A Meta Model for Multiple Transaction Models

Justin Spiteri
(justinspiteri@gmail.com)

Department of Computer Science and Artificial Intelligence
University of Malta

Abstract. It has been widely recognized that traditional transaction models with ACID (Atomicity, Consistency, Isolation, and Durability) properties are generally not applicable to transactions which have a compound nature. This has led researchers both in industry and academy to create a series of specialized models which cater for this type of transactions, each oriented towards a particular range of applications, according to the needs of the particular developer. This paper discusses the problems which have arisen in the transaction management field due to this approach in handling compound transactions, and proposes a solution which is an attempt at standardizing transaction model description methods. The solution includes an XML based transaction workflow modeling language specification, and a prototype transaction coordination engine which proves the proposed concepts. The full source code and dissertation can be found on: http://transitmodel.sourceforge.net.

Project Background

A number of supporting theories, concepts, models and frameworks exist that aid developers to create traditional transaction enabled applications. If however a developer needs to create a solution which handles complex transactions including multiple parties, possibly spanning over a long period of time, various difficulties may arise. This has led to the research and development of middleware solutions which abstract transaction management issues from developers. Various transaction management systems are currently available, each with distinct transaction models and standards. Accompanying the project is a literature review which analyses a selection of transaction models and systems from both a theoretical and a practical point of view in an effort to produce a solution which eliminates shortcomings present in the field of transaction processing.

The main problem identified through the literature review is that unlike traditional transaction modeling techniques where one model suited a very wide range of applications, a long running transaction model only fits a narrow band of applications to the extent that in some cases, a completely customized model must be developed for an application. This is due to the fact that while ACID oriented models are based on a fixed workflow consisting of two parties, these

models must cater for multiple parties, possibly resulting in a different workflow for each application, thus requiring a redesign of the model every time.

At present, to redesign a model, a developer must have a solid knowledge in the field of transaction management, and apprehending all the concepts needed is a time consuming task. Moreover, the solutions currently available have been implemented without a common predefined standard, resulting in each implementation being completely different with regards to architecture and standards used in transaction model design. This causes further difficulty in the apprehension of the different transaction models, since developers need to examine each specification from scratch in order to apprehend it.

While standardization efforts have been made with workflow modeling languages such as BPEL, these still require the developer to learn the transaction oriented language syntax in order to be able to create adequate transaction models.

Project Objectives

The main objective of this project is the introduction of a series of novel concepts resulting from an extensive research which standardize the process of transaction model definition, through the design and development of a specialized Meta Model, the Transit Meta Model.

The Transit Meta Model is an XML based scripting language which allows the definition of any workflow based transaction models using classic imperative language blocks such as “if then” and “for do” statements. Being compliant to W3C XML specifications, and having well known language constructs, the language succeeds in simplifying the transaction model description process, by virtually abstracting the developer from complex transactional details such as transaction inter dependencies. The only knowledge needed by a developer to use the Transit Model Solution is basic OOP programming language knowledge, knowledge of XML syntax, and basic experience in the creation of abstract workflows.

The Transit Meta Model Specification

The following list includes a full definition of the Transit scripting language syntax. These constructs define the Transit Meta Model:

<model> … </model>	The Model tag is the parent tag which encapsulates all the other script tags, which define the transaction model in question.
<name> …	The name tag simply serves as a data holder for the current name

</name>	of the Transaction Model which is being defined in the script in question.
<decl> … </decl>	The <decl> tag's main purpose is that of providing an indicator for a global or local variable declaration present in the script.
<activityList> … </activityList>	The activityList tag has the sole purpose of defining an abstract list of activities, which will be used in order to create the model. The activityList tag contains an XML attribute, named "size" which defines the size of the list. In essence the activityList has properties of an ArrayList where each position in the list signifies an activity.
<counter value = "V"> … </counter>	The counter tag is used in the <decl> tag in order to declare a local or global variable of type Integer. The Inner Text of this tag is considered to represent the variable name, while the value is stored inside a value attribute.
<workflow> … </workflow>	The scope of the workflow tag is that of containing the actual workflow definition of the model, described using classic imperative language constructs.
** … **	The segment tag is responsible for containing the core part of the Transit Script, where the actual workflow resides. The segment tag has an XML attribute named "id" whose value represents the name of the segment. This name is used by <goto> statements in order to call the segment.
<begin> … </begin>	The begin tag is the first tag which servers as an indicator point for the parser that the actual workflow definition has begun. From this point onwards, the script takes a more "Procedural 3rd Generation Language" look.
<fordo begin = "A" end = "B" counter = "C" step = "D"> … </fordo>	The <fordo> tag is similar to the for loop in the C# and Java languages. It contains four attributes in all; the "begin" and "end" attributes indicating the starting and ending value through which to loop, the "counter" indicating the variable used to keep the current value, and the "step" attribute indicating whether the loop is ascending or descending step values.
<ifthen type = "normal" index = "A" result = "B" > … </ifthen> OR	The <ifthen> tag is also similar to the if then else loop in the C# and Java languages. However the use of if then statements in the transit model is restricted to two types; those which check the outcome of the execution of an activity, and those which evaluate expressions, as seen in the syntax formats above. The "type" attribute present in the tag has two values, "normal", which indicates that the statement is an expression outcome evaluator, or

