

An improved web-based solution for specifying transaction models for CAGISTrans

Oppgavetekst:

Oppgaven går ut på å vurdere et eksisterende web-basert brukergrensesnitt for spesifisering av transaksjonsmodeller. Spesielt skal det utredes hvorvidt det vil være fordelaktig å lage en ny løsning basert på Web Services. I tillegg skal den eksisterende løsningen vurderes på kriterier som brukervennlighet, design og løsningsarkitektur. Til slutt skal en ny løsning implementeres på bakgrunn av disse undersøkelsene.

Veileder: Heri Ramampiaro

Preface

This report contains the documentation of the master thesis “An improved web-based solution for specifying transaction models for CAGISTrans”. The work has been carried out from 20.01.2005 until 30.06.2005 at the Department of Computer and Information Science (IDI), Norwegian University of Science and Technology (NTNU). The thesis is submitted in partial fulfilment of the degree “Sivilingeniør” (MSc.).

I would like to thank my supervisor Heri Ramampiaro for his counsel and advice.

Trondheim, June 30, 2005

Thomas Eugen Bjørge

Abstract

Transactions have been used for several decades to handle concurrent access to data in databases. These transactions adhere to a strict set of transactional rules that ensure that the correctness of the database is maintained. But transactions are also useful in other settings such as supporting cooperative work over computer networks like the Internet. However, the original transaction model is too strict for this. To enable cooperation between transactions on shared objects, a framework for specifying and executing transaction models adapted to the environment in which they are running has been developed. Additionally, a web based user interface for the specification of transaction models for the framework has also been created.

In this thesis we look at how the process of specifying transaction models for the framework can be improved. More specifically, we start by carefully reviewing the current web based solution for specifying transaction models. In our review we focus on usability, design and the technical aspects of the solution. We then continue with a thorough look at Web Services in the context of the transaction model framework. Our main objective at this stage is evaluating the possibility of implementing a new solution for specifying transaction models using Web Services. The last part of our work is the actual implementation of an improved application for specifying transaction models. This implementation is based on the results from our evaluation of the current solution and our evaluation of Web Services.

We identified several issues in our review of the current solution. The main problem is that it is difficult for the user to get a good overview of the transaction model she is creating during the specification process. This is due to the lack of a visual representation of the model. The specification process is also very tedious, containing a large number of steps, a number we feel can be reduced. The technical aspects of the solution also have a lot of room for improvement. The overall design can easily be improved, and additionally utilizing different technologies would make the application less error prone, and also easier to maintain and update.

We reached the conclusion that Web Services is not an ideal technology for a transaction model specification application. The main problem is that the client needs to have a complete overview over the specification process leading to a lot of duplication of data between the client and the web service. In the end this situation leads to a very complex web service that does not improve the transaction model specification process.

Based on our results, we decided to implement a web based solution for specifying transaction models. Our solution is similar to the original one, but we had strong focus on improving its shortcomings, both on the usability side and the technical side. This meant focusing on giving the user a good overview of the transaction model during the specification process and also reducing the number of steps in the process. Additionally, we put a lot of effort into developing a solution that is based on technological best practices, leading to an application that is less error prone than the original. It should also be easier to maintain and update.

Table of contents

PREFACE	I
ABSTRACT	III
TABLE OF CONTENTS	V
LIST OF FIGURES	VII
CHAPTER 1 INTRODUCTION	1
1.1 PROBLEM DESCRIPTION.....	1
1.2 THESIS OUTLINE.....	1
CHAPTER 2 BACKGROUND	3
2.1 TRANSACTIONS.....	3
2.2 TRANSACTION MODELS.....	4
2.2.1 <i>Nested transactions</i>	4
2.3 CAGISTRANS	5
CHAPTER 3 CURRENT SOLUTION	9
3.1 DESCRIPTION	9
3.1.1 <i>The application</i>	9
3.1.2 <i>The generated XML</i>	9
3.2 REVIEW	10
3.2.1 <i>XML and DTDs</i>	10
3.2.2 <i>Manual XML editing</i>	10
3.2.3 <i>Wizard</i>	11
3.2.4 <i>Technical solution</i>	12
3.2.5 <i>Review summary</i>	13
CHAPTER 4 UTILIZING WEB SERVICES	15
4.1 INTRODUCTION	15
4.2 CORE WEB SERVICES TECHNOLOGIES.....	16
4.2.1 <i>SOAP</i>	17
4.2.2 <i>WSDL</i>	17
4.2.3 <i>UDDI</i>	17
4.3 WEB SERVICES AND CAGISTRANS	18
4.3.1 <i>Design</i>	18
4.3.2 <i>Evaluation of the Web Services solution</i>	19
CHAPTER 5 DESIGN AND ARCHITECTURE	21
5.1 TREE CONTROL.....	21
5.2 MVC PATTERN.....	22

CHAPTER 6 APPLICATION TECHNOLOGIES.....	25
6.1 JAVASERVER PAGES	25
6.1.1 JSTL and EL	27
6.2 JAVABEANS	28
6.3 JDOM	28
CHAPTER 7 IMPLEMENTATION	31
7.1 IMPLEMENTING THE TREE CONTROL.....	31
7.2 IMPLEMENTING THE MVC PATTERN	32
7.3 XML AND DTD.....	33
7.4 REDUCING THE NUMBER OF APPLICATION PAGES	35
CHAPTER 8 APPLICATION DESCRIPTION.....	37
8.1 START PAGE.....	38
8.2 ACID PROPERTIES	38
8.3 INITIAL TRANSACTION OVERVIEW.....	39
8.4 ADD MAIN TRANSACTION	39
8.5 ADD SUBTRANSACTION.....	40
8.6 ADD OPERATION	41
8.7 ADD IN-ARGUMENTS.....	42
8.8 ADD OUT-ARGUMENTS	43
8.9 CREATE NEW ARGUMENT.....	44
8.10 SELECTED TRANSACTION IN OVERVIEW	45
8.11 COMPLETED TRANSACTION SETUP	46
8.12 SELECTED TRANSACTION FOR DEPENDENCIES.....	47
8.13 DEPENDENCIES SPECIFICATION	48
8.14 AWARENESS AND LOCKING MECHANISMS.....	49
8.15 AWARENESS MECHANISMS WITH FULL ISOLATION	50
8.16 CONFLICTING OPERATIONS	51
8.17 PERMITTED CONFLICT.....	52
8.18 DEMANDS	53
8.19 COMPLETED TRANSACTION MODEL.....	54
8.20 ERROR PAGES	54
CHAPTER 9 CONCLUSIONS AND FUTURE WORK.....	57
9.1 CONTRIBUTION	57
9.2 CONCLUSIONS.....	57
9.3 FUTURE WORK	58
REFERENCES	61
APPENDIX A DTD AND XML.....	63
A.1 DTD	63
A.2 EXAMPLE XML	65
APPENDIX B SOURCE CODE	69
B.1 CASCADING STYLE SHEETS FILES	69
B.2 JAVASCRIPT FILES.....	70
B.3 JAVASERVER PAGES FILES	76
B.4 JAVA FILES.....	98

List of figures

Figure 2.1: Nested transactions	4
Figure 4.1: SOAP, WSDL and UDDI	16
Figure 4.2: Architecture overview	18
Figure 4.3: A simple temperature web service	20
Figure 5.1: Windows Explorer tree control	21
Figure 5.2: The MVC design pattern	23
Figure 6.1: Generating dynamic content with JSP elements	25
Figure 6.2: A simple JSP page using a scripting element	26
Figure 6.3: A simple JSP page using a custom tag	26
Figure 6.4: A JSP page using JSTL and EL	27
Figure 6.5: The resulting web page	27
Figure 6.6: An example XML document	28
Figure 6.7: Creating the example with standard Java	29
Figure 6.8: Creating the example with JDOM	29
Figure 7.1: The application's data model	33
Figure 7.2: Original dependency element DTD	34
Figure 7.3: Modified dependency element DTD	34
Figure 7.4: Original demand element DTD and xml example	34
Figure 7.5: Modified demand element DTD and xml example	34
Figure 8.1: Application flow	37
Figure 8.2: Start page	38
Figure 8.3: ACID properties	38
Figure 8.4: Initial transaction overview	39
Figure 8.5: Add main transaction	39
Figure 8.6: Add subtransaction	40
Figure 8.7: Add operation	41
Figure 8.8: Add in-arguments	42
Figure 8.9: Add out-arguments	43
Figure 8.10: Create new argument	44
Figure 8.11: Selected transaction	45
Figure 8.12: Completed transaction setup	46
Figure 8.13: Selected dependency transaction	47
Figure 8.14: Dependencies specification	48
Figure 8.15: Awareness and locking mechanisms	49
Figure 8.16: Awareness mechanisms with full isolation	50
Figure 8.17: Conflicting operations	51
Figure 8.18: Permitted conflict	52
Figure 8.19: Demands	53
Figure 8.20: Completed transaction model	54
Figure 8.21: No transaction selected #1	54
Figure 8.22: No transaction selected #2	55
Figure 8.23: No transaction selected #3	55

Chapter 1

Introduction

The way computers are being used has changed dramatically the last few years. The Internet has gone from being a research project to becoming a natural part of people's lives. This has also affected the way people work and the way people use computers in their work. Through the Internet vast numbers of computers are now connected together, forming an enormous network. To utilize this resource, several new computing models have emerged. These new computing models include new ways of cooperation and sharing of data between users and applications. This change has also brought with it the challenge of how to provide efficient support for these new cooperative activities.

1.1 Problem description

Transactions were originally used to ensure the correctness of databases, meaning that they are very strict. But transactions are also useful in other settings, such as supporting cooperative work over computer networks like the Internet. However, the original transaction model is too strict for such cooperation. To overcome this, less strict models have been developed for different application needs. But there is no clear answer to what model suits what application the best. As a result of this, frameworks that allow tailoring of transaction models to different application needs have emerged. One such framework is the CAGISTrans framework [1], which was specifically developed to support cooperative work.

In order to specify a model for the CAGISTrans framework a lot of information is needed. This leads to a fairly complicated specification process when defining a transaction model. The main focus of this thesis will be on this specification process. There already exists a solution for specifying transaction models for the CAGISTrans framework. This is a web based application that guides the user through the process of specifying a transaction model. We want to take a detailed look at this current solution, evaluating it on several different criteria. The criteria which will be our main focal points are usability, design and the technical architecture of the solution. Furthermore we wish to investigate Web Services in the context of CAGISTrans. More specifically, we want to thoroughly review the possibility of implementing a transaction model specification solution using Web Services. Finally, based on our evaluations of the current solution and Web Services, we will implement a new solution for specifying transaction models. The ultimate goal of this new solution is to simplify as much as possible the process of specifying transaction models for the CAGISTrans framework.

1.2 Thesis outline

- *Chapter 2* describes the background for this thesis, more specifically transactions, transaction models and the CAGISTrans framework [1].

- *Chapter 3* discusses the current solution for specifying transaction models for the CAGISTrans framework. The solution is described and then carefully reviewed.
- *Chapter 4* contains a detailed look at Web Services, both in general and in the context of the CAGISTrans framework.
- *Chapter 5* presents the design and architecture of our application for specifying transaction models.
- *Chapter 6* contains a thorough look at the technologies used in our application.
- *Chapter 7* details the implementation of our solution.
- *Chapter 8* is a detailed description of our finished application complete with screenshots.
- *Chapter 9* concludes our work by summarizing its contribution and presenting our conclusions along with future work.

Chapter 2

Background

This chapter describes the main aspects of the CAGISTrans framework [1]. Additionally, it introduces the basic concepts of transactions and transaction models. This is done to make it easier to understand the description of the CAGISTrans framework which is provided in the last section of the chapter.

2.1 Transactions

Transactions are normally used in conjunction with databases. A database is a collection of persistent data objects satisfying a set of integrity constraints [1]. A database management system (DBMS) is responsible for coordinating access to the database. Users interact with the database indirectly through special application programs. These programs interact with the database through the DBMS. This interaction results in a partial set of read and write operations, known as transactions [2]. A broad definition of a transaction is that it is a set of operations that access and update the contents of a database along with special transaction commands. These commands are usually begin, abort and commit which serve as the transaction's boundary markings. A transaction begins with a begin command and ends with either a commit or an abort command. A commit command indicates that the execution of the transaction was successful and that the changes made by the transaction should be incorporated into the database. On the other hand, an abort command indicates that the transaction failed in some way and that all of its changes must be removed by the DBMS, restoring the database to its state before the transaction begun. The operation of restoring the database to its previous state is called a roll-back. A transaction is an atomic unit of work, either every step within the transaction completes or none of them do.

Transactions are constrained by four fundamental properties, known as the ACID properties [2]. These constraints ensure that no transaction can violate the integrity constraints of the database. Additionally, they help the DBMS protect the user programs from hardware and software failures. The ACID properties are as follows [1]:

- *Atomicity* - referring to the fact that the transaction either completes successfully, i.e. commits, making the work that was performed within its scope permanent or it fails, and all its effects are rolled back. If it rolls back, it is as though the transaction never started in the first place, i.e. all or nothing.
- *Consistency* – requiring that each executed transaction always preserves the consistency of the database, i.e. transforms the database from one consistent state to another consistent state.

- *Isolation* – referring to the fact that intermediate states produced while a transaction is being executed are not visible to others. This means that a transaction must not make its updates visible to other transactions until it has successfully completed.
- *Durability* – requiring that once a transaction has committed, all its updates become permanent in the database, i.e. the results produced by the transaction are not destroyed by any type of subsequent failures.

2.2 Transaction models

A transaction that exhibits all of the ACID properties is often called a flat transaction. This is the most commonly-used and also the simplest transaction model. If an application function can be represented as a flat transaction, this model is sufficient. However, this is frequently not the case. Flat transactions are most suitably viewed as “short-lived” entities, performing stable state changes to the system; they are less suited when used in “long-lived” application functions. Long-lived, flat transactions (running for minutes, hours, or days) may reduce the concurrency of the system to an unacceptable level by holding on to resources, e.g. locks, for a long time. Additionally, if a long-lived transaction rolls back a lot of valuable work could be wasted. This has led to many proposed enhancements to the traditional flat model in the form of several new transaction models. One enhancement is nested transactions which are discussed next. For a discussion of several other transaction models refer to [1].

2.2.1 Nested transactions

The nested transaction model was introduced by [3] to extend flat transactions by splitting transactions hierarchically into several sub-transactions. This leads to a tree like structure of parents and children, see figure 2.1. If a parent aborts, all of its children must also abort, but if a child aborts, its parent is not required to abort, so the work done by the parent and/or other children that have committed is preserved. This is the main advantage of this transaction model. It is also possible for a parent to start another child transaction if one of the original children fails. Since the state-changes made by the transaction hierarchy only become durable when the root transaction commits, no special failure recovery is needed.

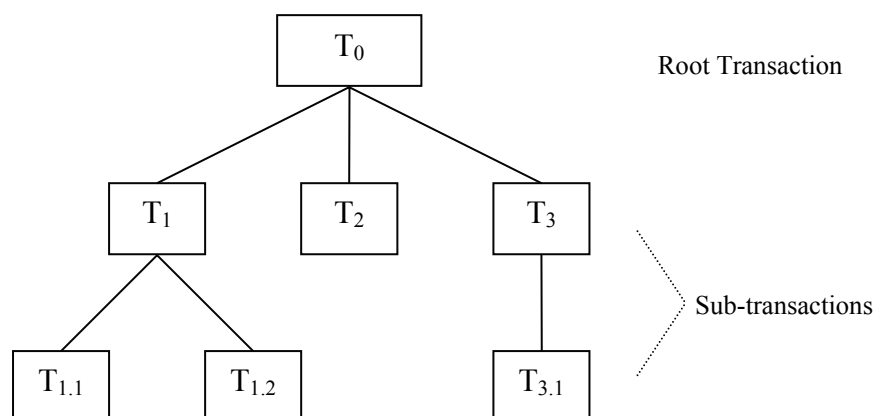


Figure 2.1: Nested transactions

The original nested transaction model can be extended by relaxing the atomicity criterion in a controlled manner. With this mechanism it is possible to invoke a transaction with root

properties from within another transaction, meaning that its results are made permanent when it commits, even if any of its parents roll back. In the event that the invoking transaction aborts, compensation may be required. The nested transaction model can also open the possibility of relaxing the isolation criterion by making the changes of a sub-transaction visible before its corresponding root transaction has committed. This enables cooperation between transactions on shared data.

2.3 CAGISTrans

CAGISTrans [1], [4] is a framework developed to provide adaptable transaction support for cooperative work. The framework allows for the specification of transaction models suiting specific applications, including runtime refinements of the transaction model to support dynamic environments. CAGISTrans uses the ACID properties as the basis for the transaction characteristics specification. Specifically, the atomicity and isolation properties are adaptable. The consistency and durability properties should not be relaxed [4].

The framework enables switching between full and relaxed atomicity. Short-lived transactions that do not involve interaction are flat, and considered atomic, meaning that they are according to the all-or-nothing law which full atomicity represents. Long-lived transactions are assumed to be nested and the aborts of such transactions can be managed in a more controlled manner. The idea is to specify transaction atomicity by means of transaction dependencies. This is already the case with nested transactions where all the children must abort if their parent aborts. The drawback of this is that all the children's work is lost, something which may not be desirable. To remedy this, abortsets are suggested. An abortset is the set of transactions that must abort if another transaction aborts, specifically:

$$Abortset(t_i) = \{t_j \in T \mid abort(t_i) \rightarrow abort(t_j), i \neq j\}$$

where $T = \{t_0, t_1, t_2, \dots, t_n\}$. This means that $Abortset(t_i)$ is the set of transactions t_j such that if t_i aborts, t_j must also abort.

To make cooperation possible, sharing of tentative data is necessary. Therefore, transactions must be able to reveal their intermediate results, i.e. relaxing the isolation property. To allow relaxed isolation abortsets are again used. However, the abortset of a transaction is not limited to only contain its children. Instead, in addition to children, the abortset can also contain other cooperating transactions that should be aborted with the transaction that the abortset belongs to. This leads to the following definition of an abortset:

$$Abortset(t_{i,j}) = \{t \in \bigcup_{i=1}^m T_i \mid abort(t_{i,j}) \rightarrow abort(t)\}$$

where $T_i = \{t_{i,0}, t_{i,1}, t_{i,2}, \dots, t_{i,n}\}$, $i = 1, \dots, m$. This means that $Abortset(t_{i,j})$ may contain transactions that are children of other transactions.

When isolation is not relaxed, the serializable correctness criterion is used. Serializability is achieved by using the 2 phase locking (2PL) protocol [2]. CAGISTrans also has awareness mechanisms which allow users to be notified when certain events occur. With full isolation these events can be when a transaction begins or terminates. The presence of awareness mechanism is not required, but is helpful in accommodating users' activities.

With relaxed isolation the serializable correctness criterion cannot be used. Instead user-defined correctness criteria are applied by utilizing flexible locking, workspaces and awareness mechanisms [4]. The awareness mechanisms available when the isolation property is relaxed are notify on begin, notify on terminate, notify on lock and notify on change. The first two are the same as for full isolation. Notify on lock means that all involved parties are informed when a transaction acquires or releases a lock on an object. Notify on change means that all involved parties are informed when a data object is being altered by a specific transaction operation.

Flexible locking means using either the user-controlled lock protocol or the collaborative lock protocol, both of which are defined in [1]. The user-controlled lock protocol is based on object locks, i.e. each object has an attribute telling which transaction is holding a lock on it. Another transaction cannot acquire a lock on a locked object until the existing lock is released. This protocol restricts cooperation since two transactions can not simultaneously access an object. To allow cooperation the collaborative lock protocol is applied. Again locks are associated with specific objects, but this protocol allows several transactions to acquire locks on the same object at the same time. If a transaction requests a lock on an object already locked by another transaction, it must specify its intention with the object, browse, incorporate or modify. If the intention is browse, the lock is granted right away. If the intention is incorporate the lock is granted, but the sequence of future actions must be specified using demands. Demands are described in the next paragraph. If the intention is modify, the arisen conflict may be resolved by negotiation between the involved parties.

When isolation is relaxed, transaction execution must still be managed in a controllable manner to achieve acceptable data consistency. This is done in CAGISTrans by a set of user-controlled correctness criteria, namely conflicts, permits and demands [4]. Conflicts define operations that cannot execute concurrently. Permits define operations that are allowed to execute at the same time in spite of being conflicting, thus allowing cooperation. Demands specify operation sequences that must appear to achieve correct transaction execution.

Another important concept in CAGISTrans is advanced operations. Advanced operations are commands specified at an abstraction level higher than, but based on, read and write operations [4]. One of the reasons for using advanced operations is that there are types of activities that cannot be easily be modeled with read and write operations. Another reason is that using advanced operations makes it possible to exploit the semantic knowledge of an operation, for instance its intention. An operation can have two types of read intentions, browse or incorporate. Browse means that the read operation will not affect other users. Incorporate, on the other hand, will probably have an impact on other users. By using this, a general conflicting rule can be specified. First of all, an advanced operation is defined as:

$$Op_i = \{OpType_i, InSet_i, OutSet_i, BrowseSet_i\}$$

where $BrowseSet_i$ contains the objects to be read by the operation with browse intention. Objects in $InSet_i$ and not in $BrowseSet_i$ are by default incorporated. Two operations are defined as compatible, i.e. not conflicting by the following rule:

$$\begin{aligned} \text{Compatible}(Op_i, Op_j) \Leftrightarrow & (OutSet_i \cap OutSet_j = \phi) \\ & \wedge (\forall Ob \in (InSet_i \cap OutSet_j), Ob \in BrowseSet_i) \\ & \wedge (\forall Ob \in (InSet_j \cap OutSet_i), Ob \in BrowseSet_j) \end{aligned}$$

This means that two operations Op_i and Op_j are compatible if they do not perform updates on any common objects and objects read by one of the operations, and updated by the other one, only are for browse with the inputting operation.

Chapter 3

Current Solution

The topic of this chapter is the current solution for specifying transaction models for the CAGISTrans system. Section 3.1 describes the main aspects of the solution, and section 3.2 contains a thorough review of these main aspects.

3.1 Description

The current solution for specifying transaction models for the CAGISTrans system is detailed in [5]. Basically it is a web application that guides the user through the setup process and produces two XML files that describe the specified transaction model.

3.1.1 The application

The application gives the user an initial choice to either follow a wizard type setup or to manually write the XML files. If the manual approach is selected, the Document Type Definitions (DTDs)¹ of the two XML files are presented to the user in two text fields where she can type in the XML. When the XML files are complete, the user is given the opportunity to validate the files with an external validator.

If the user chooses to follow the wizard, she will be guided through several steps in order to complete the transaction model specification. These steps include specifying ACID properties, transactions, subtransactions and operations and their input- and output-arguments. If the ACID properties atomicity and isolation are set to relaxed, more steps are necessary to complete the specification. These steps include specifying dependencies between transactions, awareness mechanisms, locking mechanisms and if there are conflicting operations, permits and demands. At the end of the wizard the two generated XML files are displayed to the user along with their corresponding DTDs.

To keep track of data during the specification process a SQL database is used. Specifically, transactions, operations and arguments are each stored in a corresponding database table. The data is stored and retrieved from the database using JDBC² statements.

3.1.2 The generated XML

The transaction models generated by the application are represented by two XML files. One file is the characteristics specification and the other one is the execution specification. The characteristics file contains the transaction model's ACID properties, all dependencies between transactions, locking and awareness mechanisms and all conflicts, permits and

¹ See <http://www.w3.org/TR/REC-xml/#dt-doctype>

² See <http://java.sun.com/products/jdbc/>

demands between operations. The execution file contains the transaction model's complete tree structure, describing all transactions and operations specified. Refer to [5] for an in-depth discussion of the XML files and their corresponding DTDs.

3.2 Review

In this section we will take a critical look at the existing solution, highlighting its pros and cons. Our main focus will be on two things, the first being the application's usability. Here we will take a close look at the wizard approach, reviewing the user experience when using the wizard to specify a transaction model. Our second point of focus will be the technical solution itself. More specifically, we will look at which technologies have been used to implement the application, which technologies it utilizes, and how well-suited all of these technologies are to handle the tasks they perform. Additionally, we will also review the XML and DTD portions of the application. The ultimate goal of this review is to uncover all aspects of the current solution that are not optimal and thus can be improved. We start with a brief discussion of the application's output, the two XML files and their corresponding DTDs. Following this, we discuss the manual XML editing part of the application and then look at the wizard part, which is the main component of the application. Finally we finish with a discussion of the technical aspects of the entire application.

3.2.1 XML and DTDs

The XML documents generated by the application are quite good. They contain all necessary information about the created transaction model. The DTDs define a structure that is both logical and easy to understand. Elements are organized in a sensible fashion, containing everything that is necessary, but nothing more. One exception to this is the way demands are specified. A demand is a specific sequence that operations must be executed in. This is stored in the XML file the following way: The first operation is the main element, the second operation is the first's child element, the third operation is the second's child element, and so on. The recursive structure complicates an otherwise very clean and simple XML file.

We also feel that abort dependencies could be represented differently. Abort dependencies are represented by a general dependency-element which can have the following type attributes: commit, begin, atomabort or isolabort. We feel that it is not necessary to differentiate abort dependencies, i.e. to explicitly store if the cause of the abort dependency is relaxed atomicity or relaxed isolation. This differentiation is only important when deciding which transactions can be abort dependent on other transactions, but not when a dependency actually has been specified.

3.2.2 Manual XML editing

The manual XML part was probably intended as an alternative to the wizard allowing faster production of the two XML files. We feel however that this functionality should not have been included in the application. The main reason for this is that other than displaying the DTDs the application does not help the user in any way. Additionally, it relies on an external XML validator available online, which may not be functioning or removed at any time. We feel that when manually creating the XML files the use of an XML editor is a far better solution. XML editors generally offer syntax-highlighting and more importantly validation. If the XML files must be fed to the CAGISTrans system through the web application, it could provide a simple file upload mechanism.

3.2.3 Wizard

The wizard part of the application is the main part. One of the biggest advantages of the wizard approach is that the user is informed of any errors during the specification and gets the chance to correct these errors before moving on to the next step. This approach ensures that the generated XML files do not contain any errors, neither structurally or logically. The other big advantage of the wizard approach is that the user is not required to know anything about XML, making the application easier to use.

The application is developed with user guidelines in focus. User guidelines refer to user-interface elements such as error-messages, alarms, prompts and so on. The fundamental objectives of user guidelines are to promote quick and accurate use of the full capabilities of the system [5]. Using results from the Human Computer Interface (HCI) community these guidelines are incorporated into the application. This focus on HCI and usability makes the application's interface quite good. Everything is clearly presented and help is readily available at all steps of the specification process. Each step is also quite simple and at no point does the application overwhelm the user with information or tasks that need to be performed. The application also provides good feedback to the user, clearly stating if the interaction succeeded or failed.

There are however also several aspects of the application that can be improved. First of all, it can sometimes be difficult to get a good overview of the actual specification. When specifying new transactions, subtransactions and operations it is important to have a good understanding of the tree structure being created. The way the tree structure is presented to the user is far from optimal. Only a textual representation is available, where the tree is presented for one root transaction and its children and operations. The textual representation makes the tree structure hard to visualize, especially if the structure is complicated, i.e. with several levels and many operations and transactions. It is also unfortunate that it is only possible to view the tree structure of one root transaction at a time, meaning that the user can not get a complete overview of all specified transactions and operations at once.

The pages for specifying dependencies between transactions do not even have the possibility of displaying the textual representation of the model's tree structure. Without a clear picture of the relationships between the different transactions, it is difficult to specify which transactions are dependent on each other. The same situation arises when specifying permits and demands for conflicting operations, since these pages share the same weakness as the dependencies pages. The lack of a good overview makes these steps of the process a lot more difficult than they need to be.

The wizard that the user needs to complete in order to specify the transaction model contains very many steps. In fact the user visits 22 different pages if she is specifying a model which has relaxed atomicity and isolation and also contains conflicting operations. Some of these 22 pages also need to be visited several times to complete the specification. Additionally, we have not counted the confirmation pages that appear after a transaction or operation has been successfully added to the specification, which makes the actual number of pages visited even higher. This situation is clearly not optimal, even though specifying a transaction model inevitably is a quite complex task. We feel that some of the pages in the application can be removed or combined into fewer pages. One example is transaction dependencies, where the user has to go through 4 pages specifying begin, commit, atomicity abort and finally isolation abort dependencies. It should be possible to specify these dependencies in some kind of combined way, instead of going through 4 separate steps.

3.2.4 Technical solution

The application is developed as a web application using Java Servlets³. Web applications run in standard web browsers and do not require the user to download or install any software on her computer. This is an advantage since all the user has to do is enter the web address of the application in her browser, and she is ready to start using it.

