DCP-Grid, a Framework for Conversational Distributed Transactions on Grid Environments

Manuel Salvadores[1], Pilar Herrero[2], María S. Pérez[2], and Víctor Robles[2]

[1] IMCS, Imbert Management Consulting Solutions,
C/ Fray Juan Gil 7, 28002 Madrid , Spain
[2] Facultad de Informática – Universidad Politécnica de Madrid,
Campus de Montegancedo S/N,
28.660 Boadilla del Monte, Madrid, Spain
mso@imcs.es
{pherrero, mperez, vrobles}@fi.upm.es

Abstract. This paper presents a Framework for Distribute Transaction processing over Grid Environment, called DCP-Grid. DCP-Grid complements *Web Services* with some OGSI functionalities to implement the *Two Phase Commit* (2-PC) protocol to manage two types of Distribute Transactions, Concurrent and Conversational transactions, properly in this kind of environment. Although DCP-Grid is still under development at the Universidad Politécnica de Madrid, in this paper, we present the design and the general characteristics associated to the implementation of our proposed Framework.

1 Introduction

The introduction of *Services Oriented Architectures* (SOA) [1] [2], in the last few years, has increased the use of new distributed technologies based on *Web Services* (WS) [3]. In fact, e-science and e-business processes have adopted this technology to improve the integration of some applications. The coordination of this type of processes, based on WS, needs the transactional capability to ensure the consistency of those data that are being handled by this kind of applications.

A transaction could be defined as the sequence of actions to be executed in an atomic way. This means that all the actions should finish - correctly or incorrectly- at the same time as if they were an unique action.

The four key properties associated to the transactions processing are known as the ACID properties - *Atomicity, Consistency, Isolation, y Durability*[4]. The aim of our proposal is to build a Framework, based on grid technologies, to coordinate distributed transactions that are handling operations deployed as Web Services.

The Grid Technology, which was born at the beginning of the 90's , is based on providing an infrastructure to share and coordinate the resources through the dynamic organizations which are virtually distributed[5] [6].

In order to make possible the development of DCP-Grid, we will take into account the *Grid Web Services* (GWS) characteristics. The GWS, defined in the *Open Grid Service Infrastructure* (OGSI) [7], could be considered as an extension of the WS. The GWS introduce some improvement on WS, which are necessary to the construction of standard, heterogeneous and open Grid Systems. The OGSI characteristics on which DCP-Grid has being designed and built are: Statefull and potentially transient services; Service Data; Notifications; portType extension; Grid Service Handle (GSH) and Grid Service Reference (GSR).

OGSI is just an specification, not a software platform, and therefore, we need a middleware platform, supporting this specification, in order to deploy the DCP-Grid Framework. From all the possible platforms to be used, we have decided to use the Globus Toolkit [8] platform for this Project because in the most extended nowadays. More specifically, we have being working with GT3 (version 3.2) for DCP-Grid due to its stability.

In this paper we will start describing the state of the art in the dealing area as well as their contributions to the DCP-Grid design, subsequently we will move to the architectural design of our proposal and we will give some details related to the framework implementation to finish with some conclusions, ongoing directions and future work.

2 Related Work

The standard distributed transactional processing model more extended is the X/Open [14] model, which defines three rolls (Resource Manager RM, Transaction Processing Manager TPM and Application Program AP) [9] [14].

Based on the Web Service technology two specifications to standardize the handling of transactions through open environments have arisen. These specifications are WS-Coordination [10] and WS-Transaction [13], developed by IBM, Bea and Microsoft. In them, the way to group multiple Web Services as a transaction is exposed, but the form of coordination of the transactions is not specified. On the other hand, the Business Transaction Protocol specification (BTP) [13], proposed by OASIS, defines a transactional coordination based on workflows. This specification is complex to handle and integrate [12]. Based on GT3 [8] we try to construct a simple proposal for the implementation of a transactional manager adopting the X/Open model. In the proposed design, the analogies are visible.

