A NEW APPROACH TO IMPLEMENT EXTENDED TRANSACTION MODELS IN J2EE

Xiaoning Ding, Xiangfeng Guo, Beihong Jin, Tao Huang
Institute of Software, Chinsese Academy of Sciences,China

Keywords: Extended Transaction Models, J2EE, Entity Beans.

Abstract: Extended transaction model (ETM) is a powerful mechanism to ensure the consistency and reliability of complicated enterprise applications. However, there is few implementation of ETM in J2EE. The existing research is deficient in supporting range and requires some special database supporting. This paper explores the obstacle which prevents J2EE from supporting ETMs, and argues it is because of the limitation of J2EE XAResource interface and underlying databases. To overcome the obstacle, we propose a new approach, which processes concurrency control inside J2EE application server instead of in database. Furthermore, we implement TX/E service in JBoss to validate the approach, which is an enhanced J2EE transaction service supporting extended transaction models. Compared to existing work, TX/E supports user-defined transaction models and does not require any special database supporting.

1 INTRODCTION

The traditional and most widely used transaction model, known as Flat model, has been proven powerful in On-Line Transaction Processing (OLTP) systems, such as banking and airline reservation. Flat model has a strict ACID (Atomicity, Consistency, Isolation and Durability) constraint. However, its ACID constraint is too strict. Most complicated enterprise applications in nowadays are collaborative, loosely coupled or long-running. When used for these applications, Flat model has been found some serious problems (Mohan 1994).

Numerous extensions to Flat model have been proposed to solve the problem, referred to as extended transaction model (ETM) or advanced transaction model (ATM). Each ETM is designed for a specific application domain, with a relaxed ACID constraint. Some famous ETMs include Nested, Join/Split, MMT, Pro-Motion, and etc.

J2EE is a primary middleware platform for enterprise applications. And most applications running on J2EE are complex and critical, so it is significant to support ETMs in J2EE. However, no commercial J2EE application server can support ETMs. The existing research of ETM was mostly in database and workflow systems, and little was made in J2EE. Furthermore, the little existing research in J2EE also has some serious limitations, such as the supporting range is limited, or some special database supporting is required (Marek 2002).

In this paper, we explore the implementation mechanism of ETMs in J2EE. We put forward a new approach, which processes concurrency control inside J2EE application server instead of in underlying databases. The new approach does not require any special database supporting.

The rest of this paper is organized as follows: section 2 reviews the background and analyzes the obstacles; section 3 discusses our approach; section 4 presents TX/E, the prototype system of our approach; section 5 reviews the related works; and we conclude the paper in the last section.

2 BACKGROUND

2.1 Extended Transaction Model

The traditional Flat model only supports three basic methods (or control operations, events): *Begin*, *Commit* and *Rollback*. All data operations are bracketed by these three methods. An ETM usually defines some additional methods. For example, Join/Split model supports two extra methods: *Join* and *split*. Each extra method has its specific syntax and semantics.

Each ETM is designed for a specific application area where the traditional ACID property is too strict. To achieve the efficiency and flexibility, the ETM relaxes the ACID property of the transaction in different dimensions and to different extents.

Ding X., Guo X., Jin B. and Huang T. (2006).
A NEW APPROACH TO IMPLEMENT EXTENDED TRANSACTION MODELS IN J2EE.
In *Proceedings of the Eighth International Conference on Enterprise Information Systems - DISI*, pages 118-123
DOI: 10.5220/0002496701180123

To relax ACID property, ETM requires a customized scheduling. In other words, it requires adding specific semantics to scheduling. It is necessary to set some flexible accessing roles on the data items.

For example, in Flat model, if a data item is modified by a transaction instance, then it can not be accessed by any other instances until the holder instance terminates (committed or rolled-back). The accessing role is simple.

However, let's look at the case of ETM. In Nested model, after a child transaction instance begins, it should be able to view the data items which are locked by its parent instance. In this example, the role is more complex: except the lock holders, some instances (its children instances) are allowed to view the data items, while some other instances are not allowed to.

In conclusion, to implement various ETMs, the transaction manager should provide a mechanism so that the ETMs can set their necessary flexible accessing roles.

2.2 J2EE Transaction Architecture

J2EE transaction service complies with JTS and JTA specifications. Its architecture is coming from CORBA OTS and X/Open DTP model.