Servlets were the initial Java technology for creating dynamic web pages. Servlets are Java classes that in addition to containing ordinary application logic coded in Java, also can produce HTML markup through the use of Java print statements. Servlet code that generates HTML is hard to maintain and change [6]. In order to change the layout of a page, one needs to edit Java code. Currently Servlets are less used for generating HTML, instead a newer technology, JavaServer Pages (JSP)⁴, is being utilized more and more. We discuss JSP in chapter 6.1. Servlets are by no means obsolete however, they are still commonly used in web applications, but generally only to provide application logic, and not to create the user interface.

Apart from the sole use of Servlets for creating HTML, there are other issues with the application's technical solution. The biggest issue is the design of the application. Although it consists of 12 different Java classes, 11 of these are only small helper classes. There is one main class that contains almost all the application logic and at the same time is responsible for generating every single web page presented to the user. This is a very large class that would be a nightmare to maintain and further develop because of its sheer size and the vast number of functions it performs.

The database part of the solution works quite well. The database tables contain the necessary information, and the data is stored in a logical manner. However, all the database interaction is handled by the application's main class which makes this class even more complex. But the main issue is how database connections are handled. The application opens a new database connection for each user. This is not an optimal solution especially if the application is accessed by a large number of users simultaneously. Opening up a database connection can actually take a second or two and a connection is expensive to keep open in terms of server resources such as memory [6]. Because of this, a lot of high-traffic websites today use connection-pools. A connection pool contains a finite set of database connections which are shared across the application. When a page needs database access, it requests a connection from the pool. If there are free connections, one is given to the page, and the page returns it to the pool when it is finished accessing the database. If there are no connections available, the page must wait until another page returns its connection to the pool. By separating the code that handles database connections from the rest of the application, a connection pool could be implemented more easily.

Another issue with the application design is the XML generation. As previously stated the result of the transaction model specification is two XML files. These two files are also created by the main class of the application (further highlighting the design issues already mentioned). The issue here however is not where the files are created, but rather how. The XML generation is hardcoded into the Java code as a large number of print line statements. This is not a clever way to generate XML, since it is a method prone to errors similar to HTML generation in Servlets. Additionally the method is not very flexible when it comes to

³ See <http://java.sun.com/products/servlet/>

⁴ See <http://java.sun.com/products/jsp/>

handling changes to the DTDs. If at a later time it is decided to change the structure of the generated XML, this requires tedious changes to many, if not all, of the print line statements in the Java code. Additionally, the DTDs themselves are also included at the start of the generated XML files. This means that changing the DTDs not only requires altering the code that creates the XML, but also altering the code that writes the DTDs, further increasing the possibilities of errors and complicating the process.

3.2.5 Review summary

We have pointed out several issues with the current solution that can be improved. First of all there are a couple of small changes in the DTDs we feel would improve the generated XML files. We also think the manual XML editing part of the application is superfluous, since XML editors would do a much better job of this. Moving on to the wizard part of the application, we feel that although the idea is good, the execution could be better. There are two main problems, first of all, it is difficult to get a good overview of the transaction model during the specification process. This is caused by the lack of a good visual representation of the relationship between the different transaction and operations. The second problem is the length of the specification process. We feel that the user has to go through too many different web pages to complete the transaction model specification. When it comes to the technical aspects of the current solution, there are also several issues we feel could be improved. First of all the Java Servlet technology is not optimal for creating HTML web pages. Secondly, the application is not well designed, it basically has one main class that does almost everything. Additionally, the application's database access strategy can be improved to better handle a larger number of users. The last issue of the technical solution is the way the resulting XML files are generated. The files are "manually" created by a series of tedious print statements. We feel that it would be far better to utilize some kind of higher level XML library, which would reduce the possibilities of errors and also make it a lot easier to incorporate changes to the XML structure into the application.

Chapter 4

Utilizing Web Services

In this chapter we take a closer look at Web Services. We start with a general introduction in section 4.1, before we move on to describe the core Web Services technologies in section 4.2. Finally, in section 4.3, we investigate the possibility of implementing a transaction specification application for CAGISTrans using Web Services.

4.1 Introduction

Web Services is one of the most talked about technologies of the new millennium. Although the technology is young, it is growing and maturing rapidly with lots of both commercial and research effort behind it. Web Services are also referred to as XML Web Services due to the fact that XML is the fundament on which Web Services are built. There are several definitions of what exactly a web service is, the following is given in [7]: “A web service is a piece of business logic, located somewhere on the Internet, which is accessible through standard-based Internet protocols such as HTTP or SMTP”. Microsoft gives the following definition [8]: “Web services are the fundamental building blocks in the move to distributed computing on the Internet”. These are just two of numerous definitions, however most of the definitions describe the following behavior characteristics of a web service [7]:

- *XML-based* – XML is used as the data representation layer for all Web Services protocols and technologies. This means that these technologies are interoperable at their core level eliminating language, operating system and platform bindings.
- *Loosely coupled* – The client and server logic is not closely tied together, meaning that a service can be updated without compromising the client’s ability to use it. Loosely coupled software is more manageable and allows for easier integration with other systems.
- *Coarse-grained* – A Java program, for example, consists of several fine-grained methods that together provide a coarser grained service. A web service on the other hand is coarse-grained, providing a natural way to define a service that accesses the right amount of business logic.
- *Ability to be synchronous or asynchronous* – Synchronicity refers to the binding of the client to the execution of the service. A synchronous invocation means that the client blocks and waits for the service to complete before continuing. An asynchronous invocation on the other hand, means that the client can perform other functions while waiting for the service to complete. Asynchronicity is important for enabling loosely coupled systems.

- *Supports Remote Procedure Calls* – Using an XML-based protocol, clients can invoke procedures, functions and methods on remote objects, e.g. Enterprise JavaBeans and .Net components.
- *Supports document exchange* – XML is used represent both data and complex documents, enabling transparent document exchange which facilitates business integration.

4.2 Core Web Services technologies

Currently the core of Web Services technology consists of three technologies that have become worldwide standards [7], [8]:

- *Simple Object Access Protocol (SOAP)* – Web Services expose useful functionality to Web users through a standard Web protocol. This protocol is almost always SOAP.
- *Web Service Description Language (WSDL)* – Web Services provide a way to describe their interfaces in enough detail to allow a user to build a client application to talk to them. This description is usually provided in an XML document called a WSDL document.
- *Universal Description, Discovery, and Integration (UDDI)* – Web Services are registered so that potential users can find them easily. This is done with UDDI.

Figure 4.1 shows the relationship between these three technologies.

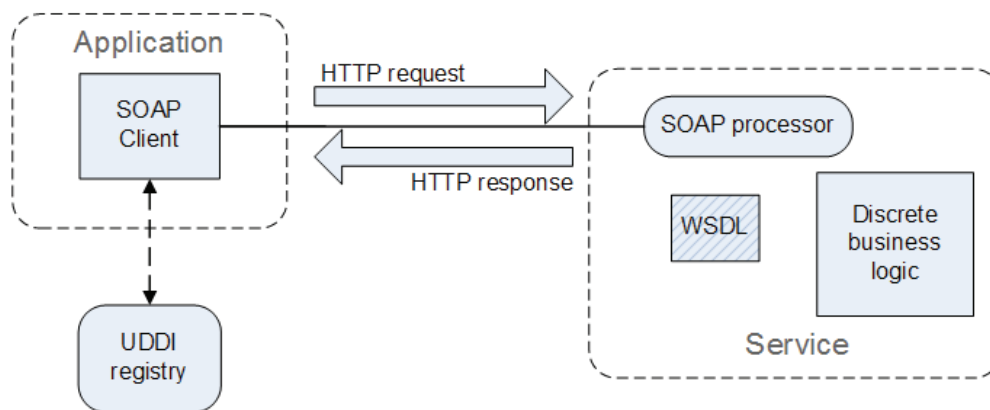


Figure 4.1: SOAP, WSDL and UDDI

The relationship can be described as follows: A client needs to locate a service somewhere on the network. It queries the UDDI registry either by name, category, identifier or specification supported. If a service matches the query, the UDDI registry returns the location of the service's WSDL document. The WSDL document describes how to contact the service. It also contains XML Schema⁵ definitions of the format of the request messages the service accepts. Finally, the client creates a SOAP message in accordance with the XML Schema found in the WSDL document and sends a request to the host where the service is.

⁵ See <http://www.w3.org/XML/Schema>

The three core technologies are described in further detail in the following sections.

4.2.1 SOAP

The SOAP specification consists of required and optional parts. The required parts of the specification define an XML format for SOAP messages. Optional parts of the SOAP specification describe how to represent program data as XML and how to use SOAP to do Remote Procedure Calls (RPC). These optional parts of the specification are used to implement RPC-style applications where a SOAP message containing a callable function, and the parameters to pass to the function, is sent from the client, and the server returns a message with the results of the executed function [8].

The last optional part of the SOAP specification defines what an HTTP message that contains a SOAP message looks like [8]. This is referred to as HTTP-binding. HTTP is the only standardized protocol for SOAP. Even though this part is optional, almost all SOAP implementations support it. SOAP does not require HTTP however, it can also be used with protocols such as File Transfer Protocol (FTP) and Simple Mail Transfer Protocol (SMTP). The advantage of using HTTP is that it is the main protocol used on the web, meaning that network infrastructures supporting it are already in place, complete with security, monitoring and load-balancing mechanisms.

4.2.2 WSDL

The official definition of WSDL from the World Wide Web Consortium⁶ is as follows [9]: “WSDL is an XML format for describing network services as a set of endpoints operating on messages containing either document-oriented or procedure-oriented information. The operations and messages are described abstractly, and then bound to a concrete network protocol and message format to define an endpoint. Related concrete endpoints are combined into abstract endpoints (services)”.

A less technical definition is that a WSDL file is an XML document that describes a set of SOAP messages and how the messages are exchanged [8]. WSDL uses a XML Schema based notation to describe the message formats. This makes WSDL programming-language neutral and standards-based, meaning that it can describe Web Services written in any kind of language and accessible from any kind of platform.

4.2.3 UDDI

Universal Discovery Description and Integration is the yellow pages of Web services [8]. The UDDI specification [10] defines a way to publish and discover information about Web Services. The main component of UDDI is the Business Registry. The Business Registry contains programmatic descriptions of businesses, the services they support and the actual Web Services themselves. The actual descriptions are XML documents defined by a set of UDDI schemas. The other big part of UDDI is the API which defines the methods a programmer needs to use to register and update the XML-descriptions. The API also defines the methods a programmer needs to use to do queries to discover existing companies and the Web Services they provide.

⁶ See <http://www.w3.org/>

4.3 Web Services and CAGISTrans

Considering that the current CAGISTrans transaction model specification application is web based, and additionally produces XML output, it seemed like a possible candidate for a reimplementation using Web Services.

4.3.1 Design

The main idea was to develop a web service that would act as a front-end for transaction model specification in the CAGISTrans system. The web service would be responsible for generating the specification files for the transaction models based on input from the web service clients. The overall architecture is shown in figure 4.2. One of the results we hoped we would get from utilizing Web Services was a thinner and less complex client. By moving a lot of the application logic to a web service, the user should see the result in the form of an easier transaction model specification process.

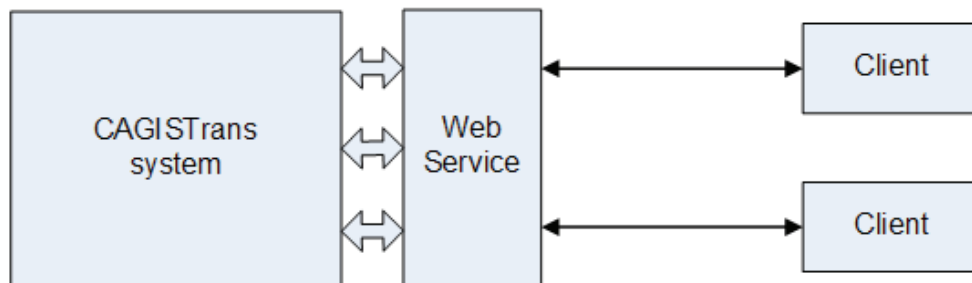


Figure 4.2: Architecture overview

The web service would make use of transaction model templates. The idea behind the templates is that there are basically four different scenarios when specifying a transaction model. Each of these scenarios are different a combination of full and relaxed ACID properties. As earlier stated the consistency and durability properties are never relaxed. The combinations are as follows:

1. *ACID* – In this scenario none of the ACID properties are relaxed, i.e. we have full atomicity and isolation.
2. *A*CID* – In this scenario the atomicity property is relaxed, but we still have full isolation
3. *ACI*D* – In this scenario the isolation property is relaxed, but we still have full atomicity. This combination of ACID properties is not recommended due to the cost of rollback [1].
4. *A*CI*D* – In this scenario both the atomicity and the isolation property are relaxed.

These different combinations of ACID properties dictate what other properties must be set in the specification of a transaction model to be executed in the CAGISTrans system. The following list gives an overview of which properties that are necessary and applicable in which of the scenarios.

- *Scenario 1* – When there is no relaxation of ACID properties, extended transaction models are not possible, meaning that a transaction model consists only of transactions and their operations.
- *Scenario 2* – When the atomicity property is relaxed, a transaction model consists of transactions and their operations, and additionally begin and commit dependencies between transactions. Also included are abortsets. Since we have full isolation, all transactions in a transaction t 's abortset must have the same root transaction as t . The last property that can be included in the transaction model is awareness mechanisms. The model can have notifications on both begin and terminate events.
- *Scenario 3* – When the isolation property is relaxed we have the same dependencies as when the atomicity property is relaxed. The abortsets of transactions are however a little different, transactions in an abortset do not have to have the same root transaction as the transaction they are abort dependent on. Relaxed isolation means that transactions with different root transactions can cooperate, and thus they can also be abort dependent on each other. Awareness mechanisms are once again included, but in addition to begin and terminate events, we also have lock and change events. The last property included in the model is the locking mechanism. If collaborative locking is applied, conflicts between operations are included in the model along with any permits and demands.
- *Scenario 4* – The transaction model generated in this scenario is almost the same as the one in scenario 3. However abort dependencies due to relaxed atomicity must also be included in the model.

4.3.2 Evaluation of the Web Services solution

Before starting the implementation process, we wanted to thoroughly evaluate our general Web Services idea, and specifically our design. During this evaluation process we became aware of some issues that might cause difficulties.

The biggest problem with our proposed solution is what we are actually trying to achieve. As mentioned earlier, Web Services are excellent for encapsulating business logic and presenting a coarse-grained service to users. Specifying a transaction model on the other hand, is not a coarse-grained process. To specify a transaction model, lots of choices must be made, and a lot of input must be given. The number of steps necessary are not easily reduced or combined into fewer higher-level steps.

The next issue is how the actual clients of the web service would work. Clients of Web Services should, as previously mentioned, be loosely coupled, meaning that they are not closely tied together with the web service logic. An example of a loosely coupled client and web service is a service that provides temperatures in specific cities, see figure 4.3. Interaction between the client and service would be as follows: The client sends a request containing the name of a city and the service returns the current temperature in the city specified. The client needs no knowledge about how the service obtains the temperature, and the service does not have any knowledge of what the client does with the returned temperature. In our case the situation is different. Similar to our example, the client does not need any information about what is happening behind the scenes at the web service. However, in order to be able to create

the transaction model specification, the web service needs to have an overview of what the client is doing. More specifically, the web service needs to keep track of all choices and input from the client during the entire specification process.

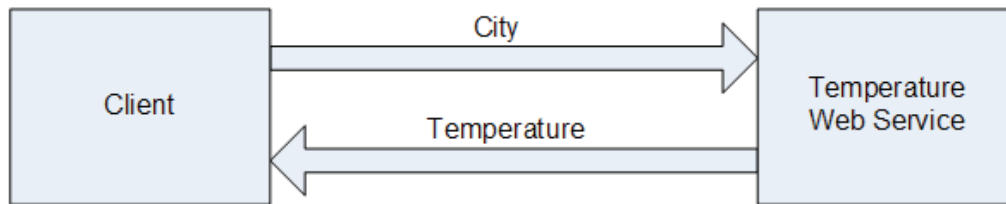


Figure 4.3: A simple temperature web service

The use of templates in our proposed solution is still a good idea. The templates can reduce the number of steps necessary to complete the specification of a transaction model. However, the use of templates does not eliminate the requirement that the web service needs to have an overview of the client. Another point about using templates is that when specifying extended transaction models, the relaxed atomicity and relaxed isolation scenario is the most useful. Thus, this scenario will most likely be the one most frequently used. Since this is also the most complex scenario, using templates will only reduce the amount of work needed to specify a transaction model in a minority of the cases.

The issues mentioned so far do not make it impossible to implement our Web Services solution. The web service would have to have a lot of methods facilitating the specification of different parts of the transaction model. Additionally we would need to create a thick client capable of maintaining some sort of state in order for the user to get an overview of the progress made in the specification process. This implementation would require quite a lot of intermediate data on both the client and server side. Now this leads to another problem, data synchronization. In order to have the same data both on the client and server side, a message must be sent to the web service every time the client performs a step in the specification process. If this message gets lost, it must be retransmitted and the client cannot continue until the message is received at the server side, and the client receives the response to the message from the server.

The web service would also require some form of session management, allowing several users to use it at once. Since a lot of intermediate data must be stored and a lot of messages sent back and forth, there is a lot of work that needs to be done by the session manager. Especially since Web Services are best suited in loosely-coupled environments, support for this kind of session management is not readily available, meaning that implementing it is far from trivial.

Although it would be possible to implement a transaction model specification application for the CAGISTrans system as a web service, we feel that there are more cons than pros for doing so. The process of specifying a transaction model is complex and involves many steps making it less suitable as a web service. Since the client would need to handle a lot of information, the web service would just be an added layer of complexity, and would not contribute to making the client less complex. Based on this we did not go ahead with an implementation effort of our Web Services design.

Chapter 5

Design and Architecture

Since we reached the conclusion that a web services solution would be less than optimal (refer to the previous chapter), a decision to implement a pure web based solution was made. We decided to use the current transaction model specification application described in [5] as our starting point, taking advantage of already existing good ideas. At the same time our main focus was addressing as many of the shortcomings of the current solution as possible. We decided to base our implementation on two main concepts, the tree control and the Model-View-Controller pattern. These concepts are discussed in the following sections.

5.1 Tree Control

Since the transaction model specifications leads to a tree structure of transactions, subtransactions and operations, we felt that the user interface should have a strong focus on this tree structure. In the original solution, only a textual representation of the tree structure is possible, and only when specifying transactions and operations. Additionally it is not possible to see the tree structure of more than one root transaction at a time. We decided to base much of our interface on a graphical representation of the tree structure. Additionally we wanted the user to be able to interact with the tree structure, e.g. selecting transactions and operations by clicking on them in the tree. This is the kind of functionality one finds in for example Windows Explorer, the Windows operating system's file management utility. Figure 5.1 shows an example of the tree control in Windows Explorer. We decided to use a similar tree control for representing transactions and operations in our application.

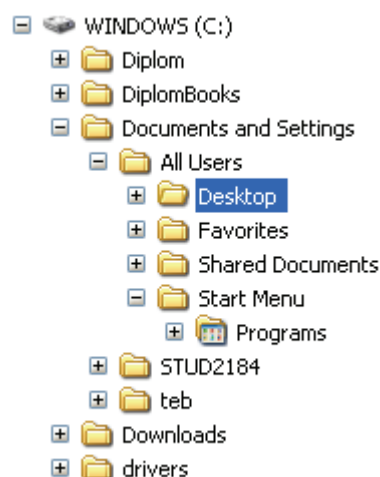


Figure 5.1: Windows Explorer tree control

5.2 MVC Pattern

Since the current solution suffers from poor design, we decided to pay close attention to this aspect of our solution. This meant trying to incorporate some best practices of software design. More specifically we wanted to base our solution, at least to a certain degree, on the Model-View-Controller (MVC) design pattern.

The Model-View-Controller pattern was created by Xerox PARC for Smalltalk-80 in the 1980s. The pattern was invented for decoupling the graphical interface of an application from the code that actually does the work [11]. It has since then become a widely used software design pattern, becoming particularly popular in web application development. MVC is the recommended model for Sun's J2EE platform⁷.

As the name implies an application is divided into three components, also called participants [12], [13]:

- *Model* – The model represents enterprise data and business rules that govern access to and updates of this data. This is where most of the processing takes place when using MVC. Databases and component objects like Enterprise JavaBeans⁸ fall under the model. The model does not apply any formatting to data, meaning that the data returned from the model is display-neutral.
- *View* – The view is the user interface the user sees and interacts with. Technically speaking, the view renders the content of a model by accessing enterprise data through the model and specifying how that data should be presented. No processing takes place in the view, however it is the view's responsibility to update its presentation when the model changes.
- *Controller* – The controller translates interactions with the view into actions to be performed by the model. Based on user interactions and the outcome of the model's actions, the controller determines what formatting to apply to the resulting data by selecting an appropriate view.

A graphical presentation of the MVC pattern is shown in figure 5.2 [12].

⁷ See <http://java.sun.com/blueprints/patterns/MVC.html>

⁸ See <http://java.sun.com/products/ejb/>

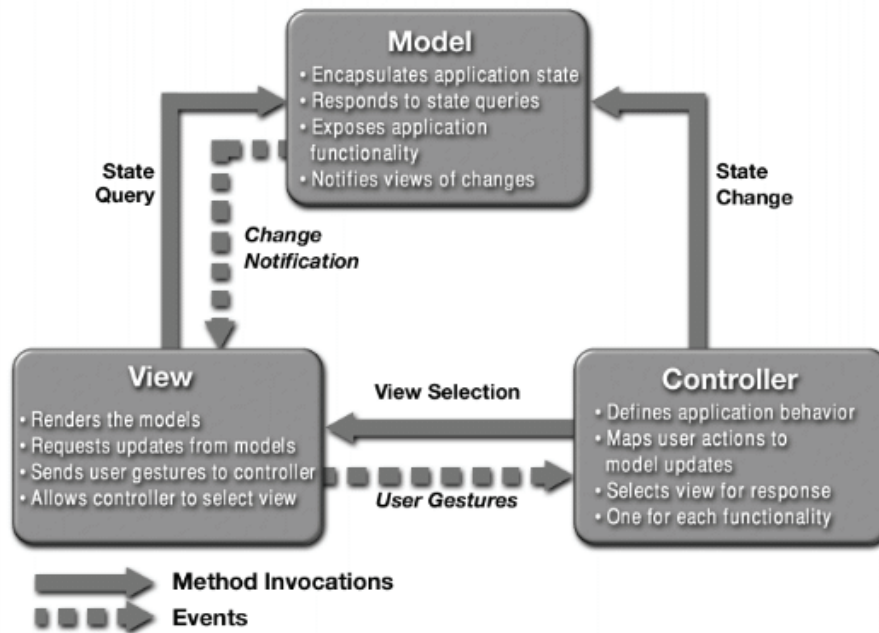


Figure 5.2: The MVC design pattern

There are several reasons to use the MVC pattern. First of all it enables multiple views that rely on the same model. This means that different types of clients can access the same data, e.g. you can have a HTML interface and a WAP interface that both rely on the same underlying data. This saves you the effort of writing and maintaining separate data models for each view. It also makes it a lot easier to add a new type of client, all that needs to be done is create the view and modify the controller.

The second advantage of using the MVC pattern is that the model can be changed without the need to change the rest of the application. An example of this is changing the underlying database from one vendor to another. If written correctly, the view does not care where its data comes from.

The drawback of the MVC controller is that it introduces quite a lot of complexity into an application. It also makes the planning effort necessary to develop an application larger, since you have to pay extra attention to the details of how different parts of the application will interact. This means that a full blown MVC implementation may be overkill for small applications, but extremely useful for large, enterprise applications.

Chapter 6

Application Technologies

Since we decided to base our implementation on the current solution for transaction model specification for the CAGISTrans system [5], Java and XML would be the main technologies at the foundation of our application. But where the current solution utilizes Java Servlets and a SQL database, we decided to implement our solution with JavaServer Pages, JavaBeans and JDOM. These three technologies are discussed in the following sections.

6.1 JavaServer Pages

Sun's definition [14]: "JavaServer Pages (JSP) is the Java 2 Platform, Enterprise Edition (J2EE) technology for building applications for generating dynamic web content, such as HTML, DHTML, XHTML, and XML. JSP technology enables the easy authoring of web pages that create dynamic content with maximum power and flexibility".

Basically what this definition says is that JSP is a technology for developing web pages that include dynamic content, as opposed to pure HTML pages which are static. A JSP page contains standard markup language elements, such as HTML tags, just like a regular web page. However, a JSP page also contains special JSP elements that allow the server to insert dynamic content in the page. When a user asks for a JSP page, the server executes the JSP elements, merges the results with the static parts of the page, and sends the dynamically composed page back to the browser, as illustrated in figure 6.1.

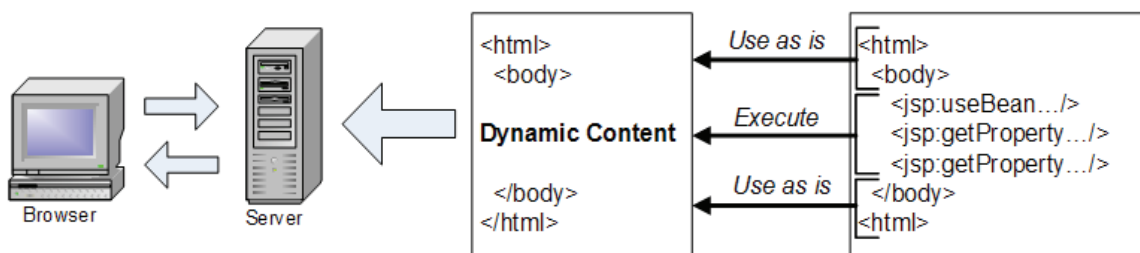


Figure 6.1: Generating dynamic content with JSP elements

The JSP elements that provide dynamic content can be divided into three categories, standard action elements, scripting elements and custom action elements. Standard action elements are basic tags, one example is the `<jsp:forward>` element, which forwards the client's browser to a specified webpage. Refer to [6] for a complete list of the JSP standard action elements.

The scripting elements are very powerful and quite easy to use. A scripting element is basically a `<% %>` tag with regular Java code inside it. The entire standard Java API is

available, allowing complex Java code to be embedded into web pages. Figure 6.2 illustrates a very simple page that uses a scripting element to display the current time and date.

```
<html>
  <body>
    Today is <%= new java.util.Date() %>
  </body>
</html>
```

Figure 6.2: A simple JSP page using a scripting element

Custom action elements are usually referred to as custom tags. Custom tags are a JSP mechanism for abstracting Java code away from the actual JSP page. A custom tag is inserted into a JSP page like a standard HTML tag, but when the user requests a page containing a custom tag, the tag's handler is called. A tag handler is simply a Java class that contains the code to be executed. To illustrate, let us look at the previous JSP example again. The outputting of the current time and date can be encapsulated by a custom tag, whose tag handler would contain the `new java.util.Date()` statement. Figure 6.3 shows the JSP page using the custom tag.

```
<html>
  <body>
    <mytags:date />
  </body>
</html>
```

Figure 6.3: A simple JSP page using a custom tag

There are some issues with the scripting elements and the custom action elements. Combining Java code and HTML with scripting elements leads to some problems. First of all a page may become very complex and hard to both understand and maintain if it contains lots of Java code intermingled with HTML. Secondly, this mixing of Java and HTML makes it very difficult to differentiate between Java development and web page development. It is desirable to let Java programmers do the Java coding and web designers do the HTML coding. This is hard to achieve with extensive use of scripting elements.

Now custom tags may seem like a good solution for separating Java code and HTML, avoiding the problems of scripting elements. The problem with custom tags however, is that the tag handlers are quite complex and difficult to implement. In addition to implementing that actual functionality of the custom tag, the implementer needs to handle the tag's lifecycle, i.e. how the tag is created, how and when it is destroyed and so on. This has turned out to be a difficult task, hindering widespread use of custom tags.

To address these JSP issues, JSP version 2.0 was released at the end of 2003. This version contains several new features which have big impact on the development of JSP pages. This is indicated by the move from 1.x versioning to 2.x. The driving force behind these new features was the desire to have JSP pages that do not contain actual Java code without the need for writing complex custom tag handlers. This is mainly achieved through the inclusion of a built-

in expression language (EL) and the JSP Standard Tag Library (JSTL). We take a closer look at these new features and their impact on the JSP page in the next section.

6.1.1 JSTL and EL

The JSTL is basically a set of custom tags. In most situations these tags include enough functionality to eliminate the need for developing your own custom tags. The tags are organized in four different libraries [15]:

- *Core* – The core tag library supports output, management of variables, conditional logic, loops, text imports, and URL manipulation.
- *XML* – The XML-processing tag library supports parsing of XML documents, selection of XML fragments, flow control based on XML, and XSLT transformations.
- *Database* – The database tag library supports database queries, updates, and transactions.
- *Formatting* – The internationalization-capable formatting tag library supports localized formatting, fine-grained control over the display of numbers and dates, and internationalization of text messages.

