transaction is not failed. If vital service fails, then the whole transaction fails.

To combine information flow between services, the messages and method invocation parameters need to be

described. In our ontology, data translations are used. The data translation means that outputs from previous services are given to translator and its output is used as a message. As a translator technology, XSLT can be used for example.

5. APPLICATION EXAMPLE: CASE BUSINESS TRIP RESERVATION

This section uses our meta ontology in real life problem. We use a business trip reservation as an example. In our case, the service user wants to order flight tickets, to book a hotel room, and optionally to reserve a car.

A flight ticket service is operating in the traditional 2PC-model. Figure 5 visualized this workflow. In the figure, squares are activities and arrows are control links. Data links are not shown. There are three possible message exchanges: an ordering flight tickets, a committing the order, and an aborting the order. These are modeled using our meta ontology.

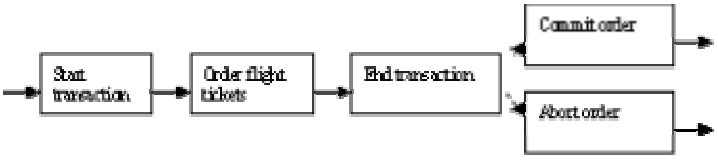

FIGURE 5: Flight Ticket Service Workflow

A hotel room service uses option model. The service has also two sequential methods that are used when reserving rooms. A car reservation operates on saga model. Therefore, there is only one method needed if everything goes well. The cancellation method is used if the transaction fails. Figure 6 presents one possible combined workflow. The notation is similar to Figure 5, except the type of control links are shown.

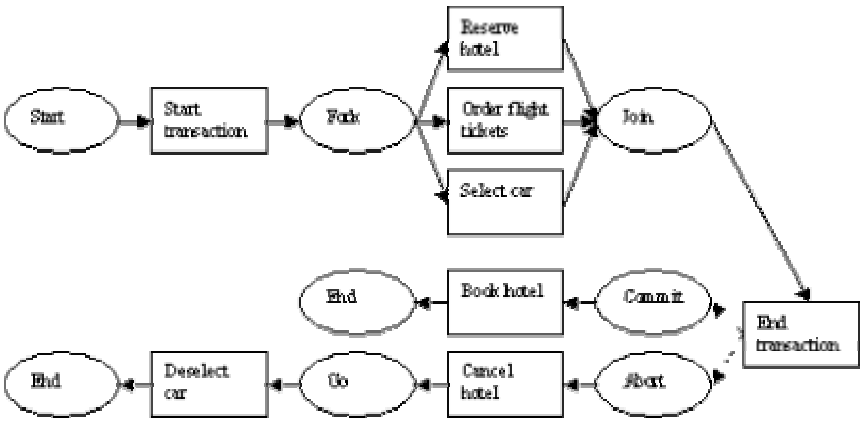

FIGURE 6: Complete Service Workflow without Data Links

In workflow, a hotel booking, an ordering flight tickets, and a reserving a car are done concurrently without any specific order. They are executed as a transaction. After doing transaction in end transaction activity, the appropriate methods are called. In this example, they are done in sequence, but the workflow generation tool could also design a workflow model where they are executed in parallel.

6. CONCLUSIONS

Here, we have described a situation where workflow and transactions are combined within Web Services

framework. We have shown that workflow and transactions can be modeled in the same web service framework. In addition, we have created a meta ontology that provided concepts for modeling workflows and transactions. Our ontology emphasizes transaction features within the workflow.

The main idea behind the AWTs is that each composite web service may choose an appropriate method for participating in the commitment protocol. In particular, we introduce a new participating state, called *reserving state*, which differs from the *prepared state* in that it is based on the semantic of the application. Our argument is that in designing web services, their possible later connections to composed web services should be taken into account. Our proposed *reserving state* would allow an elegant way for constructing transactional composed web services.

