UNIVERSITE DE VALENCIENNES
ET DU HAINAUT-CAMBRESIS

Service of advanced transactions for component based model

Version 0.9

Sergiy Nemchenko
Nadia Bennani

Rapport de recherche n°2 Juillet 2003

Université de Valenciennes et du Hainaut - Cambrésis
Le Mont Houy – B.P. 311 – 59304 VALENCIENNES CEDEX
Tél. du secrétariat 03/27/51/19/41 MME ;Aureggi Fax : 03/27/14/18/29

1. Introduction

Recently, the e-commerce is the thriving industry, especially the "Business-to-Business" applications. Transactions improve the reliability of application execution in the case of failures. Flat transaction model supports the ACID properties. It satisfies the requirements of short-lived transactions. However, the execution of long-lived transactions incorporating several remote databases is not easy to manage. Instead, remote resources may keep locked for a long period of time. Extended transaction models allow to solve these problems. Indeed, thanks to the nested transaction (CNT) model, transactions can be split into sub-transactions executed in a parallel manner. The Open Nested Transaction (ONT) model extends CNT model facilities by relaxing isolation property at sub-transaction level making resources less locked by a long-lived transaction. The main disadvantage of ONT model is the need of compensation in the case of parent rollback after a sub-transaction's commitment.

Nowadays distributed applications are often based on component model. This model simplifies the distributed applications creation. The main advantages of the component model are well-defined separation of business logic from service methods and high rate of reusability in the design of the application. However, all existing component specifications provide only flat transaction model.

The goal of this specification is the description of the Transaction Service providing the flat transactions model and open nested transactions model for component-based models. The main problem to solve by this service is the compensation possibility. The compensation mechanism, described in this specification, is based on the compensator-component. This component is aimed to compensate effects of associated business-component. The search of compensator corresponding to the required transaction is performed by means of information saved at the time of transaction commitment. This information is kept into the Compensation Data Object (CDO). The table **`CDOTable`** ensures the saving and the search of required objects CDO.

The description of proposed transactional service begins by brief state of the art of component-based models and transactional models. Then the principle of specified service, which provides the open nested transactions, is described. At the end of this document, the implementation of this model using Fractal-model is presented.

2. Existing technologies

This section briefly reminds implied technologies: component-based model and the models of transactions. Then the transactions in the component-based model conclude this section.

2.1. Component-based models

A component is an autonomous and intelligent software module. It can be executed on different platforms, with different operational systems [OHE99, HEI01]. A component is described by one or more interface(s), by configurable properties (to personalize the component) and by technical constraints (for example, security or transactional needs).

To be used, a component must be deployed on an application server, which may manages several components and proposes technical services such as transaction service, persistency service, security service, etc. The code of a component only contains the business logic. The deployment of the application consists in configuring the data sources, the distribution of tasks on a given execution platform, the policy of security, etc., for a given assembly of components.

We use the FRACTAL framework [Fractal, BCS02] to build and assembly components. A Fractal component allows dynamic composability i.e. a composition of components is seen as a component itself. Moreover, thanks to different kind of controllers, it is possible to monitor the constitution and the bindings of composite components, which are explicit and exposed. Thus, the system has a representation of itself. It also allows one component to be shared by two compositions of components. An implementation of the FRACTAL model, called Julia, is proposed by the Objectweb consortium (http://www.objectweb.org).

2.2. Overview of transaction models

Flat transactions

A transaction is a group of operations launched by the ***begin*** command and accomplished by the ***commit*** or ***rollback*** command. Transactions provide the following properties: Atomicity, Coherence, Isolation, Durability. The flat transactions model is presented in numerous works [OHE99] and is implemented in all transactional monitors available on the market. The model of flat transactions is particularly well adapted to the execution of very short transactions. On the other hand, this model is not convenient for several types of applications [JK97, DHL91]. So, for the execution of long transactions, the transactional system has to face the increase of rollback probability. The model of flat

transactions becomes then particularly penalizing because it imposes to systematically relaunch the totality of treatments.