<ifthen type =expression" expression1 = "A" operator = "B" expression2= "C"> **…** **</ifthen>**	"expression" which indicates that the statement is an expression evaluator. In the "normal" statement, the "index attribute indicates the position of the Activity in the "activityList", which is under question, while the result indicates the expected outcome.
<execute position = "A" type = "B"> **…** **</execute>**	This construct is the most important construct in the script, since it maps an activity from the LLT provided by the developer, and executes it according to the parameters defined in this statement. The execute statement has two attributes, the position, which indicates the actual position of the activity to process in the activityList, and the type, which defines till what level should the execution proceed.
<goto paramone = "A" paramtwo = B"> **…** **</goto>**	The <goto> statement has the main task of issuing calls to segments, either from the main program, or from within a segment itself. Unlike the classic "goto" statement in assembly language, this goto does not promote spaghetti code, since it can only issue segment calls, similar to a method call in C# or Java. The <goto> statement can have an indefinite number of elements, which act as parameters in order to pass values to global or local variables. The engine matches the name of the attribute (for example: paramone), to the name of a variable inside a segment, or a global variable, and propagates the parameter value to it. Thus parmeters may be passed between segments through the <goto> statement.
<cmd> **Exitscript** **</cmd>**	This is a simple command which is part of the workflow, and at present contains only one command, which is the "exitscript" command. As soon as this tag is found, its inner text is analysed, and the corresponding command is executed. Plans to extend this tag are classified as future work.
<main> **…** **</main>**	The main tag has the simple scope of containing one <goto> statement, which indicates the first segment which must be called upon initial execution.

This specified syntax can be applied to create a standard transaction model definition which may then be used by a transaction manager or workflow execution software in order to run transactions according to the defined model. A transaction model thus takes the form of an XML based script, defined physically in an XML file. The following example displays a standardized representation of the classic nested model:

```
<?xml version="1.0" encoding="utf-8" ?>

<model>
        <name>Nested Model</name>
```

```
<decl>
        <activityList size = "*n*">
                arrayOfActivities
        </activityList>
</decl>

<workflow>
        <segment id = "Start">
                <decl>
                        <counter value = "0">k</counter>
                </decl>
                <begin>
                        <fordo begin = "paramone"
                                end = "paramtwo"
                                counter = "k"
                                step = "++">

                                <execute position = "k" type = "complete">
                                        arrayOfActivities
                                </execute>

                                <ifthen index = "k" result = "rolledback" type ="normal">
                                        <goto paramone = "k-1"
                                                paramtwo = "0">

                                                        RollbackAll
                                        </goto>
                                        <cmd>exitscript</cmd>
                                </ifthen>
                        </fordo>
                        <ifthen type = "expression"
                                expression1 = "k"
                                operator = "=="
                                expression2 = "paramtwo">

                                        <goto paramone = "paramone"
                                                paramtwo = "paramtwo">

                                                CommitAll
                                        </goto>
                        </ifthen>
                </begin>
        </segment>

        <segment id = "RollbackAll">
                <decl>
                        <counter value = "0">k</counter>
                </decl>
                <begin>
                        <fordo begin = "paramone"
                                end = "paramtwo"
                                counter = "k"
                                step = "--">

                                <execute position = "k" type = "rollback">
                                        arrayOfActivities
```

```
                                        </execute>
                                </fordo>
                        </begin>
                </segment>

                <segment id = "CommitAll">
                        <decl>
                                <counter value = "0">k</counter>
                        </decl>
                        <begin>
                                <fordo begin = "paramone"
                                        end = "paramtwo"
                                        counter = "k"
                                        step = "++">
                                        <execute position = "k" type = "commit">
                                                arrayOfActivities
                                        </execute>
                                </fordo>
                        </begin>
                </segment>
        </workflow>
        <main>
                <goto paramone = "0" paramtwo = "*n*">Start</goto>
        </main>
</model>
```

One of the main targets of this project was to create a standard and simple way in which transaction models may be defined. This target has been reached since scripts defined using the Transit Meta Model are simple enough to be self explanatory.

