COMPUTING SCIENCE

Metadata Support for Transactional Web Services

M. P. Machulak, J J. Halliday, M C. Little

TECHNICAL REPORT SERIES

No. CS-TR-1158 July, 2009

TECHNICAL REPORT SERIES

No. CS-TR-1158 July, 2009

Architecting Dependable Access Control Systems for Multi-Domain Computing Environments

M. P. Machulak, J. J. Halliday, M. C. Little

Abstract

The paper describes an annotation-based Java framework for supporting Transactional Web Services. We provide a framework for automated management of compensations in Business Activity type extended transactions. Unlike rollbacks in traditional ACID transactions, these usually require substantial implementation effort by business programmers. Our annotations provide a flexible, intuitive and easy to use alternative. With a few declarative statements, users can configure the relationship between units of work and their compensations. The framework manages all aspects of the execution of the compensations, ensuring a reliable, consistent transaction outcome.

Printed and published by the University of Newcastle upon Tyne,
Computing Science, Claremont Tower, Claremont Road,
Newcastle upon Tyne, NE1 7RU, England.

Bibliographical details

MACHULAK, M., HALLIDAY J.J., LITTLE, M.C.

Metadata Support for Transactional Web Services
Newcastle upon Tyne: University of Newcastle upon Tyne: Computing Science, 2009.

(University of Newcastle upon Tyne, Computing Science, Technical Report Series, No. CS-TR-1158)

Added entries

UNIVERSITY OF NEWCASTLE UPON TYNE
Computing Science. Technical Report Series. CS-TR-1158

Abstract

The paper describes an annotation-based Java framework for supporting Transactional Web Services. We provide a framework for automated management of compensations in Business Activity type extended transactions. Unlike rollbacks in traditional ACID transactions, these usually require substantial implementation effort by business programmers. Our annotations provide a flexible, intuitive and easy to use alternative. With a few declarative statements, users can configure the relationship between units of work and their compensations. The framework manages all aspects of the execution of the compensations, ensuring a reliable, consistent transaction outcome.

About the author

Maciej Machulak received his MSc in Computing Engineering from Wroclaw University of Technology in Poland in 2007. During his studies he was an Erasmus Exchange Student at Newcastle University. Maciej Machulak additionally completed the Advanced MSc degree in "System Design for Internet Applications" (SDIA) in 2007 at Newcastle University and his thesis was awarded Best 2007 SDIA Thesis. This degree included an industrial placement at Red Hat UK Ltd. Maciej's main task was to develop a framework for transactional Web Services. Before commencing his PhD studies Maciej was also employed as an intern at Red Hat and worked on embedded tools for transaction monitoring inside JBoss Application Server.

Maciej Machulak is currently a PhD student working with Dr. Aad van Moorsel on the Trust Economics project, funded by the UK Technology Strategy Board (TSB). Maciej's project is concerned with building a novel authorization system for Web environment which allows users to flexibly adapt access control for their Web resources to their particular security requirements.

Suggested keywords
METADATA, TRANSACTIONAL WEB

Metadata Support for Transactional Web Services

Maciej P. Machulak, Jonathan J. Halliday, Mark C. Little
JBoss, a division of Red Hat.
{mmachulak,jhalliday,mlittle}@redhat.com

Abstract

The paper describes an annotation-based Java framework for supporting Transactional Web Services. We provide a framework for automated management of compensations in Business Activity type extended transactions. Unlike rollbacks in traditional ACID transactions, these usually require substantial implementation effort by business programmers. Our annotations provide a flexible, intuitive and easy to use alternative. With a few declarative statements, users can configure the relationship between units of work and their compensations. The framework manages all aspects of the execution of the compensations, ensuring a reliable, consistent transaction outcome.

1. Introduction

Transaction processing is at the core of business. Every exchange of money for goods is a transaction, as are most other activities within commerce, military, and science. Transaction processing technology ensures that any activity's operations on data are recorded consistently on computer systems, so that the systems remain as reliable indicators of the "real world" as their paper-based antecedents did, or at least as closely as possible given the vagaries of the electronic medium.

