Advanced Transactions
in Component-Based Software Architectures
Ph.D. thesis

Marek Procházka

Distributed Systems Research Group
Charles University, Prague, Czech Republic

http://nenya.ms.mff.cuni.cz/
Goals of the Thesis

• Focused on transaction processing in component-based software architectures

• A proposal for
  – A flexible way for the transaction propagation policy specification
  – Specifying transaction propagation policy as a part of the component interface
  – Support for advanced transaction models
  – Proof of the concept: Prototype implementation
A Component-Based Application Example
Container-Interposed Transactions: An Example

Diagram showing the interaction between a Client and a Server through a Transaction Manager and a Container. The diagram illustrates the flow of transactions and contexts between the different components.
Container-Interposed Transactions

• Client transactions
  – Explicit demarcation

• Container-demarcated transactions
  – Declarative transaction management

• Component-demarcated transactions
  – Not considered in this thesis
Analyzing the EJB (COM+, CCM) Approach

- Flat Transaction model only

- Transaction propagation policy specified in single transaction attribute in the component deployment descriptor
  - Several transactional factors encoded into single value
    - Client transaction context handling
    - A new (container-demarcated) transaction creation
    - Exception throwing
    - Relation among the client and container-demarcated transaction (not considered in EJB, COM+, CCM)
  - Hard-to-comprehend transaction propagation policy specification
The Proposal: Bourgogne Transactions

• A new transaction model dedicated to the environment of container-interposed transactions

• Comprises
  – Transaction propagation policy specification as a part of the component interface
  – Support for advanced transaction models
  – Employing advanced transaction models in container-demarcated transactions
  – Specification of transactional API used by both clients and containers
Bourgogne Transactions: Transaction Model

• Basic significant events: *begin*, *commit*, *abort*

• Advanced significant events – invocation of advanced transaction primitives reflecting the ACTA building blocks:
  
  – **Dependencies**: flow control bindings among transactions
  
  – **Sharing components**: allowing other transactions to access components acquired by a transaction

  – **Delegation**: transferring the responsibility to commit or abort performed operations
Bourgogne Transactions: How Does It Work

1. Obtaining \texttt{tx} interface to the Transaction Manager by means of a naming service:
   \begin{verbatim}
   tx = lookup("user transaction");
   tx.begin();
   ...
   \end{verbatim}

2. Obtaining Bourgogne transaction factory:
   \begin{verbatim}
   btxf = tx.getBourgogneTransactionFactory();
   \end{verbatim}

3. Obtaining Bourgogne transaction:
   \begin{verbatim}
   btx = btxf.getBourgogneTransaction();
   \end{verbatim}

4. Employing advanced transaction primitives:
   \begin{verbatim}
   btx.addDependency(...);
   \end{verbatim}
Bourgogne Transactions: Dependencies

- **Predefined dependencies**
  - 12 basic dependencies

  ```java
  btx1.addDependency(btx2, dependencyType);
  ```

- **User-defined dependencies**
  - Registering synchronization objects to transaction basic significant events

  ```java
  btx1.registerSynchro(synchroObject, synchroTime);
  ```

  - Synchronization time: **BeforeBegin, Begin, BeforeCommit, Commit, BeforeRollback, Rollback**
Explicit giving permissions to invoke acquired components’ methods

```java
btx1.addPermission(btx2, componentObject, method);

btx1.addPermission(btx2, componentObject);

btx1.addPermission(btx2);
...
```
Bourgogne Transactions: Component Delegation

• Transferring a component from a transaction to another transaction

\[ \text{btx1.delegate(btx2, componentObject);} \]

• Operations considered as they have been invoked by \text{btx2}

• The responsibility for commit or abort is transferred to from \text{btx1} to \text{btx2}
Transaction Propagation in EJB (COM+, CCM)

• Single attribute specified in the component deployment descriptor

• Several transactional factors encoded into single value
  – Client transaction context handling
  – A new (container-demarcated) transaction creation
  – Exception throwing
  – Relation among the client and container-demarcated transaction

• Hard-to-comprehend transaction propagation policy specification
Analyzing the EJB (COM+, CCM) Approach (cont.)

