Meta Model for Long Lived Transactions
FYP 2006 – Inception Report

University of Malta
Department of Computer Science and A.I

Developed By:

Justin Spiteri 143083(M)

B.Sc. I.T Hons Year 4

Abstract

The main aim of this report is to provide a brief insight into the world of transaction processing with emphasis on the areas which consist of long lived transaction processing in particular.

An overview of the basic concepts of transaction processing and transaction modeling is initially given, thus providing theoretical background to the subject, together with a clear picture of the different types of software specifications and implementations which deal with transaction processing, currently available.

The main problem found during the research process was mainly the fact that to present date, even though there is a vast selection of official solution specifications, there are no completely inter-operable software implementations which cater for long running transactions. This document thus serves as a prologue to an advanced software solution which caters for long running transactions. This takes place after having thoroughly analyzed the research process results, together with the general situation of the transaction processing arena, as it stands to the present date.

Table Of Contents

Chapter 1: Project Description

1.1 Introduction & Motivation

The following text introduces us to the notion of transactions and transaction modeling, which are the main area of work for this project. Let's begin by providing a definition for transaction modeling. Transaction modeling is the process in which, a real world transaction, such as the core part of a money bank transfer is modeled into a business process, composed from Units of Work. Each business process represents a formalized way in which this transaction can be carried out. Most transaction processing systems present today are based on the traditional two phase commit Transaction Model, which caters for Atomic Transactions. These systems are often completely ACID oriented, however, as Mark Little, from HP-Arjuna Technologies says in one of his online articles;

"The structuring mechanisms available within traditional atomic transaction systems are sequential and concurrent composition of transactions. These mechanisms are sufficient if an application function can be represented as an atomic, short lived transaction." (Add reference)

In simple terms, this statement refers to the fact that the transaction handling facilities present as at date, are able to cater very efficiently only for transactions of an atomic, business to client (B2C) nature, for example a typical flight booking system, where the client makes a ticket request to the airline company, and the company either commits or denies. However when it comes to long running transactions, which may involve much more complex B2B transactions, these currently available mechanisms based on ACID properties are just not adequate, due to the facts mentioned in the previous definition of long running transactions. A good practical example of the inefficiency of ACID based long running transactions would be if one takes into consideration a typical holiday planning or travel agent system, where one may book a flight, train, taxi service, or even hotel. If a client initiates a compound transaction, where he wants to book both a hotel room and an air ticket, the intricate dependencies involved possess a much higher level of complexity than that of two separate transactions, where a client first books an air ticket, and then a hotel room, in two separate processes. This example shows that the differences between long

running transactions and atomic transactions disallow the sharing of a common platform between the two. A completely ACID based long running transaction system is inadequate. Long lived transactions must have a specialized mechanism which caters specifically for them. Even though specifications for such mechanisms are available, good implementations are nowhere to be found. This led research teams to put more effort in the research of long lived transaction modeling in an attempt to enhance long lived transaction processing, thus alleviating the present day symptoms present in current transaction processing systems, which are mostly ACID based.

The main problem lies at the heart of the subject; the transaction models. A vast amount of transaction models have been proposed since 1980, varying from simple Atomic Transaction Handling Models to very complex Compensation Based models. The problem is that it's impossible to have one transaction model which caters for all possible transactional scenarios. Each proposed model fits an application, or a range of applications, and thus is most effective when a developer uses it for the relevant range of applications. In certain cases, an application may need a completely custom model, made from Advanced Transaction Model Primitives, but not conforming completely to any of them, nor to any of the true advanced transaction models currently available. This would require the developer to conceptualize and implement a transaction model for the application from scratch each time a different model variant is needed, thus of course creates a problem, since it results in inefficiency in time and resources.

The motivation of this Thesis is therefore that of providing an intermediate solution to the problems mentioned above. This can be done with the creation of a meta-model which allows the developer to either build a custom model for a transactional application under development, or use a pre-implemented template, in both cases abstracting him from the core implications of transaction handling. This would make it possible for a developer to implement the separate Units of Work in a conventional manner, without having to cater for nesting, transaction dependencies, delegation, and all issues related to transactions. The transactional behavior of each Unit of Work would then be expressed separately, possibly with the help of a specialized descriptor or scripting language. This solution in essence would be similar to the structure presented in conTract[1] models, while offering a framework, which houses similar concepts to the ACTA framework[2]. Such a meta-model would allow developers to have no restrictions on the manner of operation of the transactions required by the application under development; since a possible open – source approach could possibly be taken to enhance extensibility of the meta-model itself. Extensibility may also be applied to transactional behavior, by converting the UOW behavior script into an extensible one. Further development may include a graphical application which

[1] A particular type of transaction model. (See Appendix A1)
[2] A particular type of transaction framework. (See Appendix A)

allows the developer to graphically represent Units of work, together with the transactional behavior needed for the system in question, thus reducing the learning curve for the developer.