Nested Transactions

The Closed Nested Transaction model (CNT) [Moss81] introduces the notion of sub-transaction. In this model, the rules of beginning and completion of sub-transactions are the following [OHE99, RMo89] :

- A sub-transaction really commit only when all ancestor have committed.
- All sub-transactions are abandoned if the mother-transaction abandons.

This model proposes several advantages. At first, it facilitates the parallelism between sub-transactions. It also allows a more precise control of failure recovery. Indeed, if a sub-transaction is aborted, it is not necessary to relaunch the totality of the mother-transaction.

However, this model is not adapted for the management of long transactions between remote databases as needed by B2B applications: with such a model, resources are locked from the first beginning to the end of the root-transaction.

Open Nested Transactions (ONT) model has been proposed by J. Gray in 1981 [Gray81]. This model is based on the relaxation of the isolation constraint. Therefore, when a sub-transaction commits, all updates are visible by all the other transactions, without waiting for the validation of the root-transaction.

The ONT model increases the parallelism between transactions. Indeed, resources are available as soon as a sub-transaction is committed and so waiting transactions are blocked for a shorter period. This advantage is particularly interesting when the duration of the transactions becomes important. However, the relaxation of the isolation constraint also has some inconvenience. The main impact of this model is the need of compensation in some situations [KLS90, HW91, AVBSW94], to preserve data coherency, notably when the mother-transaction of an ONT abandons.

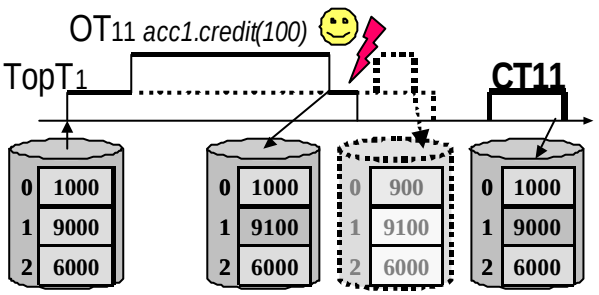

Figure 1: – Compensation example

In figure 1, we describe a simple example illustrating the need of compensation. We consider a database, which manages bank accounts. We need to transfer one hundred dollars from an account C_0 to an account C_1. To preserve database coherency, this operation must be executed in the scope of a transaction. Let us consider that this process is divided into two different stages: first the credit of the C_1 account and then the debit of the C_0 account. A root-transaction *TopT1* is created to assure the transfer. This root-transaction contains the two ONT sub-transactions, the first one for the credit operation (sub-transaction *OT11*) and the other one for the debit operation (sub-transaction *OT12*). Let us proceed to the transfer. The results of the first sub-transaction *OT11* are committed. They are then registered in the database and become visible by all the other transactions executed by the DBMS. Let us consider that a failure of the root-transaction *TopT1* arises before the commitment of the second sub-transaction *OT12*. In this case, the effects of the credit were confirmed in the database, while the effects of the debit have been abandoned. It is necessary to compensate the effects of the sub-transaction *OT11*, to maintain the database in a coherent state.

Compensation operations (*CT11*) of the effects of the open nested transaction (*OT11*) have to be executed.

It should also be noted that this example does not motivate the need of ONT. Our purpose is only to illustrate their use on a simple example.

2.3. Transactions in the component-based models

All industrial component frameworks support only the flat transactions model. These component-based models provide two modes of transaction demarcation: transactions managed by the user or by the container. In the first mode, the user calls the TM's methods. Whereas in the second mode, container (or membrane in Fractal [Fractal]) determines transaction policy by means of transaction attributes fixed in the deployment time. Works have been described in [RMe03] in order to allow transaction policies to be mapped in a transparent way for the application developer to any existing transaction services. Such solution allows to add any new policy.

3. Transaction Service Functionality for ONT

This section presents the mechanism behind the consideration of ONT, in the case of compensation management. This mechanism is based on the component-compensator. Current section includes the presentation of ONT execution and ONT compensation process.

