

Session 5: Advanced Transactions

Christoph Liebig¹ and Stefan Tai²

¹ Darmstadt University of Technology, Germany
chris@informatik.tu-darmstadt.de

² IBM T.J. Watson Research Center, New York, U.S.A.
stai@us.ibm.com

1 Introduction

The reliable and correct execution of programs is a key concern in the engineering of distributed systems. *Transactions* represent one of the mechanisms commonly used to address reliability and correctness.

A transaction groups a set of actions (such as object requests) as one single unit-of-work for which reliability and correctness properties hold. The properties of *atomicity* (*A*) and *durability* (*D*) define the notion of reliability in the context of transactions: a set of actions is either executed as one atomic unit-of-work, or none of the actions are executed, and the committed state changes caused by the set of actions are guaranteed to be persistent. The properties of *consistency* (*C*) and *isolation* (*I*) define the notion of correctness in the context of transactions: a transaction transforms the system from one consistent state into another consistent state only, and any intermediate system states of an ongoing transaction are not visible to any other concurrent transaction.

The two papers in this session describe research on transaction processing that each advance a certain kind of transaction model and transactional middleware technology. This session summary provides a short background on transactions first, and then discusses the presented research papers.

2 Background

A variety of transaction models and technologies supporting transaction processing have been proposed over the last years. These models and technologies differ in how they address and support some or all of the ACID properties. We can distinguish four major groups of kinds of transactions and transaction processing technologies:

- *Integrated database transactions*, as promoted by database management systems (DBMS),
- *Distributed object transactions*, as defined by the CORBA Object Transaction Service (OTS),
- *Component container-managed transactions*, as suggested by component technology such as Sun's Enterprise JavaBeans (EJB), and
- *Message-oriented transactions*, as promoted by message-oriented middleware such as IBM's MQSeries.

In a database context, the transaction model encompasses all of the ACID properties. DBMS use an integrated system architecture where the state of manipulated entities is under full control of the DBMS. This is the basis for integrated concurrency control and recovery. Transaction demarcation is either explicit, spanning several data manipulation statements, or implicit per statement. The transaction manager is typically closed, i.e. integrating external or remote transactional resources is not supported (besides the federation of multiple, possibly distributed, DBMS instances of the same brand).

Distributed object computing systems typically address multi-component application scenarios which are heterogeneous and open by nature. Following the paradigm of encapsulation, objects are characterized by their interface (or services provided) whereas state and persistence of state are considered to be implementation concerns. It is common practice to separate reliability and concurrency control concerns – although these may not be considered to be independent matters. Transaction managers in X/Open DTP and CORBA OTS follow this principle and provide transaction context management and 2PC coordination as their primary services. Participating resources (or resource managers) are obliged to follow the OTS protocol and guarantee recoverability or durability with respect to the decision on transaction outcome. Transaction services in distributed object systems are open in that they make no further assumption on the implementation of the participating resources. As long as an object provides the required interfaces and interacts in a 2PC conforming way it may be involved in a distributed transaction. From a transactional client point of view, transactions are demarcated explicitly and transaction services are typically realized as first class objects. It is the application designer's responsibility to draw the transaction boundaries around logical units of work.

In order to cope with the multitude of failure modes in distributed heterogeneous environments, atomicity of computations over distributed objects is a useful paradigm on its own. If additionally durability is required through persistence, this must be realized by the application programmer or typically by means of dedicated persistence services. The latter could be realized by traditional resource managers like databases or queues. But applications are not limited to the means by which they realize durability nor is the manipulation of state under control of the transaction service. The same applies for concurrency control. In fact, the concurrency control theory has been well researched for particular configurations like multi-database or multilevel systems. However, the general case for correctness and concurrency control in composite systems has been addressed in recent work, only, and is an area of ongoing research.