The JSTL makes it easy to write powerful JSP pages without writing any Java code. To make JSTL even more powerful, an expression language was created. Expressions can be used as parameters to JSTL tags. The EL is not limited to JSTL however, it can be used in all places where you previously could only use Java expressions. Basically, an expression looks like this: `${expression}`. Figure 6.4 illustrates the use of JSTL and EL, omitting some of the standard HTML code, and figure 6.5 shows the resulting web page the way it is displayed in a browser.

```
<table border="1" cellpadding="5">
  <c:forEach begin="1" end="2" varStatus="row">
    <tr>
      <c:forEach begin="1" end="3" varStatus="column">
        <td>
          row <c:out value="${row.index}"/>,
          column <c:out value="${column.index}"/>
        </td>
      </c:forEach>
    </tr>
  </c:forEach>
</table>
```

Figure 6.4: A JSP page using JSTL and EL

row 1, column 1	row 1, column 2	row 1, column 3
row 2, column 1	row 2, column 2	row 2, column 3

Figure 6.5: The resulting web page

The example is basically just a HTML table, with JSTL being used to dynamically create its contents. The JSTL and EL statements in figure 6.4 are highlighted with bold text. The `<c:forEach>` tag specifies a loop that iterates from `begin` to `end` times. The `varStatus` attribute keeps information about the loop, `row.index` and `column.index` contain the current iteration number of their corresponding loop. The `<c:out>` tag outputs its `value` attribute directly to the page. In this example the `value` attribute contains an EL expression which evaluates the contents of the `row.index` and `column.index` variables. For an in-depth look at JSTL and the EL refer to [15].

6.2 JavaBeans

JavaBeans were originally developed to be used as components in GUI builder tools. A bean would typically represent different interface elements, from simple buttons and sliders to advanced database viewers [16]. A JavaBean must adhere to a set of simple coding conventions allowing tools that have no information about a class to still use it. However, since a JavaBean is simply a Java class that follows certain coding conventions, JavaBeans are not limited to representing GUI components. JavaBeans have proven to be especially useful in web applications. JavaBeans are often used in JSP applications as the container for the dynamic content to be displayed by a web page. The beans typically represents something specific, such as a person, a product, or a shopping order [6]. JSP defines a number of standard actions for working with beans, and the JSP Expression Language accepts beans as variables in expressions. This means that a Java programmer can develop beans and a web designer with no programming experience can use them in web pages.

6.3 JDOM

To generate the two output XML files we wanted to make sure this was done in a better way than the current solution. As previously stated, the current solution “manually” creates the XML files by lots of tedious Java print statements. Even though the standard Java API contains classes for XML processing, we decided to use the JDOM [17] package to handle our XML creation. The official JDOM FAQ⁹ gives the following answer to what JDOM is: “JDOM is, quite simply, a Java representation of an XML document. JDOM provides a way to represent that document for easy and efficient reading, manipulation, and writing. It has a straightforward API, is a lightweight and fast, and is optimized for the Java programmer. It's an alternative to DOM and SAX, although it integrates well with both DOM and SAX”.

JDOM is a lot simpler to use than the standard XML classes in Java. Figure 6.6 shows very a basic XML document, figure 6.7 shows how this document is created using standard Java, and finally figure 6.8 shows how it is created with JDOM.

```
<root>This is the root</root>
```

Figure 6.6: An example XML document

⁹ See <http://www.jdom.org/docs/faq.html>

```
DocumentBuilderFactory factory =
    DocumentBuilderFactory.newInstance();
DocumentBuilder builder = factory.newDocumentBuilder();
Document doc = builder.newDocument();
Element root = doc.createElement("root");
Text text = doc.createTextNode("This is the root");
root.appendChild(text);
doc.appendChild(root);
```

Figure 6.7: Creating the example with standard Java

```
Document doc = new Document();
Element e = new Element("root");
e.setText("This is the root");
doc.addContent(e);
```

Figure 6.8: Creating the example with JDOM

From the simple example above it may not be obvious why JDOM is so much better than the approach in the current solution. Consider however that when creating a large and rather complex XML document JDOM makes it a lot easier to keep track of everything. For instance, when you create an element in JDOM, you do not need to create the end tag of the element, this is done automatically. Since JDOM abstracts away the raw XML through objects and methods, it is a lot simpler to manipulate the XML document and to create correct XML output. Using JDOM also makes it a lot easier to handle changes to the DTDs, there is no need to change lots of print statements, all that is needed is updating method calls and maybe adding or removing element objects.

Chapter 7

Implementation

This chapter gives an overview of the most interesting parts of the implementation of our solution. Section 7.1 documents how we implemented the tree control described in chapter 5.1. Section 7.2 documents how and to what extent we implemented the MVC design pattern described in chapter 5.2. In section 7.3 we describe the changes we made to the existing DTDs, and finally section 7.4 describes how we reduced the number of web pages in the specification process compared to the current solution. The complete source code for our application can be found in appendix B.

7.1 Implementing the tree control

Creating a tree control like the one found in Windows Explorer is a quite straightforward task in, for example, the Java programming language. Java has a special package that contains interfaces and classes that make the creation of a tree control quite simple. This is not the case however, when creating an HTML tree control. HTML is static, and it is not possible to create something like a tree control in pure HTML. So what about JSP, can it be used to create a tree control? JSP is great for creating dynamic web pages that respond to user interaction. However, creating a tree control in JSP is not an optimal solution due to the following problems:

- Every time the user alters the tree, e.g. by opening or closing a branch a request has to be sent to the server which generates the new view of the tree, before sending a response back to the user. This is clearly not efficient.
- Highlighting a selected node, for instance the way Desktop is highlighted in figure 5.1, is not easily achievable with just JSP and HTML. This is also the case when it comes to the layout of the tree. Creating nested levels of indentation and so on is a huge challenge using JSP and HTML.

Because of the problems with JSP, we decided to take a different approach to our tree control, and we ended up with a solution that uses JavaScript, Cascading Style Sheets (CSS) and HTML.

JavaScript is a scripting language, which means it is a lightweight programming language. It was developed by Netscape¹⁰ to add interactivity to HTML web pages. JavaScript runs on the client side, i.e. it is executed in the user's web browser without the need of server communication. All major browsers today support JavaScript. Using JavaScript you can put dynamic text into an HTML page, react to events and read and write HTML elements among other things.

¹⁰ See <http://www.netscape.com>

Cascading Style Sheets (CSS) is a simple mechanism for adding style (e.g. fonts, colors, spacing) to Web documents. The main goal behind CSS was to ease the separation of content and layout in HTML pages [18]. By storing style information in CSS stylesheets, a web page can contain only content and then use an external stylesheet to define its presentation. CSS also makes maintaining a web site consisting of several web pages easier, since all the pages can share the same stylesheet. If you for instance need to change the color of a heading, you only need to edit the stylesheet, instead of every single page.

Our tree control is made up of HTML elements whose style is determined by CSS and whose behavior is controlled by JavaScript which turns the static HTML elements into a dynamic tree control. JavaScript and CSS solves the problems that would arise by using JSP to create the tree. Since JavaScript is executed on the client side, any user interaction with the tree control will be fast, solving the efficiency issue. Furthermore CSS can determine the appearance of HTML elements down to a very detailed level. By using CSS, highlighting a node and creating the layout of the tree is no longer a difficult task. JavaScript and CSS work seamlessly together, there is full programmatically access to all CSS properties from JavaScript. This makes it possible to change CSS properties based on JavaScript event handling. This is exactly what is needed for creating the tree control. For example, when the user clicks a node in the tree, a JavaScript event is triggered causing the style of the node to be altered by changing its CSS properties. This is how we achieve the highlight effect. Similar JavaScript and CSS interaction is used to create every aspect of our dynamic tree control.

7.2 Implementing the MVC pattern

For J2EE¹¹ applications Sun defines the following strategy when implementing the MVC pattern for web-based clients such as browsers [12]: “JavaServer Pages pages to render the view, a Java Servlet as the controller and Enterprise JavaBeans components as the model”.

Since we were not creating a J2EE application, or an enterprise size application, we decided to not follow the MVC pattern 100% percent. Instead we clearly separated the model from the rest of the application, but we did not have an equally strong separation between the view and the controller. Our model was implemented as JavaBeans, where one type of bean generally refers to one entity in the application. For example, we have transaction beans that represent the transactions in the application and operation beans that represent the operations. Our data model is shown in figure 7.1.

¹¹ See <http://java.sun.com/j2ee/>

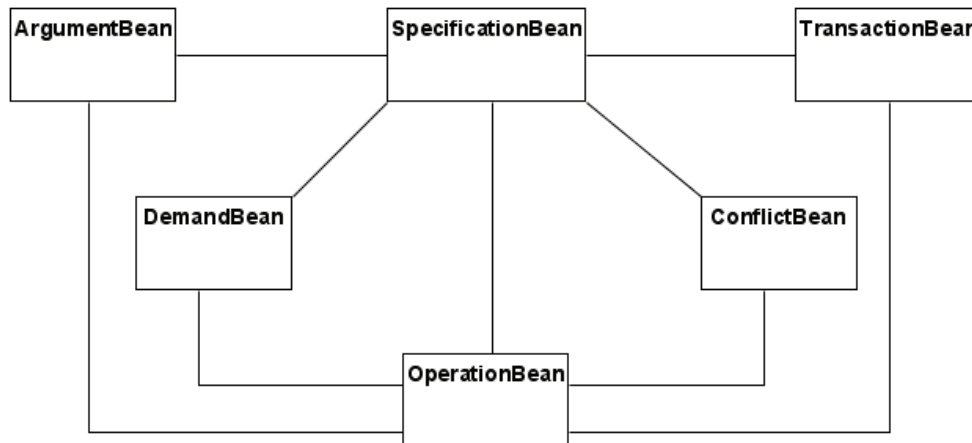


Figure 7.1: The application's data model

We also decided against using a database for storing our model during the specification process. Instead, all of the intermediate data making up our model is stored with the help of a session bean. A session bean is simply a JavaBean that has session scope. Session scope means that once the JavaBean object is created, it is available on all the pages the user accesses during her session. A session ends when the user closes her browser. The different JSP scopes are explained in detail in [6]. In our solution the `SpecificationBean` acts as the session bean, making all other JavaBeans accessible through it. This makes maintaining and accessing our data easy, since we only have to keep track of one session bean, and through that one bean we can access all of our other data. This strategy also reduces the complexity of our application, since we do not need to handle database connections and it eliminates the need for using SQL statements to query and update a database.

The view of our application was obviously implemented as JSP pages, but we also used JSP pages as controllers. Instead of having one big controller, several JSP pages are used to control the application at different parts of the specification process. The simplest JSP pages act as their own controller, whereas more complicated pages and page interaction have a dedicated JSP controller page. Even though this is not a complete separation between view and controller most of our JSP pages that are actually displayed to the user do not have any controller logic in them. Also, none of the dedicated controller JSP pages are actually displayed to the user. These pages just forward the user to the appropriate non-controller JSP pages.

7.3 XML and DTD

We decided to remove the possibility of manually creating and editing the two transaction model specification XML files from our solution. As we pointed out in chapter 3.2, there are other alternatives that are far better when it comes to manually working with the XML files. We still wanted the user to be able to view the finished files at the end of the specification process though.

The existing two DTDs that describe the transaction model specification XML files were already quite good, and we did not see the need to make any significant changes to them. In fact, we only made two modifications, both of them to the characteristics specification DTD.

The first change was modifying the dependency element to only have the type attributes commit, begin and abort. The original dependency element DTD is shown in figure 7.2 and our modified dependency element DTD is shown in figure 7.3.

```
<!ELEMENT dependency (main, dependent*)>
<!ATTLIST dependency type (begin|commit|atomabort|isolabort) "begin">
```

Figure 7.2: Original dependency element DTD

```
<!ELEMENT dependency (main, dependent*)>
<!ATTLIST dependency type (begin|commit|abort) "begin">
```

Figure 7.3: Modified dependency element DTD

The second change made was to the way demands were represented. Instead of having a nested structure of demand elements, we modified the demand element to having operation elements as children, where each operation element has a sequence number attribute. Figure 7.4 shows the original demand DTD element, followed by an xml example. Figure 7.5 shows the modified demand element DTD followed by the same xml example, but now cohering to the new DTD.

```
<!ELEMENT demand (demand?)*>
<!ATTLIST demand oid CDATA #REQUIRED>

<demand oid="4">
  <demand oid="8">
    <demand oid="2">
      </demand>
    </demand>
  </demand>
</demand>
```

Figure 7.4: Original demand element DTD and xml example

```
<!ELEMENT demand (operation)*>

<demand>
  <operation oid="4" seqNumber="1">
  <operation oid="8" seqNumber="2">
  <operation oid="2" seqNumber="3">
</demand>
```

Figure 7.5: Modified demand element DTD and xml example

The complete new versions of the DTDs can be found in appendix A.

7.4 Reducing the number of application pages

One of our goals was to reduce the number of individual web pages the user has to visit in order to complete the transaction model specification. In order to achieve this, we decided to use our template idea from the web services approach. Initially the template idea meant having 4 different starting points for the transaction model specification based on the relaxing of ACID properties. Instead of explicitly having four different templates as in the web services approach, we would have our solution choose the necessary steps needed by reacting to the user input. This is similar to the approach taken in the current solution. As we pointed out in chapter 3.2, we also felt that it should be possible to reduce the number of steps in the specification process by combining some of them. The following list sums up how we removed or combined several web pages in the current solution into fewer pages in our solution:

- *Show tree structure* – The current solution has a page for displaying the tree structure of a root transaction. This page is not necessary in our solution.
- *Transaction dependencies* – In the current solution transaction dependencies are specified on four different web pages. In our solution the number of pages has been reduced to two.
- *Awareness and locking mechanisms* – In the current solution there is one web page for awareness mechanisms and one for locking mechanisms. We have combined these pages into one.
- *Conflicts, permits and demands* – In the current solution four different web pages are used to display and specify conflicts, permits and demands. We have reduced this to two pages, one for conflicts and permits, and one for demands.

Additionally, by utilizing a graphical tree control, we eliminate the need for specific confirmation pages when adding transactions and operations. The current solution displays a confirmation page whenever a transaction or operation has been added successfully to the transaction model. In our case the newly added transaction or operation will appear in the tree control giving instant feedback to the user. This further reduces the number of pages the user needs to go through during the specification process.

Chapter 8

Application Description

Our application is described in context of specifying the transaction model used as an illustrative example throughout [4]. Figure 8.1 shows the general application flow when both the atomicity and isolation properties are relaxed. The application description is given in the chronological order of the steps in the specification process.

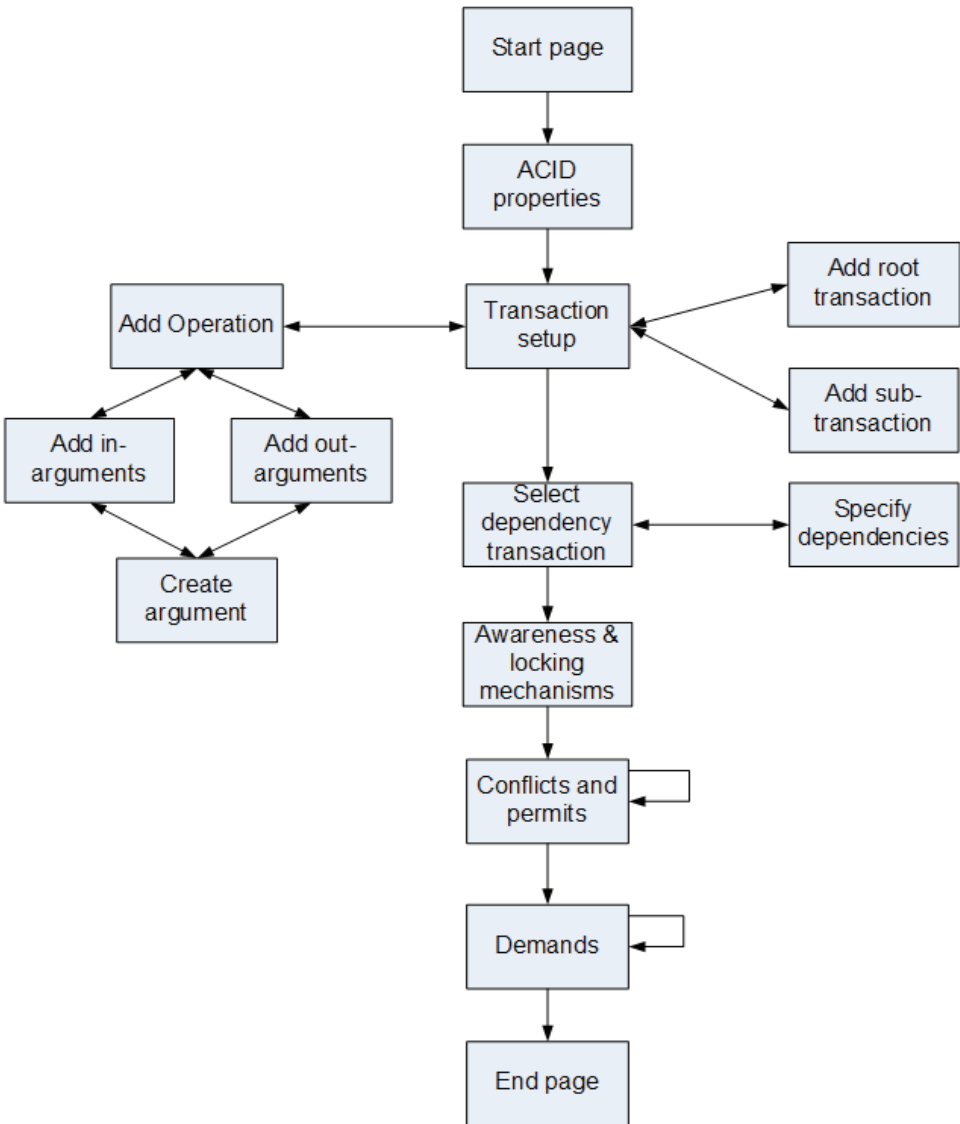
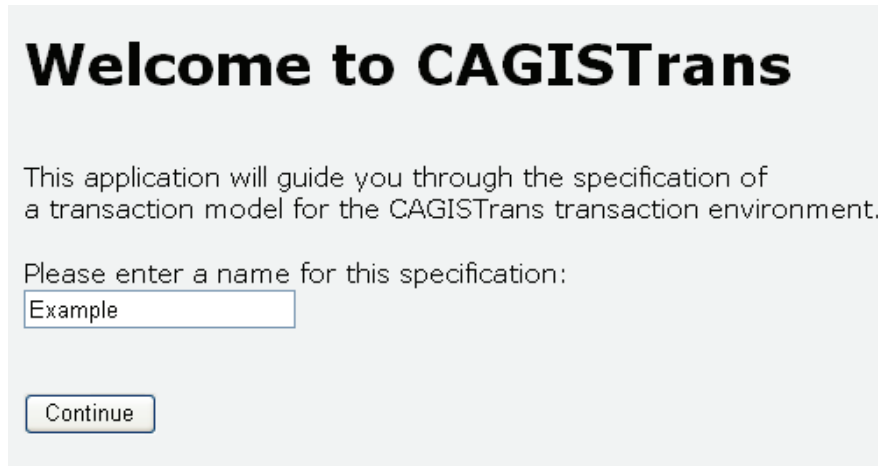


Figure 8.1: Application flow

8.1 Start page

This is the first page of the application (see figure 8.2). The user is asked to enter a name for the transaction model specification. This name will be used as the first part of the filename for the two generated XML files.



Welcome to CAGISTrans

This application will guide you through the specification of a transaction model for the CAGISTrans transaction environment.

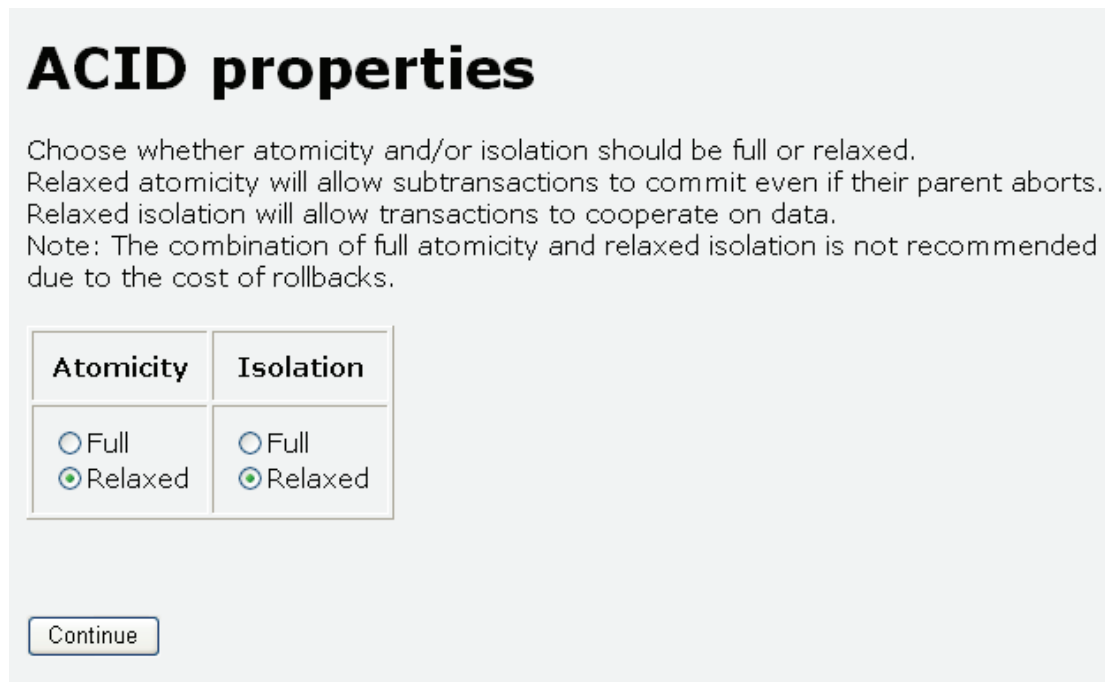
Please enter a name for this specification:

Continue

Figure 8.2: Start page

8.2 ACID properties

This is the page where the user specifies if the transaction model will have relaxed atomicity, and/or relaxed isolation (see figure 8.3). Since relaxed atomicity and isolation are the basis of extended transaction models in CAGISTrans, these options are selected by default.



ACID properties

Choose whether atomicity and/or isolation should be full or relaxed. Relaxed atomicity will allow subtransactions to commit even if their parent aborts. Relaxed isolation will allow transactions to cooperate on data. Note: The combination of full atomicity and relaxed isolation is not recommended due to the cost of rollbacks.

Atomicity	Isolation
<input type="radio"/> Full	<input type="radio"/> Full
<input checked="" type="radio"/> Relaxed	<input checked="" type="radio"/> Relaxed

Continue

Figure 8.3: ACID properties

8.3 Initial transaction overview

At this stage in the specification process, the user's only choice is to add a new main transaction to the model (see figure 8.4). This transaction will be a root transaction to which subtransactions and operations can be added.

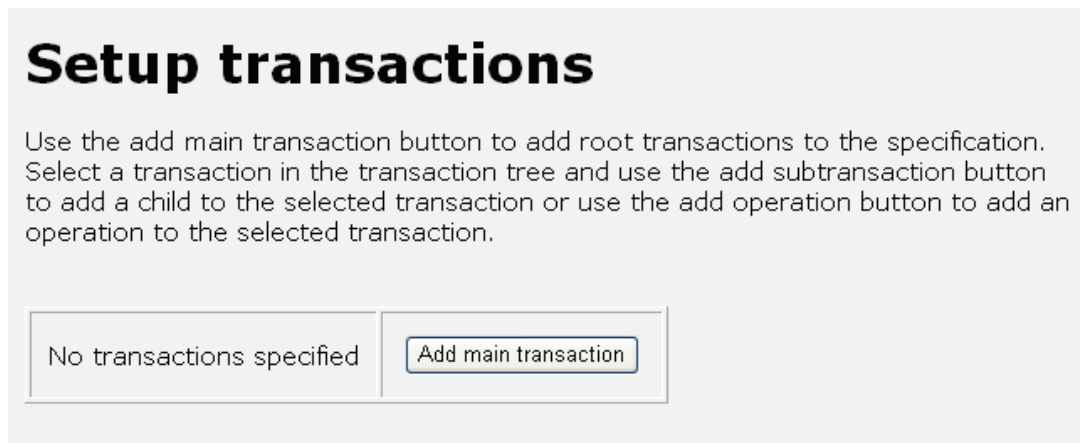


Figure 8.4: Initial transaction overview

8.4 Add main transaction

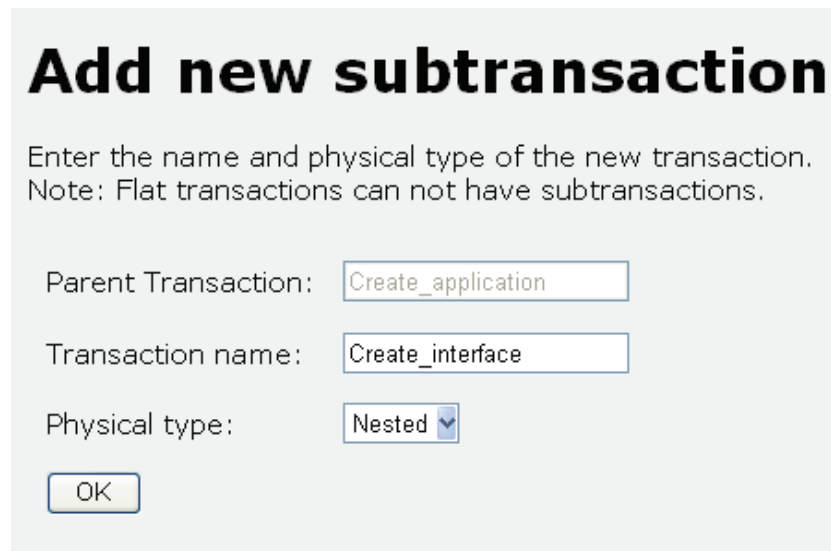
At this page the user must specify a name for the new transaction (see figure 8.5). The name does not have to be unique, all transactions are given a unique ID number hidden from the user. The user must also specify if the transaction is to be nested or flat. A flat transaction can not have any child transactions.

The screenshot shows a light gray panel with the title "Add new main transaction" in bold black text. Below the title is a paragraph of instructions: "Enter the name and physical type of the new transaction. Note: Flat transactions can not have subtransactions." Below the instructions, there are two input fields: "Transaction name:" with a text box containing "Create_application" and "Physical type:" with a dropdown menu showing "Nested" and a downward arrow. At the bottom left, there is an "OK" button.

Figure 8.5: Add main transaction

8.5 Add subtransaction

This page is basically the same as the one for adding a new root transaction. The only difference is that the subtransaction's parent transaction is displayed (see figure 8.6). The parent transaction is the one selected in the transaction tree control prior to clicking the add subtransaction button. If no transaction is selected prior to clicking the button, the error page in figure 8.21 is displayed.



Add new subtransaction

Enter the name and physical type of the new transaction.
Note: Flat transactions can not have subtransactions.

Parent Transaction:

Transaction name:

Physical type:

Figure 8.6: Add subtransaction

8.6 Add operation

This is the page where the user can specify a new operation (see figure 8.7). Initially the operation name and operation type fields are empty. The parent transaction of the operation is displayed. The parent transaction is the one selected in the transaction tree control prior to clicking the add operation button. If no transaction is selected prior to clicking the button, the error page in figure 8.22 is displayed. The user must enter a name and a type for the operation. Again the name does not have to be unique, all operations are given an unique ID number hidden from the user. This page also gives the user an overview of the operation's in- and out-arguments, initially these fields are empty. An operation must have at least one in-argument. The user can add in- and out-arguments by clicking the add iarg and add oarg buttons, which will take her to the add in-arguments and add out-arguments pages respectively.

Add new operation

Enter the name and type of the new operation.
In- and out-arguments are shown below, click the appropriate buttons to add in- and out-arguments to the operation.
Note: An operation must have at least one in-argument.

Parent Transaction:

Operation name:

Operation type:

In-arguments:

Argument	Intention
<input type="text" value="No in-arguments specified"/>	<input type="text" value="None"/>

Out-arguments:

Argument
<input type="text" value="No out-arguments specified"/>

Figure 8.7: Add operation

8.7 Add in-arguments

At this page the operation which the in-arguments are being added to is displayed along with its parent transaction (see figure 8.8). The user is presented with a list of existing arguments, if any exist, and she can select which of the arguments to add to the operation by checking the argument's checkbox. When an argument is checked, the intention selection list is enabled. The user indicates if the argument is to be browsed, incorporated or modified by selecting the intention from this list. The create new argument button allows the user to create a new argument by forwarding her to the create new argument page.