2.1 Two Phase Commit Protocol

The *Two phase commit* (2-PC) protocol is an ACID compliant protocol to manage DTs. How 2-PC works is easy to explain. A Transaction with a 2-PC protocol does not commit all the actions if not all of them are ready to be committed. This process works in two phases, as its names indicates, first phase called *Voting Phase* and second phase called *Commit Phase*.

During de *Voting Phase,* each and every action notifies to the system their intentions to commit theirs operation. This phase terminate when all the actions are ready to be committed. Then starts the *Commit Phase*, during this phase the system notifies to each and every action to be finished, the conclusion of the operation takes place when the commit message arrives from the system to each and every action.

These two components work in two different phases, during the *Voting Phase* when an application requests to the DTC to commit a Transaction, the DTC sends PREPARE_TO_COMMIT to all the RM that have uncommitted actions of the Transaction, then the DTC waits for a period to receive all the RMs responses.

There are two possible responses READY_TO_COMMIT or UNABLE_TO_COMMIT. If all the RMs responses to the DTC are READY_TO_COMMIT message then the DTC sends a COMMIT message to all RMs, but if any of the resource managers sends a UNABLE_TO_COMMIT or not response in a limit time then the DTC sends a ROLLBACK message to all the RM. In Figure 1 we can appreciate the successfully scenario of a commit transaction over 2-PC protocol.

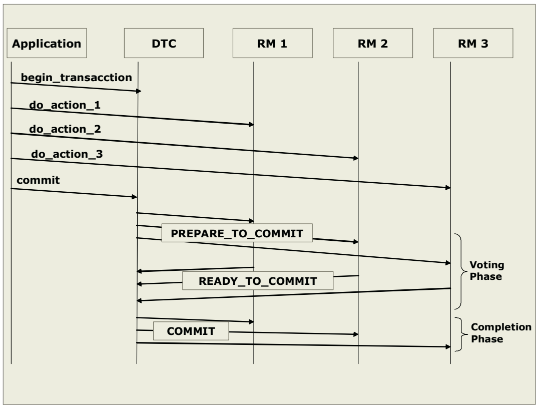

Fig. 1. 2-PC messages intereaction

Distributed Transactions and 2-PC protocol will be the pillars of our Framework because DCP-Grid will provide 2-PC protocol to support transaction processing.

2.2 Classification of Distributed Transactions

We assume two categories *Concurrent Distributed Transactions* and *Conversational Distributed Transactions*:

- *Concurrent Distributed Transactions*: are the transactions formed by actions that have not dependencies between. In this case, the different actions can be sending by the application layer in a concurrent way improving the service time processing.
- *Conversational Distributed Transactions*: are the transactions composed by dependent actions. An example of this kind of transaction could be the following. This scenario refers for any transaction in which action N depends on, at least, one or more actions previously executed, being N-i the maximum number of actions to be included in this dependence, and i a natural number representing the position.

3 Scenarios

As we mentioned in previous section of this paper, there is two different kinds of categories regarding to each proposed scenario.

3.1 Scenario 1: Concurrent Distributed Transactions

In this scenario we will concentrate on the logical operation of the Framework regarding transactions composed by independent actions.

Let's imagine a scenario composed by two actions A and B, and a client application which wants execute these actions in a Transactional way. In this case it would necessary define a transaction like Tx {idTx, coordinator, A, B} where idTx is transaction id and A, B are the actions that compose the transaction. The element coordinator references to the DTC process that coordinates the phases of the transaction.