3.1. Compensator

The main problem of ONT model is the committed transaction compensation possibility. The solution of our model is based on the concept of the compensator component. A compensator is a software module aimed to the compensation of committed ONT. It receives the descriptor of compensation process and executes the corresponding operations. This specification proposes a compensator as a component. Every business-component supporting ONT must to have an associated compensator. For this purpose, the business-component saves the identifier of corresponding component-compensator and transmits it to the transaction service in the case of ONT execution. The transaction service saves this identifier and uses it in the case of compensation necessity to search the required compensator.

A component-compensator possesses methods corresponding to the compensation of each compensable method of business component. The signature of business method and compensation method are identical. A special method starting the compensation process is provided by the interface of each compensator. This method receives information, which allows to find required compensation-method and to execute them. The parameters of compensation operation are generated according to the business-component method invoked by the client.

3.2. Search of required compensators

The previous section describes the compensator. The business-component gives the identifier of the compensator to the transaction service in the case of the ONT execution. Moreover, the parameters of compensation operation are generated in the time of ONT execution. All this information is required in order to manage the compensation. This section presents the descriptor object keeping necessary information. It is named CDO (for Compensation Data Object). The content of this object is described in the beginning of this section and the mechanisms, which allow the use of the CDO objects follows.

3.2.1. CDO descriptor

The CDO descriptor (for Compensation Data Object) is aimed to two purposes:

- Search of compensator corresponding to the required transaction;
- Description of compensation operation.

The first goal is provided by two fields of CDO: an identifier of transaction and an identifier of the compensator. In this way, a transaction is associated with a compensator. Moreover, as only one compensator can be associated with one transaction, the identifier of transaction is used as identifier of CDO also. These parameters enable to find the required compensator. After the finding of required compensator, the compensation of corresponding operation can be run. For this process, the compensator requires to know the method of compensation and its execution parameters. Two fields, containing this information are added to the CDO descriptor for this need.

The final list of descriptor fields is follow:

- ONT's identifier;
- Compensator's identifier;
- Compensation method's signature;
- List of values of compensation method.

3.2.2. The CDO table

The CDO allows to associate the committed transaction with compensator and to describe the required compensation process for the compensator. The mechanism of CDO storage and search are necessary. The table **`CDOTable`** allows to store the CDO objects of committed ONT. This table provides the possibilities to add and to delete the CDO objects. The CDO objects are recognized by the transaction identifier saved inside. A CDO search by transaction identifier is provided by **`CDOTable`**. Thus, this table allows to find the object CDO for required ONT. The list of transaction ONT, which must be compensated, is generated by means of list **`ChildrenONT`**. This list is provided by each transaction. **`ChildrenONT`** contains the identifiers of ONT committed in the scope of current transaction. The ChildrenONT object provides methods to add one or more transactions identifiers, and a method to show its content.

3.2.3. Mechanism of information saving

The process of compensation preparation consists of generation and saving of information necessary for the compensation. The CDO descriptor has to be generated at the time of ONT beginning or commitment, depending on implementation. The information required for generation of CDO object is given by the container (or membrane, in the terminology of Fractal). The transaction manager adds the identifier of transaction ONT to the generated CDO. Then, in the time of transaction commitment (CNT or ONT) operations are done for saving the required information.

Two operations must be achieved at ONT commitment. The first saves the object CDO of committed ONT to the **`CDOTable`**. The second operation saves the identifier of committed ONT to the **`ChildrenONT`** of parent transaction. If this ONT is top-level transaction, then only the saving of CDO is required for the compensation prevision. The commitment of CNT sub-transaction calls the copying of **`ChildrenONT`** contents to the **`ChildrenONT`** of parent transaction. These mechanisms allow to save all information required to the execution of compensation process in the case of transaction rollback.

3.3. Generic model

The component-based application supporting the transactions is composed at minimum of two elements: 1) the applicative components and 2) the transaction service. In the case of ONT support, the applicative component is composed of business-component and component-compensator. The applicative-component uses the transaction service for management of transactions. Transaction service providing the ONT model is composed by:

- Transaction manager;
- Compensation manager;
- Table, keeping the compensations' descriptors (CDOTable).