In the architecture, clients (including stand-alone Java Application and Session Bean) access the underlying databases through the native database interface, referred to as JDBC (Java Data Base Connectivity) interface, while J2EE transaction manager accesses the databases through XAResource interface. Both interfaces are implemented by the database vendors. Another way client access database is entity bean. Entity bean is an object-style encapsulation to the relational-data from underlying database, and it is synchronized to database through JDBC.

Figure1 shows the J2EE transaction service architecture.

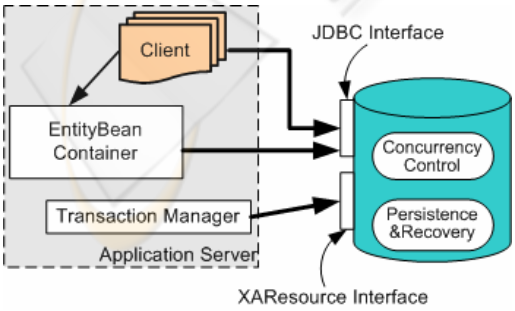

Figure 1: J2EE transaction service architecture.

The XAResource interface is a simple java mapping of X/Open XA interface. In essential, it is just a set of functions.

According to JTS and JTA specification, J2EE transaction manager is only responsible for managing transaction identifiers and transaction branches. The transaction manager performs its work through the XAResource interface, such as *start(Xid), prepare(Xid), commit(Xid).* J2EE transaction manager is not aware of the detailed accessed data and its concurrency control in the underlying databases. And all of its work is just issuing instructions to databases through XAResource interface. i.e., the underlying database is a black-box to J2EE transaction manager.

2.3 Discussion

Unfortunately, XAResource interface does not include any functions to set the flexible accessing roles. For example, J2EE transaction manager have no way to inform the underlying database that a child instance should inherit all locks of its parent instance. It is also why the exiting research of ETM in database can not be applied to J2EE directly. In Data Base Management System (DBMS), we can invoke lock manager to set the roles. However, as mentioned above, we have no similar ability in J2EE.

A straightforward way is to extend XAResource interface so that it can pass any advanced concurrency control instructions as we need. However, it still can not work even we extend the interface. Since in nowadays most commercial databases only support Flat transaction model, and do not support any advanced instructions. It is meaningless to define an interface that few databases can support it.

In conclusion, the difficulty of implementing ETM in J2EE roots in the limitation of XAResource interface and the underlying databases.

Some existing research was also troubled by this obstacle. For example, in Bourgogne (Marek 2000), a framework supports ETMs in J2EE, the system provides an concurrency control instruction (called *primitive*) *delegate*. The primitive is used to transfer uncommitted data objects from one transaction instance to another instance. However, since the limitation mentioned above, there is no way to inform databases the role. Thus the primitive *delegate* can only be used between two transaction branches with the same Xid. It is thought to be difficult to break through this obstacle under the current commercial databases (Marek 2002).

3 SOLUTION

3.1 Approach

As the J2EE official Object-Relational Mapping (ORM) framework, Entity Bean is widely used in recent years. It provides a "pure" object-oriented paradigm and reduces the programming code.

Entity beans are managed by EJB container. EJB container supplies the service of persistence, query, concurrency control, etc. The concurrency controls performed on entity beans is simple, usually a thread-lock policy, i.e., when an entity bean is accessed by a thread, it is locked and no other threads can access it until the holder thread release the lock.

While we face the limitation of XAResource interface and underlying databases, how about we implement primitives inside J2EE application server by enhancing the concurrency control of entity beans?

If we process the concurrency control inside J2EE application server, we can avoid the above two difficulties: 1) The flexible concurrency control is implemented inside J2EE application server, and we do not have to extend the XAResource interface to pass through these instructions 2) The underlying databases are not required to support ETMs and just responsible for the data persistence and recovery.

Our approach is based on this idea. We restrict entity beans as the only entrance for data accessing and enhance its concurrency control. The work can be divided into three parts:

1. J2EE transaction manager provides a set of primitives; users invoke primitives to set the desired accessing roles.
2. J2EE transaction manager maintains those roles, and some necessary data structures, such as lock table, log.
3. When clients access entity beans, we perform concurrency control based on these roles, instead of a simple thread-lock policy.

The difference between two architectures is illustrated as figure 2.

3.2 Benefits and Side-effects

In the new approach, all primitives are implemented inside J2EE application server. It does not pass through XAResource interface and does not require support of databases. Thus we overcome the obstacle.

