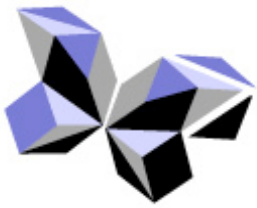


Digital Business Ecosystem

Contract n° 507953

Workpackage 24: DBE Implementation

Deliverable D24.5: DBE Distributed Transaction Model

Information Society Technologies

Project funded by the European Community under the “Information Society Technology” Programme

Contract Number: 507953

Project Acronym: DBE

Title: Digital Business Ecosystem

Deliverable N°: D24.5

Due dates: 08/2006

Delivery Date: 01/2007

Short Description:

This document reviews current transaction models, and highlights their weaknesses in the context of supporting long-term business transactions that involve SMEs participating in a Digital Business Ecosystems.

The document then proposes an alternative transaction model that preserves the local autonomy of participating SMEs, and also provides important advantages in support for forward recovery and explorative composition.

We conclude with a short analysis of the requirements for a next generation P2P architecture that will enable the benefits of the proposed transaction model to be fully realised.

Author: UniS

Partners contributed:

Made available to: Public

VERSIONING		
VERSION	DATE	AUTHOR, ORGANISATION
0.1	**21/09/06**	**AMIR RAZAVI, PAUL KRAUSE, UNIVERSTIY OF SURREY**
1.0	**23/10/06**	**AMIR RAZAVI, PAUL KRAUSE, SOTIRIS MOSCHOYIANNIS, UNIVERSITY OF SURREY**
2.0	**19/01/07**	**PAUL KRAUSE, SOTIRIS MOSCHOYIANNIS, AMIR RAZAVI, UNIVERSITY OF SURREY**

Quality check:

1st Internal Reviewer : Paul Malone, WIT

2nd Internal Reviewer: Miguel Vidal, TI

Table of Contents

1. Introduction

The Digital Business Ecosystem (DBE) project [DBE06] is concerned with building an open source environment, through which businesses, in particular small to medium enterprises (SMEs), can interact within a pan-European ecosystem. The aim is to provide access to arbitrary services that the DBE helps compose together to meet particular needs of the various partners. This collaborative software environment is being targeted primarily towards SMEs, who will be able to concatenate their offered services within service chains formulated on a pan-European basis.

Within a DBE a number of long-running multi-service transactions are expected to take place and therefore, of interest is the issue of providing support for long-term business transactions involving open communities of SMEs. A business transaction in this paradigm can be either a simple usage of a web service (rarely in B2B relationships) or a mixture of different levels of composition of several services from various service providers. We will argue that the current transaction and business coordination frameworks can lead to issues with tight coupling and violation of local autonomy for the participating SMEs.

Web services are the primary example of the Service Oriented Computing (SOC) paradigm [Pap03]. The goal of SOC is to enable applications from different providers to be offered as services that can be used, composed, and coordinated in a loosely coupled manner. Web services in fact provide a realisation of SOC. Although recent years have seen significant growth in the use of Web Services, there are some very significant technological constraints that are stopping their full potential from being realised within a DBE environment.

Current transaction models that are targeted to web services provide inadequate support for long-term distributed transactions [VZG+05], [FuG05]. This is not perhaps as visible as it may be, due to the dominance of a relatively small number of highly visible service providers. The impact is much higher with regard to the restriction in adoption of SOC by small to medium sized enterprises (SMEs). Since SMEs contribute 50% of Europe's GNP, this is of significant concern.

The conventional transaction view is based on ACID properties: Atomicity; Consistency; Isolation; Durability. However, these properties are too constraining for business transactions. The long-term nature of business transactions, in a highly dynamic and distributed environment, is the major challenge for constructing/defining a consistent transaction model. Consider, for example, the atomicity and isolation properties. In long-term transactions, partial results may need to be shared between different transactions before their final termination (commitment). Failure to do this may at best lead to unacceptable delays in related transactions, and at worst leave a provider open to denial of service attacks (by locking their data in a non-terminating transaction. Hence, we must weaken the isolation property, and this then poses further challenges with regard to ensuring consistency of the underlying transaction model.

The lack of consideration for the primary characteristics of SOC (such as loose-coupling) or ignoring some important business requirements (such as partial results), are important objections to the proposed transaction models, which also suffer from unnecessary complications of implementing a consistency model on top of the service-realisation boundary. In addition, the feasibility of a heavy coordinator framework causes some transaction models to use a centralised (or limited decentralised) coordination model.

In this report, we propose a novel distributed transaction model for DBE, which has been developed with the SOC principles in mind. It is expressive enough to cover various forms of composition types to capture evolving business requirements and supports well-known behaviour patterns. Particular thought has gone into the coordination mechanism which is performed *locally* and allows for a loose-coupled binding with the Participant. We also describe the compensation capabilities of the proposed transaction model based on the graphs used for capturing dependencies within and between transactions, and show how omitted results are handled without breaking the local autonomy of the underlying services.

The proposed transaction model for DBE, by introducing a consistency graph for a transaction (Internal Dependency Graph - IDG) implements a distributed coordinator model in which a transaction can be performed through the cooperation (sharing of results) of the local coordinators. Furthermore, the possibility for realising partial results between different transactions is considered by introducing an External Dependency Graph, which in combination with the IDG can provide a global consistency model. On the localised design of the coordinator we introduce:

- Two locks (I-Lock, C-Lock) for providing consistency connection between the distributed logs (EDG and IDG) and the local concurrency control model.
- A lock (T-Lock) for covering the dynamic aspects of the highly dynamic environment (covering omitted results in common distributed events such as temporary disconnections).
- A lock (R-Lock) for recovery which supplies an isolated two-phase recovery management routine.

Forward recovery, an integrated compensation mechanism and coverage of a variety of composition types are other features of the proposed model. The aim is to provide a foundation for further work on business expansions, the optimisation of business processes and providing competitive advantages in the E-market.

This report is structured as follows. In **Chapter 2**, we provide an extensive review of existing transaction models and discuss their characteristics in terms of long-running business transactions required for DBE. In **Chapter 3**, we outline the basic characteristics of service-oriented architectures and then discuss transaction models that have been developed with web services in mind. We argue that existing transaction models for web services are tailored to the needs of large-scale enterprises (LEs) and raise barriers to the adoption of such technology by SMEs. More specifically, such models rely on coordination frameworks that are not fully distributed and do not respect the loose-coupling of the underlying services.

The proposed distributed transaction model for DBE is presented in **Chapter 4**. At the heart of this is the coordination model which has been customised for SMEs in a SOA for DBE environments. In **Chapter 5**, we describe the formal underpinning of the coordination aspects and lay the foundations for a formal foundation of the DBE transaction model. **Chapter 6** provides implementation support for the coordination aspects of the transaction model through various lock controls that ensure local consistency. Compensation and recovery are based on these locking schemes so that consistency can be guaranteed and local progress-to-date is preserved as much as possible.

Naturally, the next step is to look for network architectures that can support this collaborative software environment of the DBE, where collaboration is conducted based on the DBE transaction model. We are concerned with such aspects in **Chapter 7** which in a certain important sense paves the way for future extensions of the work presented in this report. Peer to Peer (P2P) architectures offer significant potential for B2B interactions in SOC. However, the preliminary analysis of existing P2P architectures to date in Chapter 7 shows them to have critical shortcomings with respect to supporting a fully decentralised transaction model which allows for various forms of service composition and respects the loose-coupling of the underlying services. **Chapter 8** contains some concluding remarks and a brief discussion on future directions.

2. Distributed Transaction Models Review

The conventional definition of a transaction [Date1996] is based on ACID properties[1]. However, as we have indicated earlier, in advanced applications these properties can present unacceptable limitations and reduce performance [Elmagarmid1994]. We will focus on three specific problems with conventional transaction models [Moss1985], [Kakeshita and Haiyan Xu1992], [Haghjoo and Papazoglou1992, Elmagarmid1994]: Long-lived transactions; Lack of partial results; and Omitted results.

The necessity of change is derived from the nature of business-, as opposed to database-, transactions. For example, the specification of a transaction may allow it to be completed over a period of hours or even days (a "long-lived transaction"). In addition, the obligation for cooperation between transactions can be specified in a business process rule (a requirement for availability of "partial results"). Finally, the instability of the internet environment can define a new requirement for keeping important results even when the connection between two platforms is lost ("omitted results").

2.1 Transaction Models

Eliot Moss introduced the first revolutionary answer to the Long-lived transactions problem in 1981 with the title of "Nested Transactions" [Moss1985]. Their Nested Transactions broke the atomicity of conventional transactions, but required the definition of new integrity rules instead. However, many problems remained unsolved. Different non-conventional transaction models were subsequently derived from the Nested-Transactions model, which changed the face of Database world.

In the Nested transaction model, each transaction can have a tree structure (including many sub-transactions) and each node can share its results others in the same transaction (Figure 2-1). However, one Nested transaction cannot share its results with any of the other nested transactions. Therefore each nested transaction still was atomic and isolated as far as other nested transactions were concerned and hence the ACID properties were still applied, although at a different level.

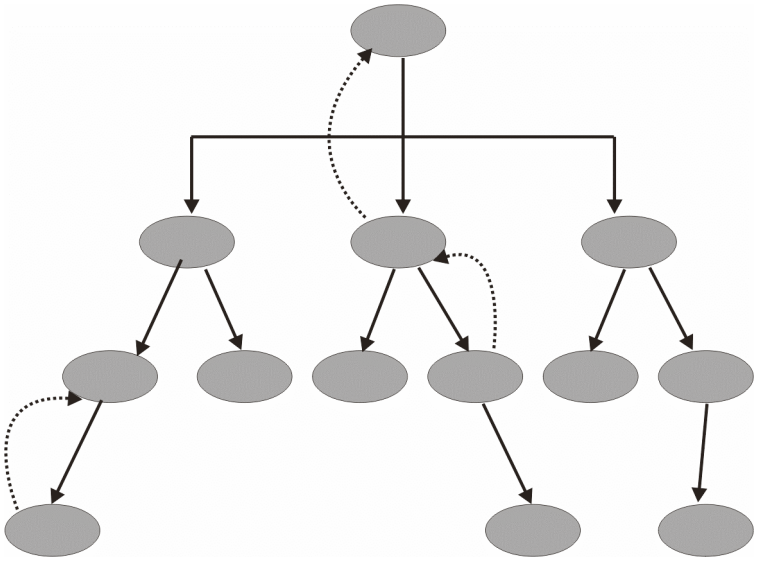


Figure 2-1 Nested Transactions

[1] **A**tomicity – either all tasks in a transaction are performed, or none of them are; **C**onsistency – data is in a consistent state when the transaction begins, and when it ends; **I**solation – all operations in a transaction are isolated from operations outside the transaction; **D**urability – upon successful completion, the result of the transaction will persist.

One of the important requirements in the distributed computing domain is the ability to access to information before transaction commitment. In the Nested Transaction model, in order to maintain database consistency this facility is available only internally within a transaction. This means only the parent of a sub-transaction can receive the results of its child and all other transactions have to wait for commitment of the main transaction. This can dramatically reduce overall performance if cases where there is potential for a number of transactions to be executed concurrently, which require access to shared or related data. This problem arises through unavailability of 'Partial Results' [KaX92], [HaP92], [Elm94].

In addition, if a Transaction is aborted (for any reason) all of its sub-transactions must also be aborted. This means all of the intermediate results produced up until that point must be rolled back to the initial values. In turn, if that transaction is restarted, all operations must also be performed again; a huge overhead! This problem is known as 'Omitted Results' [KaX92], [HaP92].

2.1.1. Different Answers to the problem

The Partial Result problem is perhaps the most complicated problem in a distributed, long-life transaction. Of course, database consistency and integrity are hard to maintain if Transactions can access the uncommitted results of the some other Transactions. Not only does the overhead of this consideration increase in the large scale, but also very complicated integrity rules are necessary if we are to allow release of partial results and still maintain global consistency. Two different approaches have been made to answering this problem [Elm94]:

- **Accepting Nested Transactions rules**: In this case, the structure of Nested Transaction is maintained and the integrity rules are the same as for a Nested Transaction. Instead, the database environment (definition) itself is structured carefully, and the definition and creation a transaction structured so that execution of one transaction will not lead to the locking of partial results that ma be needed by another transaction. The best-known model in this approach area is *Multidatabase* [No93], [NoZ94], [WZB+01].

- **Open Nested Transaction**: The basic structure of Nested Transaction is used, but transactions are able to release their uncommitted results. In this case, the Integrity rules need to be completely different (as well as the structure of models). Some of the best known transaction models in this category include: *Cooperative Transaction model* ([HPS93], [RaN99]); *Multilevel Transaction Model* [DSW94]; *Coordinating Transaction Model* [QXL02]; *Extended-Saga Transaction Model* [G-MG+91]; *Mega Transaction Model* [Hag96]; *DOM Transaction Model* [Elm94].

2.1.2. Sagas (and Extended Sagas) Transaction Model

In the primary Sagas transaction model [G-MS87], [Elm94], [LiZ04] the structure of a transaction is based on a linear, sequence of serial sub-transactions that are run one after the other. The important properties of these sub-transactions was compensability.

Let transaction S (as a Saga transaction) include sub-transactions T1, T2, T3, …, Tn. Each sub-transaction Ti, as an atomic unit, has a corresponding compensation transaction, Ci. If any fault arises, the Compensating transactions will be run in the opposite sequential order (Figure 2-2).

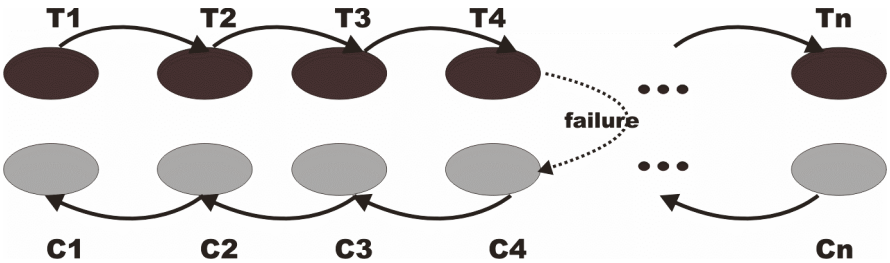


Figure 2-2 Saga Transaction Model

The main ideas behind Saga have been reused in several models. Firstly, any Saga transaction can be imagined as a compensatable unit (although this suggestion is not actually used in the Saga transaction model). This idea is used in Mega transaction and in recovery management of the Co-operative Model. Secondly, the linear structure of a Saga transaction model is used in the Coordination model and in the Mega transaction model as a scheduler (Serial scheduler).

The structure of Saga (linear sub-transactions) can cover many different transactions in a distributed environment. However, in reality we cannot model every transaction as a series of compensatable sub-transactions (because the nature of some transactions is not compensatable) [G-MG+91]. The extended-saga transaction model tries to solve these problems. By introducing a tree structure, extended-saga improves the model in several areas (mostly extended-saga is called ACTA, in Workflow technology [Eder and Liebhart1995], whereas distributed transactional component based applications in Enterprise JavaBeans use ACTA [EmT00]). But the isolation of a Saga transaction is still is a major inhibitor to releasing partial results during the lifetime of a Saga transaction.

2.1.3. Multi-level Transaction Model

The Multi-level Transaction model [DSW94] has the same structure as the Nested transaction model, but with an extra facility for solving the partial result problem. A sub-transaction of a Multi-level transaction can release results to the same level in the other Multi-level transactions (Figure 2-3).

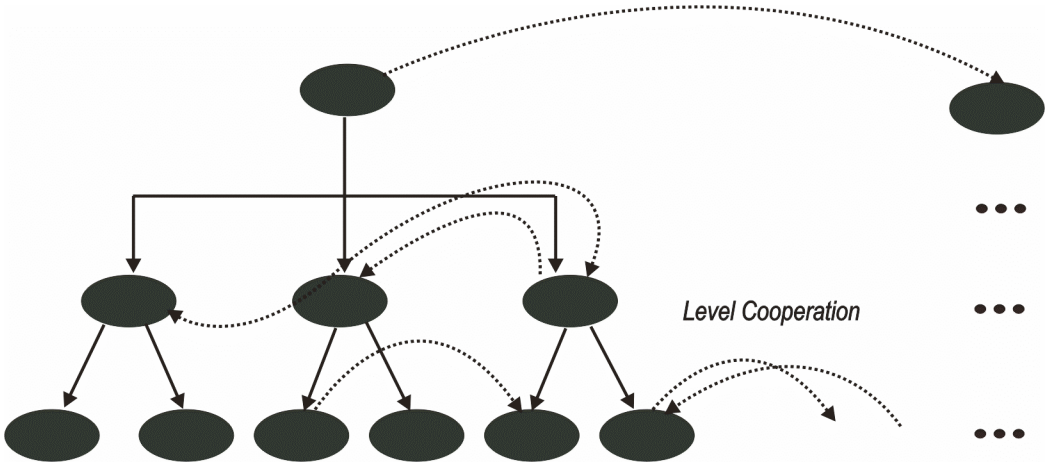


Figure 2-3 Multi-Level Transactions

Therefore there is a virtual network (a graph structure) for the same level of children (sub-transactions). These important graphs try to give an “info-structure” to compensate for the lack of isolation and atomicity (compared with a conventional transaction model). The Multi-level transaction model has been used in several applications.

In general, the nature of an application must be compatible with the Multi-level definition. Unfortunately, mostly this does not happen (or we have to follow some instructions for making these compatibilities and modifying our main application). Not only do these overhead structures (Multi-level graphs) reduce the performance (through the construction of parallel graphs that are sometimes not used at all), but also the need for deadlock control will cause a huge overhead in any practical situation (depending on the nature of the application and the value of releasing data in the same level).

2.1.4. Contract Transaction Model

The Contract Transaction Model [VeP98] is designed for controlling complicated and long calculations in specific environments such as CAD and Office automation. The Contract Transaction Model is one of the

best-known models for several motivational ideas (even though it is little used now in practical situations). For example, one innovative feature is that when a failure occurs, Contract transaction tries to continue its running by finding alternative ways to proceed; 'Forward Recoverable behaviours'.

The Contract model gives permission for commitment of partial results before final commit of a Contract transaction (releasing partial results). And it tries to warranty consistency and integrity with forward recovery. There are two types of sub-transactions:

- **Compensating transactions:** These transactions try to undo the effect of unsuccessful/unnecessary sub-transactions.

- **Contingency transactions:** When a sub-transaction fails, this alternative can be run in an attempt to avoid an abort (forward recovery).

The Contract model introduces contingency and Vital / Non-Vital transactions (with alternative sub-transactions) as two important ideas. Unfortunately, in a particular environment, structural limitations and huge overheads are the main problems for implementation of this model (even some sources categorize this model as just an extension to the conventional model that adds some control mechanism to several ACID transactions in the Contract of a bigger transaction) [Dat96], [LiZ04].

2.1.5. Split Transaction Model

This model was introduced for supporting 'Open–ended applications' like VLSI designs, CAD/CAM projects and so on. Any application with the following three properties is defined as an 'Open–ended application' (and theoretically can be modelled by the Split transaction model):

1. **Uncertain Duration:** The transaction life time can be hours or months; you can not determine any specific time limitation on the run time.

2. **Uncertain Development:** There is no prediction about operations and limitations on them.

3. **Interaction with other Concurrent Activities:** Concurrent operations can affect each other and share their results at any time (not just after commit, which will not be at a specific time, based on property 1).

The manager of the Split Transaction Model splits a transaction into two ordered transactions and divides the resources between them [Elm94]. To split transaction T into transactions A and B, where A is ordered before B, the following properties must be satisfied (for splitting resources):

$AwriteSet \cap BwriteSet \subseteq BwriteLast$
$AreadSet \cap BwriteSet = \varnothing$
$BreadSet \cap AwriteSet = ShareSet$

The first property says if A and B write on an object, B must write after A. In other words A cannot write on B's output. The second property says A cannot see B's results. The third property says B may see some of A's results. These properties guarantee A is ordered before B. If ShareSet and BwriteLast are nil (in this sense, A and B are independent) A and B can be ordered anyway or they can be run completely individually.

If ShareSet or BwriteLast are not nil (this is called a serialized situation), A can be ordered before B. In this order, objects in ShareSet must be unchanged (by A). On the other hand, B with using uncommitted A's data can commit. If A aborts, B must be aborted too (because B can read A's written data). The main aim of Split transactions is commitment of one of the Split transactions (A in this example) and releasing useful results for the main transaction which can continue without interruption.

Split transaction introduces several important ideas that are used by the other models;

1. Adaptive recovery: Tasks (application) will be committed by strings of (sub) transactions that will not be disaffected by some failures.
2. Reduce Isolation: Releasing resources (data items) by committed parts of transaction.
3. Ordered access to resources.

2.1.6. Flex Transaction Model

A Flex Transaction is a composition of a task set. Each task is equivalent to a series of sub-transactions and it can commit successfully just when one of these sub-transactions is committed [BEK93]. A Flex transaction will commit when all of its tasks finish successfully. In the Flex model, the main transaction can use a description of the dependencies between sub-transactions (task's sub-transactions).

Three important dependencies in the Flex model are: 'Failure Dependency'; 'Success Dependency'; and, 'External Dependency'. Failure and Success dependencies determine the order of running sub-transactions in different situations (failure and successful finishing). External dependency shows dependencies of sub-transactions to transactions that do not belong to the main transaction. These dependencies make the creation of a compensating mechanism possible.

The Flex model was implemented in VPL (Vienna Parallel Logic) for the first time.

2.1.7. Cooperative Transaction Model

The Cooperative transaction model [HPS93], [RaN99] is used for several different structure / mechanisms. A Cooperative transaction model is a hierarchy of sub-transactions (similar to nested transaction) where there is a possibility for releasing data-items in sub-transactions before commitment of the main transaction (for reaching the result, cooperation between transactions is necessary and this cooperation is in effect the sharing of uncommitted data-items).

The common architecture of Cooperative transactions enables release of partial results for other transactions (Figure 2-4). However, how they do this, what rules must be applied, which structure design is considered for concurrency control and recovery management and so forth, can be completely different in each customized model.

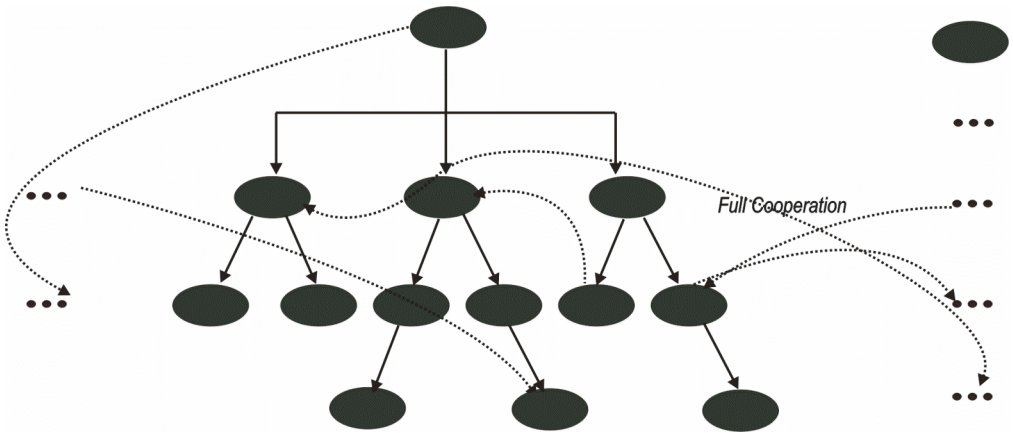


Figure 2-4 Cooperative Transaction Model

In real situations, for implementation of Cooperative model, different structures are selected. The limitation for applying the customized model (suitable for a specific application environment) for saving

database consistency and integrity must be considered [QXL02]. The important points of each Cooperative transaction models are the structure of:

- Logs (data structure) for chains of dependent sub-transactions (those releasing data-items and/or using the other's released data items).
- Recovery Management implementation
- Concurrency control structure
- Optimizing overheads and logs
- Mechanisms for breaking loops/ deadlocks and any other illegal events

2.1.8. Coordinating Transaction Model

The idea of categorising sub-transactions in the Cooperative Transaction Model [QXL02] inspired a new type of model, called the Coordinating transaction model. In the coordinating transaction model, sub-transactions are divided to two categories; Control nodes (sub-transaction) and executive nodes (sub-transactions). Control nodes can determine the order of running/termination of their children and/or how (validation/mechanism) they could release their results to their siblings or even to other transactions (partial results) [QXL02] [DTL+03].

Coordinating transactions work based on grouping transactions with three types of management (in the commitment condition). Database consistency is achieved through the concept of scheduling transactions (grouping). There are two different notifications for 'writing' (modification/creating). The written data items can be shared between group members, or can be private to modifiers (owners). There are different implementations and derived models based on the Coordinating transaction model. One of the best-known developed versions of the coordinating model is Mega transaction. We will review this model in the following section with more structural detail.

2.1.9. Mega Transaction Model

The Mega Transaction Model [Hag96] is an open nested coordinating model that was introduced for special distributed environments (aircraft design projects, Open-ended projects, CSCW etc). The structure of the model is a tree that consists of compositions of schedulers in the root and middle nodes, and executive / delegations (agents) in the leaves (Figure 2-5 symbols are from chains of researches at [HPS93], [Hag97], [Hag95], [VeP98], [PDB+97], [PDH+96]).

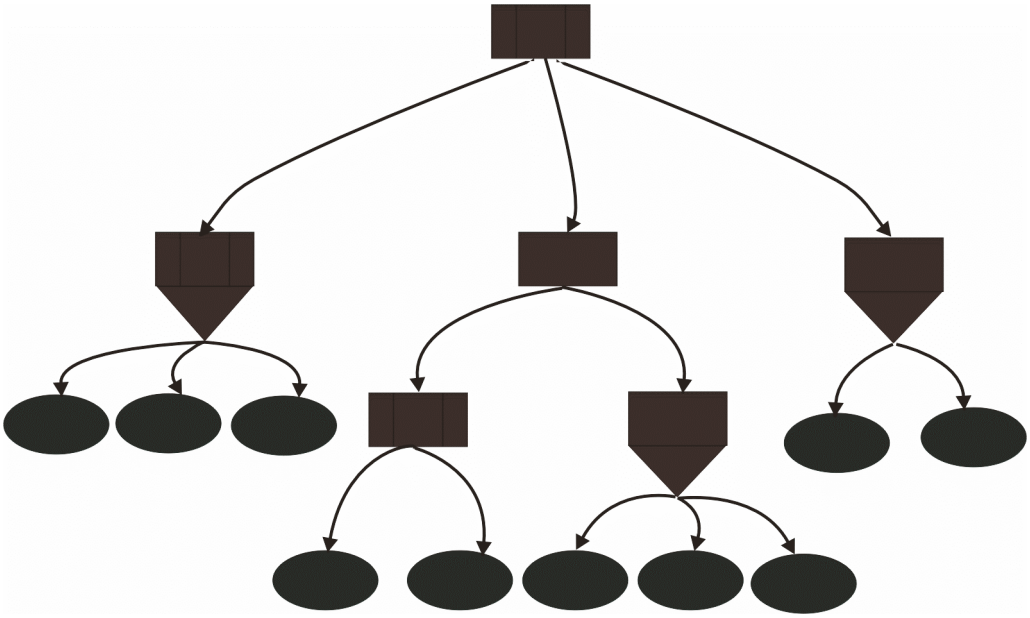


Figure 2-5 Mega Transaction Model

The possibility of any composition of schedulers gives a powerful advantage to this model for covering a huge range of applications. Different derivations of this model have bee implemented and customised for different requirements.

Mega Transaction uses compensation as a part of Recovery Management [Hag97], [Raz99], and cooperation between different transactions is available in any level (by considering the integrated recovery manager). Delegation is considered as a blanket mechanism for providing heterogeneous facilities. The restriction of dependencies to the scheduler gives the opportunity for optimization [PDB+97], [Hag97] but this does introduce some side effects for the coverage of real-time applications. Mega transaction could be considered as a wise counsel for designing customised Coordinating transaction models in different environments (depending on requirements).

2.1.10. DOM Transaction Model

This model is an improved Cooperative transaction model. It was designed in GTE laboratories for the management of distributed object projects in a heterogeneous environment [Elm94]. Some versions of the DOM transaction model are popular as a strong structured transaction model on the Internet. Theoretically it can have different heterogeneous components such as different types of Database structures (Relational and Object-Oriented data sets) and even non-Database parts (like different file/record based systems).