These elements use two classes of objects:

- Descriptor of compensation (CDO for Compensation Data Object);
- and Identifier of transactions (TxId).

The figure 2 presents the collaboration between these elements.

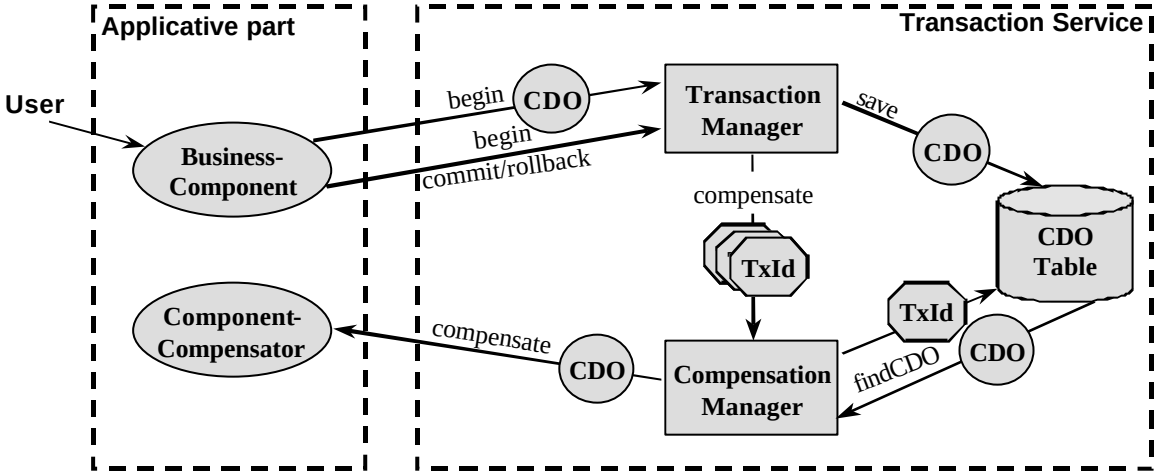

Figure 2: Component-based application using the ONT model

Business-Component

The *business-component* without any non-functional code contains only business-logic application.

Component-Compensator

The *component-compensator* contains the compensation-methods associated to the *business-component* methods and the method of compensation managment. It can be called only by *Transaction Service*. Every *business-component* supporting the ONT has one associated *component-compensator*.

CDOTable

The *CDOTable* saves the descriptors of compensations. Each descriptor is represented by *Compensation Data Object*. *CDOTable* provides the methods of saving the new objects and search of saved objects.

Transaction Manager

The *transaction manager* provides the operations necessary for the transactional demarcation and control of concurrency transactions. In addition, it prepares the possible

compensations of committed ONT and detects the necessity of ONT compensations for the ONT management.

Compensation Manager

The *compensation manager* is called by *transaction manager* in the case of compensations need. The *compensation manager* receives the identifiers of transactions to compensate and then it prepares the compensation process and launches it.

Compensation Data Object

The object *CDO* (for *Compensation Data Object*) describes the operations of the transaction compensation. Each *CDO* is associated with one transaction of type ONT.

Transaction Identifier

The object *Transaction Identifier* represent unique identifier of transaction or sub-transaction. It is generated by *Transaction Manager* at the transaction beginning. Moreover, this object identifies the objects CDO because each CDO is associated only this one transaction or sub-transaction.

3.4. Compensation managment

Execution without failure

There are two possibilities of new transaction (or sub-transaction) beginning. The *Transaction Manager* provides the methods ***begin()*** and ***begin(cdo)***. The call of command ***begin()*** implies the creation either a) of new top-level transaction or b) of new close nested transaction. The first case corresponds to the call outside of a transactional context and second case corresponds to the call within the scope of a transaction. The execution of the command ***begin(cdo)*** is similar to the first command execution. The difference is in the type of created transaction or sub-transaction. This command creates the open nested transactions. The received object CDO is saved into created transaction. This object saves the identifier of this transaction among the other information.

The commitments of transaction and sub-transactions are activated by command ***commit()*** of *Transaction Manager*. The completion of CNT and root-transactions is described by the OTS specification. The accomplishment of ONT sub-transaction definitely commits the modification realized by this sub-transaction.