<table>
<thead>
<tr>
<th></th>
<th>Client transaction</th>
<th>Client transaction handling</th>
<th>Container-demarcated transaction</th>
<th>Exception throwing</th>
<th>EJB 2.0 transaction attribute value</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>No</td>
<td>N/A</td>
<td>NotCreated</td>
<td>IfClientTx</td>
<td>Never</td>
</tr>
<tr>
<td>2</td>
<td>No</td>
<td>N/A</td>
<td>NotCreated</td>
<td>IfNotClientTx</td>
<td>Mandatory</td>
</tr>
<tr>
<td>3</td>
<td>No</td>
<td>N/A</td>
<td>NotCreated</td>
<td>Never</td>
<td>NotSupported, Supports</td>
</tr>
<tr>
<td>4</td>
<td>No</td>
<td>N/A</td>
<td>Created</td>
<td>IfClientTx</td>
<td>not considered</td>
</tr>
<tr>
<td>5</td>
<td>No</td>
<td>N/A</td>
<td>Created</td>
<td>IfNotClientTx</td>
<td>N/A</td>
</tr>
<tr>
<td>6</td>
<td>No</td>
<td>N/A</td>
<td>Created</td>
<td>Never</td>
<td>Required, RequiresNew</td>
</tr>
<tr>
<td>7</td>
<td>Yes</td>
<td>Propagated</td>
<td>NotCreated</td>
<td>IfClientTx</td>
<td>N/A</td>
</tr>
<tr>
<td>8</td>
<td>Yes</td>
<td>Propagated</td>
<td>NotCreated</td>
<td>IfNotClientTx</td>
<td>Mandatory</td>
</tr>
<tr>
<td>9</td>
<td>Yes</td>
<td>Propagated</td>
<td>NotCreated</td>
<td>Never</td>
<td>Supports, Required</td>
</tr>
<tr>
<td>10</td>
<td>Yes</td>
<td>Propagated</td>
<td>Created</td>
<td>IfClientTx</td>
<td>N/A</td>
</tr>
<tr>
<td>11</td>
<td>Yes</td>
<td>Propagated</td>
<td>Created</td>
<td>IfNotClientTx</td>
<td>not considered</td>
</tr>
<tr>
<td>12</td>
<td>Yes</td>
<td>Propagated</td>
<td>Created</td>
<td>Never</td>
<td>not considered</td>
</tr>
<tr>
<td>13</td>
<td>Yes</td>
<td>Suspended</td>
<td>NotCreated</td>
<td>IfClientTx</td>
<td>N/A</td>
</tr>
<tr>
<td>14</td>
<td>Yes</td>
<td>Suspended</td>
<td>NotCreated</td>
<td>IfNotClientTx</td>
<td>not considered</td>
</tr>
<tr>
<td>15</td>
<td>Yes</td>
<td>Suspended</td>
<td>NotCreated</td>
<td>Never</td>
<td>NotSupported</td>
</tr>
<tr>
<td>16</td>
<td>Yes</td>
<td>Suspended</td>
<td>Created</td>
<td>IfClientTx</td>
<td>N/A</td>
</tr>
<tr>
<td>17</td>
<td>Yes</td>
<td>Suspended</td>
<td>Created</td>
<td>IfNotClientTx</td>
<td>not considered</td>
</tr>
<tr>
<td>18</td>
<td>Yes</td>
<td>Suspended</td>
<td>Created</td>
<td>Never</td>
<td>RequiresNew</td>
</tr>
</tbody>
</table>
• Two-attribute approach to the transaction propagation policy specification
  – Invocation with No client Transaction (NT)
    ThrownException
    DoNothing
    CreateNew
  – Invocation in a Client Transaction (CT)
    ThrownException
    Suspend
    Propagate
    SuspendAndCreateNew
    Advanced
Transaction Propagation Policy: When To Specify?

- EJB, COM+, CCM: Specifying in the component deployment descriptor
  - Transaction propagation policy not known at the time of coding the component
  - A client is not able to determine transaction propagation policy from the component interface

- NT&CT: Specifying in the component interface
  - The author of the component code is responsible for specifying transaction propagation policy
  - A client should determine transaction propagation policy from the component interface
  - Hiding some NT&CT attributes for client if security is an issue
interface BankAccount {
    int getBalance() {
        NT: DoNothing;
        CT: SuspendAndCreateNew;
    }
    void withdraw(int iAmount) {
        NT: CreateNew;
        CT: Propagate;
    }
    void deposit(int iAmount) {
        NT: CreateNew;
        CT: Propagate;
    }
}
• Advanced CT attribute composed of *subattributes* (specifying relations between client and container-demarcated transactions)
  – Dependencies
    ClientDependency
    CdtDependency
  – Sharing components
    ClientPermissions
    CdtPermissions
  – Delegation
    ClientDelegate
    CdtDelegate

• A particular combination of subattribute values reflect a used transaction model
<table>
<thead>
<tr>
<th>CT subattribute</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>CdtDependency</td>
<td>None, a set of methods of the same interface</td>
</tr>
<tr>
<td>ClientPermissions</td>
<td>None, All</td>
</tr>
<tr>
<td>CdtPermissions</td>
<td>None, a set of methods of the same interface</td>
</tr>
<tr>
<td>ClientDelegate</td>
<td>None, All</td>
</tr>
<tr>
<td>CdtDelegate</td>
<td>None, BeforeCommit, BeforeAbort, BeforeCompletion</td>
</tr>
</tbody>
</table>
CT Subattributes: An Example