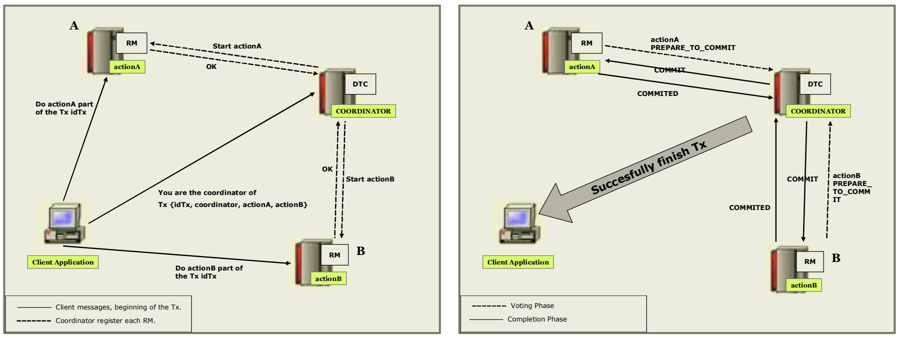
A
RM
actionA
Start actionA
OK
DTC
COORDINATOR
Do actionA part of the Tx idTx
OK
Start actionB
You are the coordinator of Tx {idTx, coordinator, actionA, actionB}
Client Application
Do actionB part of the Tx idTx
RM
actionB
B
Client messages, beginning of the Tx.
Coordinator register each RM.
A
RM
actionA
actionA PREPARE_TO_COMMIT
COMMIT
COMMITED
DTC
COORDINATOR
Succesfully finish Tx
COMMIT
actionB PREPARE_TO_COMMIT
COMMITED
Client Application
RM
actionB
B
Voting Phase
Completion Phase

Fig. 2. Scenario for Concurrent Distributed Transactions

Figure 2 shows how the transaction starts, first the client applications sends to each RM the respective action and the idTx associated with the Tx. At same time, the client, sends to the coordinator all the information of the Tx, with this information the coordinator register in the DTC all the actions of the Tx and sends a message to each RM for start their associated actions.

Once each and every action is ready to be committed, it will send a message to the DTC notifying the current state (PREPARE_TO_COMMIT), as it were mentioned previously this is the *Voting Phase*. After all RMs have sent their status message the DTC will decide about finish the transaction sending a COMMIT message to each RM. The stage meanwhile the DTC is sending the COMMIT message is the *Commit Phase* of the 2-PC protocol.

The case of ROLLBACK in this scenario will given by the overcoming of period (TIMEOUT) or if any RM sends a fail message (UNABLE_TO_COMMIT) to the DTC, in this situation a ROLLBACK message will be send to each RM.

It is easy to see a small modification over the previous explication of 2-PC, in this scenario due to the concurrency, are the RM's who sends the message PREPARE_TO_COMMIT not the DTC.

3.2 Scenario 2: Conversational Distributed Transactions

This scenario describes a DT composed by the same two previous actions A and B which can't working currently, due to this, these actions are sent sequentially because of its dependencies, the client recovers the partial results and it invokes other actions with these results. Due to this, is the client who decides when the transaction must finalize, and when the 2-PC protocol must begin their commit phases (*Voting* and *Completion*).

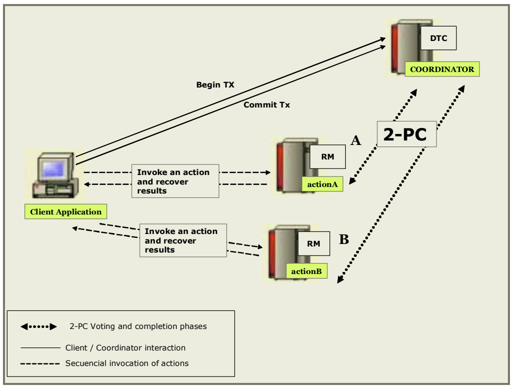
DTC
COORDINATOR
Begin TX
Commit Tx
2-PC
A
RM
actionA
Invoke an action and recover results
Client Application
Invoke an action and recover results
RM
B
actionB
2-PC Voting and completion phases
Client / Coordinator interaction
Secuencial invocation of actions

Fig. 3. Scenario for Conversational Distributed Transactions

The *Figure 3* shows the process in this scenario. First the client application invoke the coordinator to start the transaction Tx, after, call sequentially the necessary actions and to finish the transaction invoke the coordinator with a COMMIT message. At this moment, the DTC manages the 2-PC protocol to cross the *Voting* and *Completion* phases like in scenario 1. In case of failure of some action, the DTC will send a ROLLBACK message to the rest of them.