Each transaction possesses the list (named ***ChildrenONT***) of ONT-children, which are committed in the scope of this transaction. This list is used in the search of ONT to compensate in the case of rollback of current transaction. At commitment, the identifier of the ONT sub-transaction is saved in ***ChildrenONT*** of its parent-transaction. The commitment of close nested sub-transaction adds the contents of its ***ChildrenONT*** to the ***ChildrenONT*** of parent-transaction. In addition, the commitment of open nested transaction provokes saving of associated CDO object to the ***CDOTable***. An example of these processes is presented on the figure 3.

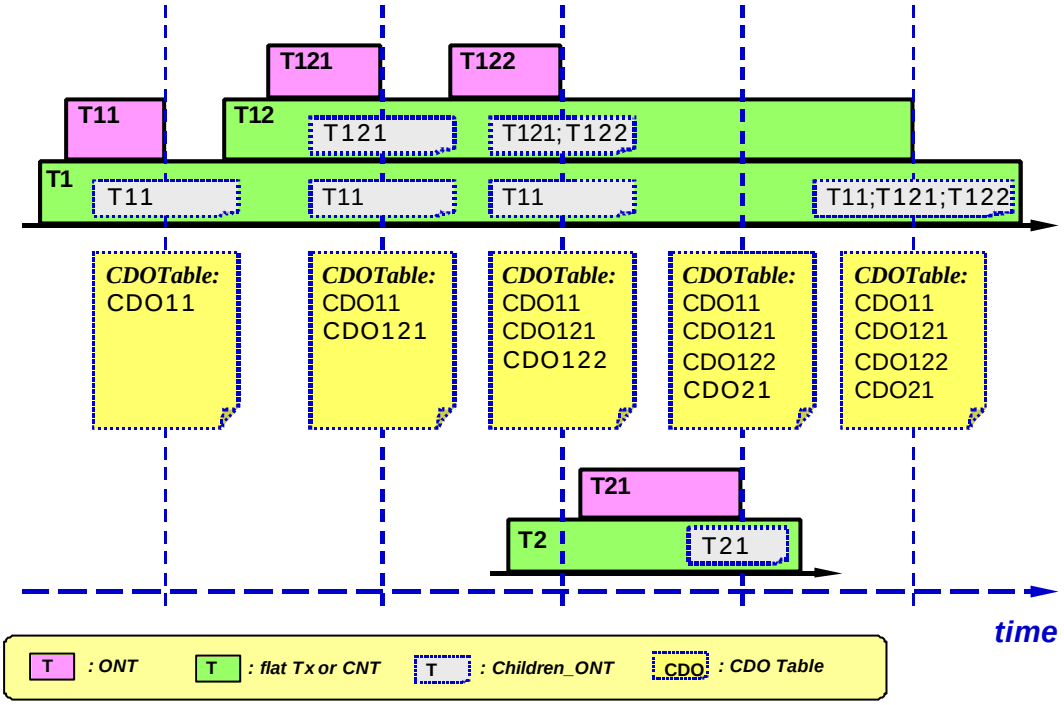

Figure 3: An example of saving the identifiers in the time of commitments

Compensation

The abort of transaction is provoked by call of method ***rollback()*** of *Transaction Manager*. This process is described by specification OTS. In addition, it can evoke the compensation process, as it was presented above. If the list ***ChildrenONT*** of aborted transaction is not empty, then the process of compensation is launched. The transaction identifiers saved in this list are sent to the *Compensation Manager*. The *Compensation Manager* finds the corresponding descriptors CDO. For this purpose, it uses the *CDOTable***.** After receiving the CDOs, the *Compensation Manager* calls the *compensators* described by these CDOs. The method **`compensate(CDO)`** of the *compensator* is invoked. This method receives the descriptor of compensation process and executes compensation method described by CDO. When compensation method is found, this method is executed with the attributes described by the CDO. If such method is absent, then an exception **`NonCompensableMethodException`** is generated. The appearance of a failure during execution of compensation-method generates the exception **`CompensateException`**.