1.2 Background

The following chapter provides a very brief review of the literature carried out, which includes both a theoretical overview of transactions and transaction modeling, and an analysis of the currently available solution specifications.

- **What is a transaction?**

A transaction in general, may be defined as a dedicated business oriented interaction between two or more parties, in which all stakeholders involved will be affected in some way. A more technical definition of a transaction can be found on Microsoft's MSDN web site, which claims the following:

"A transaction is a set of one or more related tasks that either succeed or fail as a unit. In transaction processing terminology, the transaction commits or aborts." – MSDN (Microsoft)

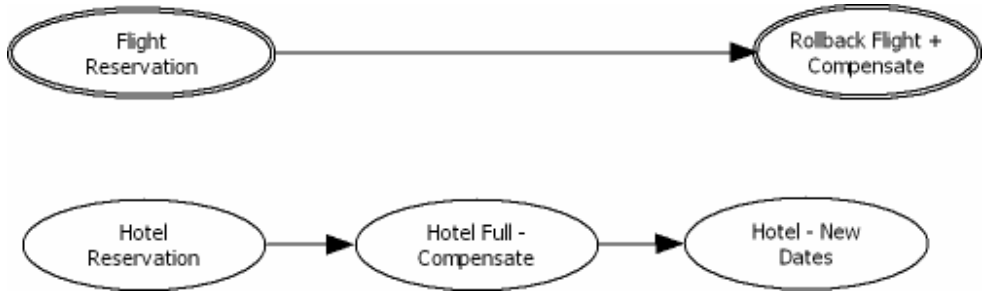
Transactions can be classified into two main categories, atomic transactions and Long Running Transactions. As Mike Chapple says in his article entitled "Your Guide to databases" (www.about.com);

"The concept of atomic transactions is based on one of the oldest but still relevant concepts of database theory, that is, the idea of ACID properties."

In fact, in order to adhere to these properties, any resources which are shared between multiple transactions must be protected, and thus locked when in use by one user. A typical Atomic transaction, being ACID based, takes a short amount of time to complete, and is usually based on a "commit or reject" philosophy. On the other hand, long running transactions have a higher degree of complexity than Atomic transactions, due to the fact that a single long running transaction can be made up of several stake holders, and potentially lasting hours or even days. This length of time makes the resource locking manifested in ACID based transactions inappropriate, since situations can arise where all resources are blocked, with no conclusive transactions, as they all wait for each other to free resources, in a massive inter-networked deadlock. Besides, as in a long running transaction, partial roll back of a part of the transaction may be

needed, thus, invalidating the concept of the scoping mechanism present in ACID based transactions which provides the "all or nothing" semantics. These differences present various implications when trying to build software models which handle these types of transactions. Following are two illustrations of both types of transactions:

Figure 1: A typical travel agent system (Ref: JSR95)



In this scenario the process consists of parallel running transactions, where each module, being an atomic transaction, has an impact on any other transaction running in parallel to it, thus full rollback and recovery capabilities are a must. Thus a long running transaction can also be composed of multiple ACID based transactions, which do not manifest ACID properties when considered as a whole. This situation brings up the following question: Is a completely ACID based system good or not?.

As seen in the example above, it's practically impossible to follow ACID properties throughout the compound transaction as a whole. Thus it can be said that the ACID model, even though powerful for short lived transactions, has its limitations when it comes to long lived and compound transaction scenarios.

The solution to this issue lies in taking a different approach when modeling the activities of the solution. It has been confirmed that strict ACID models are not the best way to tackle the problem. However should ACID properties be completely scrapped in the search for a new model which handles long lived transactions or should alternative models which extend the ACID model's capabilities be defined? In actual fact, there are various generic model template definitions which are aimed at solving this problem. These include nested model designs, split join model designs, cooperative transaction group model designs, and SAGA model designs amongst others. Most of the models or model specifications available today fall under, or are variants of, one of these model categories. More detail about each category will be given through further documentation. However it can be stated that the most discussed category is certainly SAGA. SAGA's notion is that of loosening the rigidity of strict ACID properties, however not completely scrapping them. In fact a typical SAGA:

"approximates atomicity over a long period of time, however not providing the isolation property". ("Acid is good, take it in short doses" – Mark Little, Arjuna Technologies)

This leaves us with a twofold problem; selecting a suitable transaction model, and selecting a suitable framework in which the model may operate.