DOM (Distributed Object Management) includes different structure blocks that can combine for creating more complex structures. Depending on requirements; these blocks can act like a conventional transaction model, a nested transaction model or even an open nested transaction model. DOM provides these facilities through a structure that includes five different levels transaction:

1. **Top transactions:** A Top transaction is a root of the (closed) nested transaction model. It is visible to the system where it is run. Top transactions can be defined by the user or be the result of other transactions. They can be mixed by Multi transactions but they cannot release partial result by themselves (this can be done by Multi transaction).

2. **Multi transactions:** These are for long-lived transactions and the accessing of information from external databases (over which DOM cannot have any control). Sub-transactions within a Multi transaction can potentially release their results outside of the Multi transaction (to other

transactions outside of the main transaction). Transactions in a Multi transaction can in turn be a Multi transaction or a Top transaction.

3. **Compensating transactions:** There is a compensating transaction for each Top transaction in a Multi transaction. In fact the compensating transaction is a Top transaction that can undo the effect of other Top transactions. When a Multi transaction aborts, a compensating transaction of each committed Top transaction must be run.

4. **Vital and Non-vital transactions:** Multi transaction components (sub transactions) can be Vital or Non-vital transactions. If a Vital transaction aborts the main Multi transaction must be aborted, but if a Non-vital transaction aborts, the main Multi transaction can continue.

5. **Contingent transactions:** Contingent transactions can be alternatives for some main transactions in a Multi transaction. If a transaction aborts and has a Contingent transaction, the Contingent transaction will be run automatically.

2.2 Analysis Parameters

There are three different parameters in structuring the design of any transaction model: what is the main structure of model; which type of transactions are supported in the model; and, how it supports/optimises distribution?

2.2.1. Model Structure

At the first level, advanced transaction models are categorised into two different structural classes [Elm94]: those models that try to solve the long-life problem by making a hierarchical structure of tasks (and probably controls); and those that follow a linear structure of tasks. The Cooperative model, Mega transaction, DOM are some models that are using a hierarchical structure, whereas Saga transaction, and Extended Saga use the second category.

The hierarchical structure had problems for a long time in applications with linear structure (for implementation overhead that is for saving a tree structure in hierarchical foundations), which the coordinating models tried to solve by introducing a serial controller in the middleware nodes (sub-transactions). On the other hand, the linear structure has many foundational problems. Mostly the latter are derived from Saga (at least the main Saga motivation which tries to divide long-lived transactions into a series of compensating sub-transactions) and therefore they have same problems (2.1.2).

Another categorization for model structure is the ability for supporting dynamic decisions in the model. This was the main reason for defining the Split model. Dynamic structure is one of the important factors for supporting heterogeneous environments and optimizing the network cost (if suitable structure can be defined as ADT that could have possible implementations).

2.2.2. Transaction Types Support

The variety of sub-transactions is one of the most important items not only for model generalization but also for performance/flexibility of models in different environments. Some of these sub-transactions are contingent transactions, compensating transactions, Vital/Non-Vital transactions.

Contingency helps transaction models to avoid some failures by selecting alternative sub-transactions or improving their performance by the abortion of slow transactions. The efficient usage of these sub-transactions is in coordinating models when there is control node [Elm94], [TeI03].

A compensating mechanism not only gives the possibility of safe release of partial results to a transaction model, but also can help recovery management when failure occurs. This can even work in some models where the whole recovery will be done just with using this mechanism (as in the Saga model or DOM transaction model).

Vital/Non-Vital transactions can give some type of priority to sub-transactions and improve the performance of a transaction model. This also improves robustness as abortion of the main transaction will not happen when a low priority (Non-Vital) sub-transaction aborts.

2.2.3. Distribution

The methodology for distribution can be a main factor for performance calculation (by optimizing network costs). The mechanisms for code migration, selective serial process in contrast to parallel processing, global information, control logs for distribution paths and etc are some of the different subjects in this parameter. One of the newest subjects in transaction distribution is mobility ([SHD+01], [Liao, Liu, Wang and ChuJi2003], [vdMDD+03], [WZB+01]) which introduce new mathematical modelling for localization and transaction migration in large scale.

2.3 Conclusion

This chapter has surveyed the main distributed transaction models. The DBE requirements for support of long-life transactions, with the corresponding needs for releasing partial results, and cashing potentially useful results on failure of a transaction, have been partially addressed in many of these models. However, non e of them provides a full and efficient solution.

We include a summary of their main properties in Table 2.1.

Transaction Model	Structure	Sub-transactions	Distribution
Conventional	-	-	-
Nested Transaction	Hierarchical Static	Contingency Non-Vital	Central control
Sagas Transaction	Linear Static	Compensating	Locality
Multi-level	Limited Hierarchical Static	Contingency	Central Control
ConTract Model	Linear Static	Contingency Non-Vital Compensating	Not any specific method
Split	Linear Dynamic	-	NA
Flex	Hierarchical Dynamic	Contingency Non-Vital Compensating	NA
Cooperative	Hierarchical Static	Contingency Non-Vital Compensating	Mobility (migration)
Coordinating	Hierarchical Dynamic	Contingency Non-Vital Compensating	NA
Mega Transaction	Hierarchical Dynamic	Contingency Non-Vital Compensating	Delegation
DOM	Hierarchical Dynamic	Contingency Non-Vital Compensating	Open to design

Table 2-1 Transaction Model Parameters

3. Service Oriented Computing and Business Transactions

Web services exemplify the hypothesis of *Service-Oriented Computing* (SOC) [PaG03], [Pap03], [CKM+03] which is concerned with applications from different providers offered as services that can be used, composed, and coordinated in a loosely coupled manner. In this paradigm, web services are fundamental elements for developing applications/solutions. They are self-describing, platform-agnostic computational elements that support rapid, low-cost composition of distributed applications.

Services perform functions, which can be anything from simple requests to complicated business processes. But the minimum requirements which they have to provide are (base on Papazoglou's paradigm of SOC):

- *Technology neutrality:*
  - They must be invocable through standardized lowest common denominator technologies that are available to almost all IT environments. This implies that the invocation mechanisms (protocols, descriptions and discovery mechanisms) should comply with widely accepted standards.
- *Loose coupling:*
  - They must not require knowledge of any internal structures or conventions (context) at the client or service side.
- *Location transparency:*
  - Services should have their definitions and location information stored in a repository such as UDDI and be accessible by a variety of clients that can locate and invoke the services irrespective of their location.

3.1. Service Oriented Architecture

The actual architectural approach of SOC is called SOA and is particularly applicable when multiple applications running on varied technologies and platforms need to communicate with each other. In this way, enterprises can mix and match services to perform business transactions with minimal programming effort. SOA is a way of reorganizing software applications and support infrastructure into an interconnected set of services, each accessible through standard interfaces and messaging protocols. The basic SOA is not only about architecture of services, it is a relationship of three kinds of participants: the *service provider*, the *service discovery agency*, and the *service requestor* (*client*). The interactions involve '*publish'*, '*find'* and '*bind'* operations (Figure 3-1).

In a typical service-based scenario a service provider hosts a network accessible software module (an implementation of a given service).The service provider defines a service description of the service and publishes it to a client or service discovery agency through which a service description is published and made discoverable. The service requestor uses a find operation to retrieve the service description typically from a discovery agency, i.e., a registry or repository like UDDI, and uses the service description to bind with the service provider and invoke the service or interact with service implementation.

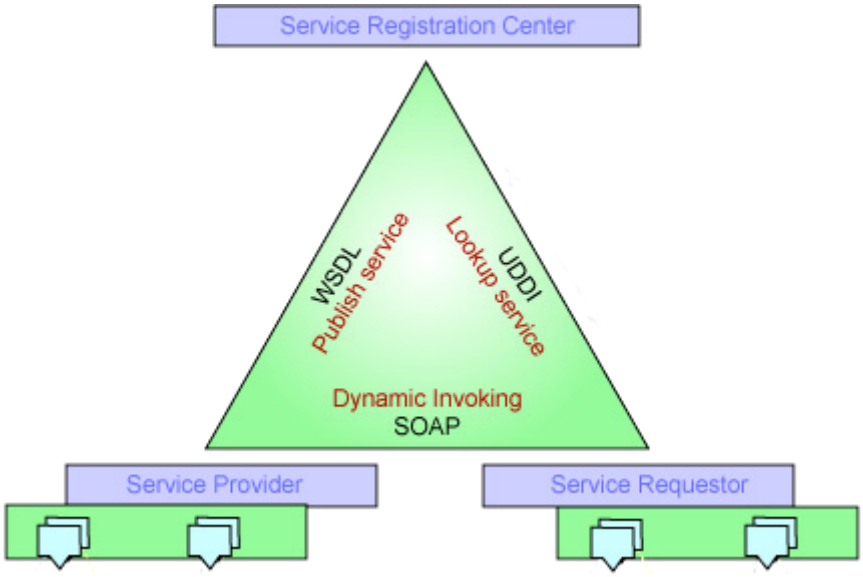


Figure 3-1 Service Oriented Architecture

3.2. Service composition, Workflow and transaction model

Figure 3-1 apart of showing the basic of SOA, does not show the reality/complexity of the situation (expectations) of web services. As a simple example multiple applications running on varied technologies and platforms need to communicate with each other. In this way, enterprises suppose mix and match services to perform business transactions with minimal programming effort and providing a consistent environment. But the story will not finish in this level of complexity, because the service environment is highly dynamic and normally large; new services may become available daily.

Service composition has several characteristics which make it very different from workflow integration, purely transactional implementation and software component integration:

- The service space is normally large and dynamic and new services regularly become available;
- There are usually multiple services which offer seemingly similar features but with some variation. For example:
  - different parameters for invocation,
  - different access interfaces,
  - different costs,
  - different quality
- Composition of services needs to be generated on the fly based on the requests of the customers;
- There are more facets of e-services that need to be considered than those of workflows or components when it comes to integration, e.g., data, functionalities, security, protocols, etc.

3.3. Current Status Analysis

There is a distinguishing between two broad aspects of services [Pap03]: *service deployment*, which is subjected to our transactional service composition, versus *service realization* (Figure 3-2). The service realization strategy involves choosing from an increasing diversity of different options for services, which may be mixed in various combinations.

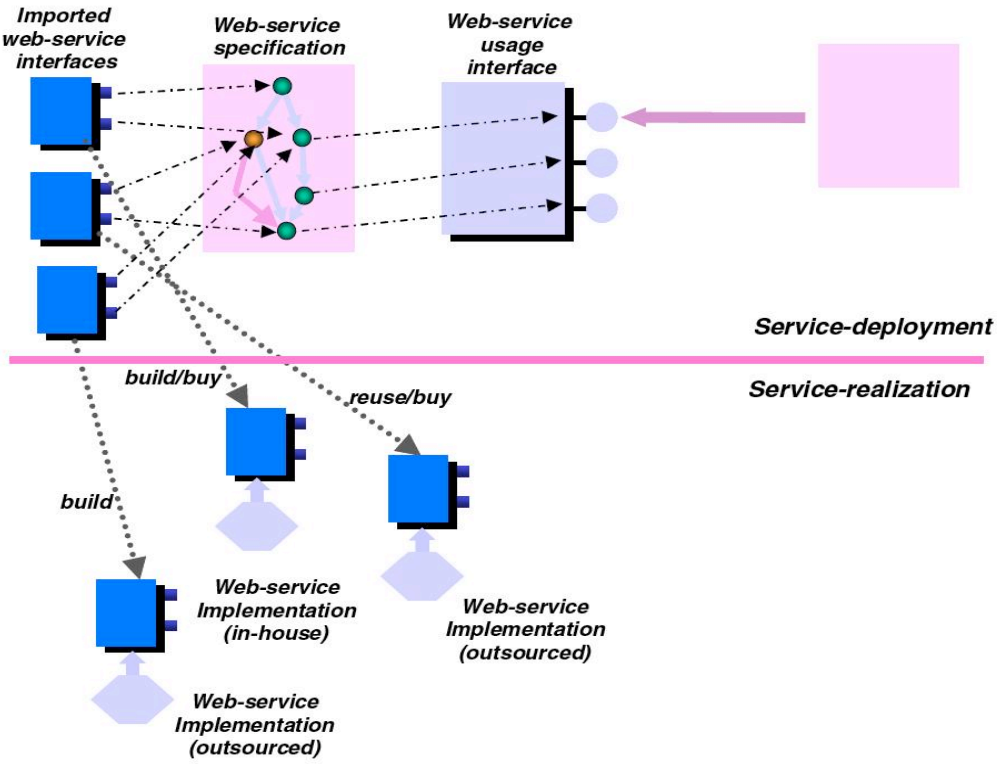


Figure 3-2 Service realization and deployment in web services

In the service realization terminology, *Service implementation* can be very involved because on many occasions organizations rely on single monolithic programs to represent a single service or service method implementation. But very often in order to fulfil the functions of a service multiple programs are involved. Application composition and integration very often is involved in fulfilling the service. At the logical level, what we consider is: 'there is a business function implemented in software somehow and this is the interface to it'.

The service in SOA is designed in such a way that it can be invoked by various service clients and is logically decoupled from any service caller (*loose coupling*). This means there are no assumptions of any kind in the service as to what kind of service a consumer is using and for what purpose and in what context. But the service callers are very much coupled with the service as they know what the services are, what they call and what they can accomplish. This can be considered as one of complexities of a service (transaction) composition manager, which has to encompass a polymorphous nature and fulfil the service callers tight coupled to them!

In service deployment, *Service descriptions* are the most important part. They are used to advertise the service capabilities, interface, behaviour, and quality. Publication of such information about available services (on a service registry) provides the necessary means for discovery, selection, binding, and composition of services. Later on, the importance and usage of service descriptions will be shown when services shall be used or composite services are created.

Based on this analysis, the research has to consider several issues such as transactional modelling, workflow manager, business models and service composition. According to the nature of web services architecture, taking into account a boundary for these issues is quite difficult and as a reflection of this fact most researches avoid to consider any boundaries. However, in practical situations somehow most web transaction models are limited to a framework such as a workflow layer (typically reflected on different version of BPEL such as BPEL4WS).

This separation causes two complexities. On one hand, a practical nested depth for workflow comes from the transactional model's ability for supporting that, therefore it is considered with some predefined limitations (for reducing transaction layer complexity). On the other hand, a workflow layer needs a reliable consistency model taking into consideration support for the long-term nature of business activities (this causes one the highest conflicts between two layers: transaction and workflow layers which mostly developers move on to completely separation between two layers [Lie06]).

One of the important side effects of first case is the failure to meet the increasing demands of business requirements. Based on practical experiences (for example within DBE), businesses have to redefine their business activities (transactions) to cover the limited nested compositions allowable in a workflow layer. As an obvious result, not only do a considerable volume of activities have to be done manually, but also many complicated computing actions have to be involved in defining the business models (these extra costs and complexities impede small and medium businesses from joining/using SOA benefits).

The important side effect of the second case is the necessity of adding an intermediate layer (mostly as a separate coordinator role [VZG+05], [FuG05], [CCJ+04], [ZLB04], [DTL+03] and [CCC+03]) for providing the primary precondition for successful running/committing business activities. In the implementation view, this layer appears as a coordinator which usually is provided by a platform. The critical demand of this architecture is a coordinator with a crucial role in any business activity (which has to in a highly reliable platform with high performance abilities; in case of processing, bandwidth, memory and etc). As a result, small and medium businesses have to rely on centralised (limited decentralised) architectures provided by large organisations such as Microsoft[2], Sun[3], and IBM[4].

Another effect of this terminology is excluding two important high level composition classes (explorative and semi-fixed compositions) [4.3] in the workflow and transaction layers. The lack of these classes not only reduces the ability of businesses for fullfilling the primary consumers requirements, but also highly reduces the possibilities for fair competition between SMEs and large businesses (retailers). In this case, even proposed solutions for covering these two compositions classes [YPH02] are reliant on a centralised registry, which cannot maintain the monopoly of large businesses.

In the reusability view, current topologies are futile for providing any facilities. After commitment of a transaction, all of the nested complex service compositions, consistency models and validity logs and etc (in the best case) will be archived as a record of a company's business activities. But the ability for reusing these costly structures or using them as a strategic resource for the introduction (inspiration of) more complex business demands are impossible. In this case, even the theoretical introduction of ESOA at 2003 [Pap03] could not challenge the current web services world.

As a result, the current architecture warranties the monopoly of huge businesses and any strains for revolutionary changes from them is unimaginable. On the other hand, the primary, centralised and low-performance service discovery (such as UDDI) has completed the rectangle of SOA in favour of these companies. In this sense, even advertising a new service by SMEs faces a bottleneck which can be solved only by a costly solution such as external advertisement through channels of these huge businesses (such as one-click, yahoo business directory or other costly policies).

In recent years, nationally, regionally and internationally, the effort for better supporting SMEs has started. But it seems the secret of success is full analysis of business requirements plus an integrated system which does not leave any hole for centralised (semi-centralised) processing. With the high bandwidth of network connections (specifically on the internet), several successful researches on the business and computing areas present an optimistic outlook for design/implementing a fully distributed model.

We will now look at the current situation with transactional models in more detail.

[2] http://msdn.microsoft.com/library/default.asp?url=/library/en-us/dnglobspec/html/wsatspecindex.asp
[3] http://developers.sun.com/techtopics/webservices/wscaf/primer.pdf
[4] http://www-128.ibm.com/developerworks/webservices/library/specification/ws-tx/

3.4. Web services Transaction models

In 2001, a consortium of companies including Oracle, Sun Microsystems, Choreology Ltd, Hewlett-Packard Co., IPNet, SeeBeyond Inc. Sybase, Interwoven Inc., Systinet and BEA System, began work on the Organization for Advance Structured Information Systems (OASIS) Business Transaction Protocol (BTP), which was aimed at business-to-business transactions in loosely-coupled domains such as Web services. By April 2002 it had reached the point of a committee specification [CDF+03] and [FDF+04].

At the same time, others in the industry, including Microsoft, Hitachi, IBM, IONA, Arjuna Technologies and BEA Systems, released their own specifications: Web Services Coordination (WS- Coordination) and Web Services Transactions (WS-AtomicTransactions and WS-BusinessActivities) [CCJ+04]. Recently Choreology Ltd. has started to make a joint protocol which tries to cover both models and by mentioning their problems in several detailed reports, tries to solve them [FuG05].

3.4.1. WS-Protocols (WS-Tx)

WS-Protocols set consists of the three protocols;

WS-Coordination [CCC+03], the most important of WS-protocols which it defined as an extensible coordination framework and its coordinating (sub) transactions in any levels.

WS-AtomicTransactions [CCC+04a], leveraging WS-Coordination(WS-C) for use with systems aware of ACID properties. Actually it cover atomic transactions which a conventional commit protocol.

WS-BusinessActivities [CCC+04b], designed for support of long-lived activities. It tries to provide enough foundations for applying long-term business activities in the transactional manner (which it is called WS-BusinessActivities).

WS-AtomicTransactions and WS-BusinessActivities use WS-Coordination framework as the coordinator protocol.

WS-Coordination specification

In WS-Coordination specifications, three roles are described for communicating parties: Initiator, Participant and coordinator. The entity aiming for a consensus among multiple Web Services is playing the Initiator role, the entity offering some service that needs to be coordinated during the interaction has Participant role and the coordinator is an entity coordinating the communicating parties to achieve the consensus.

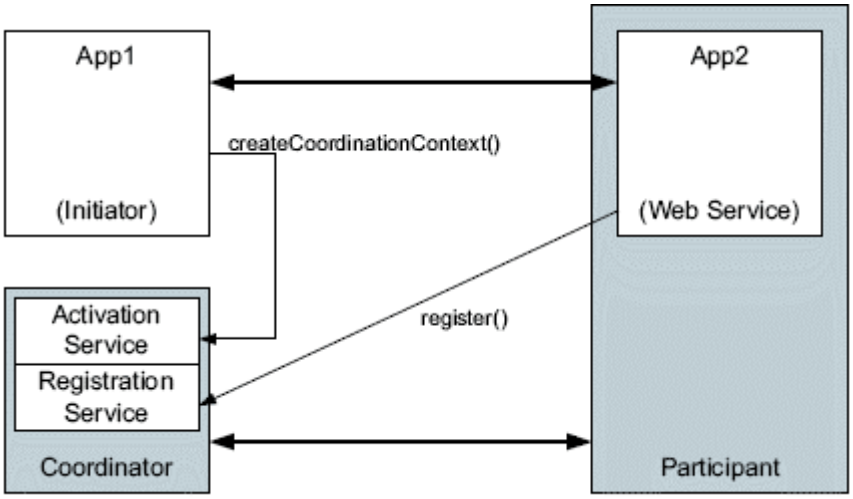


Figure 3.3: Coordination roles and message exchnages

The message exchanges have to be introduced (in the specification body), which requires the 'Activation' and 'Registration' of participants. The CoordinationContext is acquired from the Coordinator's Activation Service for the 'Activation' phase. The CoordinationContext is attached to business messages being exchanged between the communicating parties, embedded in a SOAP header.

The aim in the registration phase is to form a logical connection between Coordinator and Participant which will be done by negotiation and exchanging endpoint addresses of the Coordinator's and Participant's protocol services (in this sense, the message flow over this logical connection depends on the coordination protocol being used and is not part of WS-Coordination specification.).

WS-AtomicTransactions and WS-BusinessActivity specification

The WS-AtomicTransactions specification uses a conventional two phase commitment protocol for coordination [BeN97]. In contrast, the WS-BusinessActivity specification defines two different coordination protocols; the Business Activity with Coordinator Completion (BACC), below, and Business Activity with Participant Completion (BAPC).

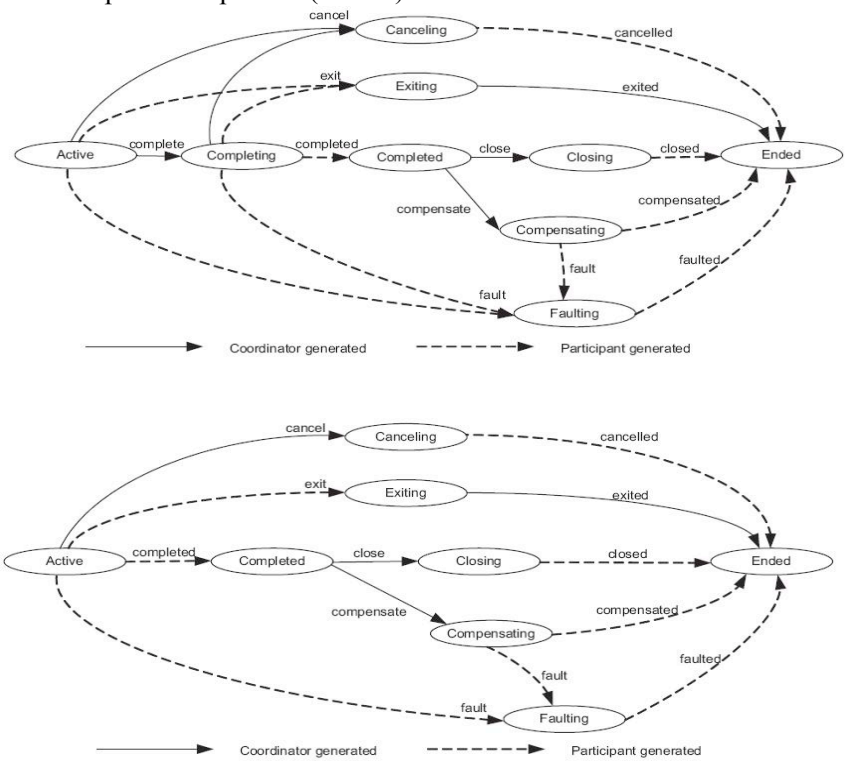


Figure 3.4: The WS-BusinessActivity coordination protocols

The differences between conventional 2PC and BAPC/BACC come back to the manner of their commitment; the first phase of BACC and BAPC, is used for exchange of business messages between the parties, then in case of BAPC, the end of the first phase occurs when the Completed message is sent from 'Participant' to 'Coordinator', indicating that the 'Participant' has completed processing and stored all data persistently. The second phase is used for confirmation or negation of results achieved during the first phase.

The difference between BAPC and BACC is the procedures for business activities; the BAPC is designed for activities in which the decision about a transition from the first to the second phase can be made by the 'Participant', but the BACC is designed for activities in which this decision is made by the Coordinator.

WS-BusinessActivity contradictions:

Based on analysis of Vogt and Zambrovski [VZG+05], WS-BusinessActivity can only support "do-compensate" behaviour patterns [FuG05] for the Participants. In the "do-compensate", the 'confirm state' has priority over the temporary states. But in the other behaviour patterns (such as "provisional-final" and "validate-do" [FuG05]), the participant establishes a (temporary) "complete" state of the application data that will be changed to the confirmed state if and when the 'Close' message is received .

As a result of applying "do-compensate", WS-BusinessActivity is not able to transit to not 'Faulting' state from 'Close' state which means any action can not be done on data when the system is in the close state! Forcing this limitation, by breaking the autonomy of local platforms and prescribing the internal behaviour to realisation level of services [3.3] has despoiled one of the primary foundations of SOA paradigm.

On the other hand, the protocol authors made some decisions about the internal build-up of communication parties as described in [CCJ+04]. The tightly-coupled Coordinator and Initiator as well as Participant encapsulating both business and transactional logic are examples of that. These presumptions are against of SOA characteristics (loose coupled and highly dynamic [Pap03], [YPH02]) particularly web-services nature, especially when SMEs are involved in a transaction.

The other side effect of the identical roles of 'Close' in "do-compensate" pattern behaviour, is the wide limitations of alternative scenarios, which not only keep a tight rein on the coverage of business requirements but also by considering the high dynamic nature of SMEs they can not be a candidate for the coordinator. By considering the designer assumption (tightly-coupled), necessity for stability, on one hand will force the system for choosing few highly stable nodes as the coordinators which in the best case will produce a decentralised system (not fully distributed as the designers were promised [CCJ+04]). On the other hand, SMEs even cannot be candidates for Initiator as well as Participant roles.

3.4.2. OASIS & BTP

The Organization for the Advancement of Structured Information Standards (OASIS) has developed the business transaction protocol (BTP) to provide a model for reaching the requirements of SOA design with emphasis on long-running collaborative business applications. BTP is designed to support long-term (running) transactions and a transaction concept that goes beyond data-centric transactions [FDF+04].

3.4.2.1 BTP Structure:

BTP by using a nested transaction model as its infrastructure has a transaction tree. In contrast to most of the transaction models, BTP uses an open-top commit [DTL+03] protocol (sometimes called a three phase protocol) which lets the application use an extra phase (as an intermediate phase) between phases. This ability is applied by expanding the range of available verbs "prepare", "confirm", "cancel", etc to include explicit control over both phases.

3.4.2.2 BTP transactions types:

BTP by using a tree structure can mix different types of (sub) transactions. To address the specific needs of business transactions, BTP makes a separation in categorization (sub) transactions by introducing two types of extended transactions:

Atom: The atom transaction type uses a classic two-phase commit protocol and the transaction is considered as the "logical unit of work" which achieves a consistent result. In this context, a unit of work can be a number of business actions which is measured as an ACID transaction (as an important emphasised feature in BTP [FDF+04] should either be completed together or reversed to a state as if not executed). The all-or-nothing semantics associated with the atom support precisely this style of interaction. Where a single party cannot make good on a business level agreement, or there is a technical failure that prevents that party from providing its service, the atom abstraction ensures that all parties involved in the interaction are freed from their obligations [DTL+03].

Cohesion: By relaxing Atomicity (in ACID properties), a 'Cohesion' transaction tries to overcome the long-term (long-running) transaction problem. It permits the terminator of transaction to confirm or cancel the selection of work based on the business model (high-level business rules). Actually, cohesion architecture tries to provide an abstraction for multiparty B2B interactions and it has a high reliance on business model ontology. Based on this logic, cohesion (in contrast by conventional transaction model) according to business model can provide participants with different outcomes which some might confirm while others cancel. Cohesion's customised version of two-phase protocol, from the high level specifies (in intermediate level) precisely which participants to prepare and which to cancel. As during transaction life time, it might meet conditions that allow it to cancel or prepare some of its units, although the cohesion might take several hours or days to arrive at its confirm-set (the set of vital participants that must confirm to successfully termination of the transaction). When the confirm-set is determined, the cohesion collapses down to an atom, and this final phase all members of the confirm-set see the same outcome. In this way, the action of Cohesion is categorised as conventional nested transactional model [2.1] with slightly flexible for commitment protocol but same view for the long-term transaction tree.