4. Component based Transaction Service

The specification of Fractal has been chosen for implementation of proposed service. This specification allows to create the composition of component and it is open for implementation of new technical services.

4.1. The component-based Transaction Service

The previous section presents the general functionality of a Transaction Service able to manage both flat and ONT transactions. This section describes components necessary to build the Transaction Service. This service is presented on figure 4 and contains the following components:

- Component *Interpreter-flat* translates the commands of Technical Service to the commands of the Transaction Manager providing flat transaction;
- Component *Interpreter-ONT* translates the commands of Technical Service to the commands of the Transaction Manager providing ONT;
- Component *Transaction Manager* (*TM*) provides the functionalities of transaction manager described in the section 3.3;
- Component *Compensation Manager* (*CM*) provides the functionalities of compensation manager described in the section 3.3;
- Component *CDOTable* provides the functionalities of CDO store described in the section 3.3.

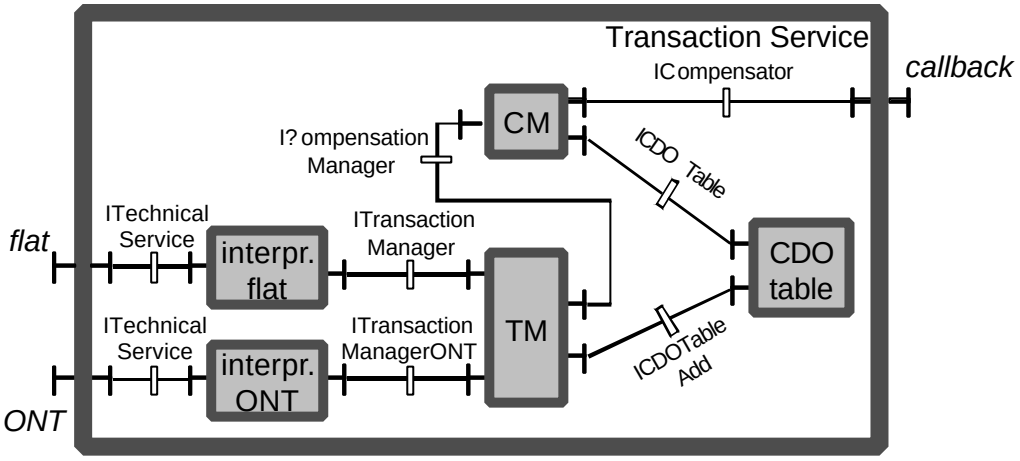

Figure 4: Transaction Service supporting ONT

The Transaction Service provides the interface of Technical Services. However, this interface is not the same that is used by the TM. The components "*Interpreter*" are put on the entry of the component *Transaction Service*. Their purpose is to translate commands received from the applicative component to the commands of transaction manager. The TM corresponding to need of Transaction Service possesses two interfaces: first for flat transaction execution and the second for the ONT execution. The *Interpreters* are connected with the corresponding interfaces providing by the *TransactionManager*.

The execution of ONT expects the CDO creation. The required information is generated by applicative component in the time of call of ONT beginning. The CDO descriptor is created by the *Interpreter-ONT* and sent to the transaction manager during the beginning of new ONT.

The component *CDOTable* keeps the CDO descriptors. The TM sends the new CDO to the *CDOTable* at the ONT commitment. The compensations are managed by *CompensationManager*. The *TransactionManager* calls its method **`compensate`** for compensation launch. The *CompensationManager* uses also the *CDOTable*, but for the CDO search. The compensator is called as callback method provided by applicative component.

4.2. Transaction service interfaces

This section presents the components used during the creation of Transaction Service. Its sub-components use following interfaces:

- ITechnicalService;
- ITransactionManager;
- ITransactionManagerONT;

- ICompensationManager;
- ICDOTableAdd;
- ICDOTable;
- ICompensator.