In addition, if some RM does not respond in a time limit then a ROLLBACK message will be sent by the DTC to each RM. By this way, we avoided blockades of long duration. The problem of blockades and concurrency access will be explained in more detail in section 4.1.

4 Our Approach to Grid Environments

Taking advantage of the OGSI characteristics, we propose a DCP-Grid to be introduced in a Grid Environment. As our first approach at the *Universidad Politécnica de Madrid* (UPM), we have decided to introduce a new interface which we have called *ITransactionSupport*. This new interface provides to Grid Services with operations that provides "rollback" and "commit" functionality. Every Grid Service which wants take part of a transactional execution will extend this interface, this solution could be achived thanks to the *PortType Extension*, provided by OGSI.

On the other hand, we have defined the element *Distributed Transaction Coordinator (DTC)*, key concept for DCP-Grid. Our solution proposes to the DTC like a *Grid Service*, we assigned to this service the name of *TXCoordinationService*, with this design we took advantage of all features of *Grid Service* to develop the DTC.

In order to separate the interface of the logical behavior our proposal introduce a new component, the engine that manages all the process around the commit protocol. This component is the *TXManagementEngine*. We can look inside the coordinator in the next figure:

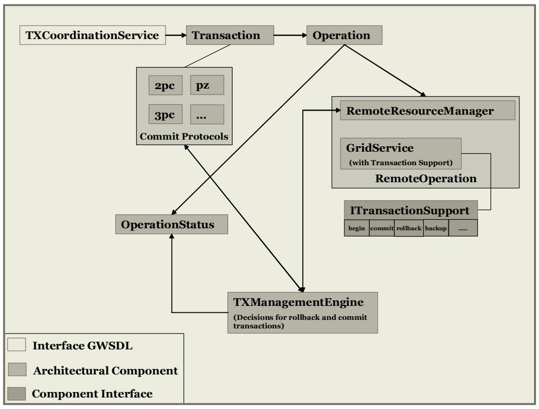

Fig. 4. DTC building blocks Architecture

The TXManegementEngine makes its decisions in function from the information updated by the *TXCoordinationService*, for example, when the TXCoordinationService receives a message PREPARE_TO_COMMIT it update the correspondent *OperationStatus*. When the *TXManagementEngine* detects that all *OperationStatus* are in PREPARE_TO_COMMIT then sends a COMMIT message to all remote resource managers.

Each remote RM can commit their operations because it implements *ITransacctionSupport*. In case of some message received by the *TXCoordinationStatus* will be a UNABLE_TO_COMMIT message then the *TXManegementEngine* will make a rollback sending the corresponding message to each remote RM.

The RM is another *Grid Service* deployed in each system that contains transactional *Grid Services* due to this the TXManagementEngine contained in the DTC can invoke the *TXResourceManagerService* for the commit or rollback interface operations and a situation of blockade will occur.

4.1 Solutions to the Situations of Blockade

A key point to be taken into account is the resolution of the interblockades; this factor could be critical in some scenarios as the following: a client application invokes a service and this execution is part of a transaction. At a certain moment, this service not finished yet because of another action associated to the same transaction is not in the commit phase. At this same moment, another client application sends the execution of a transaction in which there is the same action blocked before. If this action is not released, another execution will not be able to enter and a situation of blockade will occur.

In order to solve this situation our proposal establishes a time limits of delay to process commit once the service has been processed, this timeout will be a parameter to be established by each and every service. In future versions of DCP-Grid, we will tackle the problem deeply.

4.2 Solutions to Concurrent Access Situations

During the investigation of DCP-Grid a problem with the concurrent access appear. What happens if an RM has deployed a service that allows parallel access to different client execution, if two clients invoke the service simultaneously when the respective DTCs want to process COMMIT or ROLLBACK? How the RM knows which is the message (COMMIT / ROLLBACK) associated to each execution?