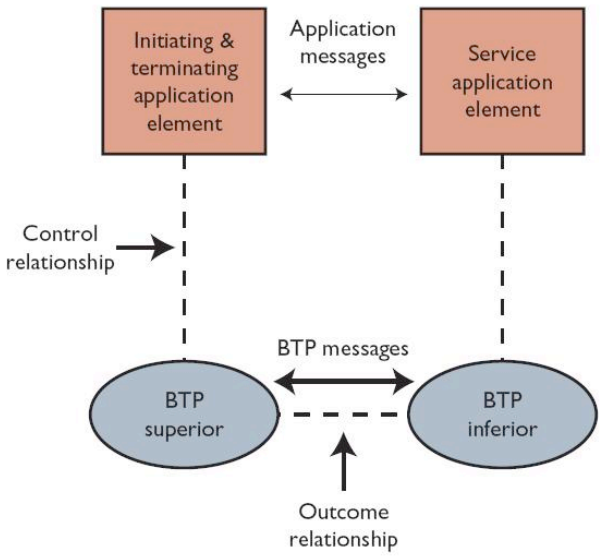


Figure 3.5: BTP Actors

3.4.2.3 BTP roles, actors and cast

A BTP role is specific to a particular relationship between software agents (known as BTP actors) participating in a business transaction. Two types of relationships exist between different roles in the BTP model; Inferior and superior. Inferior (applicable on Atom/Cohesion) is responsible for reporting to the

superior that it is prepared for the outcome, regardless of whether the associated operations' provisional effects can be confirmed or cancelled. Superior (applicable on coordinator/composer) is the party that coordinates the transaction. A superior gathers reports from its inferiors and must determine which to cancel or confirm. If the transaction is an atom, BTP's predefined rules require that every inferior be confirmed unless at least one has signalled that it wishes not to, in which case all will be cancelled. If the transaction is 'cohesion', it will not rely on just the decision made by inferiors and it has to be a combination of the decisions made by inferiors and its own business rules which is highly dependent on the business model.

Therefore BTP as a nested transaction model, stresses each inferior has only one superior, but a superior can have multiple inferiors. The tree of such relationships could thus be wide, deep, or both. According to transaction's relative position in the transaction tree (a superior coordinates an inferior). This is important because an actor's role could change during a transaction and defer different behaviour for the same logical entity.

Several other actors are introduced on BTP for handling more specific roles. One of the most important actors as is superior for an atom is called a coordinator; it is responsible for supervision of the atomic two-phase completion protocol. For cohesion, the most significant superior is a composer; (a sub-coordinator (atom) or sub-composer (cohesion)). The other important actors are 'transaction factory' (which is typically a Web service that creates the context for a business transaction and manages the coordinator associated with it), 'initiator' (which starts a business transaction at an application element's request), 'participant' (which is the entity that eventually does the real transaction work on behalf of the Web service), and 'terminator' (is an application element that inculcates the coordinator of a business transaction to either confirm or cancel the transaction). In the next part, a classic simple scenario, which it is mentioned in several references [DTL+03], will show the main structure of the model.

Simple scenario:

- **Beginning a transaction**: by sending a request (for a new business transaction) from initiator to a factory (transaction factory) a transaction begins, two different situations can arise:

  - If it is a new top-level transaction, the composer or coordinator will be the decider (the term given to the most senior coordinating entity for a transaction).
  - If the new business transaction is made the direct inferior of an existing transaction (by specifying a superior transaction as its parent in the consequent begin message), the request creates a sub-composer or sub-coordinator that is ultimately controlled by its parent composer or coordinator.

The factory sends a begun message to the initiator in reply to the begin request, indicating the related context for the new business transaction.

- **Enrolling a participant in a transaction**: For interaction in the transaction, the new transaction has to register what it is done by sending an enrol request to a superior to enrol an inferior and the superior answers with an enrolled message to point out that the inferior has been successfully enrolled in the transaction.

- **Confirming a transaction-phase one**: when association phase is finished, the first phase of 2PC confirmation protocol can be initiated by the terminator. A preparation message will be sent by the superior to its enrolled inferiors that have sent neither a cancelled nor a resign message;
  - If a coordinator or composer that has been requested to confirm has only one remaining inferior in the confirm-set, it can delegate the confirm-or-cancel decision to that inferior by sending a confirm_one_phase message, which avoids performing the two-phase exchange.
  - If the inferior is able to proceed with the work asked of it, it sends a prepared message to its superior — either unsolicited or in response to the superior's prepare message — but only once the inferior has determined that the operations associated with it can be confirmed or cancelled as instructed by the superior.

- **Confirming a transaction – phase two**: If the first phase of the completion is successful (that is, either all participants agree to confirm in the case of an atom, or the participant decisions plus business logic still allow confirmation for a cohesion), the terminator sends a confirm_transaction message to a decider to request confirmation of the business transaction, which then causes the confirm message to be propagated to each enrolled participant in the case of an atom, or to a subset of the enrolled participants (specified by the inferiors- list parameter of the confirm_transaction message) in the case of a cohesion. On receipt of a confirm message, an inferior sends a confirmed message once it applies the confirmation — whether in reply to confirm, after making an autonomous confirm decision, or in response to a confirm_one_phase. A decider sends a transaction_confirmed message to a terminator in reply to confirm_transaction if all the inferiors in the confirm-set are able to complete their associated work (and, for a cohesion, all other inferiors cancel).

3.4.2.4 BTP unsolved challenges:

The first version of BTP, was not specifically about transaction for web services (the intention was it can be also used in other environments) [CDF+03]. Actually BTP defines the transactional XML protocol and must specify all of the service dependencies within the specification. Recently Choreology Ltd. has tried to point out and modify the model and consider possible connection between BTP and WS-Tx for customising the model for web services [FuG05] (the reflection of these works can be seen in the latest version of BTP too [FDF+04]).

As mentioned in 3.4.2.2 BTP is a conventional nested transaction model, which means transactions can be nested and internally they can share uncommitted results but they are isolated for the transactions on the environment. That means the partial results can not be covered in this model.

The other important point is the relationship between superior (coordinator/composer) and inferior. As there is not any specific mechanism for alternative service composition in (specifically) composers and (with less side effect) coordinators, the relationship should be tight-coupled (as it is not mentioned directly in system specification) or the whole transaction will face a regular cascading roll back danger (as a very simple example, we can check the nature of SMEs which it is dynamic with loosely coupled binding which means accessibility and characteristics of provided services can be changed regularly, which means composer/coordinator had to be roll back and in the case of composer because it is not isolated from other composers of the same transaction, it might shared some its results with them and all dependent composers had to be roll back/restart too).

Another side effect of last problem is the probability for dead-lock. As it was not mentioned the reaction of the transaction to dead-lock (and even worse, it is not clear the algorithm for detecting deadlock), this lack of this issues can be consider as an important weakness of the model but worst than that is even if there was an internal structure for detection or prevention of deadlock with possibilities for regular abortions (rollback/restart) the probability for dead is high.

There are more issues such as recovery management algorithm, fully distributed concurrency control and etc which are unanswered in BTP.

3.5 Service Composition and transaction model terminology

Based on our expectation of transaction model, we have to clarify the varieties of service compositions which we cover. The target is to determining all different possible service compositions and covering all of them for providing proper infrastructure for Business modelling. Otherwise, the minimum result of sacrificing in service composition is necessity for dramatically manual work for applying business model to SOA (or ESOA) architecture and losing the strategic role of transactions for building up an aggregator service compositions [YPH02].

In this chapter, first the popular categorisation of service composition will be reviewed, and then we try to investigate the details of this categorization. The next high level service composition is the subject of next part of this chapter and at the end; we try to analysis the requirements for our transaction model based the relationship between service composition as the model requirements and the effect of them on consistency requirements.

3.5.1. Service Composition

As the service composition will be performed at the service deployment level (3.3), it requires the combination of service descriptions and specification of the ways in which they can combine. Therefore clear service specifications (descriptions) are needed to determine the possible service compositions. A full service description can include [YPH02]:

Service properties, i.e., service general information (e.g., identification, owner), service access information (e.g., service location -URL, the maximum time for a conversation between the service and a service requester, public key certificate), and contact information.

Service ontology, i.e., what is the service about and the terminology that is needed to discover the service.

Service cost, i.e., which is the estimated cost for using the service or the information provided by the service.

Payment, i.e., the way the service receives the payment from the customers.

Actors, i.e., which is the people or organizations who are using the service.

Authorization/security/visibility, i.e., who can see/use what (service contents and functions).

Service contents, which specifies the content and the structure of the underling service, e.g., the attributes, objects, the constraints on use of attributes/objects, etc.

Service capability, which specifies the service structure/components, the conditions of using the service, and the order of component invocation.

Jian Yang, Mike P. Papazoglou and Willem-Jan van den Heuvel introduce a service description language (SDL) which tries to cover all aspects of service deployment needs [YPH02]. Meanwhile its DTD can be found in [HYP01].

Based on a specification such as SDL in [YPH02], service composition can be considered along the following dimensions: data, process, security, protocol. But by considering the main concerns of transaction model, our main apprehensions are *data* and *process* composition. We leave security and protocol compositions as an open discussion on the top of transactional layer. In particular, with process oriented service composition, we shall discuss the following aspects:

- *Order:* which indicates whether the composition is serial or parallel;
- *Dependency:* which indicates whether there is any data or function dependency among the component services;
- *Alternative service execution:* This indicates whether there is any alternative service in the service composition that can be invoked. Alternative services can be tried either in a sequential or parallel manner.

3.5.2. Data and Process Service Composition in details:

In spite of what current technology (in particular) support, we try to cover different theoretical data and service compositions, which based on [YPH02] are;

a) **Data oriented service composition**
b) **Sequential process oriented service composition**
   - **Sequential with commitment dependency (SCD)**
   - **Sequential with data dependency (SDD)**
c) **Parallel process oriented service composition**
   - **Parallel with data dependency (PDD)**
   - **Parallel with commit dependency (PCD)**
   - **Parallel without dependency (PND)**
d) **Sequential alternative composition (SAt)**
e) **Parallel alternative composition (PAt)**

a) **Data oriented service composition.** The data generated from service realization (3.3) level are released to cover different aspects service composition. According to transactional aspects this type of composition can be atomic or even considered as partial result for another transaction which force the necessity of compensability of this type of service composition. Typically Service composition in this case involves reorganization of the information and provides additional functions such as summarization and analysis of data.

b) **Sequential process oriented service composition.** This type of service composition invokes services sequentially. The execution of a component service is dependent on its previous service, i.e., one cannot begin unless the previous service commits. There are two possible scenarios for this composition:

i. **Sequential with commitment dependency (SCD).** The relationship between component services is just sequential invoking orders without any interaction between them, which means one service has to wait for another service to commit before it can be executed. In the transactional sense just the order of execution should be considered in a coordinator.
ii. **Sequential with data dependency (SDD).** This type of composition deals with more operational complexity. One component service relies on other component service's outputs as its inputs. As the result, in transactional view, the serializability of data item has to be applied.

c) **Parallel process oriented service composition.** In this service composition, all the component services can be executed parallel but different scenarios can be considered which can make different situations (implementations) in the transactional outlook:

i. **Parallel with data dependency (PDD).** Component services are executed parallel but one (or more) service(s) must wait for a specific data value from another service to arrive before it continues executing. On the other hand, the data serializability is an issue but there is not order execution in the transaction outlook.
ii. **Parallel with commit dependency (PCD).** This situation happens when two services may go in parallel to a certain extend, but one may not commit unless the other commits first.

iii. **Parallel without dependency (PND).** In this case, all the component services can run and commit in parallel without any interaction between them.

d) **Sequential alternative composition (SAt).** This type of service composition indicates that there are alternative services to be combined, and they are ordered based on some criteria (e.g., cost, time, etc). Each will be attempted in succession until one service produces the desired outcome. In a transactional scenario, this can be considered as an important type for covering loosely coupled and very dynamic environment (we will discuss this in designing our transactional model – Chapter 5).

e) **Parallel alternative composition (PAt).** Contrary to the previous case, alternative services are pursued in parallel. As soon as any one of the service succeeds the other parallel services are aborted (discarded).

3.5.3. High level possible service compositions and effect on transaction concept

On 2002, Jian Yang, Mike P. Papazoglou and Willem-Jan van den Heuvel introduce new service composition, in this section we try to review the effect of this new type of service composition on a web service transaction and feasibility of these high level service compositions;

- *Explorative composition*:
  - The service composition is generated on the fly based on the customer's request. The customer specifies the desired service; specifications have to be translated to the service specifications, then the service discovery (broker) can compare the translated specifications with the potential matches and make feasible composition plans. These composite services can be picked up by customer or he/she can add more specifications to his/her primary specifications. This service composition is one of the best facilities for B2C relationships or B2B (considering service requester is small business) when the service composition requester does not know exact details of its request and/or do not have enough knowledge about service specification or business modelling.
  - With this service composition, customer can choose and even find out his/her desired service, meanwhile there is possibility for giving ranked feedback in the candidates service compositions. But in technical view, it needs reconsideration in the concept of transaction, because this business action not only is interactive (and obviously can not be atomic) but also does not seem to fit in conventional two phased commit protocol.

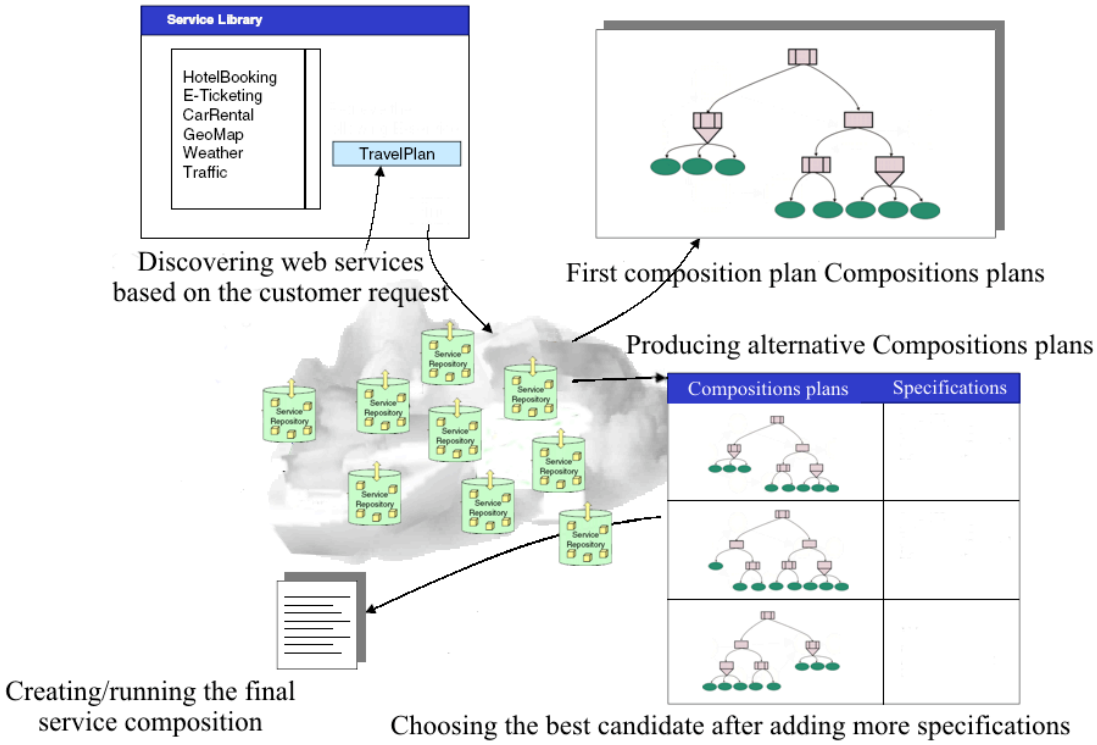


Figure 3.6: Explorative composition

- *Semi-fixed composition:*
  - In this service composition, we deal with a formal composition definition (specification) from requester. But the actual service binding needs to be decided at run time and based on availability of services (or choosing alternative-equivalent services) during composition. Not only does this composition covers the dynamic nature of web services on the internet but it is also an ideal solution for SMEs when availability of services is an important issue. In this case, the definition of the composite service is registered in the marketplace, and it can be used just as any other normal services, i.e., it can be searched, selected, and combined with other services.
  - This type of service composition can lead us to motivation for forward recovery and can considerably improve the reusability in SOA (which it is one of criticisable concept of SOA architecture). In transaction terminology, this type of composition can be covered by the contingent transaction concept. However, one must consider the potential for a paradigm shift from the conventional view of a transaction as a predefined set of actions, to a dynamically instantiated set of actions should be considered.

- *Fixed composition:*
  - This type of service composition is a conventional model of service composition which is implemented and supported in most areas. A fixed composite service requires the component services to be integrated in a fixed way. The composition structure and the component services are statically bound. Requests to such composite services are performed by sending sub-requests to component services. Even in this case still there several unsolved problems and limitations in transaction prospectus such as limitation in depth of nested compositions, centralised coordinators and etc.

3.6 Conclusion

This chapter has analysed a number of transaction models that support composition of services for achieving a high-level business goal. Our analysis has been driven from the perspective of supporting an ecosystem of SMEs in a way that supports and maintains their respective autonomy. In addition, we wish to make the responses to service requests robust against temporary unavailability of one or more components on a composite service through dynamic mechanisms of forward recovery. We propose a transaction model that is customised to the needs of Digital Business Ecosystems in the next section.

4. Coordination and DBE Transaction model

4.1. Preconditions and Motivation

4.1.1. Nested compositions and level of centralisations

In different transactional models and workflow managers, there exist levels of centralizations by using centralised (limited level of decentralised) coordinator [chapter 3]. Inherently this causes limitations on composition, damages automated different types of compositions [chapter 3], reduces platform autonomy and the range of partial results (which it is another side effect for losing full coverage of business requirements).

Therefore, with respect to additional architectural complications, we try to apply a fully distributed structure, which enables full control of autonomy. Our motivation is in using local coordinators but our challenges are in loosely coupled binding, a very dynamic environment, consistency, contrasting with coordinator cooperation and partial results. Considering variations in bandwidth is another huge challenge for the model and distribution of coordinators.

4.1.2. Loosely coupled, dynamic environment and delegations

In the previous section, a loosely coupled and dynamic environment is considered a challenge for local coordinators. Our solution is in using delegation. By delegation, the full control will transfer to another local coordinator, which not only give full autonomy to the destination platform but also provide proper facilities for overcoming a variety of bandwidths.

Part or whole service composition can be delegated to another platform, based on relevancy of provided services on the platform. The destination platform has full control on composition (including applying delegations). The autonomy provided with this ability is not dissimilar to coordinator cooperation (as introducing the distributed concurrency control can overcome the sharing and distribution of data items).

On the other hand, another parameter for delegating the service composition to the other platform can be lack of bandwidth, which means either for traffic bottleneck or low bandwidth connection, the platform can delegate whole of part of service composition to the other platform. With this ability, we can cover B2C relationships, even with low-grade mobile consumer connections (by full transaction delegation).

4.1.3. Data dependencies, partial results and compensations

During different service compositions in a transaction, data dependencies are an important part of consistency. Especially during delegation, this can be complicated. For solving this problem, a new lock (I-Lock) and a graph for keeping the chains of dependencies are introduced. The lock keeps specifications of the owner's platform address and coordinator's identification which after sharing the data, this information will be kept on the graph (Internal dependencies graph).

Another business requirements which is not considered by other web-based transaction models, is releasing data between uncommitted transactions. Despite this situation, it does not happen regularly, but it is a crucial requirement which has to be automated. Otherwise not only does it force the user to design really complicated transaction but also he/she has to change the business model for defining new transactions which does not match with the nature and reality of business. Alternatively the business has to execute an important part of its requirements manually.

For partial result releasing, the conditional commit mechanism is introduced. This mechanism relies on two new introduced structures; C-Lock and a special graph (External Dependency Graph) for this external consistency (these two will be explained in next sections).

4.1.4. Platforms failures, Forward Recovery, Contingencies plan, Timeouts

Platform failures (even inaccessibility of platform in small and medium enterprises) are one of the natural events in web services that can easily cause a transaction to fail. But according to the user specification request, range of accessible alternative web services or even business model, which triggered the transaction, most of these incidents should not cause the transaction to fail (either can be covered by alternative web services or expected by user in a period of time).

For example, if one of the composed services is supposed to be a web service from UK's IBM platform and it is not available, the same web service can be deployed from US's IBM platform. Or if one of the composed services is buying a specific model of goods from a retailer and the retailer web services fails during the service composition (or transaction commitment), an alternative retailer which fulfilled the user description can provide an alternative web service for the transaction (service composition).

Another example is about the small and medium enterprises, which provide services only within specific periods of time per day (according to Sun Microsystems, this is more than 60% of SMEs in Spain and based on their study, much the same proportion in the rest of EU). Therefore in some period of time, any transaction requiring a web service from these SMEs will fail, even when this behaviour is expected and a user could wait for this specific service time period.

In the current web service transaction models and workflow managers, none of these situations are covered; the user must manage them manually. Apart of the nature of these requirements which are relatively new, and not analysed properly by practical transaction models, technical reason for this absence are:

- Most often transaction models and workflow managers use a two phase commit protocol, which in the second phase needs to finalise the transaction by committing all vital sub-transactions [VZG+05], [LiF03], [FuGr05], [FDF+04].
- Compensation mechanisms do not exist or have no practical implementation/design, which can automatically rollback to specific part of a nested service composition.
- Lack of low level implementation in concurrency control for holding data item for period of time without generating a cascading side effect on the whole coordinator consistency model.

First of all, non-vital sub-transactions can be addressed in the model (by using two coordinators which will be explained in the next section). A compensation mechanism is provided through the use of internal and external graphs (next section). The other important element in this proposed model is the consideration of alternative sub-transactions, which is applied by alternative coordinators. By using compensation mechanisms and alternative coordinators, designing forward recovery becomes possible. In any step of a transaction, if a platform fails (and as a result a sub-transaction fails), the model is able to compensate the sub-transaction and try to chose alternative routes for committing the sub-transaction.

Introducing new locks in local-concurrency control, not only provides an effective mechanism for applying loosely coupled [T-Lock] and cooperation between other local coordinators (virtually decentralised), but also gives the sense of globally isolated recovery management (which again is applied virtually). The model tries to overcome all weaknesses of current architectures of SOA on the web services and provides the possibility for fully-automated service compositions with any complexity and depth in nesting.

4.2. Model Structure

4.2.1 Local Coordination

At the heart of this transactional model are the local coordinators. They have to handle the complexities of the model and control/generate all logs. At the same time, they should have enough flexibility for handling the low bandwidth (and low processing power) limitations from some nodes in the network.

-Coordinator structure:

Based on different types of compositions, we use different type of coordinators. Therefore a transaction will split to a nested group of sub-transactions with a tree structure (nested transaction model). The root of this tree is a main composition, which is a coordinator and each sub-transaction is either a coordinator or a simple service (in the leaf). There are five different coordinator types plus delegation coordination for handling delegation:

- **Data oriented coordinator:** This coordinator is specifically working on data oriented service composition; including fully atomic and simple service oriented which is dealing with released data item inside of a transaction or using partial results, released by other transactions.
- **Sequential process oriented coordinator:** This coordinator is invoking its sub-transactions (services) sequentially. The execution of a sub-transaction is dependent on its previous service, i.e., one cannot begin unless the previous sub-transaction commits. In fact this coordinator handles Sequential process oriented service composition by covering both Sequential with commitment dependency (SCD) and Sequential with data dependency (SDD) [chapter 4].
- **Parallel process oriented coordinator:** In the Parallel oriented coordinator all the sub-transaction (component services) can be executed in parallel but different scenarios can be considered which can make different situations (implementations) in the transactional outlook which covers; Parallel with data dependency (PDD), Parallel with commit dependency (PCD) and Parallel without dependency (PND) [chapter 4].
- **Sequential alternative coordinator:** This coordinator indicates that there are alternative sub-transactions (services) to be combined, and they are ordered based on some criterion (e.g., cost, time, etc). They will be attempted in succession until one sub-transaction (service) produces the desired outcome. In fact it is for supporting Sequential alternative composition (SAt) [chapter 4] and it may use dynamically for forward recovery.
- **Parallel alternative coordinator:** Unlike the previous coordinator, alternative sub-transactions (services) are pursued in parallel. As soon as any one of the sub-transaction (service) succeeds the other parallel sub-transactions are aborted (as it has clear, this coordinator rely on reliable compensation mechanism). Actually the Parallel alternative coordinator handles Parallel alternative composition (PAt).
- **Delegation coordinator:** The whole transaction or a sub transaction can be delegated to another platform; delegation can be by sending request specification or service(s) description.

Figure 4-1 shows the symbolic view of the model (apart from 'data oriented coordinator', symbols are standardised in [HPS93], [Hag97], [Hag95], [VeP98], [PDB$^+$97], [PDH$^+$96]).

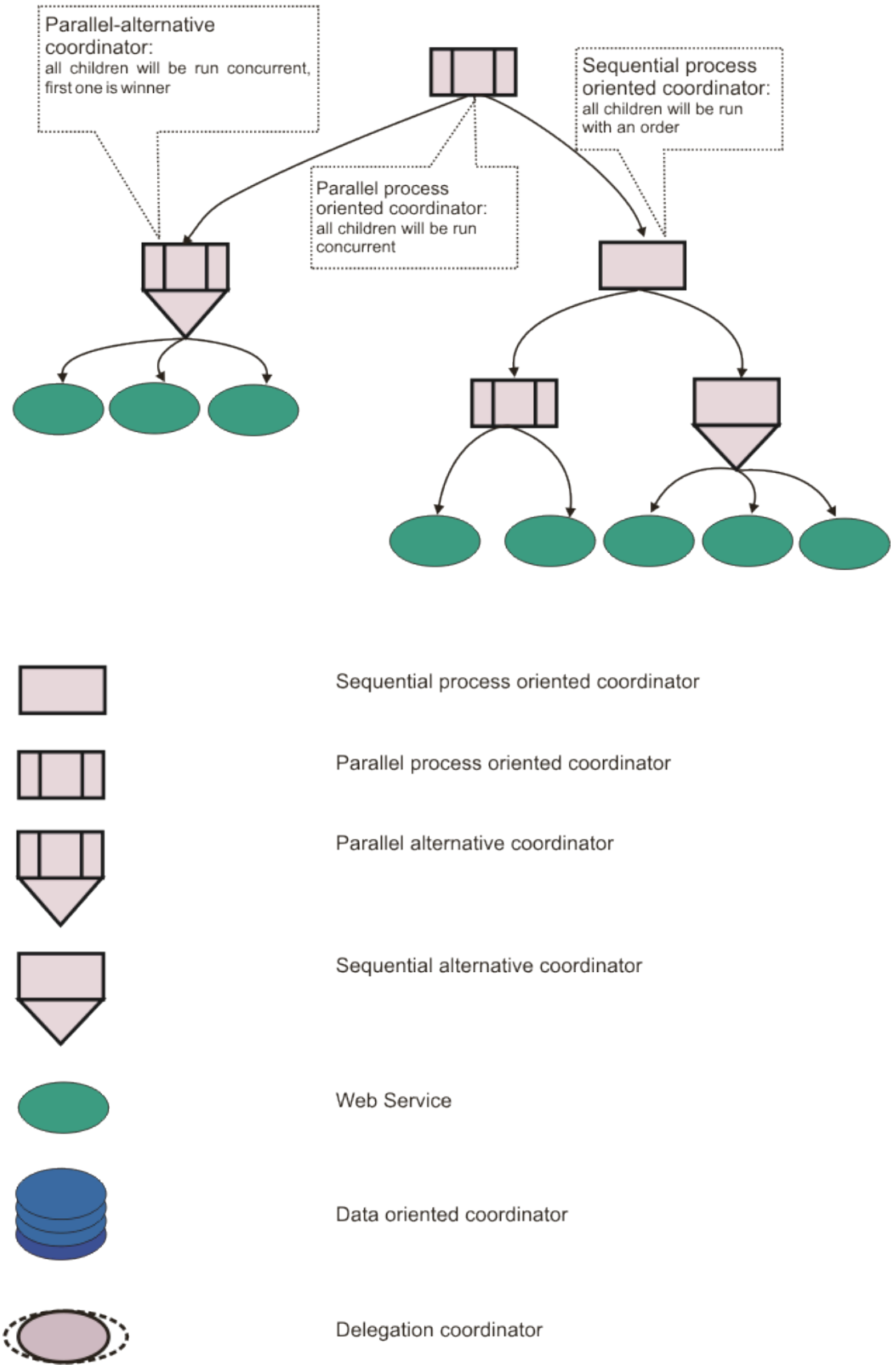


Figure 4-1 Proposed transaction model structure

4.2.2. Data dependencies and Local Consistency Graph

In Conventional transaction database, by using classic mechanism (such as lock-based concurrency control, Log-based/Shadow page Recovery management, etc), ACID properties are supported and consistency of the environment is guaranteed. Meanwhile for covering other problems such as deadlock, more structure (mostly graph) and mechanism are used.