For example, a bookstore's database must always accurately reflect in store inventory of what's on the shelves. Manual processes to confirm inventory are costly, as are errors in bank transfers, telephone billing, and manufacturing line preparation. Transaction processing technology helps all of these now automated activities run smoothly and as expected, despite system failure, which can be counted upon to regularly occur. Computers, being basically unstable electronic devices, are subject to all manner of problems.

This article was originally submitted for Middleware for Web Services Workshop 20 August 2007 and was presented 16 October 2007.

Atomic transactions are a well-known technique for guaranteeing consistency in the presence of failures. The ACID properties of atomic transactions (Atomicity, Consistency, Isolation, Durability) ensure that even in complex business applications consistency of state is preserved, despite concurrent accesses and failures. This is an extremely useful fault-tolerance technique, especially when multiple, possibly remote, resources are involved.

However, as Web Services have evolved as means to integrate processes and applications at an interenterprise level, traditional transaction semantics and protocols have proven to be inappropriate. Web services-based transactions differ from traditional transactions in that they execute over long periods, they require commitments to the transaction to be "negotiated" at runtime, and isolation levels have to be relaxed.

It has long been realised that ACID [1] transactions by themselves are not adequate for structuring longlived applications. As a result, the OASIS WS-TX standard for implementing transaction support for Web Services provides an extended transaction model: Business Activity (BA), which is designed specifically for long-duration transactions, where exclusively locking resources is impossible or impractical. In this model services are requested to do work, and where those services have the ability to undo any work, they inform the BA such that if the BA later decides to cancel, it can instruct the service to execute its undo behaviour. Consistency can still be maintained through compensation, though the task of writing correct compensating actions (and thus maintaining overall system consistency) is delegated to the developers of the services.

The coordination of Business Activities relies upon a reliable messaging layer to ensure robust communication. Message exchanges and state transitions are reliably recorded for crash recovery purposes. Despite this it can be a substantial burden on business programmers to ensure correct compensation in BAs, particularly compared to the largely automatic rollback handling in ACID transactions. This is due in

large part to the relative immaturity of support for Business Activities in application server and database implementations, compared to the pervasive support for the XA protocol used by ACID transactions. We aim to address this issue.

2. Problem Statement

Business focused programming environments, such as enterprise Java application servers and relational databases, provide high level abstractions to support construction of ACID transactions [2]. EJBs, for example, require the business programmer to simply declare their transactional behaviour with a few annotations. The application container and database then provide all the necessary plumbing to ensure ACID properties for the execution of that business logic, including difficult issues such as serialisation of state, concurrency control, locking and versioning of data, logging, rollback and crash recovery handling. The business programmer need not write code for the participants (resources) that respond to transaction protocol messages such as prepare, commit and rollback – these are provided for them.

Writing transaction-aware Web Services with accordance to the Business Activity model [3] is much more complicated and problematic. In particular, it requires the business programmer to not only provide the compensation code, but also to provide a participant implementation capable of processing transaction control messages (cancel/compensate) to correctly and reliably apply these compensations. This frequently results in business logic being mixed with transaction-related code.

The objective of our work is to provide a framework through which the business programmer may declaratively wire up compensation methods for their business activities by using annotations. The framework then works with the underlying transaction manager to ensure the reliable and correctly ordered execution of these compensations as and when necessary.

3. Related Work

The use of annotations for declaratively configuring both Web Services and transactional behaviour is widespread in both Java and .NET. Focusing on Java, the EJB3 specification provides annotations to add transactional and persistent behaviour to business logic. The JSR-181 specification [4] similarly provides annotations for exposing business logic as Web Services. Taken together, they provide for transactional Web Services using the WS-AT protocol. However, we are not aware of any current work addressing annotation based configuration of business logic for participation in WS-BA driven Business Activities. This then is the niche that we address with our work.

4. Framework

By using a clean and intuitive annotation based API we aim to leverage programmer familiarity with e.g. EJB3 and JSR-181 to reduce the learning curve for new users of our framework. We assume a one-to-one relationship between original methods and their compensation actions.