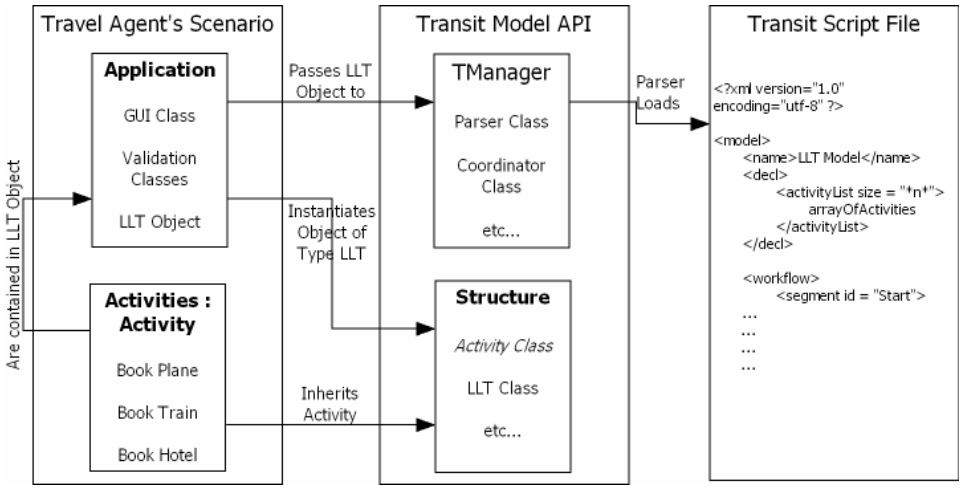
Proof of Concept

The applicability of the Transit Meta Model is proved by introducing the Transit Model Solution, a prototype compound transaction coordination system based on the Transit Model. Secondary objectives of this project included the experimentation with advanced transaction handling features, such as the introduction of long running transaction suspension and resumption mechanisms into the transaction coordination engine, together with the introduction of novel architectural concepts, such as the separation of transaction system engine design from model definitions, thus creating a “pluggable architecture” where a management system can switch transaction models. This makes the system suitable to a wider range of applications.

Project Methodology

While this project is heavily research oriented, the formal methodology used in the development of the Transit Meta Model and Solution has been based on a

hybrid of spiral and evolutionary prototyping. The general architecture of the solution is displayed below:


Travel Agent's Scenario
Application
GUI Class
Validation Classes
LLT Object
Activities : Activity
Book Plane
Book Train
Book Hotel
Are contained in LLT Object
Passes LLT Object to
Instantiates Object of Type LLT
Inherits Activity
Transit Model API
TManager
Parser Class
Coordinator Class
etc...
Structure
Activity Class
LLT Class
etc...
Parser Loads
Transit Script File
<?xml version="1.0" encoding="utf-8" ?>
<model>
<name>LLT Model</name>
<decl>
<activityList size = "*n*">
arrayOfActivities
</activityList>
</decl>
<workflow>

...
...
...
...

The transit model solution has been designed in the form of an API, in order to be easily integratable into top level solutions. In the example above, a Transit Enabled Travel Agent System's architecture can be observed. The transaction model used by the Travel Agent's System is defined in the Transit Script File, while the Transit Model API provides necessary facilities for the creation of a long running transaction, which is then executed according to the model currently "plugged" into the Transaction Manager.

The Transit Meta Model as an Open Source Project

One of the initial goals of this project has been that of posting the resulting research and code to the open source community, with the intention of creating a stream of feedback from experts competent in this area from this community. This feedback would allow the project to be improved from a series of aspects, including inclusion of previously unthought-of features and the discovery and fixing of any bugs gone undetected amongst others. Open source projects must also conform to a set of features such as formal versioning of file releases, patches and bug fixes amongst others, which could be of benefit both to the project and to the developers who download and use it.

It has been felt that the best way to transform the Transit Model Solution into an open source conformant project is by posting it the http://www.sourceforge.net open source community, through a project submission application for provision of space on their servers. This also served as an exercise to gauge the quality of the project, since project submission to source forge are reviewed by a series of technical staff, before being approved. The submission to source forge brought about the following changes/feature additions to the Transit Model Solution:

- **Web Site:** Open source practice (as stated by sourcefource) includes the creation of a web site which presents the project to the open source community. This should include a brief project description, together with links to the appropriate documentation, source code, and binary files. Links to the various open source communication tools, in this case offered by sourceforge should also be provided. The site for the Transit Model Project has been implemented and uploaded to the space provided on: http://transitmodel.sourceforge.net. For a screenshot of the site, see the appendices section, appendix H.

- **Source & Binaries**: For a project to be classified as open source, both its source code and its binary files must be posted to the open source community. In this case, source forge provides project subscribers with a standard file release system to which project administrators can post both source and release files. Source and release files for the Transit Model Solution have been made available on : http://sourceforge.net/projects/transitmodel/

- **File Versioning:** When submitting source, binary files, or bug fixes, they must be appropriately versioned, typically using an incremental numbering system. Source forge provides a standard versioning structure through its specially implemented file release system. There is currently only one version release for the Transit Model Solution, that is, version 1.0, available publicly in the downloads section of the source forge site.