In the proposed transaction model, not only are the above concepts (consistency, durability, deadlock and etc) considered, but also a method is designed for releasing some of its modified/generated data even before transaction commit. In the concurrency control [4.2.6], new Locks are used for full coverage of consistency and environment requirements. Furthermore, using two different graphs provides the possibility for applying data dependencies in a consistent way (which shows internal data structure of coordinators). Meanwhile, (taking into consideration the relationship between locks and these graphs) applying different anti-deadlock mechanisms is possible. In this section, the internal data dependencies and the designed graph for that will be analysed and in 4.2.4 the other type of graph will be explored.

-Internal dependency Graph (IDG):

For keeping tracks on data (value) dependencies, providing possibility for reverse action, possibility for applying deadlock control mechanism and transparency during delegation, two different graphs are introduced. In fact, these graphs are very important system logs, which are stored locally on a coordinator and will be effected locally (in term of local faults, forward recovery and contingencies plan) and globally (abortion, restart or etc). This Internal Dependency Graph keeps logs on value dependencies in a transaction tree.

The Internal Dependency Graph is a directed graph in which each node represents a coordinator and the direction shows the dependency between two nodes. Alternatively, when a coordinator wants to use a data item belongs to another coordinator, two nodes have to be created in IDG (if they do not already exist) and an edge generated between them (with the direction which shows the dependency between two coordinator).

-SDD (Sequential with data dependency) and IDG:

In the sequential coordinator, the children will be executed after each other and they can use the result of the previous children. If the serial scheduler i has n children, the children can work on different data item with the order of releasing data items (like). If IDSi and IDSi+1 are two children of a sequential coordinator, IDSi+1 can use the result release by IDSi and if IDSi is aborted IDSi+1 must be aborted too (because it is used the released values by IDSi). This dependency will be showed in IDG like Figure 4-2 and it means IDSi+1 is dependent to IDSi.

When there is commit-dependency in sequential coordinator (for example between two children: IDSi+1 and IDSi+2), a directed link will be created between two nodes (from dependent nodes to the other one).

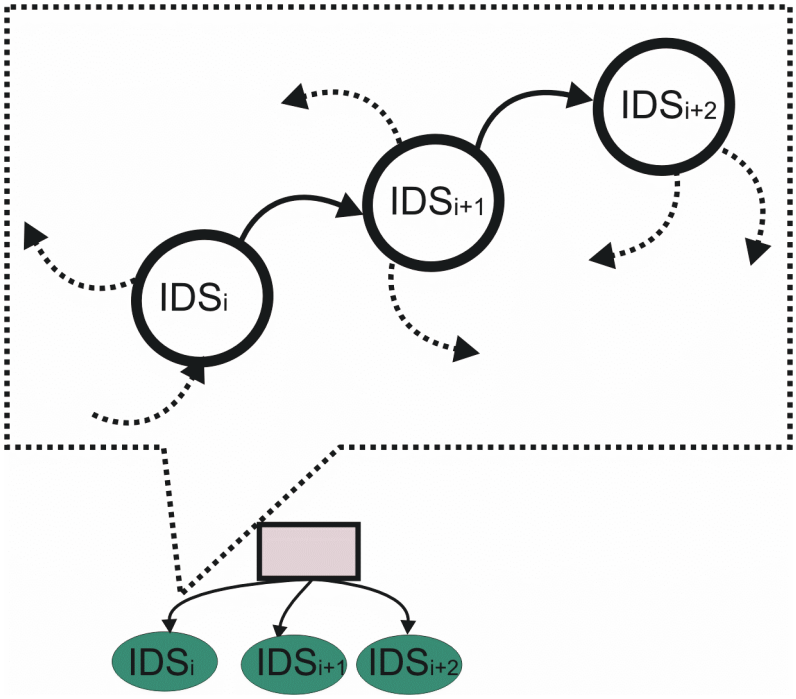


Figure 4-2 Sequential Data Dependency Coordinator and its associated Internal Dependency Graph

-PDD (Parallel with data dependency) and IDG:

With the parallel coordinator (with data dependency) when value-dependencies occur, if one of the children is aborted (or restarted) the dependent children (to that sub-transaction) must also be aborted (restarted). Therefore we must have a structure for showing these dependencies and the way that recovery management can rollback the system. IDG (Internal Dependency Graph) is a directed graph that is introduced for reaching this aim.

If IDSi and IDSj are two children of a parallel coordinator, IDSi can use the result released by IDSj and if IDSj is aborted IDSi must be aborted too (because it used the released values by IDSj). This dependency will be shown in IDG like Figure 4-3 and it means IDSi is dependent on IDSj. When there is a commit-dependency in the parallel scheduler (for example between two children: IDSi and IDSj), a directed link will be created between two nodes (from dependent nodes to the other one).

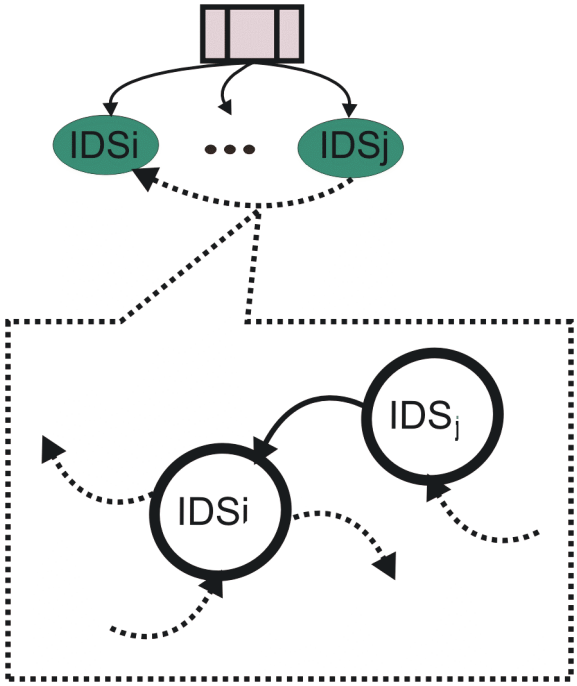


Figure 4-3 Parallel Data Dependency Coordinator and its associated Internal Dependency Graph

4.2.3. Conditional Commit, External dependency and External Consistency Graph

One the best known methods for this process is conditional commitment [PDH+96], [PDB+97], [Hag97], [Raz99]. The support mechanism can be different depending on environment and considered data structure. We are using a compensating mechanism for providing integrity and consistency. Therefore data integrity is maintained through a compensation routine when a failure occurs.

Actually compensatable sub-transactions release partial-results in conditional commit, and that means there is a routine of compensation when a DBE transaction is aborted (abortion for any reason). C_Lock [4.2.6] is used for a data item that was released in conditional commit, but for making it possible to create a suitable compensation mechanism, we need to define another log structure what we call it External Dependency Graph (EDG). This log structure records the dependency of different DBE transactions to each other (especially in low level). Then during a recovery routine, there is a possibility to run a compensating procedure.

-Conditional Commit mechanism for releasing partial result

When a sub-transaction wants to access a released data item (by C_Lock) which belongs to another DBE transaction this dependency is shown by creating a directed link between these two nodes from the owner to user of data item (Figure 3-4).

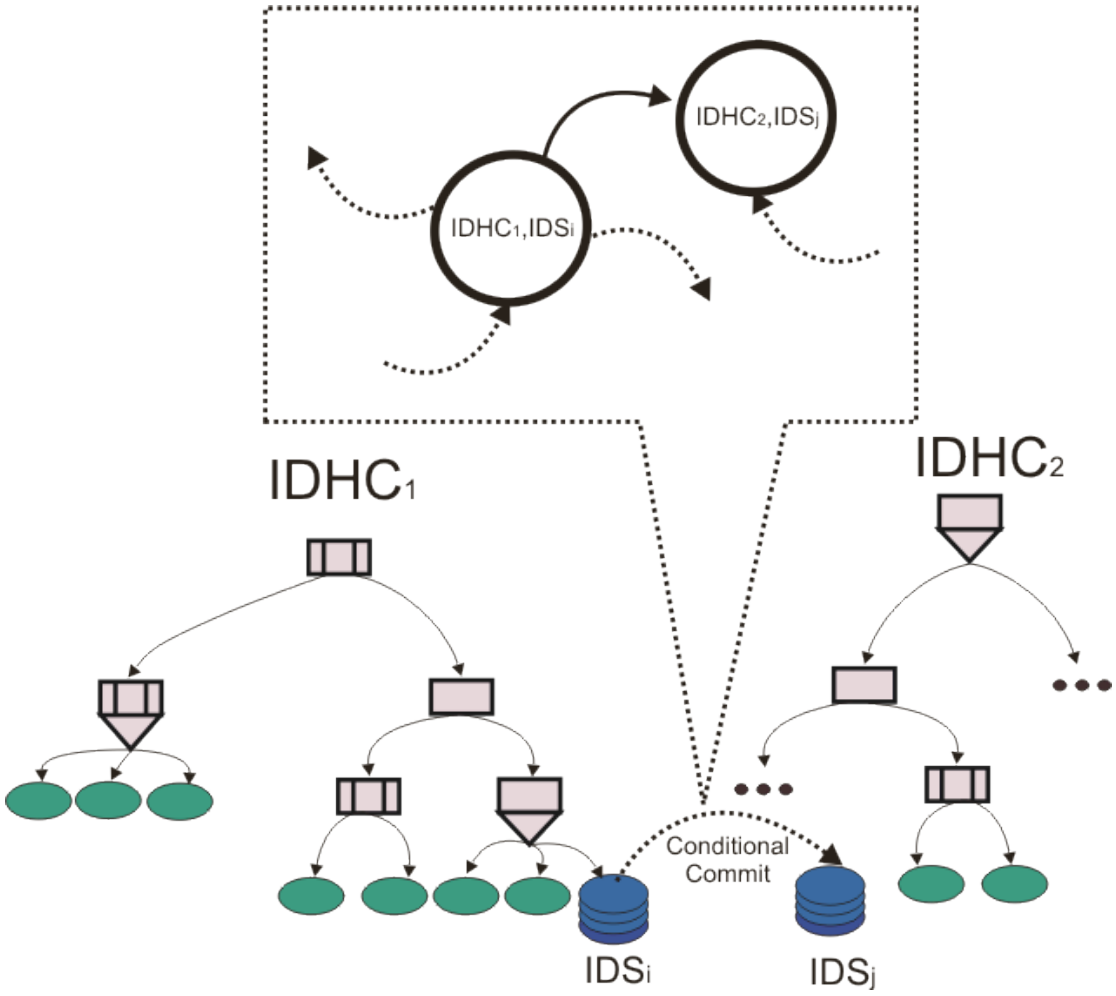


Figure 4-4 EDG for releasing partial results (Compensating transaction)

If each of these nodes are absent in EDG, they must be added and if nodes and connection between them already have created, there is no need for repetition. The most important usage of this graph is in the creation of compensatory transactions during a failure.

4.2.4. Compensatory Sub-transactions

Compensability is one the most important motivations behind this model which is supported by creating virtual compensatory transactions. These transactions are generated using a mixture of External Dependency Graph and Internal Dependency Graph for reverse operation of local coordinators. Normally Compensability is possible before second phase of commitment.

-Creating a Compensation Tree by using IDG and EDG

A compensation transaction is created/executed whenever a sub-transaction is aborted. This could be for one of a range of reasons. For example, this could happen when: alternative transactions for Alternative parallel coordinators fail to meet the specified conditions; a platform has failed and its coordinators do not respond; or a platform itself made this decision because of some internal problem such as deadlock. Therefore, conventionally the failing coordinator has to be aborted which means any sub transactions (children) of that coordinator have to be aborted. However, this can be complicated depending on sub-transactions of that particular transaction (including the transactions which were using partial results generated by this coordinator) being aborted too and recursively same thing has to happen for the dependent transactions.

For applying this algorithm by using EDG and IDG we create a tree which the failed coordinator is the root of the tree. Suppose Tci is a compensatory sub-transaction that belongs to DBE1 transaction (Figure 3-5). Tci is releasing partial results when conditional commit but DBE1 has not committed yet. Tcj is another a compensatory sub-transaction that is a sub-transaction of DBE2 transaction and is using the released data from Tci. With the same scenario a sub-transaction of another transaction can use this partial result, etc.

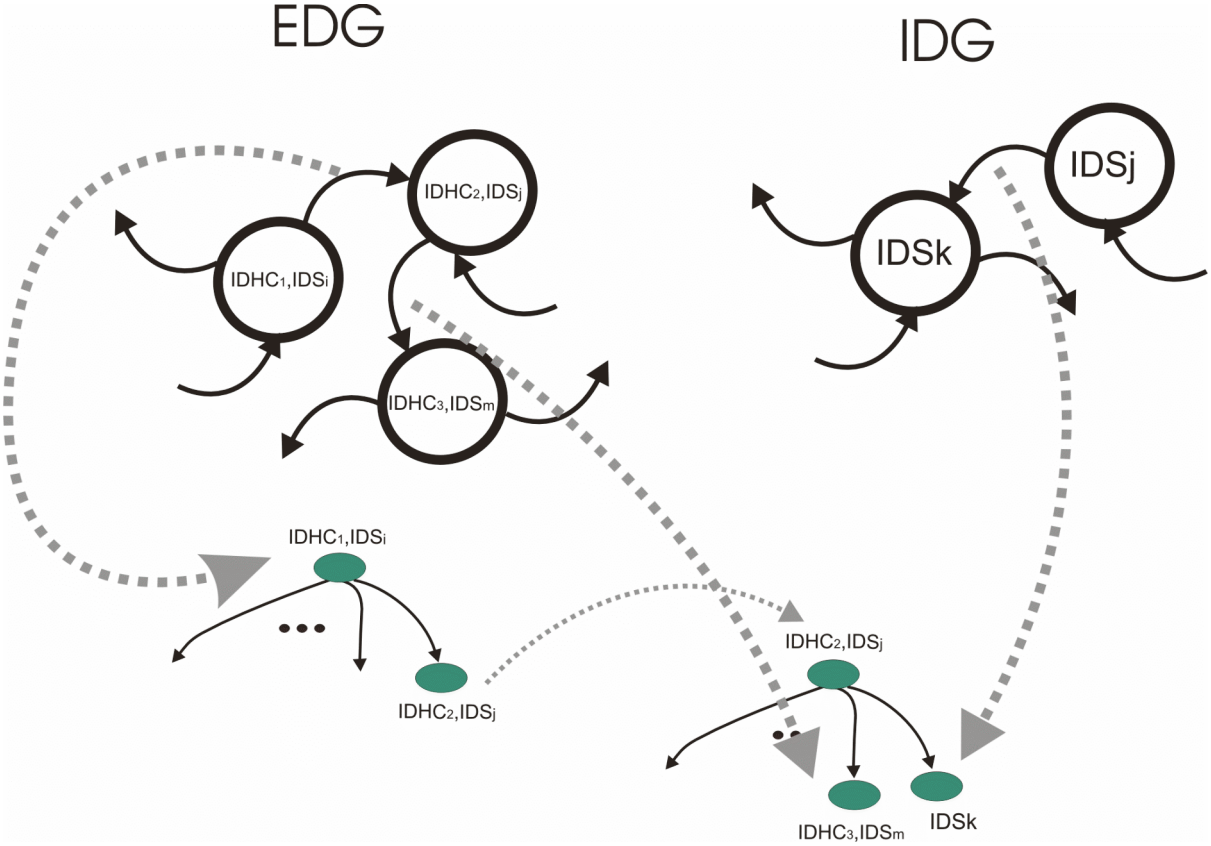


Figure 4-5 Creating compensating routines by using EDG and IDG

If Tci aborts with a failure, all a compensatory sub-transaction that were using its partial results must be aborted. In fact this could be modelled like a tree when the root is Tci (Figure 4-5). We call this transaction a compensating transaction. The compensate transaction is created by using EDG and following the directed links. During the creation of compensating transaction if we meet a sub-transaction twice, it will be skipped. The priority and locks used in a compensating transaction are rather like Recovery Management.

5. Coordinator Formal model

In this section, we lay the foundations for a model of long-running transactions showing how the sub-transactions (local coordinators and/or basic services) are orchestrated to achieve the goal of the transaction in question. The proposed formal model draws upon ideas on a truly-concurrent framework for describing interactions between communicating entities [Mos05] and is adopted to underpin the local coordination required for long-running multi-service transactions of DBE.

The service-oriented architecture for distributed transactions reinforces our interest in all environmentally observable actions within and between transactions. That means it is appropriate to consider that any action within the transaction model has no significant duration, in the sense that (i) it either occurs as a whole or not at all; (ii) it occurs either wholly before, or wholly after, or wholly in parallel with, every other action.

We have seen that a transaction can be represented by a tree structure that allows us to exemplify the local coordination that is required for the services involved to be performed in unison in accomplishing the goal prescribed by the transaction. Fig. 5-1 shows a transaction tree with four basic services whose order of execution is determined by the five coordinator types employed. We have adopted the notation of [PDH+96] extended with a symbol for the data-oriented coordinator.

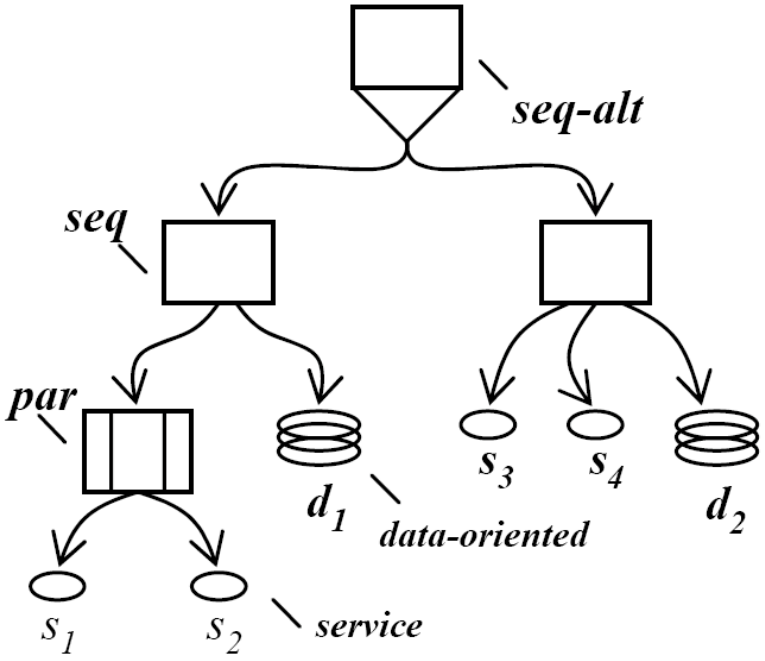


Fig. 5-1 Transactions in a tree structure

In the transaction tree of Fig.5-1, s3 and s4 are children of a sequential coordinator and hence s4 can use the results released by s3. This means, as a consequence, that if s3 is aborted, then s4 must also be aborted. In Section 4 we introduced the Internal Dependency Graph (IDG) for capturing such dependencies. The dependency between s3 and s4 is shown in the IDG of Fig.5-2(i).

When value dependencies exist between children of a parallel coordinator, then if one of the children is aborted, the rest of the children must also be aborted. The structure of the corresponding IDG, see Fig.5-2(ii), shows such dependencies and in this sense facilitates recovery management to rollback the system.

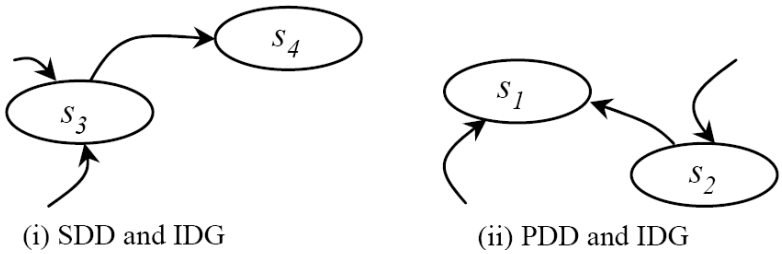


Fig. 5-2 Internal Dependency Graph

In a highly dynamic and purely distributed environment such as a Digital Business Ecosystem, it is often the case that a sub-transaction requires access to a data item released (possibly as a partial result) by a sub-transaction belonging to a different transaction. In other words, dependencies may exist not only within a transaction but also between transactions (which, may take place on different platforms). For example, consider the case of (compensatory) sub-transactions that release partial results in a conditional commit state [PDH+96].

To capture such dependencies we introduced the External Dependency Graph (EDG) in Section 4. This keeps track of dependencies between (services or coordinators of) different transactions. The log structure it provides can be used in recovery routines for running a compensating procedure.

Fig. 5-3 shows part of the EDG for the transaction trees T1 (of Fig.5-1) and T2. The data-oriented coordinators d1 and d2 of T1 release partial results that are required by d3 of T2.

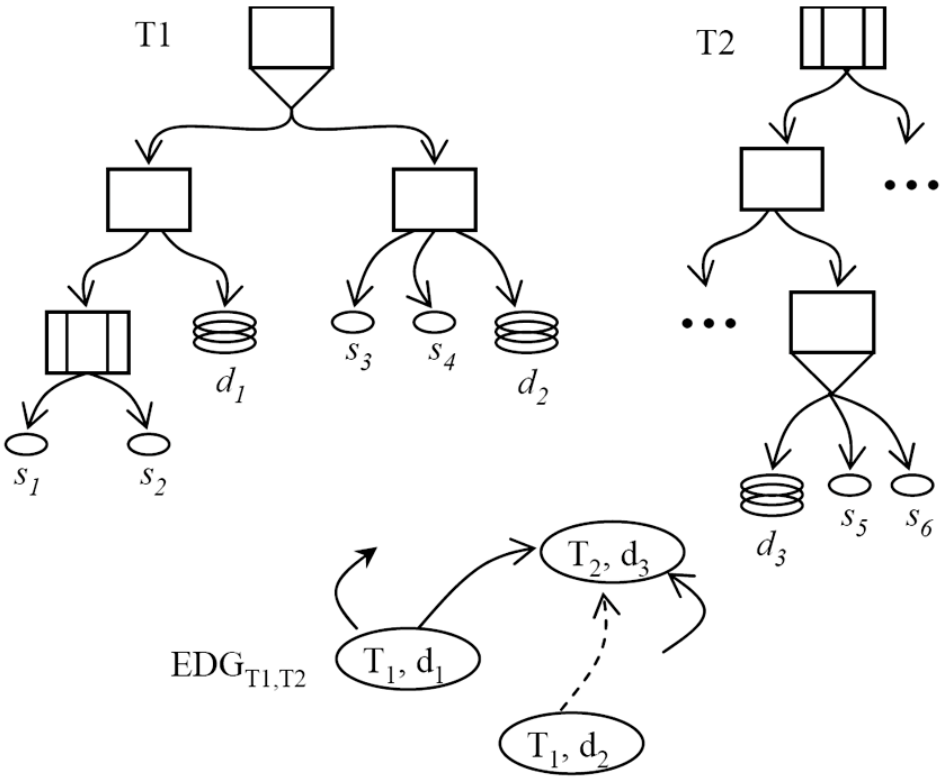


Fig. 5-3 EDG for releasing partial results between T1 and T2

Now, if for some reason d1 (or any other sub-transactions on which d1 depends, for that matter) was aborted, then d3 should also be aborted along with any sub-transactions of T2 which depend on it. Based on the log information provided by the EDG and the corresponding transaction trees, we would like to recalculate d3 based on the data items released by d2 and defer from aborting (at least part of) transaction T2.

5.1 The Model

In our behavioural model of a transaction it suffices to use formal notation for the leaves only. The aggregation coordinators (nodes) are manifested in the structure of the resulting formal construction, and there is no need for additional notation. A transaction T, then, is associated with a set of leaves L which consists of a set of basic services S, a set of data-oriented coordinators D and a set of delegation coordinators Dlg. Thus, $L = S \cup D \cup Dlg$.

A transaction is also associated with a finite set of events that may occur (on its sub-transactions) upon activation, e.g. service invocation, initialisation, commitment, service return, release result (return), termination, abort, etc. We denote this set by M. These events take place on the leaves and therefore it seems appropriate to say that each leaf is in turn associated with a set of events that may occur on that leaf, depending on its nature. We denote this set by $\mu(l)$, $l \in L$, and require that $\bigcup_{l \in L} \mu(l) \subseteq M$.

In any behaviour of a transaction T, each sub-transaction on the leaves will be activated and experience a sequence of events formed over the corresponding set $\mu(l)$, $l \in L$. We may thus describe the behaviour of the transaction by assigning such sequences to each of its leaves.

Definition 1. Let T be a transaction. We define V_T to be the set of all functions $\underline{v}: L \rightarrow M^*$ such that $\underline{v}(l) \in \mu(l)^*$. We refer to elements of V_T as transaction vectors.

By $\mu(l)^*$ we denote the set of finite sequences over $\mu(l)$. Mathematically, the set V_T is the Cartesian product of the sets $\mu(l)^*$, for each l. Effectively, transaction vectors are n-tuples of sequences where each coordinate corresponds to a leaf in the transaction tree (hence, n is the number of leaves) and contains a finite sequence of events that have occurred on that leaf.

When an event occurs on a leaf, that is to say when an action associated with some sub-transaction takes place, it appears on a new transaction vector at the appropriate coordinate. For example, the vector (s_1, Λ, Λ) describes that portion of behaviour of the transaction in which an action s_1 (e.g. service invocation) has taken place on the sub-transaction allocated to the first coordinate. The vector (s_1, s_2, Λ) describes that portion of behaviour in which both s_1 and s_2 have happened on the corresponding sub-transactions while the vector (s_1 s_1, s_2, Λ) describes two occurrences of s_1 and an occurrence of s_2.

It can be seen that there is already ordering among actions on a particular sub-transaction (e.g. s_1 followed by another s_1). This vector-based behavioural description of transactions can also capture the orderings between different sub-transactions, which amounts to events on different vector coordinates. This requires however a more careful consideration of the mathematical properties of such vectors which we briefly describe in the following section.

Before that we introduce a specific kind of transaction vectors, which is used in our model to describe actions (events or activations) within a transaction.

Definition 2. Let T be a transaction and V_T its set of transaction vectors. We define

$$A_T = \{\underline{a} \in V_T \setminus \{\underline{\Lambda}_T\} : l \in L \Rightarrow |\underline{a}(l)| \leq 1\}$$

where $|x|$ denotes the length of sequence x. We refer to elements of A_T as *column vectors*.

Thus, the vectors of Definition 2 are themselves transaction vectors, but have the additional constraint that each of their coordinates is either the empty sequence or a single action. For example, the vector (s_1, Λ, Λ) represents the occurrence of an action s_1 on the sub-transaction associated with the first coordinate.

5.1.1 Order-theoretic properties of transaction vectors

In what follows we describe the basic order-theoretic properties of transaction vectors and show how the order structure of sets of such vectors expresses ordering constraints on the activation of sub-transactions.

We have seen that transaction vectors are essentially tuples of sequences. This can be exploited in defining operations on the vectors in terms of well-known operations on sequences.

Definition 3. For $\underline{u}, \underline{v} \in V_T$, we define

- $\underline{u}.\underline{v}$ to be the unique vector $\underline{w}$ such that $\underline{w}(l) = \underline{u}(l).\underline{v}(l)$, for each $l \in L$ (*concatenation*)

- $\underline{u} \leq \underline{v}$ iff $\underline{u}(l) \leq \underline{v}(l)$, for each $l \in L$ (*prefix ordering*)

- *glb*($\underline{u},\underline{v}$) to be the vector $\underline{w}$ such that $\underline{w}(l)=\min(\underline{u}(l),\underline{v}(l))$, for each $l \in L$

- *lub*($\underline{u},\underline{v}$) (if it exists) to be the vector $\underline{w}$ such that $\underline{w}(l)=\max(\underline{u}(l),\underline{v}(l))$, for each $l \in L$

- if $\underline{u} \leq \underline{v}$, then we define $\underline{v} / \underline{u}$ to be the unique element $\underline{z} \in V_T$ such that $\underline{u}.\underline{z} = \underline{v}$ (*right-cancellation*)

Thus, concatenation on vectors is defined in terms of the concatenation of sequences appearing on their respective coordinates. For example,

$$(s_1 s_3, s_2, \Lambda).(\Lambda, s_4, \Lambda) = (s_1 s_3, s_2 s_4, \Lambda)$$

Similarly, the ordering amongst vectors is defined in terms of the usual prefix ordering operation on sequences appearing on their coordinates. For example,

$$(s_1, s_2, \Lambda) \leq (s_1 s_3, s_2, \Lambda) \text{ since } s_1 \leq s_1 s_3 \text{ and } s_2 \leq s_2 \text{ and } \Lambda \leq \Lambda$$

In other words, the vector $\underline{v}$ wins on the first coordinate while the two vector draw in all other coordinates. It is not hard to see that some vectors will be incomparable. For example,

$$(s_1 s_3, s_2, \Lambda) \text{ and } (s_1 s_5, s_2, \Lambda)$$
or
$$(s_1, \Lambda, \Lambda) \text{ and } (\Lambda, s_2, \Lambda)$$

It turns out that such vectors describe either parallel or alternative behaviours of the transaction in question, and this will be further discussed in Section 5.2.