• Employing Nested Transactions:

```c
void withdraw(int iAmount) {
    NT: CreateNew;
    CT: Advanced {
        ClientDependency = CommitDependency;
        CdtDependency = WeakAbortDependency;
        ClientPermissions = All;
        CdtPermissions = None;
        ClientDelegate = None;
        CdtDelegate = BeforeCommit;
    }
}
```
BT in Generalized Component-Based Architectures

• Multiple provides- and requires-interfaces

• Hierarchical components

• Component ties:
  – *Binding* of requires- to provides-interface
  – *Delegation* of provides-interface to a nested component provides-interface
  – *Subsumption* of requires-interface to a parent component requires-interface
Multiple Provides-Interfaces

• Example 1: a component with two provides-interfaces

```java
interface BankAccount {
    int getBalance();
    int getHighestBalance();
    void withdraw(int iAmount);
    void deposit(int iAmount);
}

interface ModifyAccount {
    int addMoney(int iAmount);
    int removeMoney(int Amount);
}
```
Multiple Provides-Interfaces (cont.)

- Example 2: another component with two provides-interfaces

```java
interface BankAccount {
    int getBalance();
    int getHighestBalance();
    void withdraw(int iAmount);
    void deposit(int iAmount);
}

interface AccountProperties {
    string getName();
    string getAddress();
    int getMaxInternetWithdraw();
    ...
}
```
Multiple Provides-Interfaces (cont.)

• Solution: dividing each component to *transactional units*

• Rules for transactional units (TU):
  – Each provides-interface is associated with a TU
  – At most one transaction can enter a single TU
  – Concurrent transactions may execute in different TUs

• Example 1: each provides-interface in separate TU
• Example 2: both provides-interfaces in the same TU
Considering Requires-Interfaces: An Example
1. NT&CT attributes used to specify requirements for transaction propagation
   - New *Empty* value
   - A list of acceptable NT&CT attributes

2. Rules for binding:
   - *Functional equivalence* required
   - *Empty* requirements compatible with any NT&CT value
   - Non-*Empty* requires-interface list should contain the NT (CT) attribute value of the provides-interface’s NT (CT) attribute

3. Requires-interfaces of a component are not associated with component’s transactional units
Hierarchical Components: An Example

Client

Request

Response

Server

Container

Provides interface

Requires interface

Top-level component

Subcomponent

C

Subsumption

Delegation

D

A

Component

B

Component

27
Hierarchical Components: Key Questions

1. Does locking of a component lead to locking of all its subcomponents and vice versa? Will transactional units work with hierarchical components?

2. Will transaction propagation be policy specified for subsumption or delegation ties?
Hierarchical Components: Another Example
Hierarchical Components: Yet Another Example
1. There is no lock propagation in hierarchical component architectures
   A component can cross its parent TU boundary

2. Rules for delegation:
   – *Functional equivalence* required
   – Equality of transaction propagation policy required

Rules for subsumption:
   – *Functional equivalence* required
   – *Empty* requirements of nested component compatible with any NT&CT value of the subsumed requires-interface of its parent
   – *Non-Empty* requirements: the nested component’s requires-interface list should be a subset of the parent component requires-interface list
EJB provides the environment for container-interposed transactions

- Components are called beans
- EJB provides a non-hierarchical component model
- Every EJB component contains a single provides-interface and no requires-interface
- Each bean contains always single default transactional unit
- The transaction propagation policy is specified in the bean deployment descriptor
Prototype Implementation in EJB

• Implemented in JOnAS 2.3
  – Part of the PEPiTA/ITEA Eureka project
  – Partners: France Telecom, Bull, Alcatel, INRIA, …

• Inter-transaction dependencies
  – Predefined dependencies
    • Abort-enforcing
    • Commit-enforcing
  – User-defined dependencies
    • Registering synchronization objects to the BeforeBegin, AfterBegin, BeforeCommit, AfterCommit, BeforeRollback, and AfterRollback events
Conclusion

• Proposed a new transaction model dedicated to container-interposed transactions settings

• Bourgogne Transactions
  – Specify the transaction propagation policy using NT&CT attributes
  – Transaction propagation policy as a part of the component interface
  – Transaction propagation and locking technique for generalized components
  – Support for advanced transaction models (both client and container-demarcated transactions)

• Prototype implemented in EJB
Related Work

• ACTA: framework for specifying and reasoning about advanced transaction models

• University of Valenciennes, France: New EJB attributes supporting Nested and Open-Nested transaction models (part of PEPiTA/ITEA)

• Business Transaction Protocol for web services

• CORBA OTS
  – OTSPolicy: Requires, Forbids, Adapts
  – InvocationPolicy: Either, Shared, Unshared
Future Perspectives

- Component-demarcated transactions
- Asynchronous transaction processing
- Extending prototype to support delegation and sharing components
  - Requires advanced persistent store
- Prototype for our SOFA component-based architecture
  - More provides- and requires-interfaces
  - Hierarchical components