Add in-argument(s)

Select arguments from the list to add them to the operation's in-arguments. Also select the intention the operation has with the argument. New arguments can be added to the list by clicking the Create new argument button. Note: Already selected arguments appear preselected in the list.

Parent transaction:

Operation name:

Select from existing argument(s):

<input type="checkbox"/>	GUI_j	<input type="text" value="Browse"/>
<input type="checkbox"/>	P_j	<input type="text" value="Browse"/>
<input type="checkbox"/>	GUI_o	<input type="text" value="Browse"/>
<input type="checkbox"/>	GUI_c	<input type="text" value="Browse"/>
<input checked="" type="checkbox"/>	P_c	<input type="text" value="Incorporate"/>
<input type="checkbox"/>	P_o	<input type="text" value="Browse"/>

Figure 8.8: Add in-arguments

8.8 Add out-arguments

This page is basically the same as the add in-arguments page, except that there are no intentions to be specified (see figure 8.9). Intentions only apply to in-arguments.

Add out-argument(s)

Select arguments from the list to add them to the operation's out-arguments. New arguments can be added to the list by clicking the Create new argument button. Note: Already selected arguments appear preselected in the list.

Parent transaction:

Operation name:

Select existing argument(s):

- GUI_i
- P_i
- GUI_c

Figure 8.9: Add out-arguments

8.9 Create new argument

At this page the user must enter the information necessary to create an argument (see figure 8.10). An argument is a system object, like a file or a web document. The argument name must be unique, since arguments are not tied to transactions, and thus it does not make sense to have two arguments with the same name. Also, the type, workspace state, physical address and owner of the argument must be entered.

Create new argument

Enter the name, type, workspace state, address and owner of the new argument
Note: All argument names must be unique.

Argument name:

Type:

Workspace state:

Address:

Owner:

Figure 8.10: Create new argument

8.10 Selected transaction in overview

This is again the transaction overview page, but now several transactions and operations have been added to the transaction model (see figure 8.11). The create_process subtransaction is highlighted in yellow, meaning that this is the transaction subtransactions or operations will be added to if the corresponding buttons are clicked.

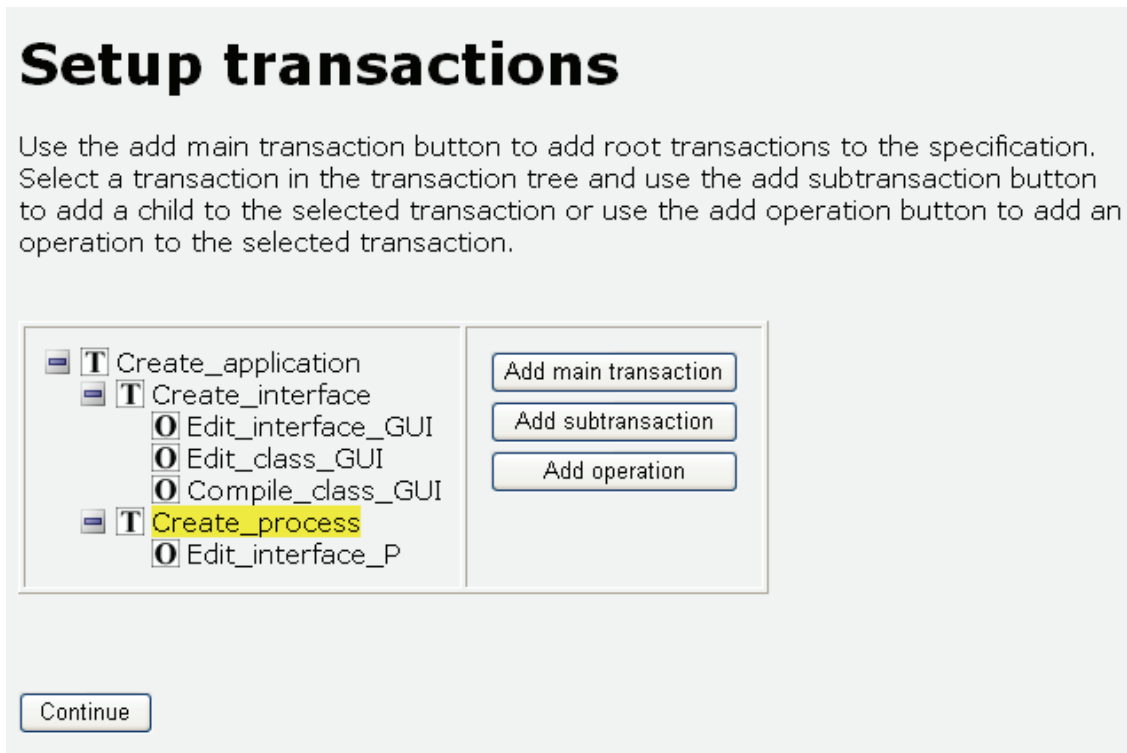


Figure 8.11: Selected transaction

8.11 Completed transaction setup

The transaction model is complete in terms of transactions, subtransactions and operations (see figure 8.12).

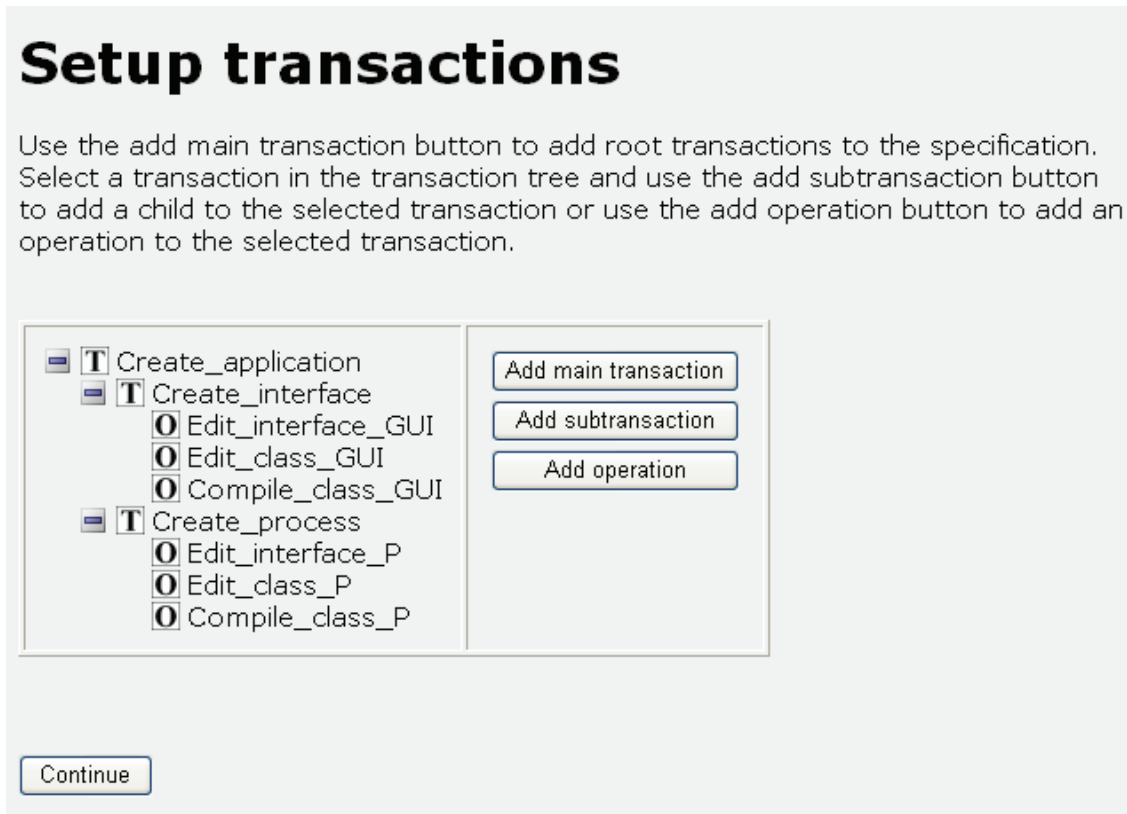


Figure 8.12: Completed transaction setup

8.12 Selected transaction for dependencies

At this page the user selects a transaction to which dependencies will be added (see figure 8.13). If the specify dependencies button is clicked prior to a transaction being selected, the error page in figure 8.23 will be displayed. The figure shows the transaction model tree partially closed, not displaying the operations in the model.

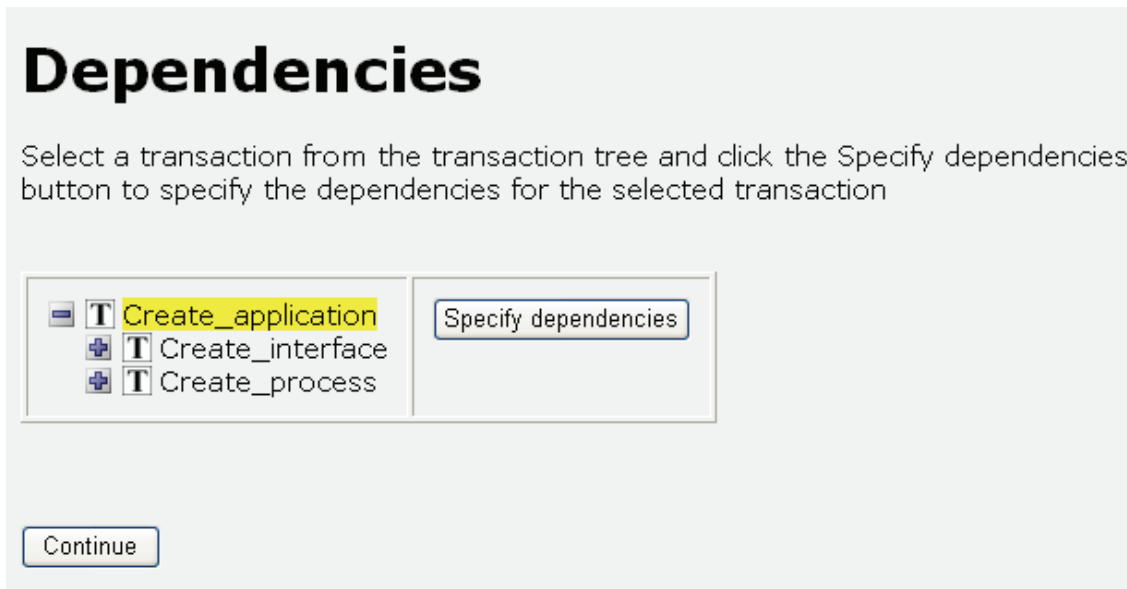


Figure 8.13: Selected dependency transaction

8.13 Dependencies specification

This is the page the user is taken to after selecting a transaction and clicking the specify dependencies button on the previous page (see figure 8.14). The selected transaction is highlighted in green. To add a dependency the user must click on the other transactions in the desired tree, e.g. to add a begin dependency, the user must select a transaction in the begin dependencies tree. Transactions that are dependent on the main transaction are highlighted in yellow. To avoid confusion, it is not possible to select the operations in any of the three trees.

Dependencies

Indicate the current transaction's dependencies by clicking on the other transactions in the corresponding transaction trees.

- Any transactions selected in the Begin dependencies tree will be begin dependent on the current transaction, i.e. they can only start after the current transaction.
- Any transactions selected in the Commit dependencies tree will be commit dependent on the current transaction, i.e. they can only commit after the current transaction.
- Any transactions selected in the Abort dependencies tree will be added to the abortset of the current transaction, i.e. if the current transaction aborts, so will the selected transactions.

Begin dependencies	Commit dependencies	Abort dependencies
<ul style="list-style-type: none"> <input type="checkbox"/> T Create_application <ul style="list-style-type: none"> <input type="checkbox"/> T Create_interface <ul style="list-style-type: none"> <input type="radio"/> Edit_interface_GUI <input type="radio"/> Edit_class_GUI <input type="radio"/> Compile_class_GUI <input type="checkbox"/> T Create_process <ul style="list-style-type: none"> <input type="radio"/> Edit_interface_P <input type="radio"/> Edit_class_P <input type="radio"/> Compile_class_P 	<ul style="list-style-type: none"> <input type="checkbox"/> T Create_application <ul style="list-style-type: none"> <input type="checkbox"/> T Create_interface <ul style="list-style-type: none"> <input type="radio"/> Edit_interface_GUI <input type="radio"/> Edit_class_GUI <input type="radio"/> Compile_class_GUI <input type="checkbox"/> T Create_process <ul style="list-style-type: none"> <input type="radio"/> Edit_interface_P <input type="radio"/> Edit_class_P <input type="radio"/> Compile_class_P 	<ul style="list-style-type: none"> <input type="checkbox"/> T Create_application <ul style="list-style-type: none"> <input type="checkbox"/> T Create_interface <ul style="list-style-type: none"> <input type="radio"/> Edit_interface_GUI <input type="radio"/> Edit_class_GUI <input type="radio"/> Compile_class_GUI <input type="checkbox"/> T Create_process <ul style="list-style-type: none"> <input type="radio"/> Edit_interface_P <input type="radio"/> Edit_class_P <input type="radio"/> Compile_class_P

OK

Figure 8.14: Dependencies specification

8.14 Awareness and locking mechanisms

In the case of relaxed isolation and atomicity this is the page displayed to the user (see figure 8.15). Any of the four awareness mechanisms can be chosen, along with either user-controlled locking or collaborative locking.

Awareness and locking mechanisms

- Select the desired awareness mechanisms, all involved parties will be notified when the events occur.
- Select the desired locking mechanism, user-controlled locking allows for negotiation if conflicts occur at runtime. Collaborative locking lets you deal with conflicts in advance with the help of permits and demands.

Awareness Mechanisms	Locking Mechanism
<input checked="" type="checkbox"/> Notify on begin <input checked="" type="checkbox"/> Notify on terminate <input checked="" type="checkbox"/> Notify on lock <input checked="" type="checkbox"/> Notify on change	<input type="radio"/> User-controlled locking <input checked="" type="radio"/> Collaborative locking

Figure 8.15: Awareness and locking mechanisms

8.15 Awareness mechanisms with full isolation

When relaxed atomicity and full isolation is chosen, this is the page displayed to the user (see figure 8.16). Only two awareness mechanisms can be chosen.



Awareness mechanisms

Select the desired awareness mechanisms, all involved parties will be notified when the events occur.

Awareness Mechanisms

- Notify on begin
- Notify on terminate

Continue

Figure 8.16: Awareness mechanisms with full isolation

8.16 Conflicting operations

If there are any conflicts, they are displayed on this page (see figure 8.17). The conflicting operations are highlighted in red, and the argument or arguments causing the conflict are displayed in the list on the right hand side. If there is more than one conflict, the number of the current conflict is displayed along with the next conflict button. Clicking this button will highlight the next pair of conflicting operations and display the argument or arguments causing this conflict. To allow conflicting operations to be performed concurrently a permit must be specified. This is done by clicking the permit button.

Conflicting operations

Conflicting operations are highlighted in red in the transaction tree.
Click the next conflict button to see the next conflict if there are more than one.
Note: Permitted conflicts are highlighted in green.

Conflict 1 of 3

<ul style="list-style-type: none"> <input type="checkbox"/> T Create_application <li style="padding-left: 20px;"><input type="checkbox"/> T Create_interface <ul style="list-style-type: none"> <input type="checkbox"/> Edit_interface_GUI <input type="checkbox"/> Edit_class_GUI <input type="checkbox"/> Compile_class_GUI <li style="padding-left: 20px;"><input type="checkbox"/> T Create_process <ul style="list-style-type: none"> <input type="checkbox"/> Edit_interface_P <input type="checkbox"/> Edit_class_P <input type="checkbox"/> Compile_class_P 	<h4 style="margin: 0;">Conflicting argument(s)</h4> <hr style="border: 0; border-top: 1px solid #ccc; margin: 5px 0;"/> <p>P_i</p>
---	--

To allow the conflicting operations to be performed, press the permit button below.

Figure 8.17: Conflicting operations

8.17 Permitted conflict

This page shows what a conflict looks like after it has been permitted (see figure 8.18). The conflicting operations are now highlighted in green, indicating the permit that has been given for them to execute concurrently.

Conflicting operations

Conflicting operations are highlighted in red in the transaction tree.
Click the next conflict button to see the next conflict if there are more than one.
Note: Permitted conflicts are highlighted in green.

Conflict 1 of 3

<ul style="list-style-type: none"> <input type="checkbox"/> T Create_application <input type="checkbox"/> T Create_interface <ul style="list-style-type: none"> <input type="checkbox"/> Edit_interface_GUI <input type="checkbox"/> Edit_class_GUI <input type="checkbox"/> Compile_class_GUI <input type="checkbox"/> T Create_process <ul style="list-style-type: none"> <input type="checkbox"/> Edit_interface_P <input type="checkbox"/> Edit_class_P <input type="checkbox"/> Compile_class_P 	<p>Conflicting argument(s)</p> <hr/> <p>P_i</p>
--	--

Figure 8.18: Permitted conflict

8.18 Demands

At this page the user can specify demands (see figure 8.19). A demand is a sequence that operations must be executed in. To create the sequence the user selects an operation in the transaction model tree and clicks the add op to demand button. The operation is then added to the new demand sequence list on the upper right hand side. These steps are repeated until all desired operations have been added to the sequence. The user can then click the sequence finished button which will move the sequence from the new demand sequence list to the existing demand sequence(s) list in the lower right hand side. This process is repeated until all demand sequences to be included in the transaction model have been specified.

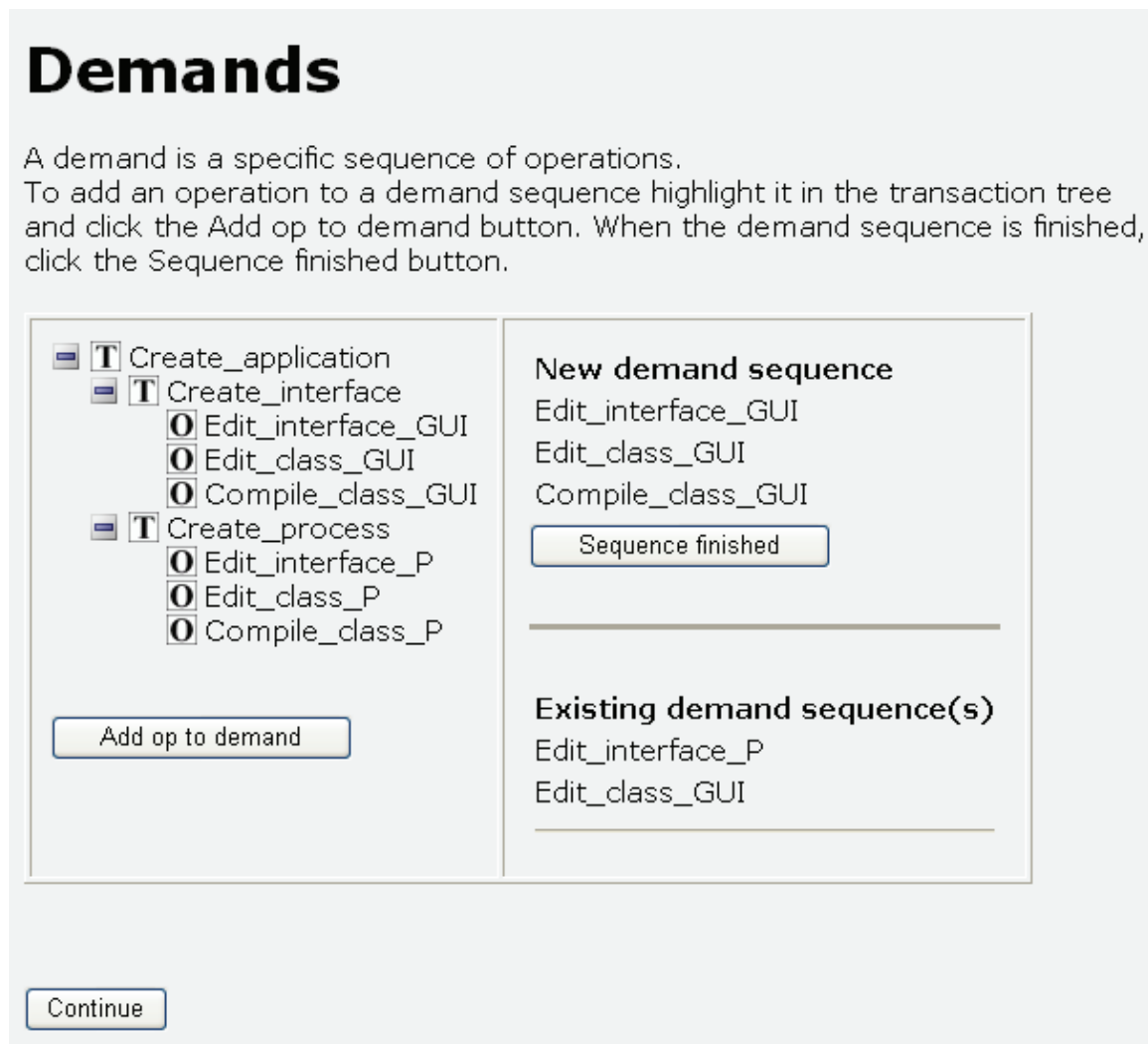


Figure 8.19. Demands

8.19 Completed transaction model

This is the last page displayed to the user (see figure 8.20). The page has links to the two resulting XML files that specify the transaction model. These files can either be viewed in the browser¹² or they can be downloaded and saved at a convenient location. Two example XML files can be found in appendix A. These files are the actual results of the transaction model specification process highlighted by the screenshots in this chapter.

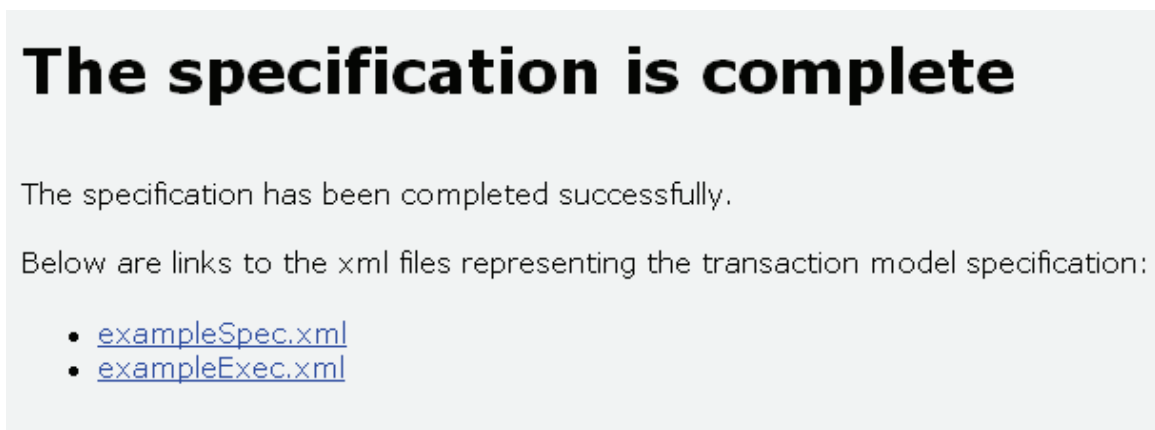


Figure 8.20: Completed transaction model

8.20 Error pages

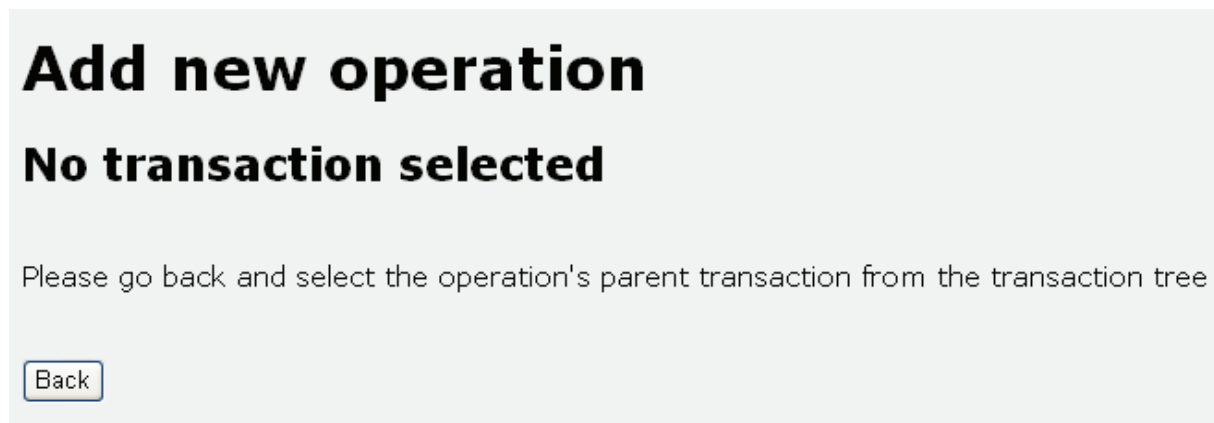


Figure 8.21: No transaction selected #1

¹² Internet Explorer and Firefox both support viewing XML, whereas Opera does not.

Add new subtransaction

No parent transaction selected

Please go back and select a parent transaction from the transaction tree

[Back](#)

Figure 8.22: No transaction selected #2

Dependencies

No transaction selected

Please go back and select a transaction to add dependencies to from the transaction tree

[Back](#)

Figure 8.23: No transaction selected #3

Chapter 9

Conclusions and Future Work

This chapter sums up this thesis and the work behind it. We start in section 9.1 with a description of the contribution of the thesis. Section 9.2 presents our concluding remarks, before we describe our future work in section 9.3.

9.1 Contribution

The main contribution of this thesis is the development of an improved transaction model specification application. The application is fully functional and can be used to specify transaction models for the CAGISTrans system. The application is web-based and thus requires no downloads or installation by the user, the user only needs the URL that identifies the server where the application is running. The application has been tested in three major browsers, Internet Explorer¹³, Firefox¹⁴ and Opera¹⁵, and it works equally well in all of them. Since both Firefox and Opera provide editions for Linux and Mac in addition to a Windows version, our application is cross-platform compatible. We have also tried to make the application code as understandable and maintainable as possible by using well-suited technologies. This eases the implementation of any future updates, either to the application itself or the XML files representing transaction models that the application creates.

Additionally, we have in this thesis provided a thorough review of the original application for specifying transaction models, trying to highlight both its advantages and its shortcomings. Finally, we have given a detailed discussion of Web Services and in particular Web Services in the context of the CAGISTrans framework, studying the possibility of using Web Services in the process of specifying transaction models.

9.2 Conclusions

A thorough evaluation of the current solution for specifying transaction models for the CAGISTrans system uncovered several weaknesses it possesses. First of all there is a lack of a good overview of the transaction model during the specification process. Secondly, the specification process itself is too complicated and long. And finally, the technical aspects of the solution could use much improving. Specifically, the application has one very large class that handles almost everything. This is not a good way to implement a solution. The application also uses Java Servlets to generate its HTML, which leads to complex and hard to maintain code. Finally, the application generates XML by using hard coded print statements. This can very easily lead to erroneous XML and also makes it very difficult to change the structure of the XML files.

¹³ <http://www.microsoft.com/windows/ie/>

¹⁴ <http://www.mozilla.org/products/firefox/>

¹⁵ <http://www.opera.com/>

After evaluating the current solution we investigated the possibility of implementing a new solution for specifying transaction models using Web Services. The main idea behind this was to move as much as possible of the application logic to the web service, thus creating a thinner client and a simpler transaction model specification process. After carefully evaluating this idea we reached the conclusion that the Web Service architecture is not well suited to what we were trying to achieve. There were several reasons that lead us to this conclusion, but the main issue is that specifying a transaction model is a complex process which the client needs to have a complete overview of at all times. What this means is that data on the web service would basically need to be duplicated at the client side. The end result of this is that implementing a transaction model specification solution as a Web Service would complicate the solution and it would not simplify the process of specifying a transaction model.

Based on our evaluation of the current solution and the Web Services idea, we decided to implement a web based solution similar to the existing one, focusing on improving the weaknesses uncovered in our evaluation. In order to deal with the problem of a lack of overview during the transaction model specification process we built much of our interface around a graphical tree control, similar to the one found in Windows Explorer for instance. We also put a lot of effort into reducing the number of pages the user has to visit to complete the specification. This was mainly achieved by combining several pages in the original solution into fewer pages in our solution. Additionally our tree control made some of the pages in the original solution superfluous.

We were also very meticulous when it came to the technology aspect of our solution, making sure we avoided the problems in the original solution. We based our implementation on the Model-View-Controller pattern, which helped us separate data from the rest of the application in a very clear and precise way. We also used JavaServer Pages 2.0 instead of Servlets to implement the user interface, thus cleanly separating HTML from Java code. Finally we used the JDOM package to create XML files. This makes the XML creation less error-prone and also makes it a lot easier to incorporate changes to the XML structure into our application. The result of using these technologies is that our application is easier to understand, maintain and update than the original solution.