It is important to note that these two fundamental operations, *concatenation and prefix-ordering*, on transaction vectors are performed coordinate-wise in our model as this simplifies the mathematics of it and allows for relatively straightforward proofs.

The operations *glb*() and *lub*() of Definition 3 give the greatest lower bound and the least upper bound, respectively, of u, v$\in$VT, in the usual sense of lattices and domain theory [DaP90]. These have an important role to play in defining the properties that ensure the well-formedness of the behavioural description as will be discussed in Section 5.1.2.

The right cancellation operator '/' says that if u is a transaction vector describing an initial part of the behaviour described by v so that $u \leq v$, then v / u is the 'continuation' of u that extends it to v. This operation is particularly useful in deriving a transition relation that allows to associate the vector-based description of behaviour with automata and asynchronous transition systems [Shi85], in giving a state-based description of the interactions involved [MSK05].

It can be shown (by arguing coordinate-wise) that a set of transaction vectors equipped with the operations of concatenation and prefix ordering forms a monoid and a partial order. $\underline{\Lambda}_T$ is used to denote the empty vector which has the empty sequence on each of its coordinates.

Proposition 1. A set of transaction vectors V_T is
- a monoid under '.' and identity $\underline{\Lambda}_T$
- a partial order under $\leq$ and bottom element $\underline{\Lambda}_T$

The incomparable vectors in the partial order $(V_T, \leq)$ allows to introduce a notion of independence between transaction vectors, which is central to expressing ture-concurrency within our model. This builds on earlier work on describing parallel behaviour in Shield's *behaviour vectors* [Shi97] where the notion of independence found in Mazurkiewicz traces [Maz88] is lifted to vectors.

Definition 4. Let $\underline{u}$, $\underline{v}$ be transaction vectors in V_T. We define $\underline{u}$ and $\underline{v}$ to be *independent* and we write $\underline{u}$ *ind* $\underline{v}$, if and only if

$$\forall l \in L : \underline{u}(l) > \Lambda \Rightarrow \underline{v}(l) = \Lambda$$

Effectively, the independence relation implies that behaviours which may take place concurrently engage distinct parts (subtransactions) of the transaction. In the case of column vectors (Definition 2), used to represent events, independence captures the fact that the actions in each vector are independent (not related in any way). If such independent vectors occur consecutively, one after the other, then they are concurrent.

In further explanation, whenever two events are independent (not ordered in any way), then their corresponding column vectors commute, i.e. $\underline{a}_1.\underline{a}_2 = \underline{a}_2.\underline{a}_1$, and in the resulting behaviour the sub-transactions involved are executed concurrently. In fact, (A_T, *ind*) is a *concurrent alphabet* in the sense of [Maz88].

It is important to note that independence alone does not guarantee concurrency – there is the additional requirement that the independent actions are both enabled after some behaviour and they occur consecutively (one after the other). Further details can be found in [Mos05]. We will have more to say about the order (or otherwise) of execution within a transaction in Section 5.2 where we examine how the relations between events in a transaction are manifested in the order structure of the corresponding transaction vectors.

5.1.2 Well-formedness of the behavioural description

In describing the behaviour of transaction we are interested in the actions (activations) on its sub-transactions. These are captured in our model using column vectors (Definition 2). Thus, instead of considering all possible transaction vectors we would like to be concerned only with those obtained by concatenations with column vectors only. This gives us the behaviour of the transaction in terms of activations or actions of its sub-transactions and can be used to enforce the coordination of the underlying services.

We have seen that transaction vectors are obtained by coordinate-wise concatenation (Definition 3), for example

$$(x_1, x_2, x_3).(y_1, y_2, y_3) = (x_1 y_1, x_2 y_2, x_3 y_3)$$

In such a behavioural description of a transaction, transaction vectors can be seen to be built up from the empty vector by a series of concatenations with the column vectors [Mos05], each of whose coordinates is either empty or contains a single event/action.

For example, the column vector $\underline{a} = (s_1, \Lambda, \Lambda)$ represents the activation of the leaf corresponding to the first coordinate. If s1 is intended to occur only after both s3 and s4 have, then this is described in the transaction vector $\underline{v} = (s_1, s_3, s_4)$ which is obtained as

$$\underline{u}.\underline{a} = (\Lambda, s_3, s_4).(s_1, \Lambda, \Lambda) = (s_1, s_3, s_4)$$

In order to ensure that vectors associated with a transaction are the result of concatenations with column vectors only, the set of transaction vectors must satisfy certain properties, namely *discreteness* and *local left-closure*. We introduce these properties next.

In describing the behaviour of a transaction in terms of the coordination of its sub-transactions, we want to capture the fact that a system's computations always have a starting point, and ensure that only a finite number of events may occur within finite time. This turns out to be the case if whenever two vectors describe an earlier part of behaviour than a third, also on the set, then their least upper and greatest lower bounds are also in the set. This is formally put in the following definition.

Definition 5. Let $V \subseteq V_T$, then V is *discrete* if and only if, $\underline{\Lambda}_T \in V$ and whenever $\underline{u}, \underline{v}, \underline{w} \in V$ such that $\underline{u}, \underline{v} \leq \underline{w}$ then

(i) $lub(\underline{u}, \underline{v}) \in V$

(ii) $glb(\underline{u}, \underline{v}) \in V$

Note that $lub(\underline{u}, \underline{v}) \in V$ is understood as asserting that $lub(\underline{u}, \underline{v})$ is defined, i.e. the least upper bound of $\underline{u}, \underline{v}$ exists. This property builds on the notion of consistently complete subsets, as discussed in [Shi97], and further requires that the least upper and greatest lower bounds belong to the set.

It can be seen that discreteness imposes a finiteness constraint in the sense that it excludes infinite ascending or descending chains of actions with respect to time ordering. It ensures that situations like those resulting in Zeno-type paradoxes will never arise. The famous Zeno paradoxes, in which the philosopher seeks to demonstrate the impossibility of motion, are examples of a non-discrete representation of system behaviour[5].

In order to obtain a precise description of discrete behaviour, we further require that every occurrence of an event (e.g. service invocation, partial result, commitment) is recorded in the set of vectors associated with the transaction. This guarantees that any earlier part of behaviour is itself a behaviour and motivates the following definition.

Definition 6. Let $V \subseteq V_T$, $l \in L$ and $x \in \beta(l)^*$. Then, V is locally left-closed if and only if, whenever $\underline{v} \in V$ and $\Lambda < x \leq \underline{v}(l)$, then there exists $\underline{u} \in V$ such that $\underline{u} \leq \underline{v}$ and $\underline{u}(l) = x$.

[5] Zeno's paradox with arrow and that involving Achilles and the tortoise are discussed in view of computer science in [Shi97]. The conclusion drawn from Zeno's arguments in this case is not that motion is impossible, but that the behavioural description used is not *discrete*.

The above definition says that whenever there is a sequence of actions on some sub-transaction (or local coordinator) which is less or equal than some other sequence appearing in some transaction vector in V, then there is some other vector in V which describes an earlier part of behaviour and has that sequence on the corresponding coordinate. In fact, 'local' comes from the fact the property is considered at the vector coordinate level and thus applies to individual sub-transactions or local coordinators and 'left-closure' reflects the fact that earlier parts of a given behaviour are themselves behaviours.

We say that the set of vectors V_T associated with a transaction T is *normal* if and only if it is locally left-closed and discrete. This reflects tha fact that the guarantees that accrue from the above properties are embedded in the behaviour of the corresponding transaction.

In fact, these properties ensure the well-formedness of the behavioural description of a transaction in our model. The idea is that in checking against these properties we may determine whether the transaction will exhibit the desired behaviour when executed or on the contrary, other non-desirable scenarios of execution are still possible. This draws upon previous work on vector languages in [Mos05].

5.2 Parallel, Alternative and Sequential Behaviour of a Transaction

The study of the order-theoretic properties of such vectors in [Mos05] shows that it is possible to express sequential, parallel and alternative behaviour. For the purpose of the present paper it suffices to understand that the ordering relation between different vectors of a transaction reflects the orderings between activations of its sub-transactions.

We have seen that the ordering relation between transaction vectors is given in terms of the coordinate-wise prefix ordering relation '$\leq$' of Definition 3. This turns the set of vectors associated with a transaction to a partially ordered set $(V, \leq)$ (see Proposition 1 – the proof is done by arguing coordinate-wise as done in [Shi97]). Since each vector describes the part of behaviour of the component in which the actions appearing in it have taken place, it is appropriate to say that whenever $\underline{u} \leq \underline{v}$, then $\underline{u}$ describes an earlier part of the behaviour described by $\underline{v}$.

Since $(V, \leq)$ is a partially ordered set some vectors may be incomparable. For example, consider the vectors $\underline{u} = (s_1, \Lambda, \Lambda)$ and $\underline{v} = (\Lambda, s_2, \Lambda)$ for which neither $\underline{u} \leq \underline{v}$ or $v \leq \underline{u}$. Such vectors describe either alternative behaviour (the last events that went into forming each are mutually exclusive) or concurrent behaviour (the last events that went into forming each are concurrent). Any pair of incomparable vectors stands in one relation or the other, and this is determined by what other vectors are in the set of vectors associated with a given transaction.

If the incomparable vectors are bounded above – in other words, if they describe earlier parts of some common later behaviour – then they describe concurrent behaviours. If they are not bounded above, then they describe alternative behaviours. It is important to note that this is determined by context, by what other vectors are included in the set for a transaction.

This is illustrated in Fig. 5-4 which uses Hasse diagrams to depict the order structure of different sets of transaction vectors for a transaction with 3 leaves. It can be seen that s1 and s2 are sequential (s2 can only be activated after s1) in Fig.5-4(i) while they are mutually exclusive (alternative) in Fig.5-4(ii) and they are concurrent in Fig.5-4(iii).

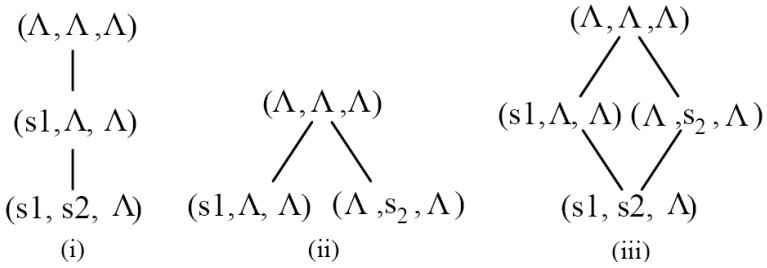


Fig. 5-4 Order structure of transaction vectors

Notice that the set of vectors in case (i) does not include the vector (Λ, s_2, Λ). This in addition with the fact that (s_1, s_2, Λ) is included implies that s2 can only happen after s1 has (sequentially).

The set of vectors in case (ii) does not include (s_1, s_2, Λ). This has as a consequence that the vectors $\underline{u} = (s_1, \Lambda, \Lambda)$ and $\underline{v} = (\Lambda, s_2, \Lambda)$ are not bounded above in this case. Hence, the actions s1 and s2 are independent but do not take place consecutively in this case. This implies that there is a choice between doing s_1 and doing s_2 on the respective coordinates (alternative execution).

In case (iii) where the vector (s_1, s_2, Λ) is included, the vectors are bounded above and this implies that they describe the concurrent execution of s_1 and s_2 leading to the behaviour described by the vector (s_1, s_2, Λ). This is indicated by the familiar lozenge shape which marks the characteristic structure of a finite lattice [DaP90]. The incomparable vectors sitting at the middle of the lozenge are both available after the same behaviour (that is $(\Lambda, \Lambda, \Lambda)$) and occur consecutively leading to the behaviour described by the vector sitting at the bottom of the lozenge shape, i.e. (s_1, s_2, Λ).

Fig. 5-4 might be instructive with regard to the subtle distinction between independence and concurrency. Independent events are concurrent only if they are both offered after the same behaviour (both enabled at the same time) and do occur one after the other (consecutively). Otherwise, they may be mutually exclusive or even sequential.

It might also be worth pointing out that the lozenge shape in Fig. 5-4(iii) exhibits the characteristic structure of a finite lattice, which is a requirement of the discreteness property (Definition 5) in the case that $\underline{u}, \underline{v}$ are independent. The vector at the bottom of the lozenge is the least upper bound of the vectors in the middle, while the vector at the top is their greatest lower bound. This shows that discreteness – in the case of independent vectors bounded above in the set – is a property inherently related to concurrency.

The transaction tree shown in Fig.5-1 has 6 leaves. The services s_1 and s_2 are to be executed in parallel (concurrently) followed by the data-oriented coordinator d_1. If the partial result released by d_1 (see Fig. 5-3) does not meet the desired outcome, then s_3 and s_4 are executed in succession (sequentially) followed by d_2.

To model the behaviour of the transaction in our formalism, we assign each leaf to a vector coordinate (from left to right here). This results in the set of 6-tuples shown in the Hasse diagrams of Fig. 5-5, which describe all possible series of sub-transaction activations in performing the transaction T1 (given earlier in Fig. 5-1). In Fig. 5-5 there is a choice between the behaviour described in the diagram on the left and that on the right, and this reflects the sequential alternative scenarios of transaction T1. This choice is deterministic and will be resolved on the basis of whether d_1 satisfies the desired outcome. Furthermore, in case some sub-transaction fails, the vector-based description is used in providing compensating transactions, taking up on the "do-compensate" and "validate-do" behaviour patterns.

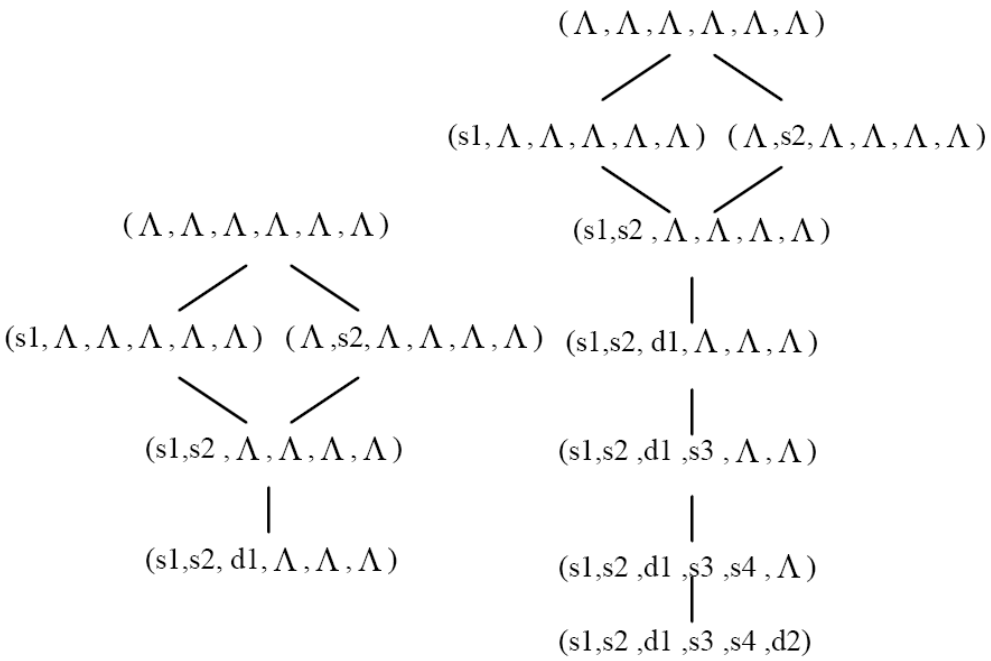


Fig. 5-5: Transaction vectors for T1

Notice the lozenge formed by s_1 and s_2 which execute in parallel (in both cases). Also, notice that the Hasse diagram on the left implies that $(s_1, s_2, d_1, s_3, \Lambda, \Lambda) \leq (s_1, s_2, d_1, s_3, s_4, \Lambda)$ which means that s_4 can only happen after s_3 has (sequentially).

The Hasse diagram depicting the order structure of the transaction vectors for T1 can be readily used for checking against discreteness and local left-closure. For discreteness, we concentrate on vectors which have more than one vector immediately underneath. Then, that vector together with its immediate predecessors (the vectors immediately below it) must constitute a finite lattice. This will be the case when the immediate predecessors are bounded above (least upper bound) and below (greatest lower bound) by some vector in the set. In our example, such a vector is $(s_1, s_2, \Lambda, \Lambda, \Lambda, \Lambda)$ which has two distinct immediate predecessors, namely $(s_1, \Lambda, \Lambda, \Lambda, \Lambda, \Lambda)$ and $(\Lambda, s_2, \Lambda, \Lambda, \Lambda, \Lambda)$. These vectors are bounded above by $(s_1, s_2, \Lambda, \Lambda, \Lambda, \Lambda)$ and are bounded below by $(s_1, s_2, \Lambda, \Lambda, \Lambda, \Lambda)$. Hence, the set V_{T1} is discrete.

For local left-closure, we look at each coordinate of the vectors. We concentrate on those which have a sequence of length greater than one. In such case, there must be some other vector in the set which, at the specific coordinate, has the same sequence but reduced by one. It can be readily checked diagrammatically that this is the case for the order structure depicted in Figure 5-5.

Therefore, in our approach given the tree structure of a transaction we may derive a formal description of its intended behaviour, in terms of activations of its sub-transactions and the coordination between them. The resulting behavioural patterns (see Fig.5-5) can be analysed before run-time as a means of preventing certain anomalies (such as race conditions) which could result in unexpected behaviour when the transaction actually takes place [Mos05].

It might be worth pointing out that our formal description of the distributed transaction model here we have been concerned with modelling individual transactions, albeit in a way that allows to capture the release of partial results to other transactions. In other words, we have been mostly concerned with the dependencies within a transaction rather than between transactions. For the latter, it would appear that we need to consider the vectors from each and compose, in a principled way, in order to get the resulting inter-transaction behaviour. Previous work on composition within the vector-based representation of behaviour [MoS04] could be exploited in this respect. Further, we note that the properties discussed in Section 5.1.2, discreteness and local left-closure, are shown to be preserved under composition of vectors in [MoS04].

6. Concurrency Control and lock system

-Coordination's Local Autonomy, concurrency control and Local locks

Conventionally a two-phase commit protocol is advised for nested transactions [Mos85]. The necessity for using two-phase commit comes from the nature of business transactions. The first phase is to prepare the transaction; declare dependencies, set up the relationships and use proper locks for indicating the boundaries and side effect of updates and second phase is to finalise or abort the transaction. But in the proposed model, there is a potential intermediate phase, when a failure occurs and one or more coordinators fail. In this situation, the conventional transaction model will abort or restart the whole transaction but we use forward recovery and try to find alternative sub-transactions (or coordinators) or restarting just the failed sub-transaction before restarting the whole transaction (one of the roles of the T_Lock).

The rest of this section gives the details of the mechanism and incentive in the kernel of concurrency control including lock system and methods for applying them. X_Lock (exclusive lock) and S_Lock (share lock) in Traditional transaction model are a sufficient way for covering the integrity and consistency of an atomic transaction. But in SOA, value dependency and conditional commit can not be covered by just these two definitions. Sections 4.2.3 and 4.2.4 provide details of conditional commitment that are used in this section and should be referred to when reading the following.

For solving this problem in our proposed model, introducing two new locks (with specific behaviour) is necessary. I_Lock (Internal Lock) and C_Lock (Conditional-commit Lock) are defined for covering value dependency and partial results releasing in conditional-commit.

-Releasing data inside of a transaction before full condition (I-Lock)

When a sub-transaction wants to release some results before commitment, have to use I_Lock. Therefore, in (for example) parallel coordinator each child not only can use S_Lock and X_Lock but also it can convert X_Lock to I_Lock and release that data item for the other children of the coordinator (applying data dependency). That means the other children can read/modify this data item as well as the owner/generator of data item.

In fact with using I_Lock, generating new logs will be possible, which could be used in creating/updating the IDG. The necessary information from the owner of each I_Lock are the unique identification of the main transaction (IDT), the identification of parent (parallel coordinator) -IDSh and the identification of sub-transaction (IDS). Therefore when another sub-transaction wants to access the data item, validation will be checked by comparing IDSh with parallel scheduler of sub-transaction.

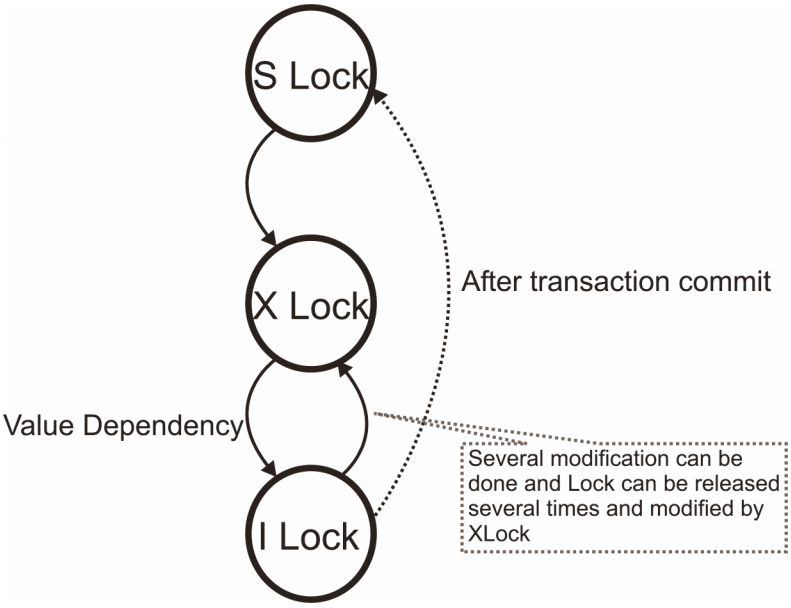


Figure 6-1: I_Lock primary schema

In the sequential coordinator with data dependency [4.2.1] I_Lock again with similar method (like parallel coordinator) is used for this type of access. When each child modifies any data item use X_Lock on it and after committing the child (sub-transaction), X_Lock will be converted to I_Lock and again just sub-transactions (children) with same parent id can access to that data item and modify it.

Alternatively,, when there is a value-dependency, a data item will be released with converting the X_Lock to ILock (Internal Lock). This means the other children of same parent (same as I_Lock owner) can use data item.

-Partial result in conditional commit by using C-lock (after 1st phase of commit)

One of the most important innovations which this transaction model introduces is partial results which are released during a long term transaction before transaction commits (conditional-commit). For integrity and consistency of database, a mechanism in concurrency control and recovery management is designed.

In the first step, a transaction can release its data item by using C_Lock on them (before commit). When a data item has C_Lock, that data item is available but some logs must be written during any usage of data (EDG). The released data item is from compensatory sub-transaction (because in the executable/leaf part of DBE transaction, there are two type sub-transactions: atomic and compensatory). If a failure occurs, the compensating mechanism must be run. In this mechanism, transactions that used the released data item must run same mechanism (rollback/abort).

In the process of conditional-commit data item with XLock can be converted to C_Lock (or I_Lock for Internal data release) for releasing partial results. The necessary information from owner of each C_Lock is unique identification of transaction (IDT) and identification of compensatory sub-transaction (IDS).

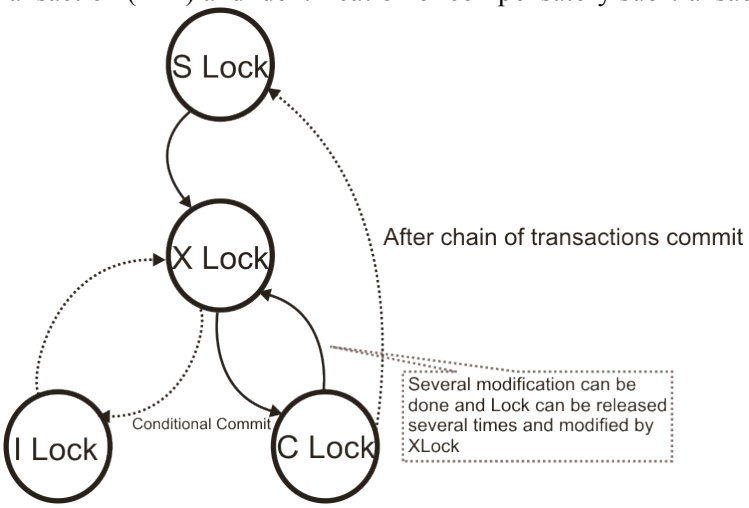


Figure 6-2: C_Lock life cycle

-Recovery Management

There are two famous methods for designing recovery management [Dat96], [Raz99], Shadow paging and Log based. As the proposed transaction is a distributed model that can be extended on a wide range, shadow paging is not a suitable method (in general) because of the value of global overheads [vdMDD+03].

Two types of information are released before proper commitment which makes for a different design for the recovery manager. The first type is released results between sub-transactions of a transaction. The second type is partial results between different transactions before their commitment. For released results in a transaction we use an internal log with the structure of graph that could show the internal routine of recovery when a failure is happen, we call this structure Internal Dependency Graph (IDG). For the partial result (released information between DBE transactions) we can use another log what is showing the external dependency, we call it, External Dependency Graph (EDG).

The graph creation, the order of recovery manager executing and the real routine of releasing result (transaction internal and partial results) is the responsibility of concurrency control. As is clear, the structure and implementation of recovery management and concurrency control are completely merged.

-Fully isolated Recovery and using R-lock

In our transaction model, naturally there are different views about Recovery Management. First of all, as this model covers long-live transaction; the Recovery Management will be a long-time routine. Accruing a

fault in a DBE transaction mostly does not mean full abortion (because of the nature of a distributed network), it could be in some sub-transaction restart or repair. On the other hand, it is important to consider some restart/repair could become a abort/restart chain (in different transaction), that is why Recovery Management is one of critical and important parts of the transaction model.

Recovery Management by using the concurrency control abilities, try to be run in two phases;

1. Preparation phase: by sending a message (abort/restart) to all sub-transaction will send them (and their data) to an isolated mode (preparing for recovery).
2. Atomic Recovery Transaction: Recovery routine will be run as a atomic-isolate procedure that can rollback or just pass (without applying any changes) a sub-transaction.

As shown, Recovery Management in the transaction model is to isolate failure transaction and related transactions (who they used its partial results directly or indirectly), determining the damaged part (where the failure occurred) and rollback to proper check point mainly happens in compensatory sub-transaction, because atomic transaction can be recovered simply like traditional transaction model (normally rollback to start point of that sub-transaction) what could be done by compensatatory transaction (but after applying the preparation phase).