4.2.1. TechnicalService

All technical services provide the interface *ITechnicalService*. One service can provide several interfaces *ITechnicalService*. Each interface corresponds to one policy of service execution. This interface is universal for all Technical Services. That is why some attributes are declared but not used.

```
public interface ITechnicalService {
   public Object preInvoke(Method m, Object[ ] args,
         ComponentIdentity componentIdentity);
   public Object postInvoke(Method m, Object[ ] args,
         ComponentIdentity componentIdentity, Object IdTS);
   public Object handleException(Method m, Object[ ] args,
         ComponentIdentity componentIdentity, Object IdTS);
}
```

preInvoke

This method is executed before execution of applicative component. In the case Transaction Service, execution of this method provokes beginning of transaction. The returned object is not used in this version. Now it returns **null**.

componentIdentity identifies the applicative component which provokes execution of ***preInvoke***. The object ***m*** is of the type *java.lang.reflect.Method*. It describes the called method of this component. ***args*** is array of ***m***'s arguments.

postInvoke

This method is executed after execution of applicative component. In the case Transaction Service, execution of this method provokes commitment of transaction. The returned object and argument ***IdTs*** are not used in this version. Now it returns **null**.

componentIdentity identifies the applicative component which provokes execution of ***preInvoke***. The object ***m*** is of the type *java.lang.reflect.Method*. It describes the called method of this component. ***args*** is array of ***m***'s arguments.

handleException

This method is executed in the case of failure during execution of applicative component. In the case Transaction Service, execution of this method provokes commitment of transaction. The returned object and argument ***IdTs*** are not used in this version. Now it returns **null**.

componentIdentity identifies the applicative component which provokes execution of ***preInvoke***. The object ***m*** is of the type *java.lang.reflect.Method*. It describes the called method of this component. ***args*** is array of ***m***'s arguments.

4.2.2. ITransactionManager

This interface of Transaction Manager corresponds to *javax.transaction.TransactionManager*. This interface allows to manage the flat transactions.

4.2.3. ITransactionManagerONT

This interface of Transaction Manager extends the interface *javax.transaction.TransactionManager*. This interface allows to manage the ONT. The method of ONT beginning is added.

```
public interface ITransactionManagerONT
    extends javax.transaction.TransactionManager {
   public void begin(cdo cdo_of_ont) throws Exception;
}
```

begin

This method creates a new ONT transaction or sub-transaction if parent transaction exists. The parameter of the method describes the compensation process of beginning transaction.

4.2.4. ICompensationManager

The interface *IC**ompensationManager*** presents the functionality of Compensation Manager.

```
public interface ICompensationManager {
   public void compensate(Vector txId_to_compensate);
}
```

compensate

This method executes the compensation of transactions described by argument. The argument contains Vector (*java.util.Vector*) of transactional identifiers (*javax.transaction.xa.Xid*).

4.2.5. ICDOTableAdd

This interface presents the possibility to add the new elements to the table of compensation descriptors.

```
public interface ICDOTableAdd {
   public void newElement(cdo cdo_to_add);
}
```

newElement

This method saves the new descriptor of compensation process. The argument is an object of class providing interface ***CDO***.

4.2.6. ICDOTable

This interface provides the possibility of search of compensation descriptors.

```
public interface ICDOTable {
   public cdo findByIdentifier(Xid tx_to_find);
}
```

findByIdentifier

This method returns the descriptor supporting the interface ***CDO*** by its identifier. The argument of the method is identifier of corresponding ONT (*javax.transaction.xa.Xid*).

4.2.7. ICompensator

The interface I***Compensator*** corresponds to the interface of callback-component. It is the interface of compensator for the Transaction Service.

```
public interface ICompensator {
   public void executeCallBack(Object IdTS, Object[] args);
}
```

executeCallBack

This method executes the compensation presented by current component and described by argument ***args***. The array ***args*** contains only one object with interface ***CDO***.