9.3 Future work

There are still a few things that can be done to improve our transaction model specification application. These are first and foremost issues relating to the usability part of the application, since all of the necessary functionality to create a full fledged transaction model already is in place. First of all, there is currently very limited support for undoing actions. More specifically, it is not possible to delete transactions or operations from the model once they have been added. Also, it is not possible to remove arguments from operations once they have been added. Similarly, a permit can not be removed after it is given to conflicting operations and demand sequences can not be removed once created.

A subtransaction will always be begin dependent on its parent, i.e. it can only start after its parent has started. Currently this dependency is not automatically specified by the application, leaving it up to the user to specify it.

There should be a way for the user to abort the specification process, and return to the start page of the application. This functionality would typically be implemented as a button present

on every page. Similarly, on the last page of the application the user should be given the opportunity to create a new transaction model, by starting the specification process all over again. Currently, these options do not exist.

There is also the issue of form validation, i.e. controlling all of the input the user gives the application. Currently there is limited form validation implemented, for example most of the time it is not checked that the user has actually entered a value in all input fields. This makes it possible to, for instance, not enter a specification name or create a new transaction with no name. This is clearly not desirable, and should be prevented by adding form validation everywhere.

References

- [1] H. Ramampiaro, *CAGISTrans: Adaptable Transactional Support for Cooperative Work*, Norwegian University of Science and Technology (NTNU), Dr.Ing. thesis, NTNU 2001:94, IDI-nr. 6/2001.
- [2] P. A. Bernstein, V. Hadzilacos, N. Goodman, *Concurrency Control and Recovery in Database Systems*, Addison-Wesley, 1987.
- [3] J. E. B. Moss, *Nested Transactions and reliable computing*, In Proceedings of the 2nd IEEE Symposium on Reliability in Distributed Software and Database Systems, 1982.
- [4] H. Ramampiaro, M. Nygård, *CAGISTrans: Providing adaptable transactional support for cooperative work - An Extended Treatment*, Information Technology & Management (ITM) Journal, 5, 23-64, 2004.
- [5] P.A. Sætre, *Human Interface and the CAGISTrans Framework*, Norwegian University of Science and Technology (NTNU), Master thesis, 2001.
- [6] H. Bergsten, *JavaServer Pages, 3rd edition*, O'Reilly, December 1, 2003.
- [7] D. Chappel, T. Jewell, *Java Web Services*, O'Reilly, March, 2002.
- [8] R. Wolter, *XML Web Service Basics*, Microsoft MSDN Library, December 2001.
- [9] E. Christensen, F. Curbera, G. Meredith, S. Weerawarana, *Web Services Description Language (WSDL) 1.1*, World Wide Web Consortium, March 2001.
- [10] UDDI Spec Technical Committee, *UDDI Version 3.0.2*, OASIS, October, 2004.
- [11] J. Bergin, *Building Graphical User Interfaces with the MVC Pattern*, <http://csis.pace.edu/~bergin/mvc/mvcgui.html>, August 19, 2003.
- [12] Sun Microsystems, Inc, *Model-View-Controller*, Java Blueprints, <http://java.sun.com/blueprints/patterns/MVC-detailed.html>
- [13] B. Kotek, *MVC Design*, <http://builder.com.com/5100-6386-1049862.html>, October, 2002.
- [14] Sun Microsystems, Inc., *JavaServer Pages 2.0 Specification (Final Release)*, November 24, 2003.
- [15] S. Bayern, *JSTL in Action*, Manning, July, 2002.
- [16] Sun Microsystems, Inc., *JavaBeans Specification (version 1.01)*, July, 1997.
- [17] JDOM, <http://www.jdom.org/>
- [18] World Wide Consortium, *Cascading Style Sheets*, <http://www.w3.org/Style/CSS/>.

Appendix A

DTD and XML

Section A.1 contains the two DTD files that define the two XML files produced by our application. Section A.2 contains two example XML files. These files specify a complete transaction model and are the actual results of the specification process described in chapter 8.

A.1 DTD

Transspec.dtd

```
<!-- Characteristics (root) element -->
<!ELEMENT characteristics (properties, structure, constraints, policies)>
<!ATTLIST characteristics name CDATA #REQUIRED>

<!-- Properties element -->
<!ELEMENT properties (atom, cons, isol, dur)>
<!ELEMENT atom EMPTY>
<!ATTLIST atom value (full|relaxed) "full">
<!ELEMENT cons EMPTY>
<!ATTLIST cons value CDATA #FIXED "full">
<!ELEMENT isol EMPTY>
<!ATTLIST isol value (full|relaxed) "full">
<!ELEMENT dur EMPTY>
<!ATTLIST dur value CDATA #FIXED "full">

<!-- Structure element -->
<!ELEMENT structure (physical, dependencies)>
<!ELEMENT physical EMPTY>
<!ATTLIST physical type (flat|nested) "flat">
<!ELEMENT dependencies (dependency*)>
<!ELEMENT dependency (main, dependent*)>
<!ATTLIST dependency type (begin|commit|abort) "begin">
<!ELEMENT main EMPTY>
<!ATTLIST main transaction CDATA #REQUIRED>
<!ELEMENT dependent EMPTY>
<!ATTLIST dependent transaction CDATA #REQUIRED>

<!-- Constraints element -->
<!ELEMENT constraints (applevel|dblevel)>
<!ELEMENT applevel (user|collaborative)>
<!ELEMENT user EMPTY>
<!ELEMENT collaborative (conflicts?, permits?, demands?)>
<!ELEMENT conflicts (conflict*)>
<!ELEMENT conflict (operation*)>
<!ELEMENT operation EMPTY>
<!ATTLIST operation oid CDATA #REQUIRED seqNumber CDATA #IMPLIED>
<!ELEMENT permits (permit*)>
<!ELEMENT permit (operation*)>
<!ELEMENT demands (demand*)>
<!ELEMENT demand (operation*)>
<!ELEMENT dblevel EMPTY>

<!-- Policies element -->
<!ELEMENT policies (rules?, mechanism)>
<!ELEMENT rules (awareness*)>
<!ELEMENT awareness EMPTY>
<!ATTLIST awareness type (begin|terminate|lock|change) "begin">
<!ELEMENT mechanism EMPTY>
<!ATTLIST mechanism type (locking|to|none) "locking">
```

Transexec.dtd

```
<!ELEMENT transactions (transaction*)>

<!ELEMENT transaction (begin, operations, manager_ops*, transaction*, end)>
<!ATTLIST transaction trid CDATA #REQUIRED parent CDATA #IMPLIED>

<!ELEMENT begin EMPTY>

<!-- Operations -->
<!ELEMENT operations (operation*, manager_ops?)>
<!ELEMENT operation (iargs, oargs?, browse_set?)>
<!ATTLIST operation oid CDATA #REQUIRED type CDATA #REQUIRED>
<!ELEMENT iargs (arg)+>
<!ELEMENT oargs (arg)+>
<!ELEMENT browse_set (arg)+>
<!ELEMENT arg EMPTY>
<!ATTLIST arg name CDATA #REQUIRED intent CDATA #IMPLIED>

<!-- Management operations -->
<!ELEMENT manager_ops (resume|suspend|delegate)+>
<!ELEMENT resume EMPTY>
<!ELEMENT suspend EMPTY>
<!ELEMENT delegate (op)+>
<!ATTLIST delegate target CDATA #REQUIRED>
<!ELEMENT op EMPTY>
<!ATTLIST op pointer CDATA #REQUIRED>

<!ELEMENT end EMPTY>
<!ATTLIST end type (commit|abort) "commit">
```

A.2 Example XML

exampleSpec.xml

```

<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE characteristics SYSTEM "Transspec.dtd">

<characteristics name="example">
  <properties>
    <atom value="relaxed" />
    <cons value="full" />
    <isol value="relaxed" />
    <dur value="full" />
  </properties>
  <structure>
    <physical type="nested" />
    <dependencies>
      <dependency type="commit">
        <main transaction="3" />
        <dependent transaction="1" />
        <dependent transaction="2" />
      </dependency>
      <dependency type="abort">
        <main transaction="3" />
        <dependent transaction="2" />
      </dependency>
      <dependency type="abort">
        <main transaction="2" />
        <dependent transaction="3" />
      </dependency>
      <dependency type="begin">
        <main transaction="1" />
        <dependent transaction="2" />
        <dependent transaction="3" />
      </dependency>
    </dependencies>
  </structure>
  <constraints>
    <applevel>
      <collaborative>
        <conflicts>
          <conflict>
            <operation oid="5" />
            <operation oid="7" />
          </conflict>
          <conflict>
            <operation oid="5" />
            <operation oid="6" />
          </conflict>
          <conflict>
            <operation oid="9" />
            <operation oid="8" />
          </conflict>
        </conflicts>
        <permits>
          <permit>
            <operation oid="5" />
            <operation oid="7" />
          </permit>
        </permits>
        <demands>
          <demand>
            <operation oid="7" seqNumber="1" />
            <operation oid="5" seqNumber="2" />
          </demand>
        </demands>
      </collaborative>
    </applevel>
  </constraints>
  <policies>
    <rules>
      <awareness type="begin" />
      <awareness type="terminate" />
    </rules>
  </policies>
</characteristics>

```

```

    <awareness type="lock" />
    <awareness type="change" />
  </rules>
  <mechanism type="locking" />
</policies>
</characteristics>

```

exampleExec.xml

```

<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE transactions SYSTEM "Transexec.dtd">

<transactions>
  <transaction trid="1">
    <begin />
    <operations />
    <transaction trid="2">
      <begin />
      <operations>
        <operation oid="4" type="Edit_interface">
          <iargs>
            <arg name="GUI_i" intent="modify" />
          </iargs>
          <oargs>
            <arg name="GUI_i" />
          </oargs>
        </operation>
        <operation oid="5" type="Edit_class">
          <iargs>
            <arg name="GUI_i" intent="browse" />
            <arg name="P_i" intent="incorporate" />
            <arg name="GUI_c" intent="modify" />
          </iargs>
          <oargs>
            <arg name="GUI_c" />
          </oargs>
        </operation>
        <operation oid="6" type="Compile_class">
          <iargs>
            <arg name="GUI_c" intent="incorporate" />
            <arg name="P_o" intent="browse" />
          </iargs>
          <oargs>
            <arg name="GUI_o" />
          </oargs>
        </operation>
      </operations>
      <end type="commit" />
    </transaction>
    <transaction trid="3">
      <begin />
      <operations>
        <operation oid="7" type="Edit_interface">
          <iargs>
            <arg name="P_i" intent="modify" />
          </iargs>
          <oargs>
            <arg name="P_i" />
          </oargs>
        </operation>
        <operation oid="8" type="Edit_class">
          <iargs>
            <arg name="P_i" intent="browse" />
            <arg name="P_c" intent="modify" />
          </iargs>
          <oargs>
            <arg name="P_c" />
          </oargs>
        </operation>
        <operation oid="9" type="Compile_class">
          <iargs>
            <arg name="P_c" intent="incorporate" />
          </iargs>

```

```
        <oargs>
          <arg name="P_o" />
        </oargs>
      </operation>
    </operations>
  <end type="commit" />
</transaction>
<end type="commit" />
</transaction>
</transactions>
```

Appendix B

Source code

This appendix includes all of the source code for our application. The source code is listed in a separate section for each file type. Files are listed alphabetically in each section. Section B.1 contains the CSS stylesheet used by all the JSP pages displayed to the user. Section B.2 contains the 4 JavaScript files, utilized by Conflicts.jsp, Demands.jsp, Dependencies.jsp and TransactionOverview.jsp, respectively. Section B.3 contains the JSP pages and finally section B.4 contains the JavaBeans classes and the general Java classes used by the beans.

B.1 Cascading Style Sheets files

cagisStyle.css

```
body{
    font: 11pt Verdana,sans-serif;

    /*Sets the background color to grey
    This is done when capturing screen-shots*/
    background-color: #F0F0F0;
}
.branch{
    cursor: pointer;
    cursor: hand;
    display: block;
}
.leaf{
    display: block;
    margin-left: 20px;
}
.eleaf{
    display: none;
    margin-left: 20px;
}
.openclose{
    padding-right: 5px;
    vertical-align: middle;
}
```

B.2 JavaScript files

confTree.js

```
function tree(){
    this.branches = new Array();
    this.add = addBranch;
    this.write = writeTree;
}

function branch(id, text){
    this.id = id;
    this.text = text;
    this.write = writeBranch;
    this.add = addLeaf;
    this.leaves = new Array();
}

function leaf(id, text){
    this.id = id;
    this.text = text;
    this.write = writeLeaf;
}

function writeTree(){
    var treeString = '';
    var numBranches = this.branches.length;
    for(var i=0;i<numBranches;i++)
        treeString += this.branches[i].write();
    document.write(treeString);
}

function addBranch(branch){
    this.branches[this.branches.length] = branch;
}

function writeBranch(){
    var branchString = '';
    branchString += '';
    branchString += '<span id="' + this.id + '">';
    branchString += this.text;
    branchString += '</span>';
    branchString += '<span class="leaf">';
    var numLeaves = this.leaves.length;
    for(var j=0;j<numLeaves;j++)
        branchString += this.leaves[j].write();
    branchString += '</span>';
    return branchString;
}

function addLeaf(leaf){
    this.leaves[this.leaves.length] = leaf;
}

function writeLeaf(){
    var leafString = '<span style="margin-left: 21px;">';
    leafString += '';
    leafString += '<span id="' + this.id + '">';
    leafString += this.text;
    leafString += '</span>';
    leafString += '<br>';
    leafString += '</span>';
    return leafString;
}

function highlightConflict(id1, id2, hcolor) {
    var spans = document.getElementsByTagName("span");
    for(var i=0; i<spans.length; i++)
        spans[i].style.backgroundColor = "transparent";
    document.getElementById(id1).style.backgroundColor = hcolor;
    document.getElementById(id2).style.backgroundColor = hcolor;
}
```


demandTree.js

```
var curHighlighted;

function tree(){
    this.branches = new Array();
    this.add = addBranch;
    this.write = writeTree;
}

function branch(id, text){
    this.id = id;
    this.text = text;
    this.write = writeBranch;
    this.add = addLeaf;
    this.leaves = new Array();
}

function leaf(id, text){
    this.id = id;
    this.text = text;
    this.write = writeLeaf;
}

function writeTree(){
    var treeString = '';
    var numBranches = this.branches.length;
    for(var i=0;i<numBranches;i++){
        treeString += this.branches[i].write();
    }
    document.write(treeString);
}

function addBranch(branch){
    this.branches[this.branches.length] = branch;
}

function writeBranch(){
    var branchString = '';
    branchString += '';
    branchString += '<span id="' + this.id + '">';
    branchString += this.text;
    branchString += '</span>';
    branchString += '<span class="leaf">';
    var numLeaves = this.leaves.length;
    for(var j=0;j<numLeaves;j++){
        branchString += this.leaves[j].write();
    }
    branchString += '</span>';
    return branchString;
}

function addLeaf(leaf){
    this.leaves[this.leaves.length] = leaf;
}

function writeLeaf(){
    var leafString = '<span style="margin-left: 21px;">';
    leafString += '';
    leafString += '<span style="cursor: pointer;cursor: hand;"
        onclick="highlight(\'' + this.id + '\')" id="' + this.id + '">';
    leafString += this.text;
    leafString += '</span>';
    leafString += '<br>';
    leafString += '</span>';
    return leafString;
}

function highlight(selectedId) {
    if(curHighlighted != null) {
        var branchElement = document.getElementById(curHighlighted.id).style;
        branchElement.backgroundColor = "transparent";
    }
    curHighlighted = document.getElementById(selectedId);
}
```

```

    var selectedStyle = curHighlighted.style;
    selectedStyle.backgroundColor = "yellow";
}

function addOpToSeq() {
    var URL = 'DemandController.jsp';
    if(curHighlighted != null) {
        URL += '?selectedOp=';
        URL += curHighlighted.id;
    }
    location = URL;
}

```

depTree.js

```

var mainHighlighted;

function tree(){
    this.branches = new Array();
    this.add = addBranch;
    this.write = writeTree;
}

function branch(id, text){
    this.id = id;
    this.text = text;
    this.write = writeBranch;
    this.add = addLeaf;
    this.leaves = new Array();
}

function leaf(id, text){
    this.id = id;
    this.text = text;
    this.write = writeLeaf;
}

function writeTree(){
    var treeString = '';
    var numBranches = this.branches.length;
    for(var i=0;i<numBranches;i++){
        treeString += this.branches[i].write();
    }
    document.write(treeString);
}

function addBranch(branch){
    this.branches[this.branches.length] = branch;
}

function writeBranch(){
    var branchString = '';
    branchString += '';
    branchString += '<span style="cursor: pointer;cursor: hand;"';
    branchString += 'id="' + this.id + '" onclick="highlight(\' ' + this.id + '\')">';
    branchString += this.text;
    branchString += '</span>';
    branchString += '<span class="leaf">';
    var numLeaves = this.leaves.length;
    for(var j=0;j<numLeaves;j++){
        branchString += this.leaves[j].write();
    }
    branchString += '</span>';
    return branchString;
}

function addLeaf(leaf){
    this.leaves[this.leaves.length] = leaf;
}

function writeLeaf(){
    var leafString = '<span style="margin-left: 21px;">';
    leafString += '';
    leafString += this.text;
    leafString += '<br>';
}

```

```

    leafString += '</span>';
    return leafString;
}

function setMainHighlight(id) {
    mainHighlighted = id;
    document.getElementById('b'+id).style.backgroundColor = "#00FF00";
    document.getElementById('c'+id).style.backgroundColor = "#00FF00";
    document.getElementById('a'+id).style.backgroundColor = "#00FF00";
    document.getElementById('b'+id).style.fontWeight = "bold";
    document.getElementById('c'+id).style.fontWeight = "bold";
    document.getElementById('a'+id).style.fontWeight = "bold";
}

function highlight(selectedId) {
    var selTree = selectedId.charAt(0);
    if(selectedId != (selTree+mainHighlighted)) {
        var branchElement = document.getElementById(selectedId).style;
        if(branchElement.backgroundColor == "#ffff00" ||
           branchElement.backgroundColor == "rgb(255, 255, 0)" )
            branchElement.backgroundColor = "transparent";
        else
            branchElement.backgroundColor = "#ffff00";
    }
}

function getSelected() {
    var URL = '';
    URL += "DepController.jsp";

    URL += "?main=";
    URL += mainHighlighted;

    var cnt = 0;
    for(var i=0; i<document.all.length; i++) {
        var tmpElement = document.all[i];
        tmpElementStyle = tmpElement.style;
        if(tmpElementStyle.backgroundColor == "#ffff00") {
            var tmpId = tmpElement.getAttribute("id");
            if(tmpId.charAt(0) == "b") {
                URL += "&begin=";
                URL += tmpId.substring(1);
            }
            if(tmpId.charAt(0) == "c") {
                URL += "&commit=";
                URL += tmpId.substring(1);
            }
            if(tmpId.charAt(0) == "a") {
                URL += "&abort=";
                URL += tmpId.substring(1);
            }
        }
        cnt++;
    }
    location = URL;
}

```

tree.js

```

var openImg = new Image();
openImg.src = "Images/tree_open.gif";
var closedImg = new Image();
closedImg.src = "Images/tree_closed.gif";
var curHighlighted;

function tree(){
    this.branches = new Array();
    this.add = addBranch;
    this.write = writeTree;
}

function branch(id, text){

```

```

    this.id = id;
    this.text = text;
    this.write = writeBranch;
    this.add = addLeaf;
    this.leaves = new Array();
}

function leaf(id, text){
    this.id = id;
    this.text = text;
    this.write = writeLeaf;
}

function writeTree(){
    var treeString = '';
    var numBranches = this.branches.length;
    for(var i=0;i<numBranches;i++)
        treeString += this.branches[i].write();
    document.write(treeString);
}

function addBranch(branch){
    this.branches[this.branches.length] = branch;
}

function writeBranch(){
    var branchString = '<span class="branch"
        onClick="showBranch(\'L' + this.id + '\')"'';
    branchString += '>';
    branchString += '';
    branchString += '<span id="' + this.id + '"
        onclick="highlight(\'' + this.id + '\')">';
    branchString += this.text;
    branchString += '</span></span>';
    branchString += '<span class="eleaf" id="L';
    branchString += this.id + '">';
    var numLeaves = this.leaves.length;
    for(var j=0;j<numLeaves;j++)
        branchString += this.leaves[j].write();
    branchString += '</span>';
    return branchString;
}

function addLeaf(leaf){
    this.leaves[this.leaves.length] = leaf;
}

function writeLeaf(){
    var leafString = '<span style="margin-left: 21px;">';
    leafString += '';
    leafString += this.text;
    leafString += '<br>';
    leafString += '</span>';
    return leafString;
}

function showBranch(branch){
    var objBranch = document.getElementById(branch).style;
    if(objBranch.display=="block")
        objBranch.display="none";
    else
        objBranch.display="block";
    swapFolder('I' + branch);
}

function swapFolder(img){
    objImg = document.getElementById(img);
    if(objImg.src.indexOf('tree_closed.gif')>-1)
        objImg.src = openImg.src;
    else
        objImg.src = closedImg.src;
}

function highlight(selectedId) {

```

```
    if(curHighlighted != null) {
        var branchElement = document.getElementById(curHighlighted.id).style;
        branchElement.backgroundColor = "transparent";
    }
    curHighlighted = document.getElementById(selectedId);
    var selectedStyle = curHighlighted.style;
    selectedStyle.backgroundColor = "yellow";
}

function addSubTrans() {
    var URL = 'AddSubTrans.jsp';
    if(curHighlighted != null) {
        URL += '?parentId=';
        URL += curHighlighted.id;
    }
    location = URL;
}

function addOperation() {
    var URL = 'AddOperation.jsp';
    if(curHighlighted != null) {
        URL += '?parentId=';
        URL += curHighlighted.id;
    }
    location = URL;
}

function addDependencies() {
    var URL = 'Dependencies.jsp';
    if(curHighlighted != null) {
        URL += '?selectedTrans=';
        URL += curHighlighted.id;
    }
    location = URL;
}
```

B.3 JavaServer Pages files

ACID.jsp

```

<%@page contentType="text/html"%>
<%@page pageEncoding="UTF-8"%>
<%@taglib prefix="c" uri="http://java.sun.com/jsp/jstl/core"%>

<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN"
"http://www.w3.org/TR/html4/loose.dtd">
<html>
  <head>
    <title>CAGISTrans</title>
    <link href="cagisStyle.css" type="text/css" rel="stylesheet">
  </head>
  <body>
    <jsp:useBean id="specification" class="no.diplom.SpecificationBean"
      scope="session"/>

    <c:set target="${specification}" property="specName"
      value="${param.specname}" />

    <h1>ACID properties</h1>

    Choose whether atomicity and/or isolation should be full or relaxed.<br />
    Relaxed atomicity will allow subtransactions to commit even if
    their parent aborts.<br />
    Relaxed isolation will allow transactions to cooperate on data.<br />
    Note: The combination of full atomicity and relaxed isolation is not
    recommended<br /> due to the cost of rollbacks.
    <br /><br />

    <form method="post" action="TransactionOverview.jsp">
      <input type="hidden" name="fromAcid" value="1" />
      <table border="1" cellpadding="10">
        <tr>
          <th>Atomicity</th>
          <th>Isolation</th>
        </tr>
        <tr>
          <td>
            <input type="radio" name="atom" value="full">Full
            <br>
            <input type="radio" name="atom" value="relaxed" checked>Relaxed
          </td>
          <td>
            <input type="radio" name="isol" value="full">Full
            <br>
            <input type="radio" name="isol" value="relaxed" checked>Relaxed
          </td>
        </tr>
      </table>
      <br /><br /><br />

      <input type="submit" value="Continue">
    </form>
  </body>
</html>

```

ACIDcontroller.jsp

```

<%@page contentType="text/html"%>
<%@page pageEncoding="UTF-8"%>
<%@ taglib prefix="c" uri="http://java.sun.com/jsp/jstl/core"%>

<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN"
"http://www.w3.org/TR/html4/loose.dtd">
<html>
  <head><title>CAGISTrans</title></head>
  <body>
    <jsp:useBean id="specification" class="no.diplom.SpecificationBean"

```

```

        scope="session"/>

<c:set var="atom" value="\${specification.atomicity}" />
<c:set var="isol" value="\${specification.isolation}" />

<c:choose>
  <c:when test="\${(atom eq 'full') and (isol eq 'full')}">
    <jsp:forward page="CreateXML.jsp" />
  </c:when>

  <c:otherwise>
    <jsp:forward page="DependencyTrans.jsp" />
  </c:otherwise>
</c:choose>
</body>
</html>

```

AddIarg.jsp

```

<%@page contentType="text/html"%>
<%@page pageEncoding="UTF-8"%>
<%@ taglib prefix="c" uri="http://java.sun.com/jsp/jstl/core"%>

<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN"
"http://www.w3.org/TR/html4/loose.dtd">
<html>
  <head>
    <title>CAGISTrans</title>
    <link href="cagisStyle.css" type="text/css" rel="stylesheet">
    <script language="javascript">
      function handleClick(id) {
        if(document.getElementById(id).disabled == true)
          document.getElementById(id).disabled = false;
        else
          document.getElementById(id).disabled = true;
      }
    </script>
  </head>
  <body>
    <h1> Add in-argument(s)</h1>
    <jsp:useBean id="specification" class="no.diplom.SpecificationBean"
      scope="session"/>

    Select arguments from the list to add them to the
    operation's in-arguments.<br />
    Also select the intention the operation has with the argument.<br />
    New arguments can be added to the list by clicking the Create new
    argument button.<br />
    Note: Already selected arguments appear preselected in the list.
    <br /><br />

    <table border="0" cellpadding="5" cellspacing="5">
      <tr>
        <td>Parent transaction:</td>
        <td>
          <c:set var="parentTrans"
            value="\${specification.transactionsById[param.parentId]}" />
          <input type="text" disabled value="\${parentTrans.trid}">
        </td>
      </tr>
      <tr>
        <td>Operation name:</td>
        <td>
          <input type="text" disabled value="\${param.opid}">
        </td>
      </tr>
    </table>

    <br />
    <form method="post" action="OperationController.jsp">
      <table border="0" cellpadding="0" cellspacing="5">
        <input type="hidden" name="newOp" value="0" />
        <input type="hidden" name="iarg" value="1" />
        <input type="hidden" name="parentId" value="\${param.parentId}" />
        <input type="hidden" name="opid" value="\${param.opid}" />

```

```

<input type="hidden" name="opType" value="\${param.opType}" />
<input type="hidden" name="uniqueId" value="\${param.uniqueId}" />

<tr>
  <td>Select from existing argument(s):</td>
</tr>
<tr>
  <td>
    <table>
      <c:set var="args" value="\${specification.arguments}" />
      <c:choose>
        <c:when test="\${not empty args}">
          <c:set var="currentOp"
            value="\${specification.operationsById[param.uniqueId]}" />

          <c:forEach items="\${currentOp.iargsNames}" var="oparg"
            varStatus="status">
            <tr>
              <td>
                <input type="checkbox" name="dummy"
                  value="\<c:out value="\${oparg}" />" disabled checked />
                  <c:out value="\${oparg}" />&nbsp;
                </td>
              <td>
                <select name="dummy2" disabled>
                  <option value="browse"
                    <c:if test="\${currentOp.
                      iargsIntentions[status.index] eq 'browse'}"&>
                      selected
                    </c:if> >
                    Browse</option>
                  <option value="incorporate"
                    <c:if test="\${currentOp.
                      iargsIntentions[status.index] eq 'incorporate'}"&>
                      selected
                    </c:if> >
                    Incorporate</option>
                  <option value="modify"
                    <c:if test="\${currentOp.
                      iargsIntentions[status.index] eq 'modify'}"&>
                      selected
                    </c:if> >
                    Modify</option>
                </select>
              </td>
            </tr>
          </c:forEach>

          <jsp:useBean id="argList" class="no.diplom.ArgumentListBean" />
          <c:set target="\${argList}" property="spec"
            value="\${specification}" />
          <c:set target="\${argList}" property="currentOpId"
            value="\${param.uniqueId}" />

          <c:forEach items="\${argList.notIncludedIArgs}" var="arg"
            varStatus="status">
            <tr>
              <td>
                <input type="checkbox" name="arguments"
                  value="\<c:out value="\${arg.id}" />"
                  onclick="handleClick(\<c:out value='\${status.index}' />)"
                />
                <c:out value="\${arg.id}" />&nbsp;
              </td>
              <td>
                <select id="\<c:out value='\${status.index}' />"
                  disabled name="intention">
                  <option value="browse">Browse</option>
                  <option value="incorporate">Incorporate</option>
                  <option value="modify">Modify</option>
                </select>
              </td>
            </tr>
          </c:forEach>
        </c:when>
        <c:otherwise>