The framework's API consists of the following set of annotations that can be used to mark Web Service methods to expose them as Business Activity tasks:

- @BAService specifies the service in terms of its state management (if it modifies any data or not);
- @BAAgreement specifies the agreement protocol the service is willing to participate in;
- @BACompensatedBy provides information about compensation type and method;
- @BACompensationManagement marks a BA compensation manager to enable transparent dependency injection. This manager assists the business programmer by making data from the original method call available to the corresponding compensation method, facilitating e.g. restoration of a prior state;
- @BAParam annotates the service's parameter to be processed by the compensation mechanism;
- @BAResult annotates the service's return value to be processed by the compensation mechanism;

To address the problem of management of intermediary data that can potentially be required by compensation actions we have designed additional operations, which can be invoked on a BA compensation manager:

- put(id,Object) supports storing arbitrary objects with given identifiers;
- get(id) used in compensation tasks for retrieving objects with given identifiers.

4.1. Design

To understand the structure that comprises our framework it would be beneficial to present a sequence of events (Figure 1) that occur during a client call, which is a part of a BA transaction. When such call is made it is intercepted before it reaches the actual Web Service method. For this purpose we have decided to use the Aspect-Oriented Programming (AOP) concept as it is a natural way to intercept events and trigger appropriate functionality based on those events. To guarantee that transaction-related functionality is applied only for specific and correctly annotated Web Service methods, we have defined complex

compositional pointcuts. Those pointcuts specify exact join points within the application code where the framework's mechanisms should be executed.

When a call to a Web Service is intercepted it is associated with a unique task identifier. Interceptor then delegates transaction management for this specific invocation to a different component, called the Business Activity Manager. This component holds information about all Web Services that have been invoked, their transactional requirements, the order of their invocation and their associated participants. It can identify participants by a *{txId,serviceId}* tuple, where the first value represents the transaction Id and the other one identifies a Web Service method. With this information given, the Business Activity Manager is able to recognise whether a method has been already executed within a scope of the given transaction. If yes, then the invocation, which is being processed, is simply associated with the existing participant. Otherwise, it is associated with a newly instantiated participant, which is also enlisted to play role in this specific Business Activity. As already stated, the order of method invocations within a transaction is preserved. This enables execution of dependent Web Services whose compensation actions must be performed in a reverse order.

When the interceptor gets hold of a reference to the appropriate participant it wraps it with a compensation manager. Such manager is made accessible for the target object on which the actual method should be invoked. It provides a lightweight API for reliably storing additional objects that might be used during compensation (put() and get() methods).

After the described sequence of events the actual business logic of a method is invoked. Typically, business programmers do not have to concern themselves with any transaction-related operations. However, in case additional flexibility of compensation data management is required, previously mentioned simple operations provided by the compensation manager can be used.

When the method's execution is finished, its return value is intercepted. Any parameters and return values marked with @BAParam and @BAResult annotations respectively are automatically remembered in a reliable manner by the framework's mechanisms in case the Business Activity is to compensate by client's request. Finally, the return value can be passed back to the method's call originator.

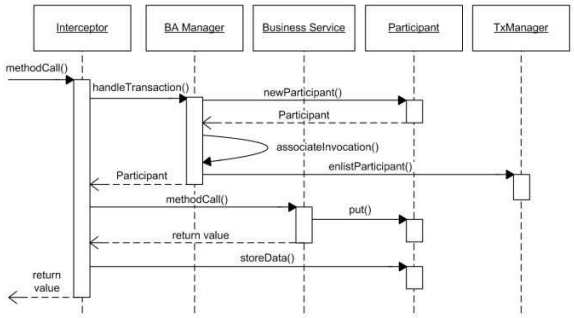

Figure 1. Business Activity task management.

When work within a certain Business Activity has been completed the client may decide that BA should either close or work should be compensated. If the BA is requested to close then participants are simply disassociated and all compensation-related data is destroyed. However, if the client requests the BA to compensate then a more complex set of actions occurs (Figure 2).

Participants communicate the necessity of compensation to the Business Activity Manager, which knows the correct order of method invocations within a specific transaction. The manager then drives the compensation process and asks participants in a given order to perform single compensation actions for their tasks. When a participant is asked to compensate a specific task then it matches compensation data of this task with the input parameters required by a specific compensation method and invokes it. The framework also seamlessly associates previously described compensation manager with the compensation action so that any additional data, which might be used in the business logic, can be retrieved.

When the execution of a single compensation action finishes, the task identifier is removed from the list handled by the participant and compensation data is destroyed. Control is returned to the BA manager, which removes this task from its list as well. It may then proceed with the next task, or if no more tasks are to be compensated then the whole compensation process ends.