- **Transaction Models**

  There is a vast amount of transaction models present today, some extending ACID models, and some which have been redesigned from the ground up. These can be categorized into various sets, according to their different nature and properties. Below is a comprehensive list of the standard, most commonly renowned transaction models, categorized into their various sets. Most of these models are currently available as specifications on www.omg.org.

  - **Traditional Transaction Models**
    - Two Phase Commit Model
  - **Advanced Transaction Models**
    - Nested Transaction Model
    - Saga Transaction Model
    - Split Join Transaction Model
    - ACTA Model/Framework
  - **True Advanced Transaction Models**
    - BTP Atom Model
    - BTP Cohesion Model
    - WS-AT Model
    - WS-BA Model
    - TX-ACID Model
    - TX-LRA Model
    - TX-BP Model
    - conTract Model
    - Bourgogne Model

Note: There are several other models in existence which have not been referenced here, due to the fact that there are too many. Such models include the DOM Model, Flex Transactions, the CORD model, Cooperative Transaction Hierarchy Models, and H-Models amongst others. An accurate review of each of these models can be found in Marek Prochazka's PHD thesis entitled "Advanced Transactions in Component Based Software Architectures." These models are based on transaction models classified as "Advanced transaction models", and do not introduce new concepts.

- **Transaction Framework Selection**

Assuming that a transaction model has been selected, the second part of the problem involves finding a way in which the solution can be implemented using the selected model/s. As with the choice of a model, choosing a framework in which to implement transaction handling is an application specific task. Following is a list of reviewed transaction framework specifications.

- Activity Service Specification (IBM/SUN)
- Oasis Business Transaction Protocol Specification
- Oasis OTS Specification.
- Web Services Composite Application Framework Spec.
- Web Services Business Administration Framework Spec.
- ACTA Model/Framework

While implementation attempts exist for most of these frameworks, no really practical application can be found which enables developers to construct a system which supports Long Lived Transactions in an easy manner. This resulted in the motivation for this project, which has been described in the previous section.

P.T.O.

1.3 Aims and Objectives

The main objective in this project is that of providing a solution to the problem described in the previous section, through a meta-model which provides developers a way of expressing workflow of a long lived transaction, thus applying it to the desired application. This would allow developers to either define their own custom compound transaction models, or use ready made templates. The following further objectives must be met in order for the project to be successful:

- Advanced Research on Current technologies.
- Development of a Workflow Descriptor Language.
- Abstraction of Transaction Business Logic from the Application Developer.
- Provision of solution which integrates transactional modules with Workflow.
- Provision of overall simplicity and flexibility of use of the solution.

1.4 Methods

The meta-model solution can be developed by developing a Workflow Descriptor, which could possibly consist of a marked up language similar to a typical scripting language. Primitives and keywords for this language would allow developers to define behavioural characteristics and interdependencies of each of the transactionally relevant modules of the system under development through a simple script. The desired result would be that advanced Workflows could be easily modeled by conventional developers, without needing professional knowledge in the subject, thus abstracting the transactional business logic issues away from the developer, having separated them from the actual application code.

Upon system integration, the script is parsed, and the transactions within the system are run using sequence defined by the script. Thus the script can be said to wrap around an application's transactionally relevant modules thus providing a very flexible framework, in essence similar to J2EE's Java Beans

framework, however handling long lived transactions. The model may possibly be based on a selection of OCCAM primitives or other Pi-Calculus based Languages, and similar amongst others, due to their rigid structure, and extensive support for parallel running transactions. Below is a diagram which illustrates the main concept of the solution:

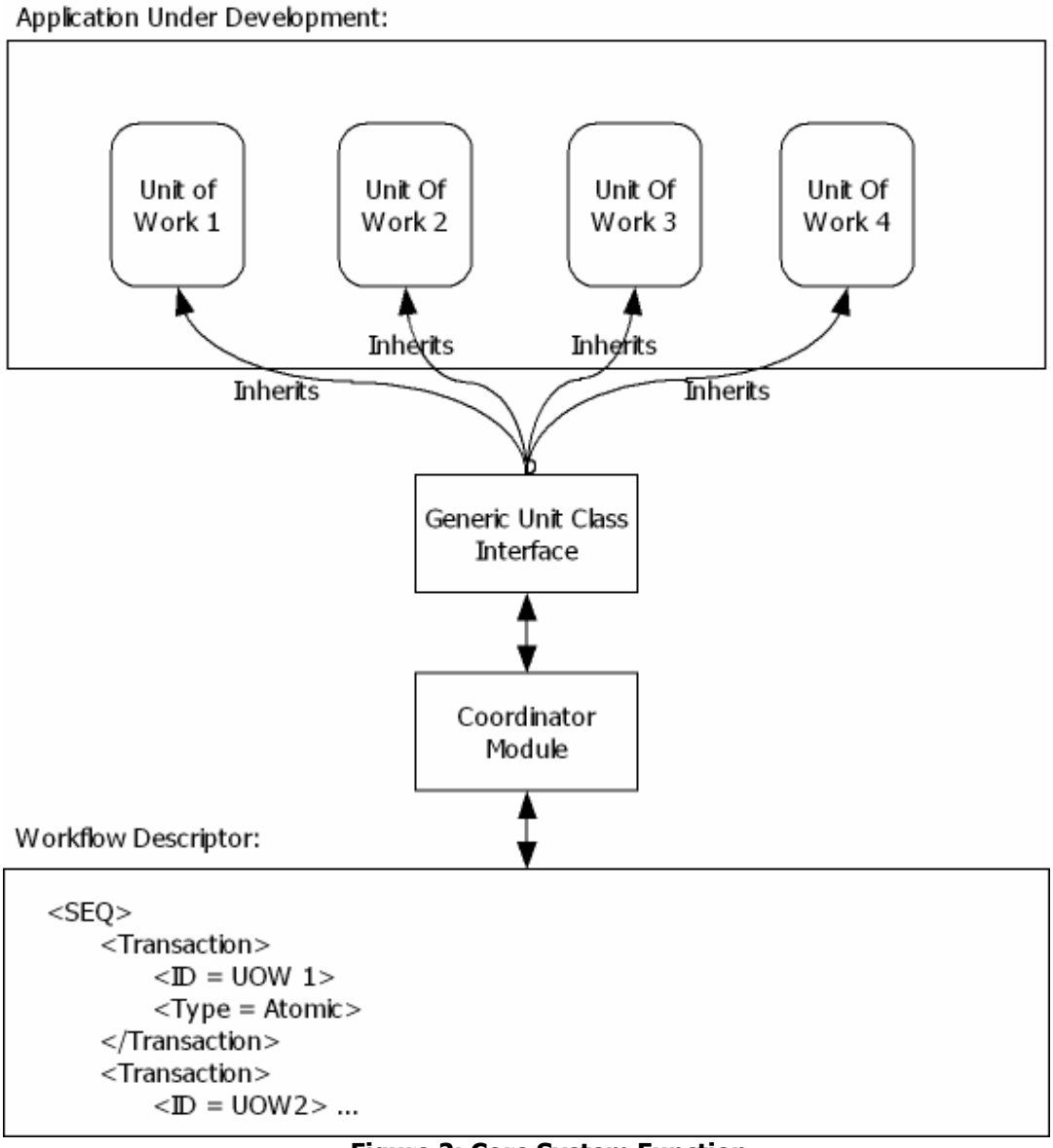


Figure 2: Core System Function

The coordinator module would be responsible for parsing the script, and executing each application module accordingly. The instructions contained in the Units of work are irrelevant to the coordinator module, since this is only concerned with Workflow Coordination.

1.5 *Deliverables*

Deliverables for this project include the following components:

- FYP Proposal Sheet
- Inception Report
- Dissertation
- Synopsis
- Software Components
    - Workflow Descriptor Language.
    - Coordinator Module
    - Framework which encapsulates developer's application.
    - Possible GUI support for generation of script and integration of modules.

P.T.O.

Chapter 2: Work Plan

This section illustrates a generalized timeline for project milestones which include the following steps:

- FYP Proposal
- General Research about the subject.
- Identification of Problem.
- High Level Design of Solution.
- Research for Technologies which will be applied.
- Generation of Inception Report.
- Detailed System Design (Class Diagram Level)
- Development
- Testing
- Finalization of Dissertation

FYP Proposal	1 October 2005
General Research	October – December 2005
Identification of the Problem	Mid December 2005
High Level Design of Solution	Late December 2005
Technology Research	Early January 2006
Inception Report	Mid January 2006
Detailed System Design	Mid January – Mid February 2006
Development	Mid February 2006 – Mid May 2006
Testing	Mid May – End of May 2006
Finalization of Dissertation Document	End of May 2006

Figure 3: Timeline

Please note that the dissertation has not been included in the table, since dissertation writing is an ongoing process throughout the whole project.