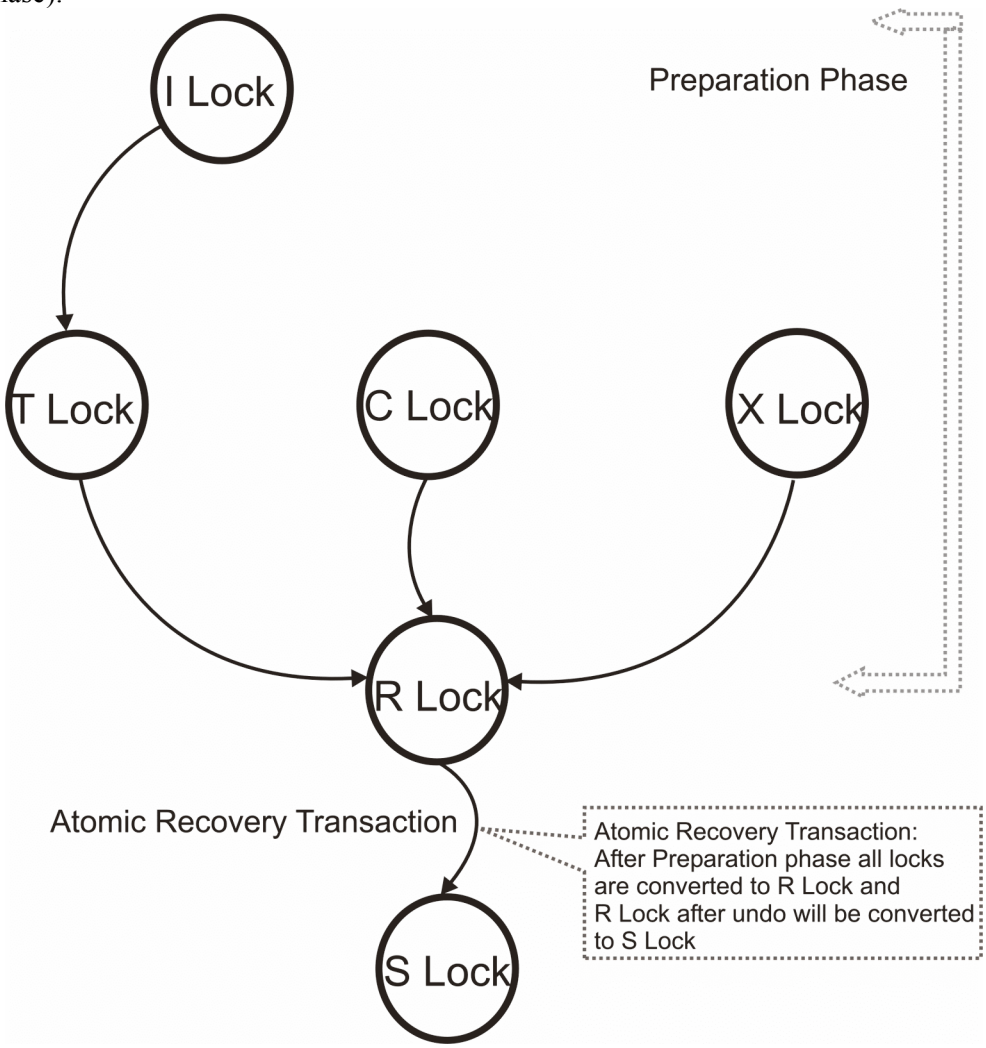


Figure 6-3: Recovery Management as a transaction in concurrency control view

Another benefit of a two phase recovery management is the possibility for saving valuable results providing by safe sub-transactions until the transaction is restart.

-Two Phase Recovery Routine

In the first phase, Recovery Management tries just to isolate the damaged (or failure) part of system by distribution a message that can isolate worked data-items of those sub-transactions. In the transaction model, modified data items can be locked by two different locks; I_Lock and C_Lock.

As it was shown, data items that are locked by I_Lock, can be used just internally (IDG), therefore when the transaction is aborted (or restarted) there is not any danger for misuse of these data items by the other transactions (because they do not have access to these items). These data items naturally can be considered atomic. They will be rolled back (if it was necessary) by using IDG. The only issue is whether we need to rollback all data items? In a transaction just the damaged part (and related data items) must rollback (and related parts can be recognised by IDG).

The other type of modified data items is locked by C_Lock and they are available for all other transactions. Meanwhile by following the EDG, the other transactions which used these partial results are in danger of abortion (or a restart) at least in some parts of transaction. Therefore they must be marked as soon as possible (exactly in preparing phase) because the procedure of rollback for C_Lock can be chains of rollback operations what can take time to commit.

-Solutions

For the critical part of the problem (C_Lock), the lock must be converted to R_Lock (Recovery Lock) by using EDG and without any processing on data. The considered nature for R_Lock must restrict access to data just to Recovery Management transaction. Therefore problem (failure) propagation will be stopped until the Recovery routine is finished.

For the I_Lock optimization, we define T_Lock (Time-out lock) by unique abilities in DBE transaction. T_Lock is like giving a time-out before rollback data item and on the other hand the access to data item will be limited to Recovery routine (avoiding failure propagation). Before finishing considered time-out, Recovery Management has opportunity for reconverting T_Lock to I_Lock (if rollback is not necessary) but after finishing the time-out the data item will be rolled back automatically.

-T-Lock role in failures

In the preparation phase, C_Lock (in all related transations) data items were converted to R_Lock by using EDG and all I_Lock data items were converted to T_Lock by using IDG. Therefore Recovery Management in phase two is like full ACID transaction that is full isolated in the lifetime of transaction.

On the other hand, with a suitable data structure, the recovery manager transaction is optimized as much as possible by providing not only special concurrent (by introducing automatic T_Lock structure) operations, but also possibility for saving the valuable results of some sub-transactions even when the transaction has failed and been restarted.

The phase two transaction will be done by traversing EDG and IDG. For rollback of partial results, traversing the EDG will help to create/execute compensatory transactions. Normally the T_Lock can provide for an automatic rollback operation (after passing the time-out) but for revalidating the correct data-items before time-out, recovery manager transaction can traverse IDG and reconvert T_Lock to I_Lock

Full Lock schema

There are 6 different locks in Concurrency Control of the transaction model. Two locks (R_Lock and T_Lock) are related to Recovery atomicity and optimization. The ReadLock and XLock (eXclusive Lock) have completely similar behaviour to a conventional transaction model. But value dependency and conditional commit (partial result) can change the ReadLock /XLock behaviour (Figure 6-4).

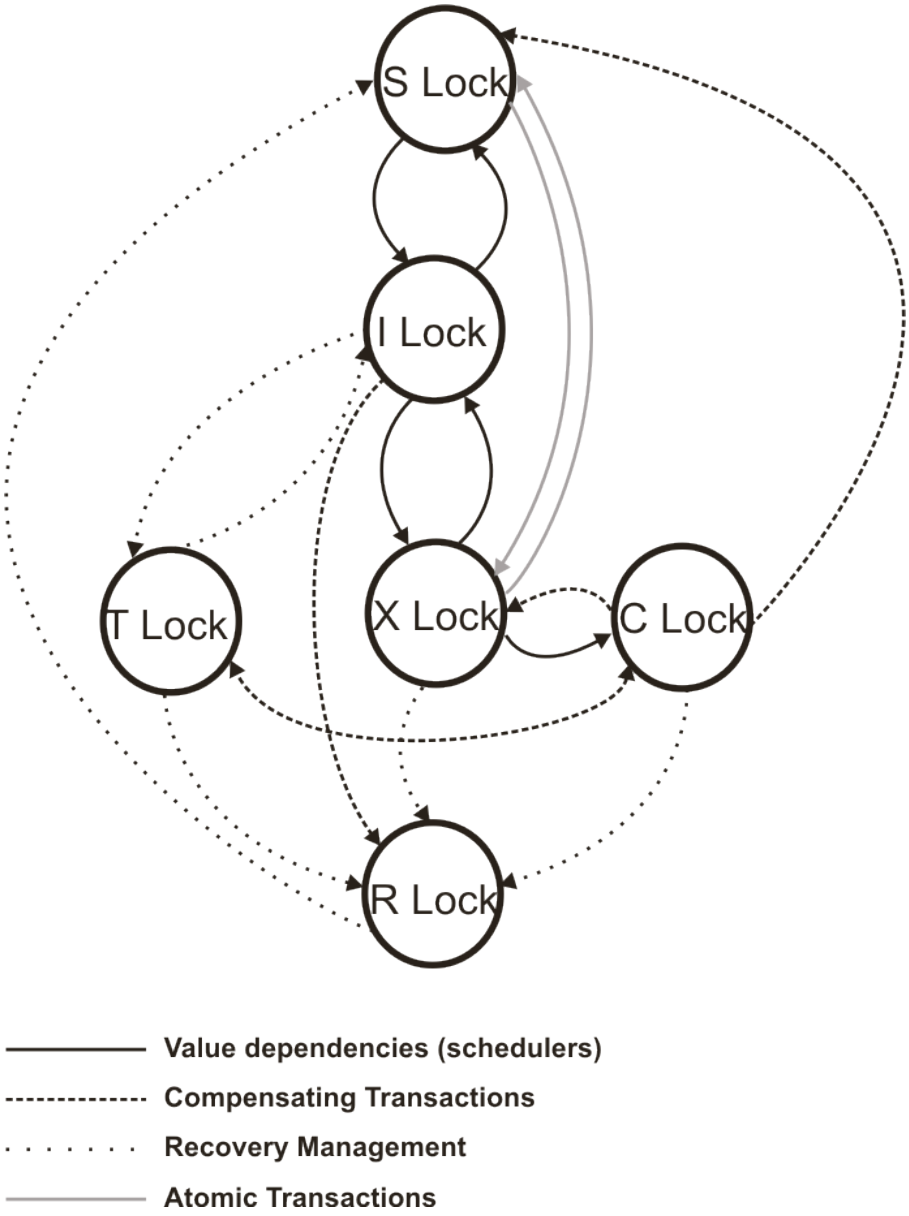


Figure 6-4: Full locks view

7. A Peer-to-Peer network for this business environment

In this section, first we analyse the requirements and necessity of a peer-to-peer (P2P) network for DBE and then we review different peer-to-peer networks, and analyse the unique characteristics of Digital business environment should fulfil and the need of a fully distributed peer-to-peer network with specific properties will be end of this section.

7.1. Peer to Peer requirements and necessities

We have described a transactional model for supporting critical requirements of Digital Business Ecosystems in the main part of this report. However, fulfilling all these requirements also puts constraints on the design of a Peer to Peer (P2P) Network structure. The central problem is the P2P network itself *must* respect the local autonomy of the participating SMEs. Unfortunately, any dependency on large enterprises for providing hubs on the network and introducing centralised (limited decentralised) discovery system such as UDDI, (even with providing distributed coordination frameworks), introduces two blocks on achieving the goals of a DBE. Firstly, this seriously impacts on the ability for small and medium businesses to advertise their services in a fair and computational environment. Secondly, such dependencies fail to provide transparent loosely coupled binding with large enterprises that is needed to preserve the local autonomy of SMEs. In addition, the current state of technology provides serious technical problems for the implementation of explorative and semi-fixed compositions. This is simply because these can cause huge amounts of traffic on any centralised registry and the whole structure will rely on few critical hubs.

A key driver behind DBE is that the whole computational infrastructure must serve the needs of the DBE community as a whole. The DBE computing environment must be provided by and for its whole community of users. There must be no opportunity for one organization, or even a small number of organizations, to be in a position to benefit from, or (intentionally or otherwise) create risks to, the routine business activities of other members of the DBE community.

Consequently:
- There must be no critical dependencies on single organizations.
- There must be no critical points of failure.
- Hence the environment should be truly distributed.

The categorized requirements for DBE network can be considered as below:

Consistency

As has been mentioned already, a key aspect of the DBE P2P environment is that it must be optimized for the needs of business, and not just the sharing of information within a community. Within a single business transaction, there may be a set of sub-transactions involving different organizations. Each sub-transaction may involve a degree of financial risk should the transaction as a whole fail (e.g. if a sub-transaction involves purchase of a flight ticket, it may not be possible to recover part or even all of the ticket cost should a later stage in the transaction lead to failure and roll back).

Hence, DBE must
- Support a highly transactional environment which can cause huge amount of traffic.
- Support long-running transactions which need replication of critical system logs.

Latency

This is a critical impact factor on the performance of the infrastructure in supporting long-running business transactions.

Local autonomy

Although the DBE network as a whole must belong to the community to enable their services to be freely published and discovered, the autonomy of individual nodes within the network must be respected.

Hence:

- There must be no limitation to the ownership or decision making of a single node's data (local node has full control over its own data)

Reliability

- Address/manage failure risks for any infrastructural services

Security

- Primary structure must fully support development of a secure transaction model

Availability

- Consistent replication

Boundaries

- In principle there should be no boundaries to the growth of the network and must be a scale-free network
- Hence requires a dynamic structure to handle bandwidth + load balancing issues (maybe not just traffic but also local conditions)

Limitations

- Infrastructure must not restrict business solutions

Technical Requirements

Completely distributed infrastructure

- Not centralized (including clusters with strong dependencies)
- Not limited decentralization

Bootstrapping

- Accessibility (ability for new members to register with and join into the network)
- Performance
- Security

Growth

- Algorithm for growth must ensure critical properties are preserved as the network grows

Maximum Distance

- Shortest path algorithm
- Scale-free

Durability

- High reliability of the whole network
- Needs a reliability model for the network to understand what guarantees can be made

Include gateway to services external to DBE

Defragmentation Algorithm

One of the serious dangers for a distributed network is that of fragmentation of the network into separate sub-networks (islands), (figure 7-2). In this situation, peer-to-peer connections between different nodes, running different transactions and queries, may fail. This could lead to critical failures in long-term

transactions, failure to provide optimal responses to queries, and ultimately a collapse of the community that the network aims to support.

There are several algorithms for de-fragmentation of a distributed network. However, as with all things prevention is better than a cure; if the network can be designed to be stable against possible de-fragmentation, then significant network traffic to monitor and repair fragmentation can be avoided. Theoretically, the potential for fragmentation can be established during the creation of the network and the main reason for fragmentation is a missing routine in the birth and growth protocols of the network.

Therefore, as a first step, DBE as a distributed network needs a Birth model that avoids fragmentation (figure 7-1). Subsequently a growth algorithm is needed that guarantees connectivity of the distributed network.

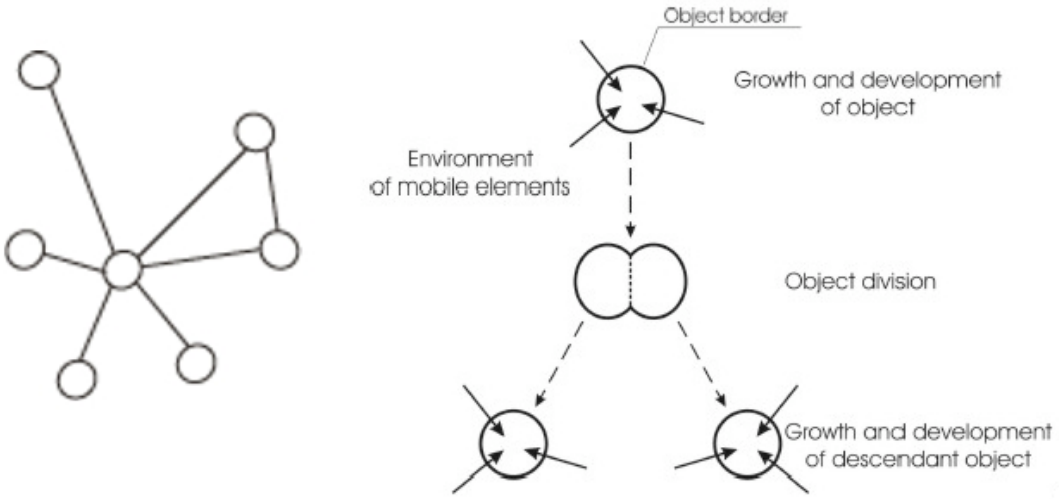


Figure 7-1: Growth through replication provides strong connectivity between nodes

To achieve some guarantee of connectivity between different nodes, we have to have strong connectivity between nodes. At the same time, for efficiency of transactions and queries we need a short mean path between different nodes. One of the best known mechanisms to achieve these desirable features is replication. Different levels of replication can not only provide strong connectivity and shorter path lengths between nodes but can also be used in the de-fragmentation algorithm (for safe and secure de-fragmentation).

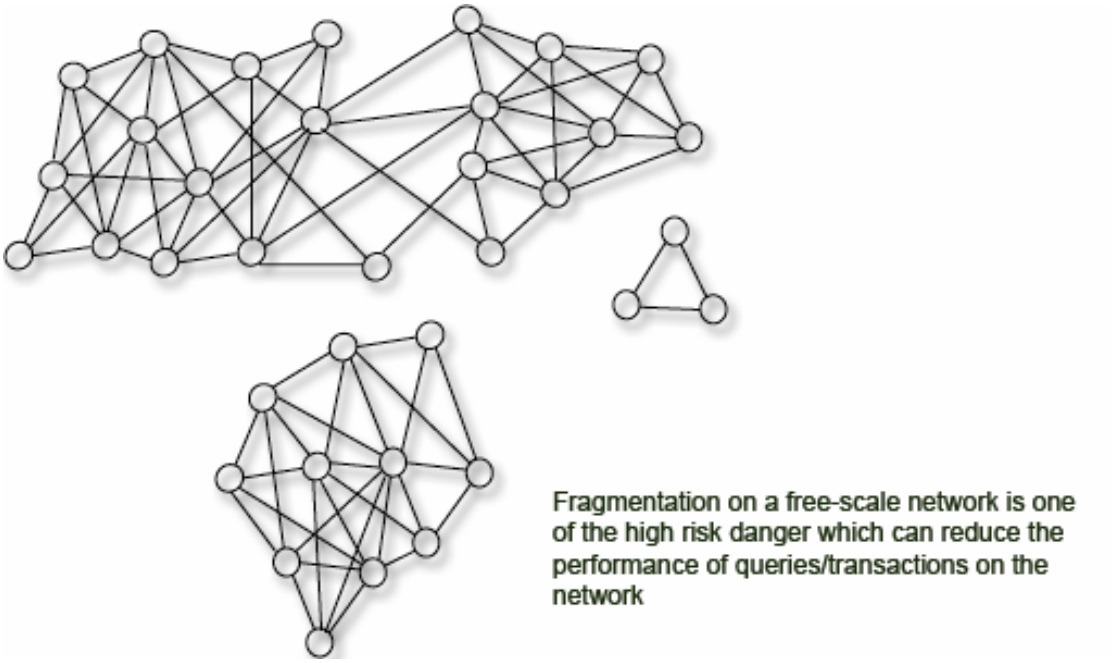


Figure 7-2: Fragmentation

In the next section, we will review the main algorithms for replication models. In addition, the relationship between a replication model and a birth-model (de-fragmentation) will be reviewed and analysed.

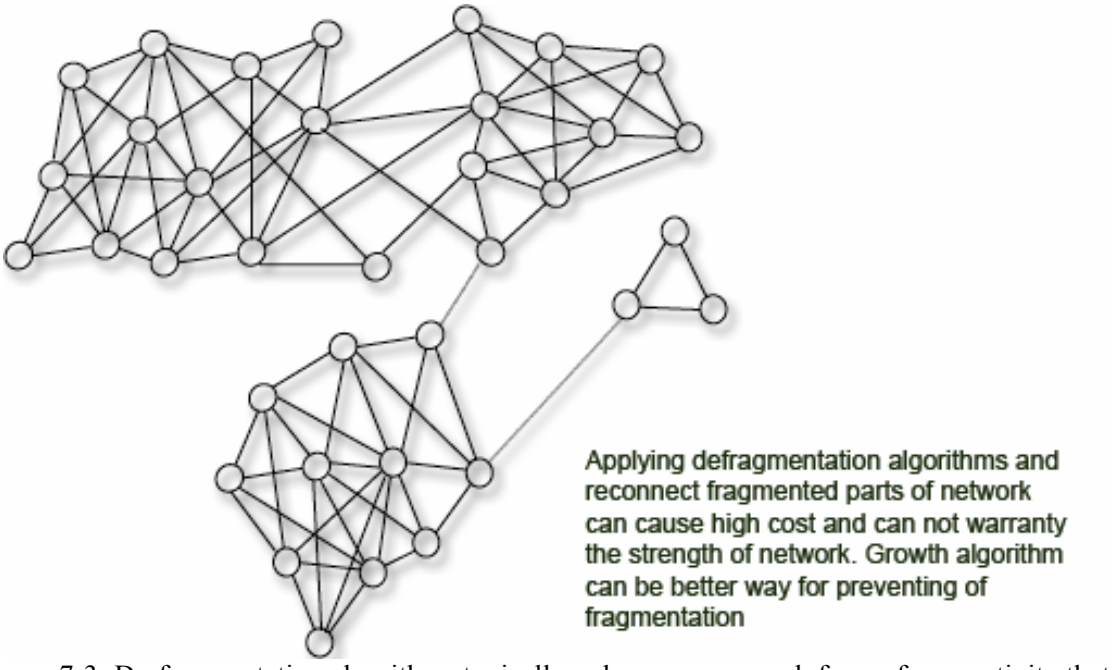


Figure 7-3: De-fragmentation algorithms typically only recover a weak form of connectivity that is not robust against subsequent fragmentation

The concepts of a Recovery Model and a Replication Model are important n the remainder of this section. Please keep the following points in mind:

Recovery Model

- Self-healing
- Needs a Good Theoretical understanding of the boundaries for recovery

Replication Model

- Replication should be consistent
- Which information is replicated?
- Supports Delegation – to enable roles or responsibilities to be rapidly delegated should a node be partially or wholly unable to fulfill them for some period of time.

Overall, we need to understand the importance of DBE as an environment that must be designed so that critical concepts stay alive. We can make a very useful analogy with the Gaia hypothesis, which proposes that the Earth as a dynamic system which will react to perturbations in such a way that the system as a whole continues to survive.

7.2. Current implementations and designs

The Intel P2P working group gave the definition of P2P as "The sharing of computer resources and services by direct exchange between systems" [Kan01]. This thus gives P2P systems two main key characteristics:

- Scalability: There is no algorithmic or technical limitation to the size of the system; for example, the complexity of the system should be somewhat constant regardless of the number of nodes in the system.
- Reliability: The malfunction on any given node will not affect the whole system (or maybe even any other nodes).

Based on the research of Schmidt and Parashar [ScP03], P2P can be categorized at least into two groups by the type of model: pure P2P and hybrid P2P. A pure P2P model, such as Gnutella and Freenet, does not have a central server. The hybrid P2P model, such as Napster, Groove, and Magi, employs a central server to obtain meta-information such as the identity of the peer on which the information is stored or to verify security credentials. In a hybrid model, peers always contact a central server before they directly contact other peers. We will refer back to these two categories in the following review.

➢ Napster:

In May 1999, Shawn Fanning and Sean Parker created Napster Inc., thus beginning an unforeseen revolution. Napster devised a technology for using peer-to-peer connections to exchange compressed music files (MP3s). Due to the fact that Napster is not an open source application, it was only possible to build up a similar application to reveal the Napster protocol through reverse engineering (OpenNap project; Turcan, 2002; [DNB03]). Most analyses have been based on a reverse engineering exercise and so cannot guarantee that they capture the exact architecture of Napster.

In respect of its underlying centralized directory model, the early Napster (Napster, 2000 [DNB03]) can be viewed as a nearly perfect example of a hybrid P2P system in which a part of the infrastructure functionality, in this case the index service, is provided centrally by a coordinating entity. The moment a peer logs into the Napster network, the files that the peer has available are registered by the Napster server. When a search request is issued, the Napster server delivers a list of peers that have the desired files available for download. The user can obtain the respective files directly from the peer offering them.

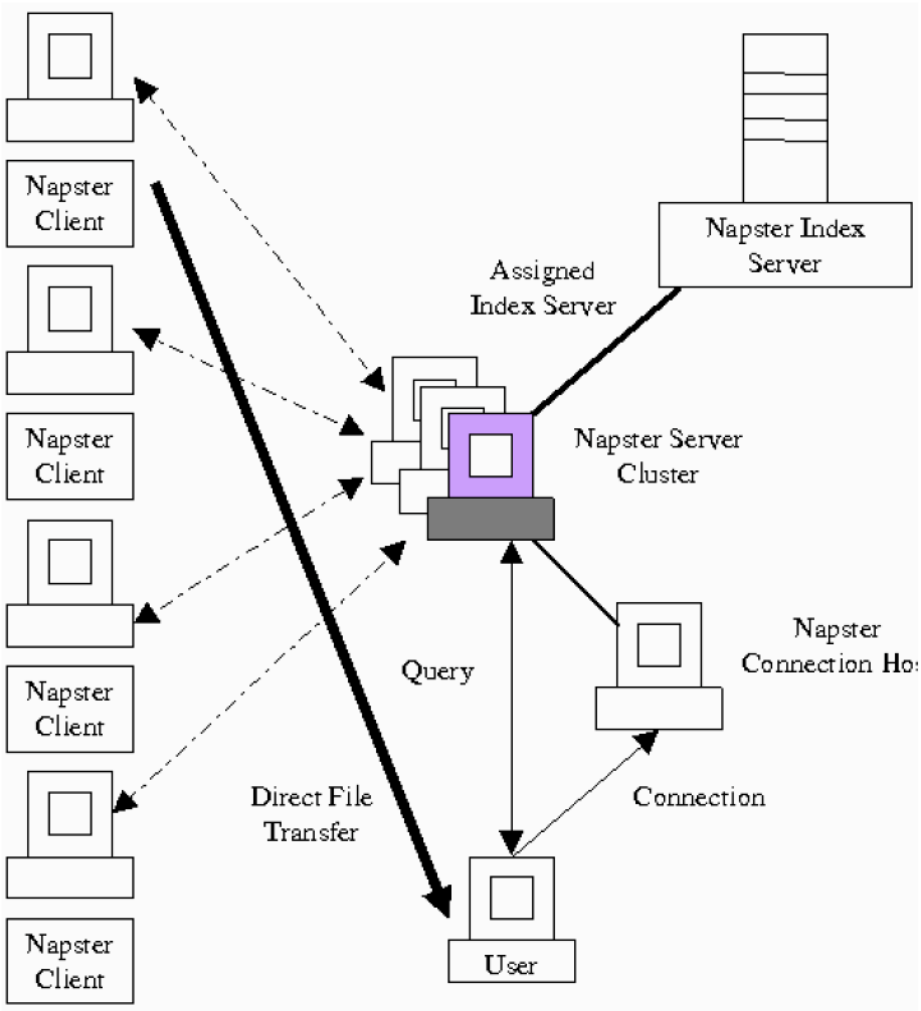


Figure 7-4 "Napster" structure. We use quotes to emphasise this based on a re-engineering of the original Napster

In an efficient "Napster" design (Figure 7-4), there is a server-client structure where there is a central server system, which directs traffic between individual registered users. The central servers maintain directories of the shared files stored on the respective PCs of registered users of the network. These directories are updated every time a user logs on or off the Napster server network. Clients connect automatically to an internally designated "metaserver" that acts as common connection arbiter. This metaserver assigns at random an available, lightly loaded server from one of the clusters. Servers appear to be clustered about five to a geographical site and Internet feed, and able to handle up to 15,000 users each. The client then registers with the assigned server, providing identity and shared file information for the server's local database. In turn, the client receives information about connected users and available files from the server. Although formally organized around a user directory design, the Napster implementation is very data-centric. The primary directory of users connected to a particular server is only used indirectly, to create file lists of content reported as shared by each node [Bar01].

- **Gnutella:**

In early March 2000, Justin Frankel and Tom Pepper, who were both, working under Gnullsoft, created Gnutella which is one of AOL's subsidiaries. AOL halted distribution shortly after it was published, but the short duration where Gnutella was made online was enough to allow interested programmers to download and later reverse engineer Gnutella's communication protocol. As a result, a number of Gnutella clones with improvements were introduced (for example, LimeWire, BearShear, Gnucleus, XoloX, and Shareaza).

P2P networks that are based on the Gnutella protocol function without a central coordination authority. All peers have equal rights within the network. Search requests are routed through the network according to the flooded request model, which means that a search request is passed on to a predetermined number of peers. If they cannot answer the request, they pass it on to other nodes until a predetermined search depth (ttl = time-to-live) has been reached or the requested file has been located. Positive search results are then sent to the requesting entity, which can then download the desired file directly from the entity that is offering it. A detailed description of searches in Gnutella networks, as well as an analysis of the protocol, (can be found in [RLF02] and [Rip01]). Due to the fact that the effort for the search, measured in messages, increases exponentially with the depth of the search, the inefficiency of simple implementations of this search principle is obvious. In addition, there is no guarantee that a resource will actually be located. Operatingsubject to certain prerequisites (such as nonrandomly structured networks), numerous prototypical implementations (for example, [CrP02], [RoD01], [RFH+01], [LCC+02], [ALH02]) demonstrate how searches can be effected more "intelligently" (see, in particular, [DKR02], and also [AbH02] for a brief overview).

- **FastTrack**

The FastTrack network as a hybrid architecture is a cross between centralized and decentralized topologies ([YaG02]). Very little is known of the actual protocol used. Many attempts have been made to reverse engineer the FastTrack protocol. The most well known to date would be the giFT project[6] as they were the closest to finally cracking the protocol. Our review is based on this reverse engineering research ([Ber03]). This technology uses two tiers of control in its network. The first tier is made up of clusters of ordinary nodes that log onto Super Nodes (ordinary machines with high-speed connection). As discussed previously, this sort of connection mimics the centralized topology. The second tier consists of only Super Nodes that are connected to one another in a decentralized fashion. These Super Nodes are just ordinary nodes which can and will join or leave the network as they please. In order to ensure the constant availability of the network, there exists a need for a dedicated peer (or several of these peers) that will monitor and keep track of the network. Such a peer is called a bootstrapping node ([KuR03]) and it should always be available online.