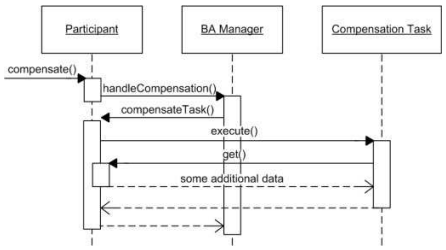

Figure 2. Compensation action management.

It is important to note that the actual design of the framework is more complex. It includes additional components responsible for caching information about already invoked services, processing compensation related data or executing compensation tasks. The framework has been built in a fully modular way with clearly separated parts responsible for only well defined subsets of tasks. Such approach enabled to encapsulate implementation specific parts which have been proven to be easily exchanged and customised.

4.2. Implementation

The current version of framework has been implemented on top of the existing XML Transaction Service (XTS) [5] component, which is part of the JBoss Transaction Service. It uses the JBoss AOP framework [6] to intercept method calls and enrich Web Services with transaction-related functionality.

4.3. Application example

An example of an application that may participate in a Business Activity is presented in Figure 3. The book() method, which is exposed as a Web Service, is simply marked with previously described annotations. Those specify its BA-related requirements and associate it with the cancel() method as its compensation task. The BACompensationManager object is used to store additional data, which may be used later along with the annotated parameter and return value if the Business Activity is requested to compensate. It is important to note that the compensation task is just another service that can be also used separately outside the scope of a Business Activity or it may be an internal method which is not exposed at all.

```
@BACompensationManagement
BACompensationManager cm;
```

```
@WebMethod
@BAService(BAServiceType.MODIFY)
@BAAgreement(BAAgreementType.PARTICIPANT_COMPLETION)
@BACompensatedBy(value="cancel",type = BACompensationType.CUSTOM)
@BAResult("reservationNumber")
public int book(@BAParam("uid") int userId, int roomNumber)
{
    … // business logic
    cm.put("refundValue",5);
    … // business logic
    return reservationNumber;
}
```

```
@WebMethod
public void cancel(@BAParam("uid") int userId, @BAParam("reservationNumber") int reservation)
{
    … // business logic
    Integer refund = (Integer) cm.get("refundValue");
    if (refund != null)
    {
        … // business logic
    }
    … // business logic
}
```

Figure 3. Application example

5. Future work

Development of our framework is an on-going research project. Although its first version has been already released, it still misses several features, which we have recognised to be potentially attractive in production environments.

One of the features which we are developing is the remote compensation. Its aim is to allow business programmers to provide URL of a service (possibly administered by a 3rd party) that is to be used as the compensation task. It is important to note that such compensation has several drawbacks which would need to be considered by developers. The remote service may not be working so that compensation will not succeed. To minimise the risk of the overall system consistency loss we are designing mechanisms, which will allow business programmers to specify multiple possible compensation actions for a single task. The framework will ensure that only one, currently available compensation action is executed.

We are also extending the framework to allow exposing existing components as Business Activity tasks. This will be achieved by allowing business programmers to specify all necessary BA-related information in XML description files instead of using annotations. We are working on a design of a relevant XML schema and necessary framework components.

6. Conclusions

Exposing business logic as Web Services requires a new approach to transaction management, which is poorly supported by current environments.

Our proposed framework provides a declarative, annotation driven approach to the problem of developing BA aware Web Services, relieving the business programmer of much of this burden.

7. References

[1] Gray, J., Reuter A., *Transaction Processing: Concepts and Techniques*,Morgan Kaufman,San Francisco,USA, 1993.

[2] Little, M., J. Maron, and G. Pavlik, *Java Transaction Processing: Design and Implementation*, Prentice Hall PTR, New Jersey, USA, 2004.

[3] WS-BusinessActivity Specification, version 1.0. August 2005. http://www.arjuna.com/library/specs/ws-tx/WS-BusinessActivity.pdf.

[4] JSR 181: Web Services Metadata for the Java™ Platform, version 1.0. June 2005. http://jcp.org/en/jsr/detail?id=181

[5] Halliday J.J. et al. JBoss XML Transaction Service. http://labs.jboss.com/jbosstm/

[6] Khan K. et al. JBoss AOP. http://labs.jboss.com/jbossaop/