Another benefit brought from our approach is the ability to detect global deadlock. Most commercial DBMS use locking as the concurrency control mechanism. It can detect a local deadlock in the wait-for graph and abort one of the deadlock transactions. However, if a global transaction in J2EE involves two or more different heterogeneous databases, such as Oracle and DB2, each DBMS transaction manager is only able to view its own local wait-for table and cannot detect the global deadlock. So it has to wait until one of the transaction instances became timed-out. It is one of the serious shortcomings in current J2EE transaction service. After the concurrency control was upgraded into J2EE application server, J2EE transaction manager is able to control the global wait-for table and detect the global deadlock.

The new approach also has some shortcomings. After the entity beans are restricted as the only entrance, accessing underlying databases through JDBC directly is forbidden if the application client is willing to employ an ETM. It may sacrifice some efficiency in certain situations. However, it is not a serious problem since the Entity Bean is efficient and powerful nowadays.

4 TX/E SERVICE

To validate the approach proposed in last section, we implement TX/E service in JBoss (Marc 2003), a famous open source J2EE application server. TX/E is an enhanced J2EE transaction service and provides all interfaces required by JTA and JTS specification. So it is compatible with the standard J2EE transaction service.

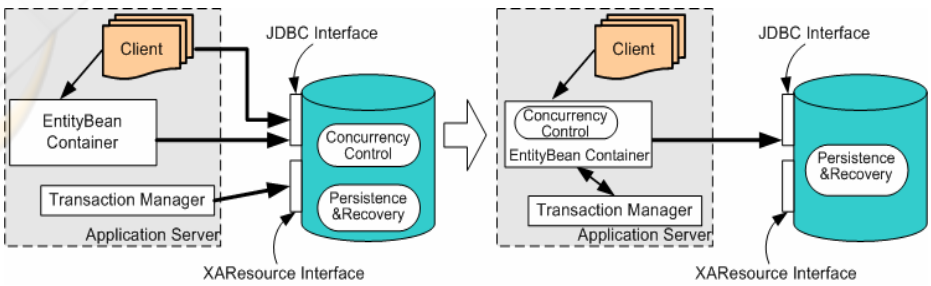

Figure 2: Evolution of the architecture.

4.1 Architecture

The TX/E service is organized in an extensible manner. Each ETM is implemented as an add-on module, named ETM adapter, and can be easily plugged into the system. The system provides a small set of primitives as the kernel, and each ETM adapter is a properly composition of the primitives. This architecture style (pluggable adapters and a set of primitives) has lots of advantages and was employed in several previous ETM frameworks, such as RTF (Roger 1995).

The ETM adapter is a java class inherited from the appointed class. We classify the users in the TX/E system into two categories: model designer and application client. The ETM adapter class is implemented by model designer, which defines the methods set and the attributes set of the ETM. The application client uses these ETM adapter classes to work.

The kernel, primitives set, is implemented inside TX/E transaction manager. The ETM adapter calls primitives to set the accessing roles on the entity beans.

4.2 Programming Interface

All of the ETM adapter classes inherit from the base class *AbstractModel*. As the root of the inheritance tree, the *AbstractModel* defines the elements which are common to every transaction model. For example, it defines some properties such as Xid and time-out. The *AbstractModel* also defines three abstract methods: *begin, commit* and *rollback*. Being abstract methods, these three methods are guaranteed to be implemented in every inherited class by Java language, i.e., every transaction model in TX/E supports these three methods.

All primitives are provided in class *Primitive*, which is organized as an inner class named *m_Primitive* in class *AbstractModel*. To ensure the consistency, *m_Primitive* is declared as *protected*. Thus, model designers have the privileges to invoke the primitives in the derived ETM adapter class, while the application clients are not able to invoke the primitives directly.

Figure 3 shows the class hierarchy after deploying three ETMs: Flat, Nested and Join/Split.

After compilation, each ETM adapter class is deployed to the specific directory with a configuration file. The TX/E transaction manager loads all ETM adapter classes during start-up, and binds each class to the JNDI name described in the corresponding configuration file. The application client gets the ETM adapter class reference using JNDI, and instances it to create an ETM transaction instance.