Resource discovery is accomplished through the act of broadcasting between Super Nodes. When a node from the second tier makes a query, it is first directed to its own Super Node, which will in turn broadcast that same query to all other Super Nodes to which it is currently connected. This is repeated until the TTL of that query reaches zero. So, if for example, the TTL of a query is set to 7 and the average amount of nodes per Super Node is 10, a FastTrack client is able to search 11 times more nodes on a FastTrack network as compared to Gnutella ([ABC+01]). This provides FastTrack clients with a much greater coverage and better search results. However, there is an important drawback to such a broadcasting method, and that is the daunting amount of data that needs to be transferred from Super Node to Super Node. This is the very same problem that has been plaguing the Gnutella network.

But by defining Super Nodes as nodes that are guaranteed to have fast connections, it tries to overcome to problem. Each of the Super Nodes that received the query performs a search through its indexed database that contains information of all the files shared by its connected nodes. Once a match is found, a reply will be sent back following the same path the search query was propagated through until it reaches the original node that issued the query. This method of routing replies is similar to Gnutella, and hence runs the risk of facing the same problem of losing replies as it is routed back through the network. This is due to the fact that the Gnutella network backbone, as mentioned previously, is made up of peers that connect and disconnect from the network very sporadically (One of the most famous P2P networks with FastTrack structure is KaZaa).

[6] See: http://sourceforge.net/projects/gift/ or, http://gift.sourceforge.net/

➢ OpenFT

Like FastTrack, the OpenFT protocol also classifies the nodes in its network into different roles, but instead of a two-tier control architecture, OpenFT added an extra tier, making it a three-tier control architecture. The classification of nodes is based on the speed of its network connection, its processing power, its memory consumption, and its availability ([ABC+01]).

The first tier is made up of clusters of ordinary machines, which we refer to as User Nodes. These nodes maintain connections to a large set of Search Nodes (ordinary machines with high speed connection). The user nodes then update a subset of the search nodes to which it is connected with information regarding files that are being shared ([DNB03]). The second tier is made up of machines that are referred to as Search Nodes. These nodes are the actual servers in the OpenFT network. These servers have the responsibility to maintain indices of files that are shared by all the User Nodes under them. On default, a Search Node can manage information about files stored at 500 User Nodes. The third tier is made up of a group that it much smaller, because the requirements to qualify for this group are much more stringent. One has to be a very reliable host that has to be up and available most of the time. These nodes are referred to as Index Nodes as their main purpose is to maintain indices of existing Search Nodes. They also perform tasks such as collecting statistics and monitoring network structure.

➢ Freenet:

Freenet ([CSW+00]) was designed to prevent the censorship of documents and to provide anonymity to users. Unlike Gnutella, which uses a breadth-first search (BFS) with depth TTL, Freenet uses a depth-first search (DFS) with a specified depth. Each node forwards the query to a single neighbor and waits for a response from the neighbor before forwarding the query to another neighbor (if the query was unsuccessful) or forwarding the results back to the query source (if the query was successful).

Searching for and storing files within the Freenet network ([CMH+02]) takes place via the so-called document routing model. A significant difference to the models that have been introduced so far is that files are not stored on the hard disk of the peers providing them, but are intentionally stored at other locations in the network. The reason behind this is that Freenet was developed with the aim of creating a network in which information can be stored and accessed anonymously. Among other things, this requires that the owner of a network node does not know what documents are stored on his/her local hard disk. For this reason, files and peers are allocated unambiguous identification numbers. When a file is created, it is transmitted, via neighboring peers, to the peer with the identification number that is numerically closest to the identification number of the file and is stored there. The peers that participate in forwarding the file save the identification number of the file and also note the neighboring peer to which they have transferred it in a routing table to be used for subsequent search requests.

➢ Groove

The Groove platform provides system services that are required as a foundation for implementing P2P applications. A well-known sample application that utilizes this platform is the P2P Groupware Groove Virtual Office. The platform provides storage, synchronization, connection, security and awareness services. In addition, it includes a development environment that can be used to create applications or to expand or adapt them. This facilitates the integration of existing infrastructures and applications (such as Web Services or .NET Framework).

Groove Virtual Office is the current best-known application for collaborative work based on the principles of P2P networks. This system offers standard collaboration functions (instant messaging, file sharing, notification, co-browsing, whiteboards, voice conferences, and databases with real-time synchronization) to those of the widely used client/server based Lotus products, Notes, Quickplace, and Sametime, but does not require central data management. All of the data created are stored on each peer and

are synchronized automatically. If peers cannot reach each other directly, there is the option of asynchronous synchronization via a directory and relay server. Groove Virtual Offce offers users the opportunity to set up so-called shared spaces that provide a shared working environment for virtual teams formed on an ad hoc basis, as well as to invite other users to work in these teams. Groove Virtual Office can be expanded by system developers. A development environment, the Groove Development Kit, is available for this purpose ([Edw02]). Similar system, IBM Sametime Server-based e-Meetings (Sametime) is server oriented. However, the process of knowledge sharing in such systems regardless of their physical architecture is often P2P.

- **Jabber**

The Jabber Open Source Project (http://www.jabber.org/) is aimed at providing added value for users of instant messaging systems. Jabber functions as a converter, providing compatibility between the most frequently used and incompatible instant messaging systems of providers, such as Yahoo, AOL, and MSN. This enables users of the Jabber network to exchange messages and present information with other peers, regardless of which proprietary instant messaging network they actually use. Within the framework of the Jabber-as-Middleware-Initiative, Jabber developers are currently working on a protocol that is aimed at extending the existing person-to-person functionality to person-to-machine and machine-to-machine communication ([Mil01]).

- **JXTA**

JXTA is an open platform that aims at creating a virtual network of various digital devices that can communicate via heterogeneous P2P networks and communities. The specification includes protocols for locating, coordinating, monitoring and the communication between peers (Project JXTA; [Gon01]).

- **Other P2P networks and current research works**

There are varieties of P2P networks and even there are different implementations and discovery system in the current P2P architecture. In this review, we mentioned little important and popular architecture but the next chapter we try to discuss about important part of all of these structures and give a brief view about them (in next report we may explain more details about this review part but unfortunately the limitation for providing this report does not allow us to explain more details and reviewing other architectures).

7.3. Formal models for Peer-to-Peer networks

In this part, we try to find a global categorization for P2P networks, especially in the term of network theory. In this section, we use different views and models from classical Erdös and Rényi model to Watts and Strogatz's small-world [WaS98], from Barabasi's Scale-free network [BAJ99] to evaluation frameworks and analysis of Adamic [AdH00a] , [AdH00b] ,[ALH02] and [ALH01] from HP labs. In the resource discovery system, boot strapping, query responds time, network growth limitations, fragmentation possibilities and even probability of transaction abortion, effect of this general models can be shown.

For a long time, the modelling of physical as well as non-physical systems and processes has been performed under an implicit assumption that the interaction patterns among the individuals of the underlying system or process can be embedded onto a regular and perhaps universal structure such as a Euclidean lattice (regular network). The side effect of this became apparent in the early days of Napster when a centralized server was used to keep an index which was used for client registrations, responding to queries and creating peer to peer connection between nodes (client and users). This model is more popular in another areas of P2P networking such as bootstrapping or keeping the dynamic structure of rings of servers.

Two mathematicians, Erdös and Rényi (ER), made a breakthrough in the classical mathematical graph theory. They described a network with complex topology by a random graph. Fundamentally, Gnutella uses this model

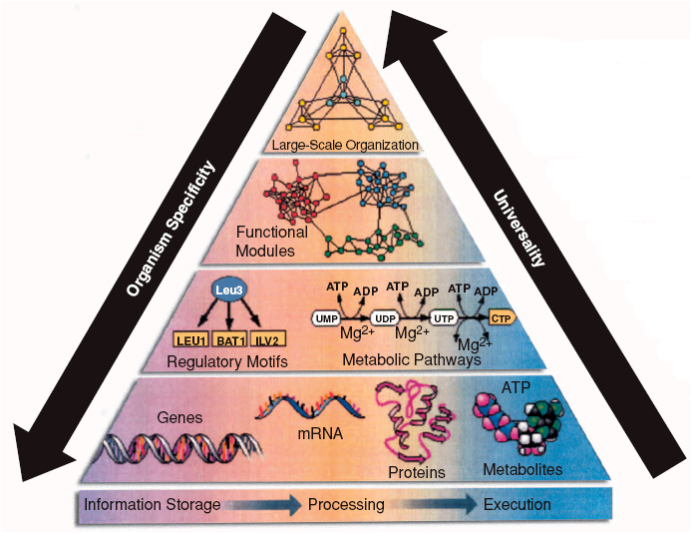


Figure 7-5 Complexity Pyramid [BaA99]

Many real-life complex networks are neither completely regular nor completely random, and in peer to peer networks even in just file sharing system, these two models had lots of problems and practical implementations showed lots of evidences for that.

A key model that motivated the development of several P2P networks (especially discovery systems) is the Small-world model. A conceptual definition of this model was introduced by Watts and Strogatz (WS-1998) [WaS98]. A prominent common feature of the ER random graph and the WS small-world model is that the connectivity distribution of a network peaks at an average value and decays exponentially. Such networks are called “exponential networks” or “homogeneous networks,” because each node has about the same number of link connections.

Scale-free networks ([BAJ99], [BaA99] and [Bar03]) or “power-law networks”, differ fundamentally in their connectivity distribution from Small worlds and random graphs. They are scale-free in the sense that their connectivity distributions are in a power-law form that is independent of the network scale [Bar03]. Differing from an exponential network, a scale-free network is inhomogeneous in nature: most nodes have very few link connections and yet a few nodes have many connections.

7.4 Key Attributes for the assessment of alternative architectures

Given the significant volume of theoretical analyses of network structures, it is not surprising that a number of attributes have been identified that can be used to characterise certain aspects of network topologies. This section provides a glossary of the main ones.

- Average path:

The *average path length L* of the network is defined as the mean distance between two nodes, averaged over all pairs of nodes.

We clearly need to minimise the average path length, in terms of number of hops needed to find the answer to a query. In addition, we must do this in a way so that the *optimal* solution to a query is found (for example, a search for the cheapest flight, should be guaranteed to find the cheapest flight available in the *whole* network, and not just a local minimum).

- Clustering coefficient

One can define a *clustering coefficient C* as the average fraction of pairs of neighbours of a node that are also neighbours of each other. Suppose that a node *i* in the network has *ki* edges and they connect this node to *ki* other nodes. These nodes are all neighbours of node *i*. Clearly, at most *ki (ki* − 1*)/*2 edges can exist among them, and this occurs when every neighbour of node *i* is also connected to every other neighbour of node *i*. The clustering coefficient *Ci* of node *i* is then defined as the ratio between the number *Ei* of edges that actually exist among these *ki* nodes and the total possible number *ki(ki* − 1*)/*2, namely, *Ci* = 2*Ei/(ki(ki* − 1*))*. The clustering coefficient *C* of the whole network is the average of *Ci* over all *i*. Clearly, *C* ≤ 1; and *C* = 1 if and only if the network is globally coupled; that is, every node in the network connects to every other node. In a completely random network consisting of *N* nodes, *C* ~ *1/N*, which is very small as compared to most real networks. It has been found that most large-scale real networks have a tendency toward clustering, in the sense that their clustering coefficients are much greater than *O(*1*/N)*, although they are still significantly less than one (i.e, far away from being globally connected). This, in turn, means that most real complex networks are not completely random. Therefore they should not be treated as completely random and fully coupled lattices alike.

In many cases the nodes in a network are clustered according to some common domain. Indeed, we typically see this in the internet, where there is a high degree of connectivity amongst nodes concerning a common subject matter. This is already pointing us towards the general network topology we need for DBE. That is, it needs to be scale-free on a global basis, but with local small-worlds of highly connected clustered nodes.

- Degree of a node:

The degree k_i of a node *i* is usually defined to be the total number of its connections. The average of k_i over all *i* is called the *average degree* of the network, and is denoted by $< k >$.

-Distribution degree (function) P(k);

The spread of node degrees over a network is characterized by a distribution function *P(k)*, which is the probability that a randomly selected node has exactly *k* edges. A regular lattice has a simple degree sequence because all the nodes have the same number of edges; and so a plot of the degree distribution contains a single sharp spike (delta distribution). Any randomness in the network will broaden the shape of this peak. In the limiting case of a completely random network, the degree sequence obeys the familiar Poisson distribution; and the shape of the Poisson distribution falls off exponentially, away from the peak value $< k >$. Because of this exponential decline, the probability of finding a node with *k* edges becomes negligibly small for $k >> < k >$.

Many empirical results showed that for most large-scale real networks the degree distribution deviates significantly from the Poisson distribution. In particular, for a number of networks, the degree distribution can be better described by a power law of the form

$P(k) \sim k^{-\gamma}$

The power-law distribution falls off more gradually than an exponential one, allowing for a few nodes of very large degree to exist. Because these power-laws are free of any characteristic scale, such a network with a power-law degree distribution is called a *scale-free network* (most real networks are not fully connected and their number of edges is generally of order *N* rather than N^2)

ER (Erdös and Rényi) model (random graph) Versus Small-World

Erdös and Rényi showed that, if the probability *p (probability for having an edge between any two nodes)* is greater than a certain threshold *pc* ~ *(*ln *N)/N*, then almost every random graph is connected, and ER random graph with *N* nodes has about *pN(N* − 1*)/*2 edges. The average degree of the random graph is $< k >= p(N-1) \cong pN$.

If L_{rand} be the average path length of a random network. Intuitively, about $< k >^{L_{rand}}$ nodes of the random network are at a distance L_{rand} or very close to it. Hence $N \sim < k >^{L_{rand}}$, which means $L_{rand} \sim \ln N / < k >$.

This logarithmic increase in average path length with the size of the network is a typical small-world effect. Because *LnN* increases slowly with *N*, it allows the average path length to be quite small even in a fairly large network.

The clustering coefficient of the ER model is $C = p =< k> /N << 1$. This means that a large-scale random network does not show clustering in general. In fact, for a large *N*, the ER algorithm generates a homogeneous network, where the connectivity approximately follows a Poisson distribution.

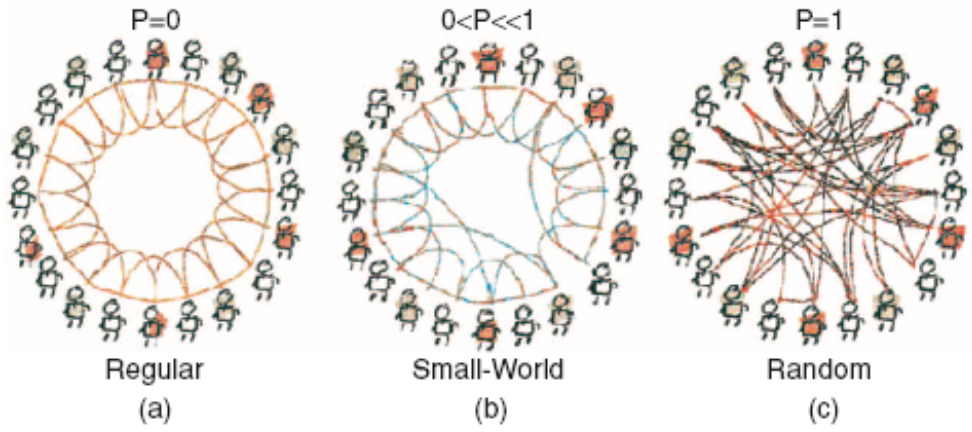


Figure 7-6 Classic comparison between regular, small-world and random network

Simple classic algorithm for WS Small-World Model:

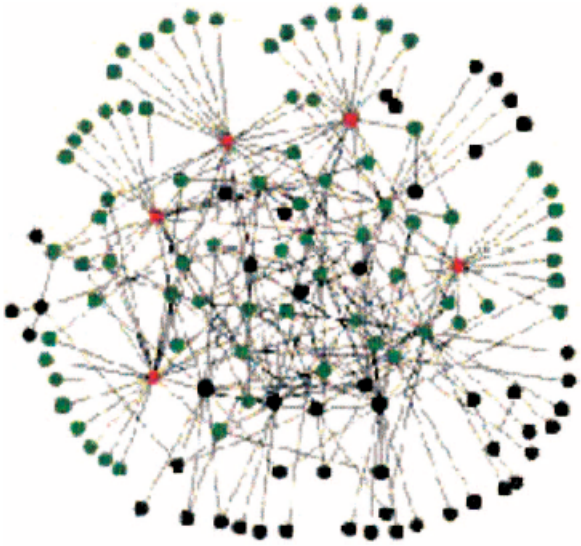
1) Start with order: Begin with a nearest-neighbour coupled network consisting of N nodes arranged in a ring, where each node i is adjacent to its neighbour nodes, i = 1, 2, … , K/2, with K being even.

2) Randomization: Randomly rewire each edge of the network with probability p; varying p in such a way that the transition between order (p = 0) and randomness (p = 1) can be closely monitored.

Simple classic algorithm for Scale-free Model:

1) Growth: Start with a small number (m_0) of nodes; at every time step, a new node is introduced and is connected to $m \leq m_0$ already-existing nodes.

2) Preferential Attachment: The probability $\prod_i$ that a new node will be connected to node *i* (one of the *m* already-existing nodes) depends on the degree k_i of node *i*, in such a way that $\prod_i = k_i / \sum_j k_j$.



A scale-free network of 130 nodes, generated by the BA scale-free model. The five biggest nodes are shown in red, and they are in contact with 60% of other nodes (green).

Figure 7-8 Scale free network

7.3 . Brief overview of global information structure of current models:

Using the attributes of the preceding sub-section, together with the requirements for supporting long-life transactions, we can now provide an evaluation of various network structures. This highlights their respective strengths and weaknesses in the context of DBEs.

a) Centralised design:

- Reasonable average path and diameter; in centralised average path is minimum (accessing to server and reaching the target) and even diameter is reasonably low. This can be considered as a reason for using a centralized system but other problems can affect the performance of such a system dramatically; indeed manipulation of huge amounts of data can be impossible in a centralised system.
- Certainty in resource/service discovery; theoretically all answers to queries can be reached (if the network cost does not affect this performance).
- High cost growths after reaching the traffic limitation; if the system is highly service oriented or resource (especially file) sharing is part of system, the network traffic will increase quickly and the central point will have to be improved regularly. Otherwise the rate of failure will affect the system consistency.
- Long term (long-running) transactions problem: the effect of Long term (long-running) transactions (as explained in Section 5), can be more than other designs (lack of replication causes/increases the problem and long waiting for other transactions, when a Long term (long-running) transaction work on a data item).
- High risk in critical situations (attacking/failure on server, exalted traffic period and etc); attack or any failure on server may destroy the whole network, server as a centre of network is critical point of network.
- High cost recovery system; recovery management will effect the performance of server and with this the whole network performance will decrease (because this network is highly dependent to its server).
- Single point of failure

b) Scale-free network with Power-law structure:

- Reasonable search performance (average path and diameter) in power-law distribution degree; one of the proven attribute of scale-free network is low average path (actually it was the point of discovering this network in the nature). This useful attribute comes from power-law distribution degree, which can side effect in different situation.
- Reasonable clustering coefficient; another proven property of scale-free networks, as discussed earlier. This is one of the important useful attributes of this type of network.
- Flat growth; dead-lock and exponential traffic generation; as we know the growth in scale-free networks is two dimensional and because of power-law nature of distribution degree, it is focused on few nodes. Then based on different scenarios (different transactions) and dependency to those few nodes (with very high degree), the probability for dead-lock increases and network traffic can increase based on same pattern of distribution degree.
- High degrees nodes drop down with the growth of degree; this is one of the most important problems in this architecture, because first of all dependency on a few nodes cause a need for improvement of bandwidth for those specific nodes (improving the bandwidth of other nodes can not help), otherwise network performance can reduce. On the other hand, based on the dynamic nature of this network if for any reason the importance of any this nodes reduces (see Adamic's statistical study in HP labs [ALH02] and [ALH01]), not only this cost (for improving the bandwidth) is wasted, but also we have to improve the bandwidth of other high degree nodes for compensating the lack of cooperation of that specific node.
- Self-attack as an exponential growth based on age of platform ([BAJ99], [BaA99] and [Bar03]); based on Barabasi's research, the degree of highest nodes will increase based on their age by power-law.
- Performance reduction; Long term (long-running) transactions, uncertainty in resource/service discovery (result of high traffic on a high degree nodes and no warranty for reaching all nodes for reaching a certain answer in a query)
- High probability for Fragmentation

c) WS Small World network (different versions of Chord)

- Reasonable average path and diameter (as discussed earlier in the current chapter).
- Reasonable clustering coefficient (as discussed earlier in the current chapter).
- Network growth problem; it is the most important problem, because the nature of WS small-world is static and if we allow a highly dynamic growth, the network will start to change to scale-free network (as a solution some P2P networks try to mix a hybrid solution with centralized system and used centralized node for some specific tasks, such as bootstrapping and system registry which again we have to deal with centralized systems problems).
- Very low performance on critical transactions (on popular services in the network); specifically in long-term transactions, this problem can reduce the performance of system and possibilities for deadlock will increase.

7.4 . Conclusions:

The primary objective of this report was to motivate and present the development of a transaction model for long-life business transactions in DBEs. However, it is important to appreciate that the full potential of such a transaction model cannot be realised without significant development of the underlying P2P architecture. The objective of this chapter has been to highlight the areas that need to be addressed in the development of a next generation architecture, by identifying the attributes that are needed for DBEs and then analysing where current models fail in meeting DBE requirements.

Work is under way to develop a P2P network architecture that addresses the concerns we have raised. This will be reported on in a separate document.

8. Concluding remarks and future directions

The nature of the transactions expected to take place in a highly dynamic and purely distributed environment such as that of a Digital Business Ecosystem (DBE) raises a number of non-trivial issues with respect to defining a consistent transaction model. In this report, we have presented the fundamentals of a transaction model and discussed the underlying architecture based on SOC principles that eases a number of issues that arise in providing a collaborative distributed software environment for SMEs.

The long-term nature of business transactions frames the concept of a transaction in Digital Business Ecosystems. We discussed traditional transaction models (nested, sagas, and others) and argued that the conventional view they take of a transaction, which is based around the ACID properties, is not suitable for a business environment.

Furthermore, the principles underlying SOC make it the principal computing paradigm for a business environment. We considered a service-oriented architecture (SOA) for a transaction model through which organisations can seamlessly access customised, potentially disposable services to aid them in realising their business processes efficiently.

The primary requirements for SOA are well-suited to serve individual SMEs and respect their local autonomy. A range of service composition types needs to be supported in order to cover evolving business requirements and offer added value for participating organisations through higher level compositions. The study of recent transaction models for SOA presented in Chapter 3 shows that the basic requirements for SOA are violated. The loose-coupling of services is not achieved and consistency is guaranteed at the expense of local autonomy. Also, the coordination mechanism deployed is not fully distributed. This makes such models suitable for Lage Enterprises, but not for SMEs whose independent existence is undermined.

Full realisation of the vision of a collaborative distributed software environment for SMEs is largely dependent on whether we are able to guarantee consistency and preserve local autonomy. In this report we have proposed a transaction model whose coordination mechanism is considered at the deployment level of SOA only, and is performed *locally*. This allows us to guarantee consistency without breaking the local autonomy of the participants since, unlike WS-Coordination, we do not have to tamper with the execution of the underlying services at the realisation level. The model thus respects the loose-coupling of services and is particularly tailored to be deployed within a service-oriented architecture.

It is also important to note that particular thought has gone into supporting well-known behaviour patterns and the model described in this report supports the ‘do-compensate’ and ‘validate-do’ patterns; the locking schemes discussed in Chapter 6 show it is possible to also support the ‘provisional-final’ behaviour pattern.

We have also described the compensation capabilities of the proposed transaction model and the way omitted results are handled through forward recovery. This is based on the directed graphs that capture the dependencies between service compositions and/or local coordinators at both the intra-transaction and the inter-transaction level, and does not require knowledge of the local design of SMEs or access to the local platforms.

In addition, the transaction model for DBE that we propose is expressive enough to capture various forms of compositions and can cope with dynamic composition of services in a straightforward manner.

In a collaborative distributed environment it is important to ensure that collaboration is conducted in an appropriate manner and that the behaviour exhibited by the participating clients is suitable in a given scenario. For multi-service transactions this means that there are certain obligations constraining the interactions between the underlying services. In DBE the services are managed by multiple partners in

different domains and therefore there is a clear interest in supporting the engineering task of setting up business transactions that involve hierarchies of service compositions whose execution needs to be orchestrated.

For this purpose, we have described work in progress on a formal model for coordinating distributed long-running transactions which can be used to analyse the behaviour of a transaction before run-time. The formal representation of a transaction in our model provides a thorough understanding of the behaviour exhibited by the underlying service compositions. In this sense, it can ease the implementation and testing of the underlying services and increase expectations of a successful outcome prior to deployment.

We have given a formal behavioural description of a transaction in DBE, in terms of activations of its sub-transactions (local coordinators, service compositions, basic services) and the orchestration required for performing the transaction in question. This formal description is based on previous work on a true-concurrent framework for interactions between communicating entities [Shi97, Mos05] and is expressive enough to capture the various forms of composition considered in the proposed transaction model. We have discussed how we may generate the various patterns of behaviour that the transaction can exhibit and can be used to verify that its design correctly reflects the necessary service compositions for the desired result. Formal reasoning against properties of the underlying mathematical constructions allows us to determine whether the behaviour exhibited by the transaction will be suitable in given scenarios or, on the contrary, certain anomalies could still occur. The generated patterns of behaviour of a transaction also allow the identification of the possible sequences of reverse actions for compensation and forward recovery.

From a more implementation-oriented point of view, we have described an extensive locking mechanism to support the proposed transaction model. This is done in a way that ensures local consistency and can be realised at the local coordinator level. More specifically, we described the use of two locks, namely *I-Lock* and *C-Lock*, for ensuring consistency between the distributed logs as provided by the IDG and EDG and the local concurrency model. We also introduced a lock, the so-called *T-Lock*, for covering omitted results in common distributed events. Finally, we described a lock for recovery, named *R-Lock*, which facilitates an isolated two-phase recovery routine.

These different locking schemes, as apart of the concurrency control, can provide implementation support for compensation and forward recovery in a way that ensures local progress-to-date is preserved as much as possible. The locking mechanism is set up in such a way that it allows us to introduce a customised 3PC communication, where the intermediate phase is used for addressing unexpected failures in the commit state.

The proposed transaction model captures several aspects that are crucial in realising the vision of a collaborative distributed software environment for business transactions conducted in a way that allows for the viability of SME's. The next step is to consider a network architecture that would support such long-running and multi-service transactions. In this report, we reviewed existing P2P networks and analysed the fundamental requirements for their adoption in a highly dynamic and purely distributed environment such as DBE.

The early analysis of current P2P networks in Chapter 7 shows them to have critical dependencies on single organisations and be subject to critical point(s) of failure. Hence, they do not provide a fully distributed environment which is a prerogative for DBE; especially when considering SMEs. New architectures for P2P networks are required for deployment within a DBE environment for SMEs. This is currently under further investigation and we identified a number of basic characteristics such networks should have in Chapter 7. In this report we have simply opened the discussion for possible future directions in this respect. These are mostly concerned with novel fully distributed architectures for P2P networks that can support higher level composition types, such as *explorative* and *semi-fixed* composition, as we believe such aspects deserve our attention in future extensions of this work.

9. References

[ABC+01] Aitken, D., Bligh, J., Callanan, O., Corcoran, D., & Tobin, J., (2001), "Peer-to-Peer Technologies and Protocols", Link At: http://ntrg.cs.tcd.ie/undergrad/4ba2.02/p2p/ (last access: 02/11/06).

[AbH02] Aberer, K. and Hauswirth, M. (2002), "An Overview on Peer-to-Peer Information Systems", Workshop on Distributed Data and Structures (WDAS-2002), 2002 - minoas.di.uoa.gr

[AJB00] Albert, R.; Jeong, H.; and Barabási, A. (2000) "Attak and tolerance in complex networks", Nature 406 378 (2000).

[Ada99] Adamic, L. A. "The small World Web." Third European Conference on Research and Advanced Technology for Digital Libraries (ECDL 99); 1999 September 22-24; Paris, France. NY: Springer Verlag; 1999; Lecture Notes in Computer Science; 1696: 443-452

[AdH00a] Adamic, L. A.; Huberman, B. A. (2000), "Power-law distribution of the World Wide Web." Science. 2000 March 24; 287 (5461): 2115

[AdH00b] Adamic, L. A.; Huberman, B. A. (2000) "The nature of markets in the World Wide Web." Quarterly Journal of Electronic Commerce. 2000; 1 (1): 5-12.