The meta ontology provides new opportunities for further studies. The transaction capabilities can be developed further. This means possibly automating a generation of compensations and an optimizing of an execution order. The studies can be done to apply these technologies in combining Web Services. Another idea for further studies is studying more on a service composition. The ontology could allow a recursive and parallel service composition and therefore, be better supportive for composing concurrent services in many layers.

REFERENCES

[1] A. Arkin. Business Process Modeling Language (BPML). BPMI.org. http://www.bpmi.org/bpmi-downloads/WD-BPML-20010308.pdf.
[2] A. Arkin, S. Askary, S. Fordin, W. Jekeli, K. Kawaguchi, D. Orchard, S. Pogliani, K. Riemer, S. Struble, P. Takacsi-Nagy, I. Trickovic, S. Zimek. Web Service Choreography Interface 1.0. http://wwws.sun.com/software/xml/developers/wsci/wsci-spec-10.pdf.
[3] A. Banerji, C. Bartolini, D. Beringer, V. Chopella, K. Govindarajan, A. Karp, H. Kuno, M. Lemon, G. Pogossiants, S. Sharma, S. Williams. Web Services Converstation Language (WSCL) 1.0. http://www.w3.org/TR/wscl10/.
[4] D. Box, D. Ehnebuske, G. Kakivaya, A. Layman, N. Mendelsohn, H. Nielsen, S. Thatte, D. Winer. Simple Object Access Protocol (SOAP) 1.1. http://www.w3.org/TR/SOAP/.
[5] L. Cardelli, R. Davies. Service Combinators for Web Computing. IEEE Transactions On Software Engineering, Vol. 25, No. 3, 1999.
[6] E. Christensen, F. Curbera, G. Meredith, S. Weerawarana. Web Services Description Language (WSDL) 1.1. http://www.w3.org/TR/wsdl.
[7] DAML Services Coalition. DAML-S: Semantic Markup For Web Services. http://www.daml.org/services/daml-s/2001/10/daml-s.html.
[8] R. Fikes, D. McGuinness. An axiomatic Semantics for RDF, RDF-S, and DAML+OIL. http://www.daml.org/2001/03/axiomatic-semantics.html.
[9] D. Florescu, A. Grünhagen, D. Kossmann. XL: An XML Programming Language for Web Service Specification and Composition. In the Proceedings of The Eleventh International World Wide Web Conference, 2002.
[10] S. Narayanan, S. McIllraith. Simulation, Verification and Automated Composition of Web Services. In the Proceedings of The Eleventh International World Wide Web Conference, 2002.
[11] O. Lassila, R. Swick. Resource Description Framework (RDF) Model and Syntax Specification. http://www.w3.org/TR/REC-rdf-syntax/.
[12] F. Laymann. Web Services Flow Language (WSFL 1.0). http://www-3.ibm.com/software/solutions/webservices/pdf/WSFL.pdf.
[13] J. Lyon, K. Evans, J. Klein. Transaction Internet Protocol Version 3.0. http://www.ietf.org/rfc/rfc2371.txt.
[14] OASIS. Business Transaction Protocol. An OASIS Committee Specification. https://www.oasis-open.org/committees/business-transactions/documents/specification/2002-06-03.BTP_cttee_spec_1.0.pdf.
[15] J. Puustjärvi. Options: a Way for Achieving Failure Atomicity in the WorkMan System. In Proceedings of the Seventh International Conference on Database Systems for Advanced Applications, 2001.
[16] J. Puustjärvi. Workflow Concurrency Control. The Computer Journal, Vol. 44, No. 1, 2001.
[17] S. Thatte. XLANG Web Services for Business Process Design. http://www.gotdotnet.com/team/xml_wsspecs/xlang-c/default.htm.
[18] Workflow Management Coalition. Workflow Process Definition Interface – XML Process Definition

Language. http://www.wfmc.org/standards/docs/xpdl_010522..pdf.