```



```

                <tr><td>
                    <input style="width: 150px;" type="text" disabled
                        value="No existing arguments" />
                    &nbsp;
                </td></tr>
            </c:otherwise>
        </c:choose>
    </table>
</td>
</tr>
<tr><td>&nbsp;</td></tr>
<tr>
    <td>
        <input style="width: 150px;" type="submit" name="newArgBtn"
            value="Create new argument" />
    </td>
</tr>
</table>
<br /><br />
<input type="submit" name="iargOkBtn" value=" OK " />
</form>
</body>
</html>

```

AddOarg.jsp

```

<%@page contentType="text/html"%>
<%@page pageEncoding="UTF-8"%>
<%@ taglib prefix="c" uri="http://java.sun.com/jsp/jstl/core"%>

<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN"
    "http://www.w3.org/TR/html4/loose.dtd">
<html>
<head>
    <title>CAGISTrans</title>
    <link href="cagisStyle.css" type="text/css" rel="stylesheet">
</head>
<body>
    <h1> Add out-argument (s)</h1>
    <jsp:useBean id="specification" class="no.diplom.SpecificationBean"
        scope="session"/>

    Select arguments from the list to add them to the operation's
    out-arguments.<br />
    New arguments can be added to the list by clicking the Create
    new argument button.<br />
    Note: Already selected arguments appear preselected in the list.
    <br /><br />

    <table border="0" cellpadding="5" cellspacing="5">
        <tr>
            <td>Parent transaction:</td>
            <td>
                <c:set var="parentTrans"
                    value="\${specification.transactionsById[param.parentId]}" />
                <input type="text" disabled value="\${parentTrans.trid}">
            </td>
        </tr>
        <tr>
            <td>Operation name:</td>
            <td>
                <input type="text" disabled value="\${param.opid}">
            </td>
        </tr>
    </table>

    <br />
    <form method="post" action="OperationController.jsp">
        <table border="0" cellpadding="0" cellspacing="5">
            <input type="hidden" name="newOp" value="0" />
            <input type="hidden" name="oarg" value="1" />
            <input type="hidden" name="parentId" value="\${param.parentId}" />
            <input type="hidden" name="opid" value="\${param.opid}" />
            <input type="hidden" name="opType" value="\${param.opType}" />
            <input type="hidden" name="uniqueId" value="\${param.uniqueId}" />

```

```
|  |
| --- |
| Select existing argument(s): |

```

<pre> <c:set var="args" value="\\${specification.arguments}" /> <c:choose> <c:when test="\\${not empty args}"> <c:set var="currentOp" value="\\${specification.operationsById[param.uniqueId]}" /> <c:forEach items="\\${currentOp.oargs}" var="oparg"> <tr> <td> <input type="checkbox" name="dummy" value="\<c:out value="\\${oparg.id}" />" disabled checked /> <c:out value="\\${oparg.id}" />&nbsp; </td> </tr> </c:forEach> <jsp:useBean id="argList" class="no.diplom.ArgumentListBean" /> <c:set target="\\${argList}" property="spec" value="\\${specification}" /> <c:set target="\\${argList}" property="currentOpId" value="\\${param.uniqueId}" /> <c:forEach items="\\${argList.notIncludedOArgs}" var="arg"> <tr> <td> <input type="checkbox" name="arguments" value="\<c:out value="\\${arg.id}" />"/> <c:out value="\\${arg.id}" />&nbsp; </td> </tr> </c:forEach> </c:when> <c:otherwise> <tr><td> <input style="width: 150px;" type="text" disabled value="No existing arguments" /> &nbsp; </td></tr> </c:otherwise> </c:choose> </table> </pre>

<pre> <input style="width: 150px;" type="submit" name="newArgBtn" value="Create new argument" /> </pre>

```

<br /><br />
<input type="submit" name="oargOkBtn" value=" OK " />
</form>
</body>
</html>

```

AddOperation.jsp

```

<%@page contentType="text/html"%>
<%@page pageEncoding="UTF-8"%>
<%@ taglib prefix="c" uri="http://java.sun.com/jsp/jstl/core"%>

<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN"
  "http://www.w3.org/TR/html4/loose.dtd">
<html>
<head>
  <title>CAGISTrans</title>
  <link href="cagisStyle.css" type="text/css" rel="stylesheet">

```

```

</head>
<body>
  <h1>Add new operation</h1>

  <c:choose>
    <c:when test="{empty param.parentId}">
      <h2>No transaction selected</h2>
      <br />
      Please go back and select the operation's parent transaction
      from the transaction tree
      <br /><br /><br />
      <input type="button" value="Back" onclick="history.back()" />
    </c:when>

    <c:otherwise>
      <jsp:useBean id="specification" class="no.diplom.SpecificationBean"
        scope="session" />

      Enter the name and type of the new operation.<br />
      In- and out-arguments are shown below, click the appropriate<br />
      buttons to add in- and out-arguments to the operation.<br />
      Note: An operation must have at least one in-argument.
      <br /><br />
      <form method="post" action="OperationController.jsp">
        <input type="hidden" name="parentId" value="{param.parentId}" />
        <input type="hidden" name="newOp" value="1" />
        <c:if test="{not empty param.uniqueId}">
          <input type="hidden" name="uniqueId" value="{param.uniqueId}" />
        </c:if>
        <table border="0" cellpadding="5" cellspacing="5">
          <tr>
            <td>Parent Transaction:</td>
            <td>
              <c:set var="parentTrans"
                value="{specification.transactionsById[param.parentId]}" />
              <input type="text" disabled value="{parentTrans.trid}" />
            </td>
          </tr>
          <tr>
            <td>Operation name:</td>
            <td>
              <c:choose>
                <c:when test="{not empty param.opid}">
                  <input type="text" disabled value="{param.opid}" />
                  <input type="hidden" name="opid" value="{param.opid}" />
                </c:when>

                <c:otherwise>
                  <input type="text" name="opid" />
                </c:otherwise>
              </c:choose>
            </td>
          </tr>
          <tr>
            <td>Operation type:</td>
            <td>
              <c:choose>
                <c:when test="{not empty param.opType}">
                  <input type="text" disabled value="{param.opType}" />
                  <input type="hidden" name="opType" value="{param.opType}" />
                </c:when>

                <c:otherwise>
                  <input type="text" name="opType" />
                </c:otherwise>
              </c:choose>
            </td>
          </tr>
        </table>
        <br />
        <table cellspacing="5">
          <tr>
            <td>In-arguments:</td>
            <td></td>
          </tr>
          <tr>
            <td align="right"><small>Argument</small></td>

```

```

        <td align="right"><small>Intention</small></td>
    </tr>
    <tr>
        <td>
            <c:choose>
                <c:when test="{not empty param.opid}">
                    <c:set var="operation"
                        value="{specification.operationsById[param.uniqueId]}" />
                    <select style="width: 175px;" disabled size="5">
                        <c:forEach items="{operation.iargsNames}" var="iarg">
                            <option>{iarg}</option>
                        </c:forEach>

                        <c:if test="{empty operation.iargsNames}">
                            <option>No in-arguments specified</option>
                        </c:if>
                    </select>
                </c:when>

                <c:otherwise>
                    <select style="width: 175px;" disabled size="5">
                        <option>No in-arguments specified</option>
                    </select>
                </c:otherwise>
            </c:choose>
        </td>
        <td>
            <c:choose>
                <c:when test="{not empty param.opid}">
                    <select style="width: 75px;" disabled size="5">
                        <c:forEach items="{operation.iargsIntentions}"
                            var="intention">
                            <option><c:out value="{intention}" /></option>
                        </c:forEach>

                        <c:if test="{empty operation.iargsNames}">
                            <option>None</option>
                        </c:if>
                    </select>
                </c:when>

                <c:otherwise>
                    <select style="width: 75px;" disabled size="5">
                        <option>None</option>
                    </select>
                </c:otherwise>
            </c:choose>
        </td>
    </tr>
    <tr>
        <td>
            <input type="submit" name="iargBtn" value="Add iarg" />
        </td>
        <td></td>
    </tr>
</table>
<br />
<table cellpadding="5">
    <tr>
        <td>Out-arguments:</td>
    </tr>
    <tr>
        <td align="right"><small>Argument</small></td>
    </tr>
    <tr>
        <td>
            <c:choose>
                <c:when test="{not empty param.opid}">
                    <c:set var="operation"
                        value="{specification.operationsById[param.uniqueId]}" />
                    <select style="width: 175px;" disabled size="5">
                        <c:forEach items="{operation.oargs}" var="oarg">
                            <option><c:out value="{oarg.id}" /></option>
                        </c:forEach>

                        <c:if test="{empty operation.oargs}">

```

```

        <option>No out-arguments specified</option>
      </c:if>
    </select>
  </c:when>

  <c:otherwise>
    <select style="width: 175px;" disabled size="5">
      <option>No out-arguments specified</option>
    </select>
  </c:otherwise>
</c:choose>
</td>
</tr>
<tr>
  <td>
    <input type="submit" name="oargBtn" value="Add oarg" />
  </td>
</tr>
</table>
<br /><br />
<input type="submit" name="okBtn" value=" OK " />
</form>
</c:otherwise>
</c:choose>
</body>
</html>

```

AddRootTrans.jsp

```

<%@page contentType="text/html"%>
<%@page pageEncoding="UTF-8"%>

<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN"
"http://www.w3.org/TR/html4/loose.dtd">
<html>
  <head>
    <title>CAGISTrans</title>
    <link href="cagisStyle.css" type="text/css" rel="stylesheet">
  </head>
  <body>
    <h1>Add new main transaction</h1>

    Enter the name and physical type of the new transaction.<br />
    Note: Flat transactions can not have subtransactions.
    <br /><br />

    <form method="post" action="TransactionOverview.jsp">
      <input type="hidden" name="root" value="1">
      <table border="0" cellpadding="5" cellspacing="5">
        <tr>
          <td>Transaction name:</td>
          <td>
            <input type="text" name="trid"/>
          </td>
        </tr>
        <tr>
          <td>Physical type:</td>
          <td>
            <select name="physType">
              <option>Nested</option>
              <option>Flat</option>
            </select>
          </td>
        </tr>
        <tr>
          <td>
            <input type="submit" value=" OK " />
          </td>
        </tr>
      </table>
    </form>
  </body>
</html>

```

AddSubTrans.jsp

```

<%@page contentType="text/html"%>
<%@page pageEncoding="UTF-8"%>
<%@ taglib prefix="c" uri="http://java.sun.com/jsp/jstl/core"%>

<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN"
"http://www.w3.org/TR/html4/loose.dtd">
<html>
  <head>
    <title>CAGISTrans</title>
    <link href="cagisStyle.css" type="text/css" rel="stylesheet">
  </head>
  <body>
    <jsp:useBean id="specification" class="no.diplom.SpecificationBean"
      scope="session"/>

    <h1>Add new subtransaction</h1>

    <c:choose>
      <c:when test="{empty param.parentId}">
        <h2>No parent transaction selected</h2>
        <br />
        Please go back and select a parent transaction from the transaction tree
        <br /><br /><br />
        <input type="button" value="Back" onclick="history.back()">
      </c:when>

      <c:when test="{specification.transactionsById[param.parentId].
        physType == 'Flat'}">
        <h2>Flat parent transaction selected</h2>
        <br />
        The selected parent transaction is flat, and can not have subtransactions
        <br /><br /><br />
        <input type="button" value="Back" onclick="history.back()">
      </c:when>

      <c:otherwise>

        Enter the name and physical type of the new transaction.<br />
        Note: Flat transactions can not have subtransactions.
        <br /><br />

        <form method="post" action="TransactionOverview.jsp">
          <input type="hidden" name="root" value="0">
          <input type="hidden" name="parentId" value="{param.parentId}">
          <table border="0" cellpadding="5" cellspacing="5">
            <tr>
              <td>Parent Transaction:</td>
              <td>
                <c:set var="parentTrans"
                  value="{specification.transactionsById[param.parentId]}" />
                <input type="text" disabled value="{parentTrans.trid}" />
              </td>
            </tr>
            <tr>
              <td>Transaction name:</td>
              <td>
                <input type="text" name="trid" />
              </td>
            </tr>
            <tr>
              <td>Physical type:</td>
              <td>
                <select name="physType">
                  <option>Nested</option>
                  <option>Flat</option>
                </select>
              </td>
            </tr>
            <tr>
              <td>
                <input type="submit" value=" OK " />
              </td>
            </tr>
          </table>
        </form>
      </c:otherwise>
    </c:choose>
  </body>
</html>

```

```

        </table>
    </form>
</c:otherwise>
</c:choose>
</body>
</html>

```

AnLcontroller.jsp

```

<%@page contentType="text/html"%>
<%@page pageEncoding="UTF-8"%>
<%@ taglib prefix="c" uri="http://java.sun.com/jsp/jstl/core"%>
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN"
    "http://www.w3.org/TR/html4/loose.dtd">
<html>
<head><title>CAGISTrans</title></head>
<body>
    <jsp:useBean id="specification" class="no.diplom.SpecificationBean"
        scope="session"/>

    <c:set target="\${specification}" property="lockingMechanism"
        value="\${param.locking}" />

    <c:forEach items="\${paramValues.awareness}" var="amech">
        <c:set target="\${specification}" property="awarenessMechanism"
            value="\${amech}" />
    </c:forEach>

    <c:choose>
        <c:when test="\${param.locking eq 'collaborative'}">
            <jsp:forward page="Conflicts.jsp" />
        </c:when>
        <c:otherwise>
            <jsp:forward page="CreateXML.jsp" />
        </c:otherwise>
    </c:choose>
</body>
</html>

```

AwarenessLocking.jsp

```

<%@page contentType="text/html"%>
<%@page pageEncoding="UTF-8"%>
<%@taglib prefix="c" uri="http://java.sun.com/jsp/jstl/core"%>
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN"
    "http://www.w3.org/TR/html4/loose.dtd">
<html>
<head>
    <title>CAGISTrans</title>
    <link href="cagisStyle.css" type="text/css" rel="stylesheet">
</head>
<body>
    <jsp:useBean id="specification" class="no.diplom.SpecificationBean"
        scope="session"/>

    <c:if test="\${specification.isolation eq 'relaxed'}">
        <c:set var="isorel" value="true" />
    </c:if>

    <h1>Awareness <c:if test="\${isorel}">and locking </c:if>mechanisms</h1>

    <c:choose>
        <c:when test="\${isorel}">
            <ul>
                <li>Select the desired awareness mechanisms, all involved parties<br />
                    will be notified when the events occur.</li>
                <li>Select the desired locking mechanism, user-controlled locking<br />
                    allows for negotiation if conflicts occur at runtime.<br />
                    Collaborative locking lets you deal with conflicts in advance<br />
                    with the help of permits and demands.</li>
            </ul>

```

```

    </c:when>

    <c:otherwise>
        Select the desired awareness mechanisms, all involved parties<br />
        will be notified when the events occur.<br />
    </c:otherwise>
</c:choose>
<br />
<form method="post" action="AnLcontroller.jsp">
    <table border="1" cellpadding="10">
        <tr>
            <th>Awareness Mechanisms</th>
            <c:if test="{isorel}"><th>Locking Mechanism</th></c:if>
        </tr>
        <tr>
            <td>
                <input type="checkbox" name="awareness" value="begin" />
                Notify on begin
                <br />
                <input type="checkbox" name="awareness" value="terminate" />
                Notify on terminate
                <c:if test="{isorel}">
                    <br />
                    <input type="checkbox" name="awareness" value="lock" />
                    Notify on lock
                    <br />
                    <input type="checkbox" name="awareness" value="change" />
                    Notify on change
                </c:if>
            </td>
            <c:if test="{isorel}">
                <td style="vertical-align: top">
                    <input type="radio" name="locking" value="user">
                    User-controlled locking
                    <br>
                    <input type="radio" name="locking" value="collaborative">
                    Collaborative locking
                </td>
            </c:if>
        </tr>
    </table>
    <br /><br /><br />

    <input type="submit" value="Continue">
</form>
</body>
</html>

```

Conflicts.jsp

```

<%@page contentType="text/html"%>
<%@page pageEncoding="UTF-8"%>
<%@ taglib prefix="c" uri="http://java.sun.com/jsp/jstl/core"%>

<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN"
"http://www.w3.org/TR/html4/loose.dtd">
<html>
    <head>
        <title>CAGISTrans</title>
        <link href="cagisStyle.css" type="text/css" rel="stylesheet">
        <script src="confTree.js" type="text/javascript"></script>
    </head>
    <jsp:useBean id="specification" class="no.diplom.SpecificationBean"
        scope="session"/>
    <c:set var="conflicts" value="{specification.conflicts}" />
    <c:choose>
        <c:when test="{empty conflicts}">
            <body>
                <h1>Conflicting operations</h1>
                <br /><br />
                <h2>There are no conflicting operations</h2>
                <br /><br />
                <form method="post" action="CreateXML.jsp">
                    <input type="submit" value="Continue">
                </form>
            </body>
        </c:when>
    </c:choose>

```



```

</c:when>
<c:otherwise>
  <c:if test="{empty numOfConf}">
    <c:set var="numOfConf" value="{specification.conflictsSize}"
      scope="session" />
    <c:set var="curConflict" value="-1" scope="session" />
  </c:if>
  <c:set var="curConflict" value="{curConflict + 1}" scope="session" />

  <c:if test="{curConflict ge numOfConf}">
    <c:set var="curConflict" value="0" scope="session" />
  </c:if>

  <c:set var="conflict" value="{conflicts[curConflict]}" />
  <c:set var="color" value="#FF0000" />

  <c:if test="{conflict.permitted}">
    <c:set var="color" value="#00FF00" />
  </c:if>

  <body onload="highlightConflict('<c:out
    value="{conflict.firstConflictOp.uniqueId}" />', '
    <c:out value="{conflict.secondConflictOp.uniqueId}" />', '
    <c:out value="{color}" />')">

  <h1>Conflicting operations</h1>

  Conflicting operations are highlighted in red in the transaction
  tree.<br />
  Click the next conflict button to see the next conflict if there
  are more than one.<br />
  Note: Permitted conflicts are highlighted in green.<br />
<br />

  <h2>Conflict <c:out value="{curConflict + 1}" /> of
    <c:out value="{numOfConf}" /></h2>
  <form method="post" action="ConflictsController.jsp">
    <c:if test="{numOfConf gt 1}">
      <input type="submit" value="Next conflict" name="nextConfBtn" />
      <br /><br />
    </c:if>

    <table border="1" cellpadding="10" >
      <tr>
        <td>
          <c:out value="{specification.jsTreeCode}" escapeXml="false" />
        </td>
        <td style="vertical-align: top;">
          <table>
            <tr>
              <th>Conflicting argument(s)</th>
            </tr>
            <tr>
              <td>
                <hr />
              </td>
            </tr>
            <c:forEach items="{conflict.conflictArgs}" var="arg">
              <tr><td>
                <c:out value="{arg.id}" />
              </td></tr>
            </c:forEach>
          </table>
        </td>
      </tr>
    </table>
    <br />

    <c:if test="{not conflict.permitted}">
      To allow the conflicting operations to be performed, <br />
      press the permit button below.<br />
      <input type="submit" name="permitBtn" value="Permit" />
    </c:if>

    <br /><br /><br />
    <input type="submit" name="contBtn" value="Continue" />
  </form>

```

```

    </c:otherwise>
  </c:choose>

</body>
</html>

```

ConflictsController.jsp

```

<%@page contentType="text/html"%>
<%@page pageEncoding="UTF-8"%>
<%@ taglib prefix="c" uri="http://java.sun.com/jsp/jstl/core"%>

<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN"
"http://www.w3.org/TR/html4/loose.dtd">
<html>
<head><title>JSP Page</title></head>
<body>
  <jsp:useBean id="specification" class="no.diplom.SpecificationBean"
    scope="session"/>

  <c:if test="${not empty param.nextConfBtn}">
    <jsp:forward page="Conflicts.jsp" />
  </c:if>

  <c:if test="${not empty param.permitBtn}">
    <c:set var="conflict" value="${specification.conflicts[curConflict]}" />
    <c:set target="${conflict}" property="permitted" value="true" />
    <c:set var="curConflict" value="${curConflict - 1}" scope="session" />
    <jsp:forward page="Conflicts.jsp" />
  </c:if>

  <c:if test="${not empty param.contBtn}">
    <jsp:forward page="Demands.jsp" />
  </c:if>
</body>
</html>

```

CreateArg.jsp

```

<%@page contentType="text/html"%>
<%@page pageEncoding="UTF-8"%>
<%@taglib prefix="c" uri="http://java.sun.com/jsp/jstl/core"%>

<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN"
"http://www.w3.org/TR/html4/loose.dtd">
<html>
<head>
  <title>CAGISTrans</title>
  <link href="cagisStyle.css" type="text/css" rel="stylesheet">
</head>
<body>
  <h1>Create new argument</h1>

  Enter the name, type, workspace state, address and
  owner of the new argument<br />
  Note: All argument names must be unique.
  <br /><br />

  <form method="post" action="OperationController.jsp">
    <input type="hidden" name="newArg" value="1" />
    <input type="hidden" name="parentId" value="${param.parentId}" />
    <input type="hidden" name="opid" value="${param.opid}" />
    <input type="hidden" name="opType" value="${param.opType}" />
    <input type="hidden" name="uniqueId" value="${param.uniqueId}" />

    <c:if test="${not empty param.iarg}">
      <input type="hidden" name="iarg" value="1" />
    </c:if>

    <c:if test="${not empty param.oarg}">
      <input type="hidden" name="oarg" value="1" />
    </c:if>

```

```

<table border="0" cellpadding="5" cellspacing="5">
  <tr>
    <td>Argument name:</td>
    <td>
      <input style="width: 150px;" type="text" name="argid" />
    </td>
  </tr>
  <tr>
    <td>Type:</td>
    <td>
      <select style="width: 150px;" name="argType">
        <option>File</option>
        <option>Webdoc</option>
        <option>Relation</option>
      </select>
    </td>
  </tr>
  <tr>
    <td>Workspace state:</td>
    <td>
      <select style="width: 150px;" name="wsState">
        <option>Private</option>
        <option>Group</option>
        <option>Public</option>
      </select>
    </td>
  </tr>
  <tr>
    <td>Address:</td>
    <td>
      <input style="width: 150px;" type="text" name="argAddr" />
    </td>
  </tr>
  <tr>
    <td>Owner:</td>
    <td>
      <input style="width: 150px;" type="text" name="argOwner" />
    </td>
  </tr>
  <tr>
    <td>
      <input type="submit" name="newArgBtn" value=" OK " />
    </td>
  </tr>
</table>
</form>
</body>
</html>

```

CreateXML.jsp

```

<%@page contentType="text/html"%>
<%@page pageEncoding="UTF-8"%>
<%@taglib prefix="c" uri="http://java.sun.com/jsp/jstl/core"%>

<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN"
"http://www.w3.org/TR/html4/loose.dtd">
<html>
  <head>
    <title>CAGISTrans</title>
    <link href="cagisStyle.css" type="text/css" rel="stylesheet">
  </head>
  <body>
    <jsp:useBean id="specification" class="no.diplom.SpecificationBean"
      scope="session"/>
    <c:set target="{specification}" property="servletContext"
      value="{pageContext.servletContext}" />
    <c:set target="{specification}" property="xml"
      value="{specification.appPath}" />
    <h1>The specification is complete</h1>
    <br />
    The specification has been completed successfully.<br />
    <br />

```

```

<c:set var="xmlLinks" value="\${specification.XMLfileLinks}" />
Below are links to the xml files representing the transaction
model specification:<br />
<ul>
  <li><c:out value="\${xmlLinks[0]}" escapeXml="false" /></li>
  <li><c:out value="\${xmlLinks[1]}" escapeXml="false" /></li>
</ul>

</body>
</html>

```

DemandController.jsp

```

<%@page contentType="text/html"%>
<%@page pageEncoding="UTF-8"%>
<%@ taglib prefix="c" uri="http://java.sun.com/jsp/jstl/core"%>

<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN"
"http://www.w3.org/TR/html4/loose.dtd">
<html>
  <head>
    <title>CAGISTrans</title>
    <link href="cagisStyle.css" type="text/css" rel="stylesheet">
  </head>
  <body>
    <jsp:useBean id="specification" class="no.diplom.SpecificationBean"
      scope="session"/>

    <c:choose>
      <c:when test="\${not empty param.selectedOp}">
        <c:set var="currentDemand" value="\${specification.currentDemand}" />
        <c:set target="\${currentDemand}" property="operation"
          value="\${specification.operationsById[param.selectedOp]}" />
        <jsp:forward page="Demands.jsp" />
      </c:when>

      <c:when test="\${not empty param.finishedSeqBtn}">
        <c:set target="\${specification}" property="demand"
          value="\${specification.currentDemand}" />
        <jsp:forward page="Demands.jsp" />
      </c:when>

      <c:when test="\${not empty param.contBtn}">
        <jsp:forward page="CreateXML.jsp" />
      </c:when>

      <c:otherwise>
        <h2>No operation selected</h2>
        <br />
        Please go back and select an operation from the transaction tree
        <br /><br /><br />
        <input type="button" value="Back" onclick="history.back()" />
      </c:otherwise>
    </c:choose>
  </body>
</html>

```

Demands.jsp

```

<%@page contentType="text/html"%>
<%@page pageEncoding="UTF-8"%>
<%@ taglib prefix="c" uri="http://java.sun.com/jsp/jstl/core"%>

<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN"
"http://www.w3.org/TR/html4/loose.dtd">
<html>
  <head>
    <title>CAGISTrans</title>
    <link href="cagisStyle.css" type="text/css" rel="stylesheet">
    <script src="demandTree.js" type="text/javascript"></script>
  </head>

```

```

<body>
  <jsp:useBean id="specification" class="no.diplom.SpecificationBean"
    scope="session"/>
  <h1>Demands</h1>

  A demand is a specific sequence of operations.<br />
  To add an operation to a demand sequence highlight it in
  the transaction tree<br />
  and click the Add op to demand button. When the demand sequence
  is finished,<br /> click the Sequence finished button.
<br /><br />

  <form method="post" action="DemandController.jsp">

    <table border="1" cellpadding="10">
      <tr>
        <td style="vertical-align: top">
          <c:out value="\${specification.jsTreeCode}" escapeXml="false" />
          <br /><br />
          <input type="button" value="Add op to demand" onclick="addOpToSeq()">
        </td>
        <td>
          <table>
            <tr>
              <td>
                <table>
                  <jsp:useBean id="demand" class="no.diplom.DemandBean"
                    scope="session"/>
                  <tr>
                    <th>New demand sequence</th>
                  </tr>
                  <c:forEach items="\${specification.currentDemand.opSequence}"
                    var="op" >
                    <tr>
                      <td><c:out value="\${op.oid}" /></td>
                    </tr>
                  </c:forEach>
                </table>
              </td>
            </tr>
          </table>
          <c:if test="\${specification.currentDemand.opSeqSize ge 2}">
            <tr>
              <td>
                <input type="submit" name="finishedSeqBtn"
                  value="Sequence finished">
              </td>
            </tr>
          </c:if>
          <tr>
            <td>
              <br /><hr noshade size="3" /><br />
            </td>
          </tr>
          <c:if test="\${not empty specification.demands}">
            <tr>
              <td>
                <table>
                  <tr>
                    <th>Existing demand sequence(s)</th>
                  </tr>
                  <c:forEach items="\${specification.demands}" var="demand">
                    <c:forEach items="\${demand.opSequence}" var="oper">
                    <tr>
                      <td><c:out value="\${oper.oid}" /></td>
                    </tr>
                  </c:forEach>
                  <tr>
                    <td><hr /></td>
                  </tr>
                </c:forEach>
              </table>
            </td>
          </c:if>
        </td>
      </tr>
    </table>
  </form>

```

```

                </tr>
            </c:if>
        </table>

        </td>
    </tr>
</table>

    <br /><br /><br />

    <input type="submit" name="contBtn" value="Continue">
</form>
</body>
</html>

```

DepController.jsp

```

<%@page contentType="text/html"%>
<%@page pageEncoding="UTF-8"%>
<%@ taglib prefix="c" uri="http://java.sun.com/jsp/jstl/core"%>

<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN"
    "http://www.w3.org/TR/html4/loose.dtd">
<html>
    <head><title>CAGISTrans</title></head>
    <body>
        <jsp:useBean id="specification" class="no.diplom.SpecificationBean"
            scope="session"/>

        <c:set var="mainTrans" value="\${specification.transactionsById[param.main]}" />

        <c:forEach items="\${paramValues.begin}" var="bdep">
            <c:set target="\${mainTrans}" property="beginDep"
                value="\${specification.transactionsById[bdep]}" />
        </c:forEach>

        <c:forEach items="\${paramValues.commit}" var="cdep">
            <c:set target="\${mainTrans}" property="commitDep"
                value="\${specification.transactionsById[cdep]}" />
        </c:forEach>

        <c:forEach items="\${paramValues.abort}" var="adep">
            <c:set target="\${mainTrans}" property="abortDep"
                value="\${specification.transactionsById[adep]}" />
        </c:forEach>

        <jsp:forward page="DependencyTrans.jsp" />
    </body>
</html>