4.2.8. CDO

The object, providing interface ***CDO***, describes the process of ONT compensation. It is associated with each ONT. The descriptor contains follow information:

- ***ComponentId***: The identifier of component, which executes the ONT;
- ***Method***: The identifier of method executed in the scope of corresponding ONT;
- ***Attributes***: The array of attributes of method ***Method***;
- ***TxId***: The identifier of ONT. This identifier serves as identifier of descriptor.

```
public interface CDO {
   public cdo (ComponentIdentity newComponentId,
        Method newMethod, Object[ ] newAttributes);
   public cdo();
   public ComponentIdentity getComponentId();
   public Method getMethod();
   public Object[ ] getAttributes();
   public Xid getTxId();
   public void setComponentId(ComponentIdentity new_componentId);
   public void setMethod(Method new_Method);
   public void setAttributes(Object[ ] new_Attributes);
   public void setTxId(Xid new_txId);
}
```

cdo

The constructor creates the new object-descriptor. If arguments are absent, then all parameters of object equal null. Else, the identifier of component, of method and list of attributes are sent as arguments.

getComponentId

This method returns the value of ***ComponentId***.

getMethod

This method returns the value of ***Method***.

getAttributes

This method returns the value of ***Attributes***.

getTxId

This method returns the value of ***TxId***.

setComponentId

This method modifies current value of ***ComponentId***. The new parameter is taken from argument of the method.

setMethod

This method modifies current value of ***Method***. The new parameter is taken from argument of the method.

setAttributes

This method modifies current value of ***Attributes***. The new parameter is taken from argument of the method.

setTxId

This method modifies current value of ***TxId***. The new parameter is taken from argument of the method.

References

[AVBSW94] G. Alonso, R. Vingralek, D. Agrawal, Y. Breitbart, A.E. Addabi, H.J. Schek, G. Weikum, Unifying Concurrency Control and Recovery of Transactions, Proceedings of the EDBT Conference, 1994.

[BCS02] E. Bruneton, T. Coupaye, J.-B. Stefani, « Recursive and Dynamic Software Composition with Sharing », *Proceedings of the 7th ECOOP International Workshop on Component-Oriented Programming (WCOP'02*), Malaga, Spain, 2002.

[DHL91] U. Dayl, M. Hsu, R. Ladin, A Transactional Model for Long-Running Activities, Proceedings of the VLDB Conference, 1991.

[Fractal] http://www.objectweb.org/fractal/index.html.

[Gray81] J. Gray, The Transaction Concept: Virtues and Limitations, Proceedings of the VLDB Conference, 1981.

[HEI01] G. T. Heineman, , W. T. Council, « Component-Based Software Engineering, Putting the Pieces Together », Addison Weysley, 2001.

[HeLe03] C.Hérault, S.Lecomte, « Gestion de Personnalités de Services Non-Fonctionnels dans un Modèle à Composants »Communication présentée à Workshop "Objets, Composants et Modèles dans l'ingénierie des Systèmes d'Information" en commun avec le congrès INFORSID 2003, Nancy, France, juin 2003.

[HW91] C. Hasse, G. Weikum, A Performance Evaluation on Multi-Level Transaction Management, Proceedings of the VLDB Conference, 1991.

[JK97] S. Jajodia, L. Kerschberg, Advanced Transaction Models and Architecture, Kluwer Academic Publishers, 1997.

[KLS90] H.F. Korth, E. Levy, A. Silberschatz, A Formal Approach to Recovery by Compensating Transactions, Proceedings of the VLDB Conference, 1990.

[Moss81] E. Moss, Nested Transactions: An approach to reliable Distributed Computing, PhD Thesis, Tech. Rep. MIT/LSCTR-260, 1981.

[OHE99] R. Orfalie, D. Harkey, J. Edwards, « Client/Server Survival Guide, 3thrd edition », Vuibert, 1999.

[RMe03] R. Rouvoy, P. Merle, « Abstraction of Transaction demarcation in Component-Oriented Platforms », ACM/IFIP/USENIX International Middleware Conference (Middleware'03), Rio de Janeiro, Brésil, 16-20 Juin 2003.

[RMo89] K. Rothermel, C. Mohan, ARIES/NT : A Recovery Method Based on Write-Ahead Logging for Nested Transactions, Proceedings of the VLDB Conference, 1989.

[Rou03] R.Rouvoy, "GOTM Architecture. Technical report", LIFL, USTL, June 2003