[ALH02] Adamic, lada A.; Lukose, rajan M. and Huberman, bernardo A. (2002), "Local Search in Unstructured Networks", arXiv:cond-mat/0204181 v2 4 Jun 2002

[ALH01] Adamic, L. A.; Lukose, R. M.; Puniyani, A. R. and Huberman, B. A. (2001), "Search in power-law networks" PHYSICAL REVIEW E, VOLUME 64, 046135

[AzM03] Azzedin, F. Maheswaran, M. (2003), "Trust modeling for peer-to-peer based computing systems", International Parallel and Distributed Processing Symposium, 22-26 April 2003, pp: 10

[BaA99] Barabasi AL and Albert R (1999) Emergence of scaling in random networks. Science 286: 509–512.

[Bag05] Baghdadi, Y. (2005), "A web services-based business interactions manager to support electronic commerce applications", Proceedings of the 7th international conference on Electronic commerce, Vol. 113, 2005, pp: 435-445

[BAJ99] Barabasi, AL.; Albert, R. & Jeong, H. (1999) "The diameter of the world wide web", arXiv:cond-mat/9907038 v1 2 Jul 1999.

[Bar01] Barkai, D., 2001. Peer-to-Peer Computing, Technologies for Sharing and Collaborating on the Net. Intel Corporation.

[Bar03] Barabasi A. (2003), Linked: How everything is connected to everything else and what it means for business, science and everyday life, PLUME, USA

[BeN97] Bernstein P.A. and Newcomer E. (1997), Principles of Transaction Processing. Morgan Kaufmann Publishers, 1997

[Ber03] Bergner, M., (2003), Improving Performance of Modern Peer-to-Peer Services,UMEA University, Department of Computer Science

[BHMcC+03] Booth, D.; Haas, H.; McCabe, F.; Newcomer, E.; Champion, M.; Ferris, C. and Orchard, D. (2003), "Web Services Architecture", 8 August 2003. Available at: http://www.w3.org/TR/2003/WD-ws-arch-20030808/

[Bro02] Broberg, J. C. (2002), "Glossary for the OASIS WebService Interactive Applications (WSIA/WSRP)", OASIS Web Services Interactive Applications TC Committees, 2002. Available at: http://www.oasis-open.org/committees/wsia/glossary/wsia-draft-glossary-03.htm

[BEK93] Bukhres, O. A.; Elmagarmid, A. K.; Kühn, e. (1993), "Implementation of the Flex Transaction Model". IEEE Data Eng. Bull. 16(2) 1993, pp: 28-32

[CCJ+04] Cabrera, F. L.; Copeland, G. Johnson, J.; and Langworthy, D. (2004), Coordinating Web Services Activities with WS-Coordination, WS-AtomicTransaction, and WS-BusinessActivity. Available at: http://msdn.microsoft.com/webservices/default.aspx, January 2004

[CCC+03] Cabrera, L. F.; Copeland, G.; Cox, W.; Feingold, M.; Freund, T.; Johnson, J.; Kaler, C.; Klein, J.; Langworthy, D.; Nadalin, A.; Orchard, D.; Robinson, I.; Shewchuk, J.; Storey, T. and Satish Thatte. (2003) “Web Services Coordination Framework (WS-Coordination)”, September 2003.

[CCC+04a] Cabrera, L. F.; Copeland, G.; Cox, W.; Feingold, M.; Freund, T.; Johnson, J.; Kaler, C.; Klein, J.; Langworthy, D.; Nadalin, A.; Orchard, D.; Robinson, I.; Shewchuk, J.; Storey, T. and Satish Thatte. (2004-a) “Web Services Atomic Transaction Framework(WSAtomicTransaction)”, January 2004.

[CCC+04b] Cabrera, L. F.; Copeland, G.; Cox, W.; Feingold, M.; Freund, T.; Johnson, J.; Kaler, C.; Klein, J.; Langworthy, D.; Nadalin, A.; Orchard, D.; Robinson, I.; Shewchuk, J.; Storey, T. and Satish Thatte. (2004-b) “Web Services Business Activity Framework (WSBusinessActivity)”, Januar 2004

[CdeS95] Campos, R.V.; de Souza e Silva, E. (1995), “Availability and performance evaluation of database systems under periodic checkpoints”, Twenty-Fifth International Symposium on Fault-Tolerant Computing (IEEE CNF), 27-30 June 1995, pp: 269 – 277

[CDF+03] Ceponkus, A.; Dalal, S.; Fletcher, T.; Furniss, P.; Green, A. and Pope B. (2003), "Business Transaction Protocol Version 1.0", An OASIS Committee Specification, 3 June 2002 Available at: http://www.oasis-open.org/committees/download.php/1184/2002-06-03.BTP_cttee_spec_1.0.pdf

[ChG94] Chen, Y.-W.; Gruenwald, L. (1994), “Research issues for a real-time nested transaction model”, Proceedings of the IEEE Workshop on Real-Time Applications, , 21-22 July 1994, pp:130 – 135

[CKM+03] Curbera F.; Khalaf R.; Mukhi, N.; Tai, S. and Weerawarana, S. (2003), “The Next Step in Web Services”, Communications of the ACM, Vol. 46, No 10, Oct. 2003, pp: 29-34 (SOC/WF)

[CMH+02] Clarke, I.; Miller, S.G.; Hong, T.W.; Sandberg, O. and Wiley, B. (2002), "Protecting free expression online with Freenet", Internet Computing, IEEE, Volume: 6, Issue: 1, Jan/Feb 2002, pp: 40-49.

[CSW+00] Clarke, I.; SANDBERG, O.; WILEY, B. AND HONG, T.W. (2000), "Freenet: A distributed anonymous information storage and retrieval system", In Proceedings of the ICSI Workshop on Design Issues in Anonymity and Unobservability (Berkeley, California, June 2000).

[CrP02] Crowcroft, J. and Pratt I. (2002), "Peer-to-Peer: Peering into the Future" In Proceedings of the IFIP-TC6 Networks 2002 Conference, Pisa, Italy., May 2002.

[CTT97] Chin, C.; Chung-Kie Tung; Shang-Rong Tsai, (1997), “Phaeton: a log-based architecture for high performance file server design”, Pacific Rim International Symposium on Fault-Tolerant Systems, 15-16 Dec. 1997, pp: 28 - 33

[Dat96] Date C. J. (1996), An introduction to Database Systems (5th edition), Addison Wesley, USA

[DaP90] Davey, B. A. and Priestley, H.A. . Introduction to Lattices and Order, Cambridge University Press, 1990.

[DBE06] Digital Business Ecosystems (DBE) EU-FP6 IST Integrated Project No 507953. Available at http://www.digital-ecosystem.org [last accessed 19 September 2006]

[DKA+04] Derbas, G.; Kayssi, A.; Artail, H.; Chehab, A. (2004), “TRUMMAR - a trust model for mobile agent systems based on reputation”, ICPS 2004. Proceedings. The IEEE/ACS International Conference on Pervasive Services, 19-23 July 2004, pp: 113 - 120

[DKR02] Druschel, P.; Kaashoek, MF. and Rowstron, AIT (2002), "Peer-to-Peer Systems" - First International Workshop, IPTPS, 2002

[DLD97] Domel, P.; Lingnau, A.; Drobnik O. (1997), “Mobile Agents Interaction in Heterogeneous Environment”, First International Workshop on Mobile Agents, April 1997, No 1219, pp: 136-148
Eder, J. and Liebhart, W. (1995), "The Workflow Activity Model WAMO", Proceeding of the 3rd Int. Conference on Cooperative Information Systems (CoopIS), 1995

[DNB03] Ding, C. H.; Nutanong, S.; Buyya, R. (2003), "Peer-to-peer Networks for Content Sharing", Technical Report, GRIDSTR-2003-7, Grid Computing and Distributed Systems Laboratory, University of Melbourne, Australia, Dec. 2003

[DSW94] Deacon, A.; Schek, H.-J.; Weikum, G. (1994), “Semantics-based multilevel transaction management in federated systems”, Proceedings. 10th International Conference on Data Engineering (IEEE CNF), 14-18 Feb. 1994, pp: 452 – 461

[DTL+03] Dalal, S.; Temel, S.; Little, M.; Potts, M. and Webber, J. (2003), "Coordinating Business Transactions on the Web", IEEE INTERNET COMPUTING, Volume: 7, Issue: 1, JANUARY - FEBRUARY 2003, pp: 30-39

[Edw02] Edwards, J. (2002), "Peer-to-Peer Programming on Groove", Addison-Wesley, 2002.

[Elm94] Elmagarmid A. (1994), Database Transaction Model for Advanced applications, Morgan – Kaufmann

[EmT00] Emmerich, W.; Tai, S. (2000), "Advanced Transactions in Enterprise Java Beans", Engineering Distributed Objects: Second International Workshop, Publisher: Springer Berlin / Heidelberg, November 2000, pp: 215

[FML+97] Fraga, J. Maziero, C. Lung, L.C. Loques Filho, O.G. (1997), “Implementing replicated services in open systems using a reflective approach”, ISADS 97., Third International Symposium on Autonomous Decentralized Systems, 9-11 April 1997, pp: 273 - 280

[FrI01] Fritzke, U., Jr. Ingels, P. (2001), “Transactions on partially replicated data based on reliable and atomic multicasts”, 21st International Conference on Distributed Computing Systems, April 2001, page(s): 284 - 291

[FuG05] Furnis, P. and Green, A. (2005), "Choreology Ltd. Contribution to the OASIS WS-TX Technical Committee relating to WS-Coordination, WS-AtomicTransaction and WS-BusinessActivity",

OASIS WS-TX Technical Committee, 16 November 2005. (avalaible at www.oasis-open.org/committees/download.php/15808/Choreology.WS-TX.TC.Contribution.2005-11-16.doc , last availability; 06.June.2006).

[FDF+04] Furnis, P.; Dalal, S.;Fletcher, T.; Green, A.; Ceponkus, A. and Pope, B. (2004), "Business Transaction Protocol, version 1.1.0", Committee Draft, November 2004. Available at: http://www.oasis-open.org/committees/download.php/9836/business_transaction-btp-1.1-spec-wd-04.pdf

[G-MS87] Garcia-Molina, H. and Salem, K., (1987), "Sagas", Proceedings of the Association for Computing Machinery Special Interest Group on Management of Data 1987 Annual Conference, San Francisco, California, May 27-29, 1987, pp: 249-259

[G-MG+91]Garcia-Molina, H.; Gawlick, D.; Klein, J.; Kleissner, K.; Salem, K. (1991), "Coordinating activities through extended sagas: a summary", Compcon Spring '91. Digest of Papers (IEEE CNF), 25 Feb.-1 March 1991, pp: 568 – 573

[GAP04] Gilbert, A. Abraham, A. Paprzycki, M. (2004), "A system for ensuring data integrity in grid environments", ITCC 2004. International Conference on Information Technology: Coding and Computing, 5-7 April 2004, pp: 435 - 439 Vol.1

[GiC00] Gillies J., Cailliau R. (2000), How the Web Was Born: The Story of the World Wide Web, Oxford University Press, England

[Gon01] Gong, L. (2001), "JXTA: a network programming environment", Internet Computing, IEEE, Volume: 5, Issue: 3, May/Jun 2001, pp: 88-95.

[Gud04] Gudgin, M. (2004), "Secure, reliable, transacted: innovation in Web Services architecture", Proceedings of the 2004 ACM SIGMOD international conference on Management of data (ACM Press), 2004, pp: 879 – 880

[LLW+03] Guo Qiong Liao, Yun Sheng Liu, Li Na Wang. Chu Ji Peng (2003), "Concurrency control of real-time transactions with disconnections in mobile computing environment", International Conference on Computer Networks and Mobile Computing (IEEE CNF), 20-23 Oct. 2003, pp: 205 – 212

[Hag95] Haghjoo M. S. (1995), Transaction Actors in Cooperative Information Systems, PHD Thesis, The Australian National University, Australia

[Hag96] Haghjoo M. S. (1996), "Scheduling and Scripting Mega Transaction", in prcds of the 2nd International Computer Conference, Iran Computer Society, Amir Kabir University, Iran

[Hag97] Haghjoo M. S. (1997), "Compensating Mega Transaction", in prcds of the 3rd International Computer Conference, Iran Computer Society, Iran University of Science and Technology, Iran

[HaP92] Haghjoo, M.S.; Papazoglou, M.P. (1992), "TrActorS: a transactional actor system for distributed query processing", Proceedings of the 12th International Conference on Distributed Computing Systems (IEEE CNF), 9-12 June 1992, pp: 682 – 689

[HPS93] Haghjoo, M.S.; Papazoglou, M.P.; Schmidt, H.W. (1993), "A semantic-based nested transaction model for intelligent and cooperative information systems", , Proceedings of International Conference on Intelligent and Cooperative Information Systems (IEEE CNF), 12-14 May 1993, Pages:321 – 331

[HST01] Herrero, J.L.; Sanchez, F.; Toro, M.(2001), “Fault tolerance as an aspect using JReplica”,FTDCS 2001. Proceedings. The Eighth IEEE Workshop on Future Trends of Distributed Computing Systems, 31 Oct.-2 Nov. 2001, pp: 201-207

[HYP01] Heuvel, W-J Van; Yang, J. and Papazoglou, M.P. (2001). Service Representation, Discovery, and Composition for E-Marketplaces, Proc. Of International Conference on Cooperative Information Systems cooPIS01),Sep, 2001.

[H-RC02] Hong-Ren Chen; Chin, Y.H. (2002),” An efficient real-time scheduler for nested transaction models”, Ninth International Conference on Parallel and Distributed Systems (IEEE CNF), 17-20 Dec. 2002, pp: 335 – 340

[JSM01] Janaki Ram, D.; Chandra Sekhar, N.S.K.; Uma Mahesh, M.(2001), “A data-centric concurrency control mechanism for three tier systems”, 2001 IEEE International Conference on Systems, Man. and Cybernetics, Volume: 4 , 7-10 Oct. 2001, pp:2402 - 2407 vol.4

[JTK89] Jenq, B.-C.; Twichell, B.C.; Keller, T.W.(1989), “Locking performance in a shared nothing parallel database machine”, IEEE Transactions on Knowledge and Data Engineering, Volume: 1 , Issue: 4 , Dec 1989 pp: 530 – 543

[Kan01] Kan, G., (2001), Gnutella, Peer-to-Peer: Harnessing the Power of Disruptive Technologies, A. Oram (ed.), O’Reilly Press, USA.

[KaX92] Kakeshita, T.; Haiyan Xu (1992), “Transaction sequencing problems for maximal parallelism”, Second International Workshop on Transaction and Query Processing (IEEE), 2-3 Feb. 1992, pp: 215 – 216

[KuR03] Kurose, J. F. & Ross, K. W. (2003), Computer Networking: A Top-Down Approach Featuring the Internet, Addison Wesley, Boston, USA.

[LaP01] Lala, C.; Panda, B. (2001), “Evaluating damage from cyber attacks: a model and analysis”, IEEE Transactions on Systems, Man and Cybernetics, Volume: 31 , Issue: 4 , July 2001, pp: 300 – 310

[LCC+02] Lv, Q.; Cao, P.; Cohen, E.; Li, K. and Shenker S. (2002), "Search and replication in unstructured peer-to-peer networks", Proceedings of the 16th international conference on Supercomputing, 2002, pp: Pages: 84 - 95

[LiT96] Liang, D.; Tripathi, S.K. (1996), “Performance analysis of long-lived transaction processing systems with rollbacks and aborts”, IEEE Transactions on Knowledge and Data Engineering, Volume: 8, Issue: 5, Oct. 1996, pp: 802 – 815

[Lie06] Lieberman, B. (2006), "SOA transaction management -- Part I: The transaction coordination service;", developer Works on SOA and Web services at IBM Ltd,15 May 2006. Available at: http://www-128.ibm.com/developerworks/rational/library/may06/lieberman/

[LiZ04] Limthanmaphon, B. and Zhang, Y. (2004), "Web service composition transaction management", Proceedings of the fifteenth conference on Australasian database, vol. 27, 2004, pp: 171-179

[LiF03] Little, M. and Freund, T. (2003), "A comparison of Web services transaction protocols", developer Works on SOA and Web services at IBM Ltd., 07 October 2003. Available at: http://www-128.ibm.com/developerworks/webservices/library/ws-comproto/

[LoC04] Losee, R.M.; Church, L. (2004), "Information retrieval with distributed databases: analytic models of performance", IEEE Transactions on Parallel and Distributed Systems, Volume: 15 , Issue: 1, Jan. 2004 pp:18 – 27

[Mad97] Madria, S.K. (1997), "Concurrency control algorithm for an open and safe nested transaction model", Proceedings of 1997 International Conference on Communications and Signal Processing (IEEE CNF), 9-12 Sept. 1997, pp: 907 - 912 vol.2

[Maz88] Mazurkiewicz, A. (1988) Basic Notions of Trace Theory. In de Baker, de Roever and Rozenberg, eds, *Proceedings of Linear time, Branching Time and Partial Orders in Logics and Models for Concurrency*, Lecture Notes in Computer Science, Vol. 354, pp. 285-363, Springer-Verlag, 1988

[Mil01] Miller, J. (2001), "Jabber: Conversational Technologies." Peer-to-Peer: Harnessing the Power of Disruptive Technologies. Andy Oram, editor. O'Reilly and Associates, Sebastopol, California. 2001.

[Mos85] Moss J. E. B. (1985), Nested transaction an approach to Reliable Distributed Computing, MIT Press, USA

[MoS04] Moschoyiannis, S. Shields, M. W. (2004) A Set –Theoretic Framework for Component Composition. *Fundamenta Informaticae* 59(4): 373-396, 2004

[MSK05] Moschoyiannis, S., Shields, M. W. and Krause, P. J. Modelling Component Behaviour using Concurrent Automata. In *Proceedings ETAPS 2005 – Formal Foundations of Embedded Software and Component-Based Architectures (FESCA'05)*, Electroonic Notes on Theoretical Computer Science, Vol. 141, pp. 199-220. Elsevier, 2005

[Mos05] Moschoyiannis, S. (2005). Specification and Analysis of Component-Based Software in a True-Concurrent Setting. PhD Thesis, University of Surrey, 2005.

[M-EGB98] Munoz-Escoi, F.D. Galdamez, P. Bernabeu-Auban, J.M. (1998), "ROI: an invocation mechanism for replicated objects", Seventeenth IEEE Symposium on Reliable Distributed Systems, 20-23 Oct. 1998, pp: 29 – 35

[MNS+04] Mustafa, M.D. Nathrah, B. Suzuri, M.H. Abu Osman, M.T. (2004), "Improving data availability using hybrid replication technique in peer-to-peer environments", AINA 2004. 18th International Conference on Advanced Information Networking and Applications, 2004, pp: 593 - 598 Vol.1

[MRW+93] Muth, P.; Rakow, T.C.; Weikum, G.; Brossler, P.; Hasse, C. (1993), "Semantic concurrency control in object-oriented database systems", Ninth International Conference on Data Engineering (IEEE CNF), 19-23 April 1993, pp: 233 – 242

[NeC02] Nektarios, G. Christodoulakis, S. (2002), "UTML: Unified Transaction Modeling Language", Proceedings of the Third International Conference on Web Information Systems Engineering (IEEE CNF), 12-14 Dec. 2002, pp: 115 - 126

[Nod93] Nodine, M.H. (1993), "Supporting long-running tasks on an evolving multidatabase using interactions and events", Proceedings of the Second International Conference on Parallel and Distributed Information Systems (IEEE CNF), 20-22 Jan. 1993, pp: 125 – 132

[NoZ94] Nodine, M.H.; Zdonik, S.B. (1994), "Automating compensation in a multidatabase", Proceedings of the Twenty-Seventh Hawaii International Conference on Software Technology (IEEE CNF), Volume: 2, 4-7 Jan. 1994, pp: 293 – 302

[Pap03] Papazoglou M. P. (2003) , “Service-Oriented Computing: Concepts, Characteristics and Directions”, 4th International Conference on Web Information Systems Engineering (WISE'03) , Rome, Italy, 2003.

[PaG03] Papazoglou M.P. and Georgakopoulos, D. (2003), "SERVICE-ORIENTED COMPUTING", COMMUNICATIONS OF THE ACM, October 2003/Vol. 46, No. 10, pp:24-28

[PDB+97] Papazoglou, M.; Dells, A.; Bouguettaya, A.; Haghjoo, M. (1997) “Class library support for workflow environments and applications”, IEEE Transactions on Computers, Volume: 46 , Issue: 6, June 1997, pp: 673 – 686

[PDH+96] Papazoglou, M.P.; Delis, A.; Haghjoo, M.; Bouguettaya, A. (1996) “Language support for long-lived concurrent activities”, Proceedings of the 16th International Conference on Distributed Computing Systems (IEEE CNF), 27-30 May 1996, pp: 698 – 705

[PaH05] Papazoglou, M.P.; van den Heuvel, W.-J. (2005), "Web services management: a survey", IEEE Internet Computing, Volume: 9, Issue: 6, Nov.-Dec. 2005, pp: 58- 64

[PaH99] Pavlova,E.; van Hung, D. (1999), “A formal specification of the concurrency control in real-time databases”, Software Engineering Conference, 1999. (APSEC '99) Proceedings. Sixth Asia Pacific, 7-10 Dec. 1999, pp: 94 – 101

[PoH01] Posselt, D.; Hillebrand, G.(2001), “Database support for evolving data in product design”, The Sixth International Conference on Computer Supported Cooperative Work in Design (IEEE CNF), 12-14 July 2001, pp: 377 – 383

[QXL02] Qiujing Li; Hong Xu; TingLong Liu (2002), “Coordinating transaction model in CDBMS”, The 7th International Conference on Computer Supported Cooperative Work in Design, 25-27 Sept. 2002, pp: 421 – 425

[RaN99] Ramampiaro, H.; Nygard, M.(1999), “Cooperative database system: a constructive review of cooperative transaction models”, International Symposium on Database Applications in Non-Traditional Environments (DANTE '99), IEEE CNF, pp: 315 – 324

[Raz99] Razavi A. R. (1999), “Design and Implementation of Recovery Management for Mega Transaction and Model implementation”, MSC Thesis, Iran University of Science and Technology, Iran

[RFH+01] Ratnasamy, S.; Francis, P.; Handley, M.; Karp, R. and Shenker, S. (2001), "A scalable content-addressable network", Proceedings of the 2001 conference on Applications, technologies, architectures, and protocols for computer communications, 2001, pp: 161 – 172

[Rip01] Ripeanu M. (2001), "Peer-to-Peer Architecture Case Study: Gnutella Network", First International Conference on Peer-to-Peer Computing (P2P'01), 2001.

[RLF02] Ripeanu, M.; Iamnitchi A. and Foster I. (2002) "Mapping the Gnutella Network" Internet Computing, IEEE, Volume: 6, Issue: 1, Jan/Feb 2002, pp: 50-57.

[RoD01] Rowstron A. and Druschel P. (2001), "Storage management and caching in PAST, a large-scale, persistent peer-to-peer storage utility", Proceedings of the eighteenth ACM symposium on Operating systems principles, 2001, pp: 188 - 201

[ScP03] Schmidt, C.; Parashar, M. (2003), "Flexible information discovery in decentralized distributed systems", 12th IEEE International Symposium on High Performance Distributed Computing, 2003. Proceedings. 22-24 June 2003, pp:226- 235

[SHD+01] Segun, K.; Hurson, AR; Desai, V.; Spink, A. and Miller, LL. (2001) "Transaction Management in a Mobile Data Access System", Annual Review of Scalable Computing, 2001

[Shi85] Shields, M. W. (1985), Concurrent Machines. *Computer Journal*, 28:449-465, 1985

[Shi97] Shields, M. W. Semantics of Parallelism (1997). Springer-Verlag, London, 1997

[SCW+04] Siau, K.L.; Chan, H.C.; Wei, K.K. (2004), "Effects of Query Complexity and Learning on Novice User Query Performance With Conceptual and Logical Database Interfaces", *IEEE Transactions on Systems, Man and Cybernetics,* Volume: 34 , Issue: 2 , March 2004, pp: 276 – 281

[Sta03] Stallings W. (2003), Cryptography and Network Security Principle and Practice (Third edition), Prentice Hall, USA

[TUH+97] Tada, H.; Uchida, K.; Higuchi, M.; Fujii, M. (1997), "28 A model of nested transaction with fine granularity of concurrency control", 1997 IEEE Pacific Rim Conference on Communications, Computers and Signal Processing, Volume: 2 , 20-22 Aug. 1997, pp: 977 - 980 vol.2

[TeI03] Terai, K. and IzumI, N. (2003), "Coordinating Web Services based on business models", Proceedings of the 5th international conference on Electronic commerce (ACM Press), 2003, vol. 50, pp:473-478

[vdMDD+03] van der Meer, D. Datta, A. Dutta, K. Ramamritham, K. Navathe, S.B. (2003), "Mobile user recovery in the context of Internet transactions", IEEE Transactions on Mobile Computing, Volume: 2, Issue: 2, April-June 2003, pp: 132 – 146

[VeP98] Verharen, E. M. and Papazoglou, M.P.(1998), "Introducing Contracting in Distributed Transactional Workflows", Thirty-First Annual Hawaii International Conference on System Sciences-Volume 7, 1998, pp:324-334

[VZG+05] Vogt, F.H.; Zambrovski, S.; Gruschko, B.; Furniss, P. and Green, A. (2005), "Implementing Web service protocols in SOA: WS-Coordination and WS-BusinessActivity", Seventh IEEE International Conference on E-Commerce Technology Workshops, 19 July 2005, pp: 21- 26

[Wan96] Wanlei Zhou (1996), "Performance evaluation of nested transactions on locally distributed database systems", Second International Symposium on Parallel Architectures, Algorithms, and Networks (IEEE CNF), 12-14 June 1996, pp: 353 – 356

[WaS98] Watts DJ, Strogatz SH, 1998, "Collective dynamics of small-world networks", Nature 363: 202-204.

[Wat99] Watanabe, T (1999), "Agent-oriented model for managing long-lived transaction, based on work-flow and task-graph", Third International Conference on Computational Intelligence and Multimedia Applications ICCIMA '99 (IEEE CNF), 23-26 Sept. 1999, pp: 10 – 13

[WWZ02] Wgrzyn, S.W. Winiarczyk, R. Znamirowski, L. (2002). "Nanotechnologies and nanosystems of informatics as a basis for self-replication", IEEE-NANO 2002. Proceedings of the 2002 2nd IEEE Conference on Nanotechnology, Aug. 2002, pp: 99 - 102

[Wik06] Wikipedia, (2006), "Web service", Wikipedia encyclopedia, Available at: http://en.wikipedia.org/wiki/Web_service (accessed date; 14/06/06)

[WZB+01] Xiao Weijun; Lu Zhengding; Li Bing; Sarem, M. (2001) "Transaction management in mobile multidatabase systems", 2001 International Conference on Computer Networks and Mobile Computing (IEEE CNF), 16-19 Oct. 2001, Los Alamitos, CA USA, pp: 513 – 518

[YaG02] Yang, B. and Garcia-Moline, H. (2002), “Designing a Super-Peer Network”, Technical Report, Standford University, February 2002.

[YPH02] Yang, Jian; Papazoglou, M P. and Heuvel, W-J van den (2002), “Tackling the Challenges of Service Composition in E-Marketplaces”, Twelfth International Workshop on Research Issues in Data Engineering: Engineering E-Commerce/E-Business Systems, 2002. RIDE-2EC 2002 (IEEE Computer society), pp:125-133

[YaM03] Yaohang Li; Mascagni, M. (2003), “Improving performance via computational replication on a large-scale computational grid”, CCGrid 2003. 3rd IEEE/ACM International Symposium on Cluster Computing and the Grid, 12-15 May 2003, pp: 442 - 448

[YK-MA+02] Younas, M. Kuo-Ming Chao Anane, R. James, A. (2002), “Multi-agent transactions for Web-based design activities”, The 7th International Conference on Computer Supported Cooperative Work in Design, 25-27 Sept. 2002, pp: 258 – 263

[YLY03] Yu, S. Zhou, W. Lan, M. Yue Wu (2003), “An architecture of Internet based data processing based on multicast and anycast protocols”, PDCAT'2003. Proceedings of the Fourth International Conference on Parallel and Distributed Computing, 27-29 Aug. 2003, pp: 104 – 110

[ZLB04] Zirpins, C.; Lamersdorf, W. and Baier, T. (2004), "Flexible coordination of service interaction patterns", Proceedings of the 2nd international conference on Service oriented computing (ACM Press), 2004, pp: 49-56

[ZoJ98] Zou H. Jahanian, F. (1998), “Optimization of a real-time primary-backup replication service”, Seventeenth IEEE Symposium on Reliable Distributed Systems, 20-23 Oct. 1998, pp: 177 - 185