```

Dependencies.jsp

```

<%@page contentType="text/html"%>
<%@page pageEncoding="UTF-8"%>
<%@ taglib prefix="c" uri="http://java.sun.com/jsp/jstl/core"%>

<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN"
    "http://www.w3.org/TR/html4/loose.dtd">
<html>
    <head>
        <title>CAGISTrans</title>
        <link href="cagisStyle.css" type="text/css" rel="stylesheet">
        <script src="depTree.js" type="text/javascript"></script>
    </head>
    <body>
        <c:if test="\${not empty param.selectedTrans}">
            onload="setMainHighlight('<c:out value="\${param.selectedTrans}' />')"
        </c:if>
        >

        <jsp:useBean id="specification" class="no.diplom.SpecificationBean"

```

```

        scope="session"/>
<h1>Dependencies</h1>

<c:choose>
  <c:when test="{empty param.selectedTrans}">
    <h2>No transaction selected</h2>
    <br />
    Please go back and select a transaction to add dependencies
    to from the transaction tree
    <br /><br /><br />
    <input type="button" value="Back" onclick="history.back()" />
  </c:when>
  <c:otherwise>
    <c:set target="{specification}" property="selectedTrans"
      value="{param.selectedTrans}" />

    Indicate the current transaction's dependencies by clicking on
    the other<br />
    transactions in the corresponding transaction trees.
    <ul>
      <li>Any transactions selected in the Begin dependencies tree will
        be begin dependent on<br />
        the current transaction, i.e. they can only start after the current
        transaction.</li>
      <li>Any transactions selected in the Commit dependencies tree will be
        commit dependent on<br />
        the current transaction, i.e. they can only commit after the current
        transaction.</li>
      <li>Any transactions selected in the Abort dependencies tree will be
        added to the abortset<br />
        of the current transaction, i.e. if the current transaction aborts,
        so will the selected transactions.</li>
    </ul>
    <br />
    <table border="1" cellpadding="10">
      <tr>
        <td><b>Begin dependencies</b></td>
        <td><b>Commit dependencies</b></td>
        <td><b>Abort dependencies</b></td>
      </tr>
      <tr>
        <td>
          <c:out value="{specification.jsBeginTreeCode}" escapeXml="false" />
        </td>
        <td>
          <c:out value="{specification.jsCommitTreeCode}" escapeXml="false" />
        </td>
        <td>
          <c:out value="{specification.jsAbortTreeCode}" escapeXml="false" />
        </td>
      </tr>
    </table>
  </c:otherwise>
</c:choose>

<br /><br /><br />
<c:if test="{not empty param.selectedTrans}">
  <input type="submit" value=" OK " onclick="getSelected()">
</c:if>

</body>
</html>

```

DependencyTrans.jsp

```

<%@page contentType="text/html"%>
<%@page pageEncoding="UTF-8"%>
<%@ taglib prefix="c" uri="http://java.sun.com/jsp/jstl/core"%>

<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN"
  "http://www.w3.org/TR/html4/loose.dtd">
<html>
  <head>
    <title>CAGISTrans</title>
    <link href="cagisStyle.css" type="text/css" rel="stylesheet">

```

```

<script src="tree.js" type="text/javascript"></script>
</head>
<body>
  <jsp:useBean id="specification" class="no.diplom.SpecificationBean"
    scope="session"/>
  <h1>Dependencies</h1>

  Select a transaction from the transaction tree and click the
  Specify dependencies<br />
  button to specify the dependencies for the selected transaction
  <br /> <br /><br />
  <table border="1" cellpadding="10" >
    <tr>
      <td>
        <c:out value="{specification.jsTreeCode}" escapeXml="false" />
      </td>
      <td style="vertical-align: top">
        <input style="width: 140px;" type="button"
          value="Specify dependencies" onclick="addDependencies()" />
      </td>
    </tr>
  </table>
  <br /><br /><br />

  <form method="post" action="AwarenessLocking.jsp">
    <input type="submit" value="Continue" />
  </form>
</body>
</html>

```

index.jsp

```

<%@page contentType="text/html"%>
<%@page pageEncoding="UTF-8"%>
<%@ taglib prefix="c" uri="http://java.sun.com/jsp/jstl/core"%>

<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN"
"http://www.w3.org/TR/html4/loose.dtd">
<html>
  <head>
    <title>CAGISTrans</title>
    <link href="cagisStyle.css" type="text/css" rel="stylesheet">
  </head>
  <body>
    <h1>Welcome to CAGISTrans</h1>
    <br />
    This application will guide you through the specification of<br />
    a transaction model for the CAGISTrans transaction environment.
    <br /><br />
    <form method="post" action="ACID.jsp">
      Please enter a name for this specification:<br />
      <input type="text" name="specname" />
      <br /><br /><br />
      <input type="submit" value="Continue" />
    </form>
  </body>
</html>

```

OperationController.jsp

```

<%@page contentType="text/html"%>
<%@page pageEncoding="UTF-8"%>
<%@taglib prefix="c" uri="http://java.sun.com/jsp/jstl/core"%>

<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN"
"http://www.w3.org/TR/html4/loose.dtd">
<html>
  <head><title>CAGISTrans</title></head>
  <body>

    <jsp:useBean id="specification" class="no.diplom.SpecificationBean"
      scope="session"/>

```



```

<jsp:useBean id="uniqueId" class="no.diplom.UniqueIdBean"
  scope="session" />
<c:if test="${not empty param.newOp}">
  <c:if test="${param.newOp eq 1}">
    <c:if test="${empty param.uniqueId}" >
      <jsp:useBean id="operation" class="no.diplom.OperationBean" />

      <c:set target="${operation}" property="oid"
        value="${param.opid}" />
      <c:set target="${operation}" property="type"
        value="${param.opType}" />
      <c:set target="${operation}" property="uniqueId"
        value="${uniqueId.ID}" />
      <c:set var="parentTrans"
        value="${specification.transactionsById[param.parentId]}" />
      <c:set target="${operation}" property="parent"
        value="${parentTrans}" />
      <c:set target="${specification}" property="operation"
        value="${operation}" />
    </c:if>

    <c:if test="${not empty param.okBtn}">
      <jsp:forward page="TransactionOverview.jsp"/>
    </c:if>

    <c:if test="${not empty param.iargBtn}">
      <c:choose>
        <c:when test="${empty param.uniqueId}" >
          <jsp:forward page="AddIarg.jsp">
            <jsp:param name="uniqueId" value="${operation.uniqueId}"/>
          </jsp:forward>
        </c:when>
        <c:otherwise>
          <jsp:forward page="AddIarg.jsp" />
        </c:otherwise>
      </c:choose>
    </c:if>

    <c:if test="${not empty param.oargBtn}">
      <c:choose>
        <c:when test="${empty param.uniqueId}" >
          <jsp:forward page="AddOarg.jsp">
            <jsp:param name="uniqueId" value="${operation.uniqueId}"/>
          </jsp:forward>
        </c:when>
        <c:otherwise>
          <jsp:forward page="AddOarg.jsp" />
        </c:otherwise>
      </c:choose>
    </c:if>
  </c:if>

  <c:if test="${param.newOp eq 0}">
    <c:if test="${not empty param.newArgBtn}">
      <jsp:forward page="CreateArg.jsp" />
    </c:if>
    <c:if test="${not empty param.iargOkBtn}">
      <c:forEach items="${paramValues.arguments}"
        var="arg" varStatus="status">
        <c:set target="${specification.operationsById[param.uniqueId]}"
          property="intention"
          value="${paramValues.intention[status.index]}"/>
        <c:set target="${specification.operationsById[param.uniqueId]}"
          property="iarg" value="${arg}"/>
      </c:forEach>

      <jsp:forward page="AddOperation.jsp" />
    </c:if>
    <c:if test="${not empty param.oargOkBtn}">
      <c:forEach items="${paramValues.arguments}" var="arg">
        <c:set target="${specification.operationsById[param.uniqueId]}"
          property="oarg" value="${specification.argumentsById[arg]}" />
      </c:forEach>

      <jsp:forward page="AddOperation.jsp" />
    </c:if>
  </c:if>

```

```

    </c:if>
  </c:if>

  <c:if test="${not empty param.newArg}">
    <jsp:useBean id="argument" class="no.diplom.ArgumentBean" />

    <c:set target="${argument}" property="id" value="${param.argid}" />
    <c:set target="${argument}" property="type" value="${param.argType}" />
    <c:set target="${argument}" property="workspaceState"
      value="${param.wsState}" />
    <c:set target="${argument}" property="address" value="${param.argAddr}" />
    <c:set target="${argument}" property="owner" value="${param.argOwner}" />

    <c:set target="${specification}" property="argument" value="${argument}" />

    <c:if test="${not empty param.iarg}">
      <jsp:forward page="AddIarg.jsp" />
    </c:if>

    <c:if test="${not empty param.oarg}">
      <jsp:forward page="AddOarg.jsp" />
    </c:if>
  </c:if>
</body>
</html>

```

TransactionOverview.jsp

```

<%@page contentType="text/html"%>
<%@page pageEncoding="UTF-8"%>
<%@taglib prefix="c" uri="http://java.sun.com/jsp/jstl/core"%>

<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN"
  "http://www.w3.org/TR/html4/loose.dtd">
<html>
  <head>
    <title>CAGISTrans</title>
    <link href="cagisStyle.css" type="text/css" rel="stylesheet">
    <script src="tree.js" type="text/javascript"></script>
  </head>
  <body>

    <jsp:useBean id="specification" class="no.diplom.SpecificationBean"
      scope="session"/>
    <jsp:useBean id="uniqueId" class="no.diplom.UniqueIdBean"
      scope="session" />

    <c:if test="${not empty param.fromAcid}">
      <c:set target="${specification}" property="atomicity"
        value="${param.atom}" />
      <c:set target="${specification}" property="isolation"
        value="${param.isol}" />
    </c:if>

    <c:if test="${not empty param.root}">
      <jsp:useBean id="transaction" class="no.diplom.TransactionBean" />

      <c:if test="${param.root eq 1}">
        <c:set target="${transaction}" property="uniqueId"
          value="${uniqueId.ID}" />
        <c:set target="${transaction}" property="trid"
          value="${param.trid}" />
        <c:set target="${transaction}" property="physType"
          value="${param.physType}" />

        <c:set target="${specification}" property="transaction"
          value="${transaction}" />
      </c:if>

      <c:if test="${param.root eq 0}">
        <c:set target="${transaction}" property="uniqueId"
          value="${uniqueId.ID}" />
        <c:set target="${transaction}" property="trid"
          value="${param.trid}" />
        <c:set target="${transaction}" property="physType"

```

```

        value="{param.physType}" />

<c:set var="parentTrans"
    value="{specification.transactionsById[param.parentId]}" />
<c:set target="{transaction}" property="parent"
    value="{parentTrans}" />

    <c:set target="{specification}" property="transaction"
        value="{transaction}" />
</c:if>
</c:if>

<h1>Setup transactions</h1>

Use the add main transaction button to add root transactions
to the specification.<br />
Select a transaction in the transaction tree and use the add
subtransaction button<br />
to add a child to the selected transaction or use the add
operation button to add an<br />
operation to the selected transaction.
<br /><br /><br />

<table border="1" cellpadding="10">
  <tr>
    <td>
      <c:set var="transactions" value="{specification.transactions}" />
      <c:choose>
        <c:when test="{not empty(transactions)}">
          <c:out value="{specification.jsTreeCode}" escapeXml="false" />
        </c:when>
        <c:otherwise>
          No transactions specified
        </c:otherwise>
      </c:choose>
    </td>
    <td style="vertical-align: top">
      <table>
        <tr>
          <td>
            <input style="width: 140px;" type="button"
              value="Add main transaction"
              onclick="location = 'AddRootTrans.jsp'"/>
          </td>
        </tr>
        <c:if test="{not empty(transactions)}">
          <tr>
            <td>
              <input style="width: 140px;" type="button"
                value="Add subtransaction"
                onclick="addSubTrans()"/>
            </td>
          </tr>
          <tr>
            <td>
              <input style="width: 140px;" type="button"
                value="Add operation"
                onclick="addOperation()"/>
            </td>
          </tr>
        </c:if>
      </table>
    </td>
  </tr>
</table>
<c:if test="{not empty(transactions)}">
  <br /><br /><br />
  <form method="post" action="ACIDcontroller.jsp">
    <input type="submit" value="Continue" />
  </form>
</c:if>
</body>
</html>

```

B.4 Java files

ArgumentBean.java

```
package no.diplom;

/**
 * This class represents the objects in the system.
 * These objects are used as in- and out-arguments in advanced operations
 * @author Thomas Bjorge
 */
public class ArgumentBean {

    private String id;
    private String type;
    private String workspaceState;
    private String address;
    private String owner;

    /**
     * Constructor is empty
     */
    public ArgumentBean() {
    }

    /**
     * Returns ID representing the unique ID of the argument
     * @return The ID string
     */
    public String getId() {
        return id;
    }

    /**
     * Set's the unique ID of the argument
     * @param id The ID String
     */
    public void setId(String id) {
        this.id = id;
    }

    /**
     * Returns the type of the argument
     * @return A String containing either
     * "File", "Webdoc" or "Relation"
     */
    public String getType() {
        return type;
    }

    /**
     * Sets the arguments type
     * @param type A String containing either
     * "File", "Webdoc" or "Relation"
     */
    public void setType(String type) {
        this.type = type;
    }

    /**
     * Gets the arguments workspace state
     * @return A String containing either
     * "Private", "Group" or "Public"
     */
    public String getWorkspaceState() {
        return workspaceState;
    }

    /**
     * Sets the arguments workspace state
     * @param workspaceState A String containing either

```

```

    * "Private", "Group" or "Public"
    */
    public void setWorkspaceState(String workspaceState) {
        this.workspaceState = workspaceState;
    }

    /**
     * Gets the arguments address
     * @return A String containing the argument's address
     */
    public String getAddress() {
        return address;
    }

    /**
     * Sets the arguments address
     * @param address A String containing the argument's address
     */
    public void setAddress(String address) {
        this.address = address;
    }

    /**
     * Gets the argument's owner
     * @return A String containing the owner of the argument
     */
    public String getOwner() {
        return owner;
    }

    /**
     * Gets the argument's owner
     * @param owner A String containing the owner of the argument
     */
    public void setOwner(String owner) {
        this.owner = owner;
    }
}

```

ArgumentListBean.java

```

package no.diplom;

import java.util.Vector;

/**
 * A simple utility class used by AddIarg.jsp and AddOarg.jsp
 * to better handle the displaying of arguments
 * @author Thomas Bjorge
 */
public class ArgumentListBean {

    private String currentOpId;
    private SpecificationBean spec;

    /**
     * The constructor is empty
     */
    public ArgumentListBean() {
    }

    /**
     * Sets the ID of the current operation that
     * arguments are being added to
     * @param currentOpId A String with the
     * unique ID of the current operation
     */
    public void setCurrentOpId(String currentOpId) {
        this.currentOpId = currentOpId;
    }

    /**
     * Returns the arguments not included in the
     * current operation's in-argument list
     */
}

```

```

    * @return An array of ArgumentBean objects
    */
    public ArgumentBean[] getNotIncludedIArgs() {
        OperationBean currentOp =
            (OperationBean)spec.getOperationsById().get(currentOpId);
        ArgumentBean [] allArgs = spec.getArguments();
        Vector notIncluded = new Vector();
        String [] iargsNames = currentOp.getIargsNames();
        for(int i=0; i<allArgs.length; i++) {
            boolean found = false;
            for(int j=0; j<iargsNames.length; j++) {
                if(allArgs[i].getId().equals(iargsNames[j])) {
                    found = true;
                    break;
                }
            }
            if(!found) {
                notIncluded.add(allArgs[i]);
            }
        }
        ArgumentBean[] notIncArray = new ArgumentBean[notIncluded.size()];
        notIncluded.toArray(notIncArray);
        return notIncArray;
    }

    /**
     * Returns the arguments not included in the
     * current operation's out-argument list
     * @return An array of ArgumentBean objects
     */
    public ArgumentBean[] getNotIncludedOArgs() {
        OperationBean currentOp =
            (OperationBean)spec.getOperationsById().get(currentOpId);
        ArgumentBean [] allArgs = spec.getArguments();
        Vector notIncluded = new Vector();
        ArgumentBean [] oargs = currentOp.getOargs();
        for(int i=0; i<allArgs.length; i++) {
            boolean found = false;
            for(int j=0; j<oargs.length; j++) {
                if(allArgs[i].getId().equals(oargs[j].getId())) {
                    found = true;
                    break;
                }
            }
            if(!found) {
                notIncluded.add(allArgs[i]);
            }
        }
        ArgumentBean[] notIncArray = new ArgumentBean[notIncluded.size()];
        notIncluded.toArray(notIncArray);
        return notIncArray;
    }

    /**
     * Gives the class a pointer to the
     * SpecificationBean class
     * @param spec The SpecificationBean object
     * used in the current session
     */
    public void setSpec(SpecificationBean spec) {
        this.spec = spec;
    }
}

```

ConflictBean.java

```

package no.diplom;

import java.util.Vector;

/**
 * This class represents conflicting operations.
 * Each conflict consists of two operations and
 * all the arguments causing the conflict

```

```
* @author Thomas Bjorge
*/
public class ConflictBean {

    private OperationBean firstConflictOp;
    private OperationBean secondConflictOp;
    private Vector conflictArgs;
    private boolean permitted;

    /**
     * The constructor explicitly sets permitted to false
     */
    public ConflictBean() {
        permitted = false;
    }

    /**
     * Gets the first conflicting operation.
     * Which operation is first makes no difference.
     * @return The OperationBean representing the
     * first conflicting operation
     */
    public OperationBean getFirstConflictOp() {
        return firstConflictOp;
    }

    /**
     * Sets the first conflicting operation
     * @param firstConflictOp The OperationBean
     * representing the first conflictin operation
     */
    public void setFirstConflictOp(OperationBean firstConflictOp) {
        this.firstConflictOp = firstConflictOp;
    }

    /**
     * Gets the second conflicting operation.
     * Which operation is second makes no difference.
     * @return The OperationBean representing the
     * second conflictin operation
     */
    public OperationBean getSecondConflictOp() {
        return secondConflictOp;
    }

    /**
     * Sets the second confliction operation
     * @param secondConflictOp The OperationBean
     * representing the second conflicting operation
     */
    public void setSecondConflictOp(OperationBean secondConflictOp) {
        this.secondConflictOp = secondConflictOp;
    }

    /**
     * Gets the arguments causing the conflict between the two operations
     * @return An array containing the ArgumentBeans causing the conflict
     */
    public ArgumentBean[] getConflictArgs() {
        ArgumentBean [] argArray = new ArgumentBean[conflictArgs.size()];
        conflictArgs.toArray(argArray);
        return argArray;
    }

    /**
     * Sets the conflicting arguments
     * @param conflictArgs A Vector containing the
     * ArgumentBeans causing the conflict
     */
    public void setConflictArgs(Vector conflictArgs) {
        this.conflictArgs = conflictArgs;
    }

    /**
     * Gets the permitted property.
     * Permits are represented in the system by this property
     * @return True if the conflict is permitted, false otherwise
     */
}
```

```

    */
    public boolean isPermitted() {
        return permitted;
    }

    /**
     * Sets the permitted property.
     * Permits are represented in the system by this property
     * @param permitted Either true or false
     */
    public void setPermitted(boolean permitted) {
        this.permitted = permitted;
    }
}

```

Conflicts.java

```

package no.diplom;

import java.util.Vector;

/**
 * This class finds any conflicts between two operations
 * by comparing the operations in their in-sets and out-sets
 */
public class Conflicts {

    private SpecificationBean spec;
    /**
     * Sets up a pointer to the current session's SpecificationBean
     * @param spec The session's SpecificationBean
     */
    public Conflicts(SpecificationBean spec) {
        this.spec = spec;
    }

    /**
     * Loops through all the existing operations and calls findConflicts
     * with all possible operation pairs. If a conflict is returned a new
     * ConflictBean is created
     */
    public void checkForConflicts() {
        OperationBean [] operations = spec.getOperations();
        for(int i=0; i<operations.length; i++) {
            for(int j=i+1; j<operations.length; j++) {
                Vector conflictArgs =
                    findConflicts(operations[i], operations[j]);
                if(conflictArgs.size() > 0) {
                    ConflictBean conflict = new ConflictBean();
                    conflict.setFirstConflictOp(operations[i]);
                    conflict.setSecondConflictOp(operations[j]);
                    conflict.setConflictArgs(conflictArgs);
                    spec.setConflict(conflict);
                }
            }
        }
    }

    /**
     * Checks for conflicts by comparing the two
     * operations' in-sets and out-sets
     * @param op1 The first operationBean to be
     * checked against the second for conflicts
     * @param op2 The second operationBean to be
     * checked against the first for conflicts
     * @return A Vector containing all the ArgumenBeans
     * causing the conflict
     */
    public Vector findConflicts(OperationBean op1, OperationBean op2) {
        Vector conflictingArgs = new Vector();
        ArgumentBean [] outset1 = op1.getOargs();
        ArgumentBean [] outset2 = op2.getOargs();
        ArgumentBean [] inset1 = argNamesToObjects(op1.getIargsNames());
        ArgumentBean [] inset2 = argNamesToObjects(op2.getIargsNames());
    }
}

```



```

ArgumentBean [] browseset1 = argNamesToObjects(op1.getBrowseset());
ArgumentBean [] browseset2 = argNamesToObjects(op2.getBrowseset());

//check if intersection of outset1 and outset two is empty
for(int i=0; i<outset1.length; i++) {
    for(int j=0; j<outset2.length; j++) {
        if(outset1[i].getId().equals(outset2[j].getId())) {
            conflictingArgs.add(outset1[i]);
        }
    }
}
//check that args read by one op and written by another
//are only for browse with the reading op
for(int i=0; i<inset1.length; i++) {
    for(int j=0; j<outset2.length; j++) {
        if(inset1[i].getId().equals(outset2[j].getId())) {
            if(!inBrowseset(inset1[i], browseset1)) {
                if(!conflictingArgs.contains(inset1[i]))
                    conflictingArgs.add(inset1[i]);
            }
        }
    }
}
//check that args read by the other op and written by the first
//are only for browse with the reading op
for(int i=0; i<inset2.length; i++) {
    for(int j=0; j<outset1.length; j++) {
        if(inset2[i].getId().equals(outset1[j].getId())) {
            if(!inBrowseset(inset2[i], browseset2)) {
                if(!conflictingArgs.contains(inset2[i]))
                    conflictingArgs.add(inset2[i]);
            }
        }
    }
}
return conflictingArgs;
}

/**
 * Checks if an argument is in an operation's browseset
 * @param arg The ArgumentBean two be tested
 * @param browseset An array of ArgumentBeans
 * @return True if the argument is in the browseset, false otherwise
 */
private boolean inBrowseset(ArgumentBean arg, ArgumentBean[] browseset) {
    for(int i=0; i<browseset.length; i++) {
        if(arg.getId().equals(browseset[i].getId())) {
            return true;
        }
    }
    return false;
}

/**
 * Converts an array of argument IDs into an array of the IDs'
 * corresponding ArgumentBeans
 * @param argNames A String array of argument IDs
 * @return An array of ArgumentBeans
 */
private ArgumentBean[] argNamesToObjects(String [] argNames) {
    ArgumentBean [] argObjects = new ArgumentBean[argNames.length];
    for(int i=0; i<argObjects.length; i++) {
        argObjects[i] =
            (ArgumentBean) spec.getArgumentsById().get(argNames[i]);
    }
    return argObjects;
}
}

```

DemandBean.java

```

package no.diplom;
import java.util.Vector;

```

```

/**
 * This class represents demands which are specific sequences of operations
 * @author Thomas Bjorge
 */
public class DemandBean {

    Vector opsequence = new Vector();

    /**
     * The constructor is empty
     */
    public DemandBean() {
    }

    /**
     * Adds an operation to the and of the demand sequence
     * @param op The OperationBean to be added
     */
    public void setOperation(OperationBean op) {
        opsequence.add(opsequence.size(), op);
    }

    /**
     * Gets the operation sequence
     * @return An array of OperaionBeans representing the sequence
     */
    public OperationBean[] getOpSequence() {
        OperationBean[] seqArray = new OperationBean[opsequence.size()];
        opsequence.toArray(seqArray);
        return seqArray;
    }

    /**
     * Gets the number of operations in the demand
     * @return The integer size of the demand sequence
     */
    public int getOpSeqSize() {
        return opsequence.size();
    }
}

```

GenerateTree.java

```

package no.diplom;

import java.util.Vector;

/**
 * This class generates the necessary javascript code to create the trees
 * representing transactions and operations
 * @author Thomas Bjorge
 */
public class GenerateTree {
    private SpecificationBean spec;
    StringBuffer jscode;

    /**
     * Empty constructor
     */
    public GenerateTree() {
    }

    /**
     * Sets a pointer to the current session's SpecificationBean
     * @param spec The session's SpecificationBean
     */
    public GenerateTree(SpecificationBean spec) {
        this.spec = spec;
    }

    /**

```

```

* Calls generateTree with no selected transaction
* @param treeType A String representing the treetype,
* either "b","c","a" or empty
* @return The generated javascript code
*/
public String getJsCode(String treeType) {
    generateTree(treeType, "");
    return jscode.toString();
}

/**
* Calls generateTree with the correct treetype and the id of the
* current selected transaction
* @param treeType A String representing the treetype,
* either "b","c","a" or empty
* @param selectedTrans A String containing the id of
* the selected transaction
* @return The generated javascript code
*/
public String getJsCode(String treeType, String selectedTrans) {
    generateTree(treeType, selectedTrans);
    return jscode.toString();
}

/**
* Generates part of the javascript code
* @param treeType A String representing the treetype,
* either "b","c","a" or empty
* @param selectedTrans A String containing the id of
* the selected transaction
*/
private void generateTree(String treeType, String selectedTrans) {
    jscode = new StringBuffer();
    jscode.append("<script language=\"JavaScript\">\n");
    jscode.append("var my" + treeType + "Tree = new tree();\n");
    TransactionBean [] transactions = spec.getTransactions();
    //if treetype abort and isolation = full,
    //only return family of selected branch
    if(treeType.equals("a") && !selectedTrans.equals("")) {
        jscode.append("my" + treeType + "Tree.add(" +
            writeTreeCode(getSelectedRoot(selectedTrans), treeType) +
            ");\n");
    }
    else {
        for(int i=0; i<transactions.length; i++) {
            if(transactions[i].getParent() == null) {
                jscode.append("my" + treeType + "Tree.add(" +
                    writeTreeCode(transactions[i], treeType) +
                    ");\n");
            }
        }
    }
    jscode.append("my" + treeType + "Tree.write();\n");
    jscode.append("</script>\n");
}

/**
* Generates the javascript code representing each transaction.
* The method calls itself recursively for each child transaction
* @param trans The TransactionBean to generate the javascript code for
* @param treeType A String representing the treetype,
* either "b","c","a" or empty
* @return A string containing the method's generated javascript code
*/
private String writeTreeCode(TransactionBean trans, String treeType) {
    jscode.append("var " + trans.getTrid() +
        " = new branch(\"" + treeType + trans.getUniqueId() +
        "\",\"" + trans.getTrid() + "\");\n");
    Vector children = trans.getChildren();
    int numOfChildren = children.size();
    if(numOfChildren > 0) {
        for(int i=0; i<numOfChildren; i++) {
            jscode.append(trans.getTrid() + ".add(" +
                writeTreeCode((TransactionBean)children.get(i),
                    treeType) + ");\n");
        }
    }
}

```

```

    }
    Vector leaves = trans.getOperations();
    int numOfLeaves = leaves.size();
    if(numOfLeaves > 0) {
        for(int j=0; j<numOfLeaves; j++) {
            OperationBean tmp = ((OperationBean)leaves.get(j));
            jscode.append("var " + tmp.getOid() + " = new leaf(\"' +
                tmp.getUniqueId() + "\",\"' +
                tmp.getOid() + "\");\n");
            jscode.append(trans.getTrid() + ".add(" +
                tmp.getOid() + ");\n");
        }
    }
    return trans.getTrid();
}
}

/**
 * Finds the root of the current selected transaction
 * @param selected The string ID of the selected transaction
 * @return The TransactionBean that is the root
 * of the selected transaction
 */
private TransactionBean getSelectedRoot(String selected) {
    TransactionBean sel =
        (TransactionBean)spec.getTransactionsById().get(selected);
    TransactionBean parent = sel.getParent();
    while(parent != null) {
        TransactionBean tmp = parent.getParent();
        sel = parent;
        parent = tmp;
    }
    return sel;
}
}
}