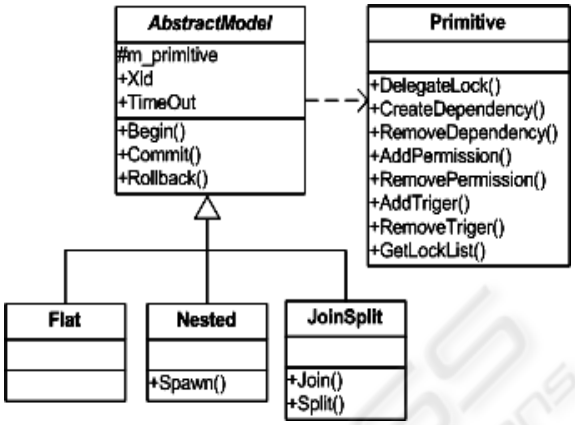

Figure 3: Class Hierarchy.

4.3 Primitives Set

The capability of TX/E service is determined by the primitives set. Primitives must be carefully selected so that the ETM adapter has enough flexibility. Fortunately, there are lots of research on which primitives should be provided (Briliris 1994, Roger 1995). In general, there are four kinds of primitives should be provided:

- Manage the life-cycle of transaction instances, such as creating an instance of a certain ETM type and terminating current instance.
- Manage the control flow of transaction instances, such as the dependency between different methods.
- Manage the conflict and visibility.
- Manage the responsibility for modified data objects.

Some important primitives defined in TX/E include *CreateDependency, RemoveDependency, AddPermission, RemovePermission, AddTrigger, removeTrigger, GetLockList, DelegateLock*, and so on.

4.4 Primitives Implementation

In TX/E transaction manager, a lock table records the locking status and waiting queue of every entity bean, and the transaction manager schedules each accessing using a basic Two-Phased Locking (2PL) algorithm.

All primitives are implemented by J2EE transaction manager. Each primitive call lock manager to set the accessing roles, and calls log manager to do some necessary logging. For example, the *DelegateLock* primitive is implemented as following:

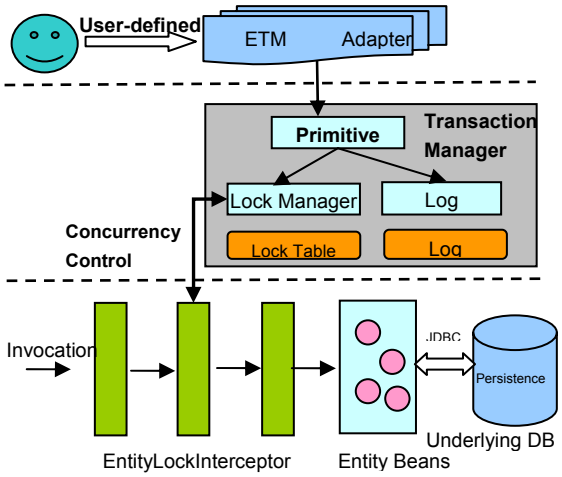

Figure 4: Implementation Architecture.

1. Get the Xid of current transaction instance from *AbstractModel*.
2. Query the lock manager, and get all lock items which holder is the Xid of current transaction.
3. Replace each lock item's holder to the Xid of target transaction instance.

By calling primitives, each ETM Adapter can set the detailed accessing roles as they want.

After the accessing roles have been set, we need to enhance the concurrency control of entity beans so that we can schedule each invocation according to the roles.

The JBoss EJB Container uses a Pipe and Filter architecture style, and each service function is implemented as an interceptor, such as logging, security and transaction. Every invocation passes the interceptor chain in turn. Its original concurrency control is just a thread-lock policy. We modify the existing interceptor *EntityLockInterceptor* of JBoss. In this interceptor, we query J2EE transaction manager to get the judgment to the fate of each access, such as granting locks, blocking the instance for locks, or aborting the instance.

The detailed architecture is illustrated in figure 4.

5 RELATED WORK

The traditional research on ETM framework is usually based on database system. ACTA (Panayiotis 1994) is a general framework for specifying the structure and the behaviour of transactions as well as for reasoning about the concurrency and recovery properties of the ETMs. ACTA is such a classic framework that nearly become a standard to formal specification and analysis of ETMs. Many concepts of ETM are introduced and discussed in ACTA, such as *dependency*, *delegation*, and *permission*. Most primitives in following research were designed according to these concepts. But unfortunately, ACTA did not discuss the approaches to implement ETMs.