Component technologies such as Sun's Enterprise JavaBeans (EJB) follow the distributed object computing model, but introduce and emphasize the concept of a "container" to manage the interactions between object instances and the server. A container is responsible for tasks such as creating new object instances, and it supports the mapping between objects and records in a database and generates code for persistent storage. With component container-managed transactions, transactions also become the responsibility of the container. The container creates new or accepts existing client transaction contexts, and performs the invoked operations in those contexts. A major difference to distributed object transactions exist in the way a transaction is demarcated

and managed. Explicit transaction demarcation as proposed by the CORBA OTS allows a client to fully control the scope and behaviour of transactions. However, the explicit model typically requires the use of complex APIs, and to write transactional code within the business logic. This may reduce the clarity and reusability of the code. *Declarative* transaction management as proposed with container-managed transactions separates transaction behaviour from business logic, thus promises for better reuse of the same business logic code in different transactional or non-transactional environments. Declarative transaction management simplifies the coding of the transaction, as all that is required is to set transactional attributes for a component's operations. These transactional attributes are set at deployment time (not development time) on the server side. However, this also implies that the transactional behaviour of a server can easily be overridden and changed at deployment, which may interfere with the transactional requirements and expectations of clients.

Message-oriented transactions and transactional publish/subscribe include enqueueing/dequeueing of messages or publishing/consumption of notifications in *units of work* with "all-or-nothing" semantics. Enqueueing/dequeueing of messages and publishing/consumption of notifications is dependent on the overall transaction outcome and vice versa. The queue manager provides its own transaction manager and associated transaction demarcation API or it acts as a transactional resource on behalf of a distributed transaction. Note that message-oriented transactions are fundamentally different from the three other kinds of transactions discussed above. Message-oriented transactions group a set of messages that are to be delivered or consumed as a whole. Message-oriented transactions do not address the processing of data (as consequences of delivery or receipt of a message) and consistency of data transformations.

3 Advanced Transactions: X²TS and Bourgoigne Transactions

The technologies presented so far all reflect the *traditional* transaction processing concepts. The traditional transaction concept has its limitations as i) it lacks support for structuring multiple interdependent transactions and ii) traditional concurrency control does not support cooperative transactions and long running units of work. In the literature, there have been various suggestions to extend the traditional concepts of transactions. The papers in this session present work that addresses such extended transaction management concepts in two different areas. Their goal has not been to introduce yet another transaction model but to synthesize some ideas of such extended transaction models on behalf of services for distributed object systems. They aim at better addressing problems commonly encountered in the engineering of distributed object systems.

The first paper in this session, "Integrating Notifications and Transactions: Concepts and X²TS Prototype" by Christoph Liebig, Marco Malva, and Alejandro Buchmann, falls into the category of advanced distributed object transactions. It addresses issues of relating event notifications and distributed object transactions. The work extends concepts which have initially been proposed in centralized and monolithic active database systems. X²TS presents a novel service-based approach for transaction-aware event-based interaction in distributed systems, CORBA in particular. While transaction

services in current middleware only support request/reply interactions, X²TS focuses on event-based systems and allows to realize flexible transactional couplings between event producing and event consuming components. The application of the suggested service for process enactment is discussed. The prototype architecture is described and implications of distribution aspects on reliable event processing are discussed.

The second paper in this session, "Advanced Transactions in Enterprise JavaBeans" by Marek Prochazka, looks at component container-managed transactions. It identifies limitations of the EJB model, in particular with respect to concurrency control, support for long-running and cooperative transactions, as well as transaction flow control. It proposes the adaptation of the ACTA specification framework to EJB transactions. A new transaction management API is proposed that allows to establish dependencies between ongoing transactions along the modes presented in ACTA. The approach is called "Bourgogne transactions". With respect to concurrency control, the programmer is given control over the visibility sets and delegation of objects that are affected by the transaction. Some of the issues discussed are not restricted to container-managed transactions, but also apply to distributed object transactions (JTA/JTS transactions for Java systems in particular).

4 Discussion

The two papers in this session raise a number of interesting and important questions for continued research. During the workshop, research issues relating to the question of integrating transactions and event notifications were primarily discussed.