```

GenerateXML.java

```

package no.diplom;

import java.io.*;
import java.util.Vector;
import org.jdom.*;
import org.jdom.output.*;

/**
 * This class generates the two xml files which
 * are the result of the specification.
 * The JDOM package (<CODE>www.jdom.org</CODE>)
 * is used to create and write the xml
 * @author teb
 */
public class GenerateXML {
    private SpecificationBean spec;

    /**
     * The constructor sets up the pointer to the
     * session's SpecificationBean
     * @param spec The current session's SpecificationBean
     */
    public GenerateXML(SpecificationBean spec) {
        this.spec = spec;
    }

    /**
     * Calls the two methods that create the two xml files
     * @param path A String representing the application's
     * directory on the server
     */
    public void createXml(String path) {
        createExecTree(path);
        createSpecTree(path);
    }
}

```

```

/**
 * Writes the JDOM xml representation to an actual xml file
 * @param path A String representing the application's
 * directory on the server
 * @param doc The JDOM Document object that represents the xml
 * @param prefix The String that differentiates the two
 * generated xml files, either "Exec" or "Spec"
 */
private void writeTree(Document doc, String prefix, String path) {
    try {
        //String filename = "..\\webapps\\Cagis\\xml\\";
        String filename = path + "xml\\" +
            spec.getSpecName() + prefix + ".xml";
        System.out.println("\n*** Writing file to: "+filename+" ***\n");
        BufferedWriter writer =
            new BufferedWriter(new FileWriter(filename));
        XMLOutputter outp = new XMLOutputter(Format.getPrettyFormat());
        outp.output(doc, writer);
        writer.close();
    } catch (IOException e) {
        e.printStackTrace();
    }
}

/**
 * Created the Execution xml by calling writeTransElement
 * @param path A String representing the
 * application's directory on the server
 */
private void createExecTree(String path) {
    Document exec = new Document();
    DocType dt = new DocType("transactions", "Transexec.dtd");
    exec.setDocType(dt);
    Element root = new Element("transactions");

    TransactionBean [] transactions = spec.getTransactions();
    for(int i=0; i<transactions.length; i++) {
        TransactionBean trans = transactions[i];
        if(trans.getParent() == null) {
            Element tr = writeTransElement(trans);
            root.addContent(tr);
        }
    }
    exec.addContent(root);
    writeTree(exec, "Exec", path);
}

/**
 * Generates the <CODE>transaction</CODE>
 * elements in the execution specification.
 * The method calls itself recursively for each child transaction
 * @param trans The TransactionBean representing
 * the element being created
 * @return The xml transaction element
 */
private Element writeTransElement(TransactionBean trans) {
    Element tr = new Element("transaction");
    tr.setAttribute("trid", String.valueOf(trans.getUniqueId()));
    tr.addContent(new Element("begin"));

    //Operations
    Element ops = new Element("operations");
    for(int j=0; j<trans.getOperations().size(); j++) {
        OperationBean operation =
            (OperationBean)trans.getOperations().get(j);
        Element op = new Element("operation");
        op.setAttribute("oid", String.valueOf(operation.getUniqueId()));
        op.setAttribute("type", operation.getType());

        Element inargs = new Element("iargs");
        String [] iargsNames = operation.getIargsNames();
        String [] iargsIntentions = operation.getIargsIntentions();
        for(int k=0; k<iargsNames.length; k++) {
            Element arg = new Element("arg");
            arg.setAttribute("name", iargsNames[k]);
            arg.setAttribute("intent", iargsIntentions[k]);
        }
    }
    ops.addContent(ops);
    tr.addContent(ops);
}

```

```

        inargs.addContent(arg);
    }
    op.addContent(inargs);

    Element outargs = new Element("oargs");
    ArgumentBean [] oargs = operation.getOargs();
    for(int l=0; l<oargs.length; l++) {
        Element arg = new Element("arg");
        arg.setAttribute("name", oargs[l].getId());
        outargs.addContent(arg);
    }
    op.addContent(outargs);
    ops.addContent(op);
}
tr.addContent(ops);

//Subtransactions
Vector subtrans = trans.getChildren();
for(int i=0; i<subtrans.size(); i++) {
    tr.addContent(
        writeTransElement((TransactionBean)subtrans.get(i)));
}
Element end = new Element("end");
end.setAttribute("type", "commit");
tr.addContent(end);

return tr;
}

/**
 * Creates the Specification xml
 * @param path A String representing the
 * application's directory on the server
 */
private void createSpecTree(String path) {
    Document specdoc = new Document();
    DocType dt = new DocType("characteristics", "Transspec.dtd");
    specdoc.setDocType(dt);
    Element root = new Element("characteristics");
    root.setAttribute("name", spec.getSpecName());

    //properties
    Element props = new Element("properties");
    Element atom = new Element("atom");
    atom.setAttribute("value", spec.getAtomicity());
    props.addContent(atom);
    Element cons = new Element("cons");
    cons.setAttribute("value", "full");
    props.addContent(cons);
    Element isol = new Element("isol");
    isol.setAttribute("value", spec.getIsolation());
    props.addContent(isol);
    Element dur = new Element("dur");
    dur.setAttribute("value", "full");
    props.addContent(dur);
    root.addContent(props);

    //structure
    Element structure = new Element("structure");
    Element phys = new Element("physical");
    if(nestedPhysType())
        phys.setAttribute("type", "nested");
    else
        phys.setAttribute("type", "flat");
    structure.addContent(phys);

    //dependencies
    Element deps = new Element("dependencies");
    TransactionBean [] transactions = spec.getTransactions();
    for(int i=0; i<transactions.length; i++) {
        //begin deps
        Vector begin = transactions[i].getBeginDeps();
        if(begin.size()>0) {
            Element bdepcy = new Element("dependency");
            bdepcy.setAttribute("type", "begin");
            Element bmain = new Element("main");
            bmain.setAttribute("transaction",

```

```

        String.valueOf(transactions[i].getUniqueId()));
    bdepcy.addContent(bmain);
    for(int x=0; x<begin.size(); x++) {
        Element bdep = new Element("dependent");
        bdep.setAttribute("transaction",
            String.valueOf(
                ((TransactionBean)begin.get(x)).getUniqueId()));
        bdepcy.addContent(bdep);
    }
    deps.addContent(bdepcy);
}

//commit deps
Vector commit = transactions[i].getCommitDeps();
if(commit.size()>0) {
    Element cdepcy = new Element("dependency");
    cdepcy.setAttribute("type", "commit");
    Element cmain = new Element("main");
    cmain.setAttribute("transaction",
        String.valueOf(transactions[i].getUniqueId()));
    cdepcy.addContent(cmain);
    for(int y=0; y<commit.size(); y++) {
        Element cdep = new Element("dependent");
        cdep.setAttribute("transaction",
            String.valueOf(
                ((TransactionBean)commit.get(y)).getUniqueId()));
        cdepcy.addContent(cdep);
    }
    deps.addContent(cdepcy);
}

//abort deps
Vector abort = transactions[i].getAbortSet();
if(abort.size()>0) {
    Element adepcy = new Element("dependency");
    adepcy.setAttribute("type", "abort");
    Element amain = new Element("main");
    amain.setAttribute("transaction",
        String.valueOf(transactions[i].getUniqueId()));
    adepcy.addContent(amain);
    for(int z=0; z<abort.size(); z++) {
        Element adep = new Element("dependent");
        adep.setAttribute("transaction",
            String.valueOf(
                ((TransactionBean)abort.get(z)).getUniqueId()));
        adepcy.addContent(adep);
    }
    deps.addContent(adepcy);
}
}
structure.addContent(deps);
root.addContent(structure);
//Constraints
Element constraints = new Element("constraints");
if(spec.getIsolation().equals("full")) {
    Element dblevel = new Element("dblevel");
    constraints.addContent(dblevel);
} else {
    Element applevel = new Element("applevel");
    if(spec.getLockingMechanism().equals("user")) {
        Element user = new Element("user");
        applevel.addContent(user);
    } else {
        Element coll = new Element("collaborative");
        //conflicts
        ConflictBean [] conArray = spec.getConflicts();
        if(conArray.length > 0) {
            Element conflicts = new Element("conflicts");
            for(int c=0; c<conArray.length; c++) {
                Element conflict = new Element("conflict");
                Element op1 = new Element("operation");
                op1.setAttribute("oid",
                    String.valueOf(conArray[c].
                        getFirstConflictOp().getUniqueId()));
                conflict.addContent(op1);
                Element op2 = new Element("operation");
                op2.setAttribute("oid",

```

```

        String.valueOf(conArray[c].
            getSecondConflictOp().getUniqueId()));
        conflict.addContent(op2);
        conflicts.addContent(conflict);
    }
    coll.addContent(conflicts);
    //permits
    Element permits = new Element("permits");
    boolean anyPermits = false;
    for(int p=0; p<conArray.length; p++) {
        if(conArray[p].isPermitted()) {
            anyPermits = true;
            Element permit = new Element("permit");
            Element op1 = new Element("operation");
            op1.setAttribute("oid",
                String.valueOf(conArray[p].
                    getFirstConflictOp().getUniqueId()));
            permit.addContent(op1);
            Element op2 = new Element("operation");
            op2.setAttribute("oid",
                String.valueOf(conArray[p].
                    getSecondConflictOp().getUniqueId()));
            permit.addContent(op2);
            permits.addContent(permit);
        }
    }
    if(anyPermits) {
        coll.addContent(permits);
    }
    //demands
    DemandBean[] demArray = spec.getDemands();
    if(demArray.length > 0) {
        Element demands = new Element("demands");
        for(int d=0; d<demArray.length; d++) {
            Element demand = new Element("demand");
            OperationBean [] opArray
                = demArray[d].getOpSequence();
            for(int o=0; o<opArray.length; o++) {
                Element operation
                    = new Element("operation");
                operation.setAttribute("oid",
                    String.valueOf(opArray[o].
                        getUniqueId()));
                operation.setAttribute("seqNumber",
                    String.valueOf(o+1));
                demand.addContent(operation);
            }
            demands.addContent(demand);
        }
        coll.addContent(demands);
    }
    }
    applevel.addContent(coll);
    constraints.addContent(applevel);
}
root.addContent(constraints);

//policies
Element policies = new Element("policies");
Element rules = new Element("rules");
Vector awareness = spec.getAwarenessMechanisms();
for(int j=0; j<awareness.size(); j++) {
    Element aw = new Element("awareness");
    aw.setAttribute("type", (String)awareness.get(j));
    rules.addContent(aw);
}
policies.addContent(rules);
Element mechanism = new Element("mechanism");
mechanism.setAttribute("type", "locking");
policies.addContent(mechanism);
root.addContent(policies);

specdoc.addContent(root);
writeTree(specdoc, "Spec", path);
}

```



```

/**
 * Utility method that checks if there are any nested transactions
 * @return True if there are nested transactions, false otherwise
 */
private boolean nestedPhysType() {
    TransactionBean [] transactions = spec.getTransactions();
    for(int i=0; i<transactions.length; i++) {
        TransactionBean trans = transactions[i];
        if((trans.getParent() != null) ||
            (trans.getChildren().size() > 0))
            return true;
    }
    return false;
}
}

```

OperationBean.java

```

package no.diplom;

import java.util.Vector;

/**
 * This class represents the operations in the system,
 * including in-arguments and out-arguments
 * @author Thomas Bjorge
 */
public class OperationBean {

    private String oid;
    private int uniqueId;
    private String type;
    //Store a Vector in this one v[0]=arg and v[1]=intent
    private Vector iargs = new Vector();
    private Vector oargs = new Vector();
    private Vector browseset = new Vector();
    private TransactionBean parent;
    private String intention;

    /**
     * The constructor is empty
     */
    public OperationBean() {
    }

    /**
     * Gets the name of the operation
     * @return A String containing the name of the operation
     */
    public String getOid() {
        return oid;
    }

    /**
     * Sets the name of the operation
     * @param oid The String name of the operation
     */
    public void setOid(String oid) {
        this.oid = oid;
    }

    /**
     * Gets the operation's type
     * @return A String containing the type of the operation
     */
    public String getType() {
        return type;
    }

    /**
     * Sets the operation's type
     * @param type A String containing the type of the operation
     */
}

```

```

public void setType(String type) {
    this.type = type;
}

/**
 * Gets the IDs of the operation's in-arguments
 * @return A String array of argument IDs
 */
public String[] getIargsNames() {
    int size = iargs.size();
    String [] iargsNames = new String[size];
    for (int i=0; i<size; i++) {
        String [] tmp = (String[]) iargs.get(i);
        iargsNames[i] = tmp[0];
    }
    return iargsNames;
}

/**
 * Gets the intentions of the operation's in-arguments
 * @return A String array containing intentions,
 * either "Browse", "Incorporate" or "Modify"
 */
public String[] getIargsIntentions() {
    int size = iargs.size();
    String [] iargsIntentions = new String[size];
    for (int i=0; i<size; i++) {
        String [] tmp = (String[]) iargs.get(i);
        iargsIntentions[i] = tmp[1];
    }
    return iargsIntentions;
}

/**
 * A utility that sets a temporary intention variable
 * @param intention A String containing either
 * "Browse", "Incorporate" or "Modify"
 */
public void setIntention(String intention) {
    this.intention = intention;
}

/**
 * Adds an argument to the operation's in-set
 * @param iarg The String ID of the argument
 */
public void setIarg(String iarg) {
    String [] iargArray = new String[2];
    iargArray[0] = iarg;
    iargArray[1] = intention;
    iargs.add(iargArray);
    if(intention.equals("browse")) {
        browseset.add(iarg);
    }
}

/**
 * Gets the operation's out-arguments
 * @return An array of ArgumentBeans representing
 * the operation's out-arguments
 */
public ArgumentBean[] getOargs() {
    ArgumentBean [] oargArray = new ArgumentBean[oargs.size()];
    oargs.toArray(oargArray);
    return oargArray;
}

/**
 * Adds an argument to operation's out-set
 * @param oarg The ArgumentBean to be added to
 * the operation's out-set
 */
public void setOarg(ArgumentBean oarg) {
    oargs.add(oarg);
}

```

```

/**
 * Gets the arguments in the operation's browseset
 * @return A String array containing the IDs of the
 * arguments in the browseset
 */
public String [] getBrowseset() {
    String [] browseArray = new String[browseset.size()];
    browseset.toArray(browseArray);
    return browseArray;
}

/**
 * Sets the operation's browseset
 * @param browseset A vector containing the ArgumentBeans
 * in the browseset
 */
public void setBrowseset(Vector browseset) {
    this.browseset = browseset;
}

/**
 * Gets the operations parentTransaction
 * @return The transactionBean that is the operation's parent
 */
public TransactionBean getParent() {
    return parent;
}

/**
 * Sets the operation's parent transaction
 * @param parent The operation's parent TransactionBean
 */
public void setParent(TransactionBean parent) {
    this.parent = parent;
    parent.addOperation(this);
}

/**
 * Gets the operation unique ID
 * @return The unique integer ID
 */
public int getUniqueId() {
    return uniqueId;
}

/**
 * Sets the operation's unique ID
 * @param uniqueId The unique integer ID
 */
public void setUniqueId(int uniqueId) {
    this.uniqueId = uniqueId;
}
}

```

SpecificationBean.java

```

package no.diplom;

import java.util.HashMap;
import java.util.Map;
import java.util.Vector;
import javax.servlet.ServletContext;

/**
 * This is the application's main class. It holds all variables
 * set during the specification and also has pointers to all the
 * transactions, operations, arguments, conflicts and demands
 * @author Thomas Bjorge
 */
public class SpecificationBean {

    private Map transactions = new HashMap();
    private Map operations = new HashMap();
    private Map arguments = new HashMap();
}

```

```

private String specName;
private String atomicity;
private String isolation;
private String lockingMechanism;
private Vector awarenessMechanisms = new Vector();
private String selectedTrans;
private Vector conflicts = new Vector();
private Vector demands = new Vector();
private DemandBean currentDemand;
private ServletContext sc;

/**
 * Empty constructor
 */
public SpecificationBean() {
}

/**
 * Gets all of the transactions specified
 * @return An array containing all existing TransactionBeans
 */
public TransactionBean[] getTransactions() {
    TransactionBean [] transArray =
        new TransactionBean[transactions.size()];
    transactions.values().toArray(transArray);
    return transArray;
}

/**
 * Adds a transaction to the list of all transactions
 * @param trans The TransactionBean to be added
 */
public void setTransaction(TransactionBean trans) {
    transactions.put(Integer.toString(trans.getUniqueId()),trans);
}

/**
 * Gets all the transactions specified
 * @return A HashMap containing all existing transactions
 */
public Map getTransactionsById() {
    return transactions;
}

/**
 * Gets all existing operations
 * @return An array of all existing OperationBeans
 */
public OperationBean [] getOperations() {
    OperationBean [] opArray = new OperationBean[operations.size()];
    operations.values().toArray(opArray);
    return opArray;
}

/**
 * Gets all existing operations
 * @return A HashMap containing all existing operations
 */
public Map getOperationsById() {
    return operations;
}

/**
 * Adds an operation to the list of all operations
 * @param op The OperationBean to be added
 */
public void setOperation(OperationBean op) {
    operations.put(Integer.toString(op.getUniqueId()),op);
}

/**
 * Gets all existing arguments
 * @return An array of all existing ArgumentBeans
 */
public ArgumentBean [] getArguments() {

```

```
        ArgumentBean [] argArray = new ArgumentBean[arguments.size()];
        arguments.values().toArray(argArray);
        return argArray;
    }

    /**
     * Gets all existing arguments
     * @return A HashMap containing all existing arguments
     */
    public Map getArgumentsById() {
        return arguments;
    }

    /**
     * Adds an argument to the list of all arguments
     * @param arg The ArgumentBean to be added
     */
    public void setArgument(ArgumentBean arg) {
        arguments.put(arg.getId(), arg);
    }

    /**
     * Gets javascript code from the <CODE>GenerateTree</CODE> class
     * @return A String containing the javascript code
     */
    public String getJsTreeCode() {
        GenerateTree treegen = new GenerateTree(this);
        return treegen.getJsCode("");
    }

    /**
     * Gets javascript code from the <CODE>GenerateTree</CODE> class.
     * The treetype is set to "b"
     * @return A String containing the javascript code
     */
    public String getJsBeginTreeCode() {
        GenerateTree treegen = new GenerateTree(this);
        return treegen.getJsCode("b");
    }

    /**
     * Gets javascript code from the <CODE>GenerateTree</CODE> class.
     * The treetype is set to "c"
     * @return A String containing the javascript code
     */
    public String getJsCommitTreeCode() {
        GenerateTree treegen = new GenerateTree(this);
        return treegen.getJsCode("c");
    }

    /**
     * Gets javascript code from the <CODE>GenerateTree</CODE> class.
     * The treetype is set to "a"
     * @return A String containing the javascript code
     */
    public String getJsAbortTreeCode() {
        GenerateTree treegen = new GenerateTree(this);
        if(isolation.equals("full"))
            return treegen.getJsCode("a", selectedTrans);
        else
            return treegen.getJsCode("a");
    }

    /**
     * Gets the name of the current specification as
     * entered by the user in the first step
     * @return A String containing the specification name
     */
    public String getSpecName() {
        return specName;
    }

    /**
     * Sets the name of the specification
     * @param specName A String containing the specification name
     */

```

```
public void setSpecName(String specName) {
    this.specName = specName;
}

/**
 * Gets the atomicity value for the specification
 * @return A string with value "full" or "relaxed"
 */
public String getAtomicity() {
    return atomicity;
}

/**
 * Sets the atomicity of the specification
 * @param atomicity A string with value "full" or "relaxed"
 */
public void setAtomicity(String atomicity) {
    this.atomicity = atomicity;
}

/**
 * Gets the isolation value for the specification
 * @return A string with value "full" or "relaxed"
 */
public String getIsolation() {
    return isolation;
}

/**
 * Sets the isolation of the specification
 * @param isolation A string with value "full" or "relaxed"
 */
public void setIsolation(String isolation) {
    this.isolation = isolation;
}

/**
 * Gets the locking mechanism specified
 * @return A String with value "user" or "collaborative"
 */
public String getLockingMechanism() {
    return lockingMechanism;
}

/**
 * Sets the locking mechanism for this specification
 * @param lockingMechanism A String with value
 * "user" or "collaborative"
 */
public void setLockingMechanism(String lockingMechanism) {
    this.lockingMechanism = lockingMechanism;
}

/**
 * Gets the awareness mechanisms for this specification
 * @return A Vector containing any combination of the Strings:
 * "begin", "terminate", "lock", "change"
 */
public Vector getAwarenessMechanisms() {
    return awarenessMechanisms;
}

/**
 * Adds an awareness mechanism to the list of mechanisms
 * @param mechanism A String with value
 * "begin", "terminate", "lock" or "change"
 */
public void setAwarenessMechanism(String mechanism) {
    awarenessMechanisms.add(mechanism);
}

/**
 * Calls the <CODE>CreateXML</CODE> class
 * @param path This is the physical filesystem path to the application
 */
public void setXml(String path) {
    GenerateXML gxml = new GenerateXML(this);
}
```

```
        gxml.createXml(path);
    }

    /**
     * Gets a temporary variable containing a selected transaction
     * @return The String ID of the selected transaction
     */
    public String getSelectedTrans() {
        return selectedTrans;
    }

    /**
     * Gets a temporary variable containing a selected transaction
     * @param selectedTrans The String ID of the selected transaction
     */
    public void setSelectedTrans(String selectedTrans) {
        this.selectedTrans = selectedTrans;
    }

    boolean conflictsFetched = false;
    /**
     * Calls the <CODE>Conflicts</CODE> class to check for conflicts
     * the first time the method is called. Any subsequent calls just
     * return the conflicts generated in the first call
     * @return An array of all ConflictBeans
     */
    public ConflictBean[] getConflicts() {
        if(!conflictsFetched) {
            Conflicts c = new Conflicts(this);
            c.checkForConflicts();
            conflictsFetched = true;
        }
        ConflictBean[] conflictArray = new ConflictBean[conflicts.size()];
        conflicts.toArray(conflictArray);
        return conflictArray;
    }

    /**
     * Gets the number of conflicts
     * @return The integer number of conflicts
     */
    public int getConflictsSize() {
        return conflicts.size();
    }

    /**
     * Adds a conflict to the list of all conflicts
     * @param conflict The ConflictBean to be added
     */
    public void setConflict(ConflictBean conflict) {
        conflicts.add(conflict);
    }

    /**
     * Adds a demand to the list of all demands
     * @param demand The DemandBean to be added
     */
    public void setDemand(DemandBean demand) {
        demands.add(demand);
        currentDemand = null;//we just added the current instance
    }

    /**
     * Gets all existin demands
     * @return An array of all existin DemandBeans
     */
    public DemandBean[] getDemands() {
        DemandBean [] demandArray = new DemandBean[demands.size()];
        demands.toArray(demandArray);
        return demandArray;
    }

    /**
     * Gets the current demand. The current demand is a temporary
     * variable used by <CODE>Demands.jsp</CODE>
     * @return The current DemandBean
     */
    public DemandBean
```

```

public DemandBean getCurrentDemand() {
    if(currentDemand == null) {
        currentDemand = new DemandBean();
    }
    return currentDemand;
}

/**
 * Sets the current demand. The current demand is a temporary
 * variable used by <CODE>Demands.jsp</CODE>
 * @param currentDemand The current DemandBean
 */
public void setCurrentDemand(DemandBean currentDemand) {
    this.currentDemand = currentDemand;
}

/**
 * Returns HTML links to the two xml files generated by the application
 * @return A String array with the two links
 */
public String[] getXMLfileLinks() {
    String [] links = new String[2];
    String specPath = "xml" +
        System.getProperty("file.separator") +
        specName + "Spec.xml";
    String execPath = "xml" +
        System.getProperty("file.separator") +
        specName + "Exec.xml";
    String specLink = "<a href=\"" + specPath + "\">" +
        specName + "Spec.xml" + "</a>";
    String execLink = "<a href=\"" + execPath + "\">" +
        specName + "Exec.xml" + "</a>";
    links[0] = specLink;
    links[1] = execLink;
    return links;
}

/**
 * Sets the application's ServletContext which is needed by getAppPath
 * @param sc The ServletContext object
 */
public void setServletContext(ServletContext sc) {
    this.sc = sc;
}

/**
 * Gets the path to the application's directory on the server
 * @return A String with the path
 */
public String getAppPath() {
    return sc.getRealPath("/");
}
}

```

TransactionBean.java

```

package no.diplom;

import java.util.Vector;

/**
 * This class represents the transactions in the application
 * @author Thomas Bjorge
 */
public class TransactionBean {

    private String trid;
    private String endType;
    private TransactionBean parent;
    private String physType;
    private Vector operations = new Vector();
    private Vector abortSet = new Vector();
}

```



```
private Vector beginDependencies = new Vector();
private Vector commitDependencies = new Vector();
private int uniqueId;
private Vector children = new Vector();

/**
 * Constructor initially sets parent of transaction to null
 */
public TransactionBean() {
    parent = null;
}

/**
 * Gets the name of the transaction
 * @return The String name
 */
public String getTrid() {
    return trid;
}

/**
 * Sets the name of the transaction
 * @param trid The String name
 */
public void setTrid(String trid) {
    this.trid = trid;
}

/**
 * Gets the endtype of the transaction
 * @return A String with value "commit" or "abort"
 */
public String getEndType() {
    return endType;
}

/**
 * Sets the endtype of the transaction
 * @param endType A String with value "commit" or "abort"
 */
public void setEndType(String endType) {
    this.endType = endType;
}

/**
 * Gets the physical type of the transaction
 * @return A String with value "flat" or "nested"
 */
public String getPhysType() {
    return physType;
}

/**
 * Gets the physical type of the transaction
 * @param physType A String with value "flat" or "nested"
 */
public void setPhysType(String physType) {
    this.physType = physType;
}

/**
 * Gets all operation that have this transaction as their parent
 * @return A Vector containing OperationBeans
 */
public Vector getOperations() {
    return operations;
}

/**
 * Adds an operation to the transaction's list of operations
 * @param op The OperationBean to be added
 */
public void addOperation(OperationBean op) {
    operations.add(op);
}

/**
```

```
* Gets the abortset of this transaction, i.e. all transactions
* that must abort if this transaction aborts
* @return A Vector containing the TransactionBeans in the abortset
*/
public Vector getAbortSet() {
    return abortSet;
}

/**
 * Adds a transaction to this transaction's abortset
 * @param dependent The TransactionBean to be added
 */
public void setAbortDep(TransactionBean dependent) {
    abortSet.add(dependent);
}

/**
 * Gets all transactions that are begin dependent on this transaction,
 * i.e. all transactions that must start after this transaction
 * @return A Vector containing TransactionsBeans
 */
public Vector getBeginDeps() {
    return beginDependencies;
}

/**
 * Adds a transaction to this transaction's begindep set
 * @param dependent The TransactionBean to be added
 */
public void setBeginDep(TransactionBean dependent) {
    beginDependencies.add(dependent);
}

/**
 * Gets all transactions that are commit dependent on this transaction,
 * i.e. all transactions that must commit after this transaction
 * @return A Vector containing TransactionsBeans
 */
public Vector getCommitDeps() {
    return commitDependencies;
}

/**
 * Adds a transaction to this transaction's commitdep set
 * @param dependent The TransactionBean to be added
 */
public void setCommitDep(TransactionBean dependent) {
    commitDependencies.add(dependent);
}

/**
 * Gets this transaction's parent
 * @return The TransactionBean that is the parent,
 * or null if this transaction is a root transaction
 */
public TransactionBean getParent() {
    return parent;
}

/**
 * Sets this transaction's parent transaction
 * @param parent The TransactionBean that is the parent
 */
public void setParent(TransactionBean parent) {
    this.parent = parent;
    parent.addChild(this);
}

/**
 * Gets this transaction's unique ID
 * @return The integer ID
 */
public int getUniqueId() {
    return uniqueId;
}

/**
```

```
    * Sets this transaction's unique ID
    * @param uniqueId The integer ID
    */
    public void setUniqueId(int uniqueId) {
        this.uniqueId = uniqueId;
    }

    /**
     * Adds a transaction to this transaction's list of children
     * @param child The Transactionbean to be added
     */
    public void addChild(TransactionBean child) {
        children.add(child);
    }

    /**
     * Gets this transaction's children transactions
     * @return A Vector containing the child TransactionBeans
     */
    public Vector getChildren() {
        return children;
    }
}
```

UniqueIdBean.java

```
package no.diplom;

import java.beans.*;

/**
 * This is a simple utility class that generates unique IDs for
 * transactions and operations
 * @author Thomas Bjorge
 */
public class UniqueIdBean {
    private int ID;

    /**
     * Sets the initial ID to 1
     */
    public UniqueIdBean() {
        ID = 1;
    }

    /**
     * Gets a unique ID by incrementing the previous one by 1
     * @return The unique integer ID
     */
    public int getID() {
        return ID++;
    }
}
```