Appendix A: General Information

Model	**conTract Model**
Orientation	Long Lived Transactions (CAD/CAM)
Released	**By Andreas Reuter – 1989**
Description	The contract model was one of the early attempts at handling long lived transactions. It moves away from the idea of ACID transactions, and makes use of the concept of forward compensation. The structure of the contract model revolves around sequences of steps and scripts, where steps represent simple Units of work, and scripts represent descriptors which cater for behaviour of each unit of work with regards to concepts like interdependencies, recovery parameters, etc. The main idea in the contract model is that of abstracting workflow issues completely to the application programmer, since the script takes care of this. The official definition of a contract model, as defined by Andreas Reuter is the following: **"Contract is a consistent and fault tolerant execution of an arbitrary sequence of predefined actions (steps) according to an explicitly specified control flow description (script)" – Andreas Reuter**
Pros	• Caters for long lived transactions. • Separates Units of work from system behaviour.
Cons	• Is limited to forward recovery. • Very complex to implement.

Model	**ACTA Model/Framework**
Orientation	Short/Long Lived Transactions
Released	1990 By Chrysanthis and Ramamritham
Description	The ACTA model is based on the unification of the split join, nested and cooperative transaction models. In their specification paper, Chrysanthis and Ramamritham define

	ACTA as being a framework which extends the functionality of the amalgamation of these models, thus allowing solutions which include hybrid custom models which manifest unique behaviour, rather than a simply new transaction model. What ACTA does is mainly : *"allow the definition of structure, and behavior of transactions", and provides;* *"reasoning for the concurrency and recovery semantics of the transactions".* It can be considered more of a framework of models, rather than just another model. The core of the ACTA model semantics concentrates on the effects of a Transaction on either another Transaction, or an Object, including interdependencies between transactions, conflicts, and delegation of information from one transaction to the other.
Pros	• Much more extensible than a conventional single model system, since it allows hybrid solutions to the models it contains.
Cons	• Complex to visualize and implement.

Appendix B: Bibliography

News Sites:

Contains news on who released what protocol/standard/framework and when:

http://xml.coverpages.org/coordination.html#specs

Mark Little's Personal Blog Site:

http://markclittle.blogspot.com/2004_11_01_markclittle_archive.html

Mark Little's Web Log on Webservices.org:

http://www.webservices.org/ws/content/view/full/52229

Current Standards used by Arjuna Technologies:

http://www.arjuna.com/standards/

Articles:

Acid is good – Take it in short doses :

http://www.theserverside.com/articles/article.tss?l=AcidShortDoses

Business Transaction Protocols – Transactions for a new age :

http://webservices.sys-con.com/read/39607.htm

An Overview of Support for Extended Transaction Models in J2EE

http://www.developer.com/java/ent/print.php/1136071

A collection of articles and papers from Arjuna Technologies :

http://www.arjuna.com/library/reading.html

Corba VS SOAP based Webservices

http://searchwebservices.techtarget.com/ateQuestionNResponse/0,289625,sid26_gci930913_tax298966,00.html?bucket=ETA

JTA and JTS:

http://www.developer.com/java/ent/article.php/2224921

A comparison of Many Transaction Frameworks by Mark Little:

http://www.webservices.org/index.php/ws/content/view/full/52213

Framework Specifications:

JSR109 Web services for J2EE Documentation:

http://jcp.org/en/jsr/detail?id=109

JSR95 Activity Service Specification:

http://jcp.org/en/jsr/detail?id=095

Java API for XML Transactions

http://www.jcp.org/en/jsr/detail?id=156

WS-CAF :

http://webservices.sys-con.com/read/39936.htm

WS-Coord :

http://www-128.ibm.com/developerworks/library/specification/ws-tx/

Implementations:

Novell Bank Implementation:

http://www.novell.com/documentation/extendas50/jbroker/tm/examples/docs/resBank-1.htm

An Implementation of WS-AT standards for IBM's websphere:

http://www.alphaworks.ibm.com/tech/wsat

HP Arjuna Transaction Service with JBOSS and CORBA based on JSR95 :

http://www.arjuna.com/products/arjunats/

Models:

Two Phase Commit Model

http://www.jguru.com/faq/view.jsp?EID=20929

ACTA Model

http://swig.stanford.edu/pub/summaries/database/acta.html

Split/Join Model

http://www2.parc.com/csl/groups/sda/projects/reflection96/docs/barga/reflect/node10.html