To solve this situation our proposal generates a unique ID that will be propagated in corresponding messages to the respective DTCs and RMs. With this ID the two components (DTCs and RMs) will be able to associate the message received with the corresponding operation. This session ID identifies uniquely each transaction. Another problem appear with this solution How can generate unique IDs across different RMs in separated system?

To solve this new problem DCP-Grid will negotiate the session ID between the RMs. The internal operation is as it is described to continuation, the DTC associated to the transaction generate the transaction ID based on the system address, after the DTC sends the start message to each RM. If some RM detects that it has an active id with same value, then this RM will send a CHANGE_ID message to the DTC, the DTC will generate a new ID only for this RM. Internally the DTC will store this information for future messages.

5 Conclusions and Future Work

In this paper we have presented our approach to implement an architecture supporting transactional Grid WS execution. This approach is based on some of the main properties of OGSI specification [7]. As ongoing work, currently we are developing a similar framework for concurrent distributed transactions on grid environments which will be presented in the Workshop on KDMG'05. So many future research lines has been opened for DCP-Grid but maybe the most interesting would be the building of an environment to support transactions on distributed and heterogeneous databases based on the concepts and ideas that we have presented in this paper.

References

[1] Douglas K. Barry, Web Services and Service-Oriented Architecture: The Savvy Manager's Guide, Morgan Kaufmann Publishers 2003

[2] Mark Endrei, Jenny Ang, Ali Arsanjani, Sook Chua, Philippe Comte, Pal Krogdahl, Min Luo, Tony Newling, "Patterns: Service Oriented Architecture", IBM RedBook SG24-6303-00

[3] "Web Services Main Page at W3C", http://www.w3.org/2002/ws/, Worl Wide Web Consortium, (consultado en dic/2004)

[4] Berntein, New Comer "Principles of Transaction Processing", Editorial: Kaufman, 1997

[5] I. Foster, C. Kesselman. The Physiology of the Grid: An Open Grid Services Arquitecture for Distributed System Integration . 2002. http://www.globus.org/research/papers/ogsa.pdf

[6] Miguel L. Bote-Lorenzo, Yannis A. Dimitriadis, Eduardo Gómez-Sánchez "Grid Characteristics and Uses: A Grid Definition", LNCS 2970, 291-298

[7] S. Tuecke, K. Czajkowski, I. Foster. Grid Service Specification. Technical Report. Jun 2003. www-unix.globus.org/toolkit/draft-ggf-ogsi-gridservice-33_2003-06-27.pdf

[8] "Globus Toolkit Project", The Globus Alliance, http://www.globus.org (consulted 2004/12)

[9] I. C. Jeong, Y. C. Lew. DCE "Distributed Computing Environment" based DTP "Distributed Transaction Processing" Information Networking (ICOIN-12) Jan. 1998

[10] F. Cabrera et al., "Web Services Coordination (WS-Coordination)" Aug. 2002, ww.ibm.com/developerworks/library/ws-coor/

[11] F. Cabrera et al., "Web Services Transaction (WS-Transaction)" Aug. 2002, www.ibm.com/developerworks/library/ws-transpec/.

[12] Feilong Tang, Minglu Li, Jian Cao, Qianni Deng, Coordination Business Transaction for Grid Service. LNCS3032 pag. 108-114 (Related Work Section)

[13] OASIS BTP Committee Specification 1.0, 3 June 2002, Business Transaction Protocol, http://www.choreology.com/downloads/2002-06-03.BTP.Committee.spec.1.0.pdf

[14] X/Open Specification, 1988, 1989, February 1992, Commands and Utilities, Issue 3 (ISBN: 1-872630-36-7, C211); this specification was formerly X/Open Portability Guide, Volume 1, January 1989 XSI Commands and Utilities(ISBN: 0-13-685835-X, XO/XPG/89/002).