- **Bug Reporting & Patch Manager:** This is the first of a series of communication tools which enables members in the open source community to report bugs to the project administrator. In our case, it is available through the source forge site for the Transit Project, together with a patch manage, which hosts similar properties to the file release system, however catering solely for project patches.

- **Feature Requests:** This utility allows community members to post suggestions to the project administrator, specifically, desirable features which the project does not possess, and which would significantly improve the project. Feature requests are also offered through the Transit Project's source forge site.

- **Screenshot Manager:** The screenshot manager allows the postage of project screenshots in a standardized image with preset dimensions and file format, ensuring view ability by all the community's members. A series of screenshots of the Transit Suspend/Resume GUI together with Sample Application Screenshots is present on the Transit Sourceforge site.

- **Forums, Mailing Lists & News:** These tools further enhance communication between the community and the project administrator, thus allowing the overall improvement and expansion of the project.

Conclusions

From the testing carried out (included in the dissertations appendices) it can be concluded that the Transit Model API provides a stable and easily integratable solution for handling various types of Long Running Transactions, through the introduction of a series of both novel concepts and concepts which have been based on the positive features of current transaction model theory and solutions. The main sources of inspiration for this thesis included ConTract Models, which introduce the notion of scripts for transaction processing, Arjuna's WS-CAF project, which has workflow based architecture, and Bell Lab's Cova TM, which is also a script based transaction management system. The main inspiration for theoretical research has been Marek Prochazka's PHD thesis on Long lived transactions, which contained invaluable information regarding currently available transaction model specifications.

The original scope of this thesis was that of the creation of a Meta Model, allowing developers with scarce transactional knowledge to easily express their own transaction models, or use ready made model templates. This scope has been reached with the development of the Transit scripting language, and the accompanying API, which proves the practical nature of the Language. At present, the project has been reviewed by source forge technical staff, and successfully approved for registration as a www.sourceforge.net open source project under an academic free licence. This effort has been undertaken in order to expose the project to the open source community, and get relevant feedback regarding ways in which the solution can be improved. The project material is available for viewing on the source forge site, at the URL's:

http://www.sourceforge.net/projects/transitmodel

or

http://transitmodel.sourceforge.net/.

References

[1] Current Standards used by Arjuna Technologies: http://www.arjuna.com/standards/

[2] Article: Acid is good – Take it in short doses: http://www.theserverside.com/articles/article.tss?l=AcidShortDoses

[3] Article: Business Transaction Protocols – Transactions for a new age: http://webservices.sys-con.com/read/39607.htm

[4] Article: An Overview of Support for Extended Transaction Models in J2EE: http://www.developer.com/java/ent/print.php/1136071

[5] A collection of articles and papers from Arjuna Technologies: http://www.arjuna.com/library/reading.html

[6] Article: Corba VS SOAP based Webservices: http://searchwebservices.techtarget.com/ateQuestionNResponse/0,289625,sid26_gci930913_tax298966,00.html?bucket=ETA

[7] Article: JTA and JTS: http://www.developer.com/java/ent/article.php/2224921

[8] Article: A comparison of Many Transaction Frameworks by Mark Little: http://www.webservices.org/index.php/ws/content/view/full/52213

[9] Framework Specification: JSR109 Web services for J2EE Documentation: http://jcp.org/en/jsr/detail?id=109

[10] Framework Specification: JSR95 Activity Service Specification: http://jcp.org/en/jsr/detail?id=095

[11] Framework Specification: WS-CAF : http://webservices.sys-con.com/read/39936.htm

[12] Framework Implementation: WS-AT standards for IBM's Websphere: http://www.alphaworks.ibm.com/tech/wsat

[13] Transaction Models: Two Phase Commit Model: http://www.jguru.com/faq/view.jsp?EID=20929

[14] Source Forge: http://www.sourceforge.net/

[15] The Contract Model (Helmut Wachter & Andreas Reuter)

[16] The ACID Model – www.about.com (Mike Chapple)

[17] CovaTM: A Transaction Model for Cooperative Transactions (Jinlei Jiang, Guangxin Yang for Bell Labs)

[18] Acta: The Acta Framework Model (Chrysanthis & Ramamritham)

[19] Advanced Transactions in Component Based Software Architectures (PHD Project by Marek Prochazka)

[20] A Meta Model for Multiple Transaction Models (Justin Spiteri May 2006)

[21] A Meta Model for Multiple Transaction Models Language Specification (Justin Spiteri May 2006)