ASSET (Briliris 1994), provided in ODE database system, presents a set of primitives and explains how to implement different ETMs upon these primitives. To some extent, ASSET can be regarded as a implementation of ACTA. ASSET does not use the object-oriented paradigm for the proposed primitives; it rather uses procedural programming style. And, ASSET was based on database system and no implementation was provided.

Another way to build ETM framework is proposed in (Anwar 1996). It demonstrates how database event-condition-action (ECA) rules can be used to support extended transaction models.

Reflective Transaction Framework (Roger 1995, Roger 1996) is another framework to support ETMs, and it is built on Transaction Processing Monitor (TP Monitor). RTF allows the extension of both functionality and interface of a conventional TP Monitor to implement advanced transaction models. The framework depends on the internal structure of the TP Monitor, such as the lock manager and log manager.

A practical framework on supporting ETMs in J2EE is Bourgogne (Marek 2002, Marek 2000). It extends the deployment descriptors set of EJB and supplies several fixed primitives based on the extension. The users can develop a few ETMs upon these primitives.

However, Bourgogne still processes the concurrency control through XAResource interface. And the framework has the following limitations: 1) some primitives defined in Bourgogne such as *Delegate* can only be used between two transaction branches with the same Xid. 2) Bourgogne does not support defining new methods. So users can only create dependencies between three fixed methods, i.e., *Begin*, *Commit* and *Rollback*, which restrict its supporting range and some flexible ETMs can not be implemented. The author realized the limitation, and argued that a database with special API supporting have to be employed (Marek 2002).

There are also some implementations of a specific ETM in J2EE such as (Jinling 2004). But they are tailored for a specific ETM, not a general framework.

J2EE Activity Service for Extended Transactions is a newly proposed specification and is still in discussion. It is a coordination framework that supports some ETMs. However it is composed of several individual atomic transactions. It can not

handle the concurrency and permission issues inside an atomic transaction instance.

6 CONCLUSION

Extended transaction models are urgently needed in J2EE to support complex enterprise applications. However, there is little research in J2EE. We studied the implementation of ETMs in J2EE in this paper.

Our contributions lie in three aspects. First, we analyses the obstacle which prevents J2EE from supporting ETMs, and argue that it is the limitation of XAResource interface and the underlying database.

Second, we propose a new approach, which handles concurrency control inside J2EE application server instead of in underlying databases. In the newly proposed approach, all primitives are implemented by J2EE transaction manager, and does not need any special APIs of underlying database. The new approach discards the XAResource interface, and databases are only responsible for persistence and recovery. We also discuss the possible side-effects brought by the new approach.

Finally, to validate the approach, we implemented an enhanced J2EE transaction service TX/E in JBoss. Compared to existing work, TX/E service supports user-defined transaction models and does not require any special support of databases.

REFERENCES

C. Mohan, 1994. Tutorial: Advanced Transaction Models Survey and Critique, In *ACM SIGMOD International Conference on Management of Data*.

Marek Prochazek, 2002. *Advanced Transactions in Component-based Software Architecture*, PhD thesis.

Marek Prochazka, 2000. Advanced Transactions in Enterprise JavaBeans. *In Proceedings of the EDO 2000 Workshop*.

Marc Fleury, Francisco Reverbel, 2003. The JBoss Extensible Server, In *Proceedings of the 2003 ACM/IFIP/USENIX International Middleware Conference*.

Panayiotis K.Chrysanthis, Krithi Ramamritham, 1994. Synthesis of Extended Transaction Models using ACTA, *ACM Transactions on Database Systems*.

Briliris, A., S.Dar N Gehani, H.V.Jagadish, K.Ramamritham, 1994. ASSET : A System for Supporting Extended Transactions, In *Proceedings of the 1994 ACM SIGMOD International Conference on Management of Data*.

Anwar, E., Chakrawarthy, S., 1996. Viveros, M., An Extensible Approach To Realizing Advanced Transaction Models, In *Proceedings of the International Workshop on Advanced Transaction Models and Architectures*.

Roger Barga, Calton Pu, 1995. A Practical and Modular Method to implement extended transaction Models, In *Proceedings of International Conference on Very Large Data Bases*.

Roger Barga, Calton Pu, 1996. Reflection on a Legacy Transaction Processing Monitor, In *Proceedings of the Reflection'96 Conference*.

Jinling Wang, Beihong Jin, Jing Li, 2004. A Transaction Model for Long Running Business Processes, In *Proceedings of the 6th International Conference on Enterprise Information Systems*.