The paper by Liebig et al. tackles the problem of integrating the CORBA OTS with the CORBA Notification Service (NOS). While transaction management is well understood for procedural interactions, support for event-based interactions is restricted to message queue integrating transactions (and transactional publish/subscribe). In that case the visibility of notifications is bound to the successful commit of a transaction: immediate reactions as well as parallel actions triggered by events are not possible. In distributed object systems, there may be many different components that subscribe to events with differing reliability and processing demands. It is therefore suggested to extend the event-action paradigm with coupling modes that determine when events become visible, in which transaction context the reaction should run in, what the dependencies between triggering and triggered transaction are, and when events are considered to be delivered and consumed. Such integration of event notifications and transactions is highly desirable, particularly in the context of process enactment and workflow management.

During discussion, another reason for such an integration has been pointed out: many enterprise applications use both object-oriented middleware and message-oriented middleware in combination. Object middleware addresses reliability through the concept of transactions that comprise synchronous object requests only, while message middleware (or, messaging services for object middleware) addresses reliability through guaranteed message delivery even in the presence of system failures. An equivalent level of reliability for the use of both middleware in combination does not exist today. It is very desirable to advance the current state-of-the-art towards transaction

processing that allows for both synchronous object requests and asynchronous messages to constitute a single transaction at the same time.

When integrating event notifications and transactions we obviously need to understand to what extent the ACID properties are still satisfied, or are violated. We believe that the property of atomicity is actually "extended" in that the atomicity sphere of the event producer may depend - in various ways - on the atomicity sphere of the reacting event consumer. Thus a powerful mechanism to structure units of work into interdependent atomicity spheres is introduced that allows to realize advanced transaction concepts such as compensations, or pluggable reliable exception handling.

Recovery in event-driven systems has not received much attention. X²TS supports forward recovery by replaying events in case of failures and provision of tuneable delivery and consumption guarantees. More research is needed to find design guidelines for a consumer in case of event recovery: which recovered notifications should be ignored or reacted to and in which manner, especially in case of complex composed events.

The flexibility of reliable event-based interactions imposes an increased complexity of systems design on the software engineer. Therefore, declarative means for introducing reactive behaviour, in particular for specifying coupling modes should be introduced. In general, distributed event-based architectures lack adequate support by current software development methods.

While an integrated approach may concentrate exclusively on the property of atomicity, the properties of consistency, isolation, and durability require particular attention as well. We need to define what these properties mean in an integrated context, and how the responsibility for assuring these properties is distributed or shared between the involved components: the (conventional) transaction service, the messaging service, integrated databases and resource managers, and the application objects.

The paper by Prochazka discusses the EJB transaction model and identifies a number of weaknesses of it. The weaknesses stated fall in two categories. On the one hand, there are weaknesses that are inherited from existing transaction systems that EJBs need to collaborate with. On the other hand, the paper questions the appropriateness of conventional transactions for (EJB) component architectures and presents EJB-specific weaknesses.

Most notably, the paper argues for more application-side flexibility and control of transactions, which could lead to conflicts with the concept of declarative transaction management of container-managed transactions. "Bourgogne transactions" introduce a new API, which suggests the preference of an explicit transaction management model as opposed to a (more restricted) container-managed one. More research is needed to understand the differences between explicit and declarative transaction management in detail, and how to employ each of them (or, their combination) effectively for component architectures.

Bourgogne transactions support the explicit management of visibility sets and delegation of object ownership at the Bean level. More research is needed, in how far the granularity of a Bean is appropriate for weakening isolation and to which extend the ab-

straction of a data object as presented in ACTA is compatible with the notion of an EJB component.

In general, more practical experience as well as conceptual work is needed in the complex field of engineering concurrency control in component based distributed and heterogeneous systems.

Acknowledgements

This paper is based in part on the comments of the following workshop attendees: Wolfgang Emmerich, Kostas Kontogiannis, Marek Prochazka, David Rosenblum, Jörn-Guy Süß, and Stanley Sutton.