Transformation of DEMO models into exchangeable format

Master's Thesis in Computer Science

by

Yan Wang

14th April 2009

Delft University of Technology

Information System Design group
Faculty of EEMCS
Delft University of Technology
Delft,The Netherlands
www.ewi.tudelft.nl

Author

Candidate:	Yan Wang
Student number:	1301080
Email:	y.wang124@gmail.com

Title

Transformation of DEMO models into exchangeable format

MSc presentation

22nd April 2009

Graduation Committee

Prof. dr. ir. J.L.G. Dietz (chair)	Information Systems Design Group, EEMCS Delft University of Technology
Dr. A. Albani (supervisor)	Information Systems Design Group, EEMCS Delft University of Technology
Dr. J. Barjis (member)	Systems Engineering, TBM Delft University of Technology

Abstract

Globalized business development requires enterprises to have more flexible and interoperable information systems. The use of reusable and marketable business components has proved valuable for the development of a high-level information system [6]. Business Component Identification (BCI) is the first step and a crucial one in the development of an information system. A well defined business domain modeling is demanded to provide the requisite information for identifying business components. The DEMO methodology, which satisfies the requirements to be a well defined domain modeling for BCI, functions at a high level of abstraction and models the essence of an organization. However the gap between platform independent DEMO models and platform specific applications for BCI requires extra human efforts. This master's thesis seeks to transform DEMO models into an exchangeable format, which will be beneficial to BCI. The model transformation approach used in this research is adopted from Model Driven Architecture (MDA), and metamodeling is used in the transformation.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

Many people would agree that it is reckless to execute an implementation without a holistic and strategic plan, no matter whether it is about building a mansion or launching the development of an information system. However, it is also true that making an ideally perfect but not implementable plan would be just wishful indulgence in reverie. Experience tells us how important a smooth and effective connection is, to bridge the distance between blueprint and operation. This also applies to developments in the field of software engineering.

1.1 Problem Statement

As worldwide development continues to globalize and integrate, enterprises must rethink and reengineer their entire business process, which includes their organizational structure, staffing, and especially their information systems and technology infrastructures on an international level [21]. As recognized by Malone [23], transactions within organizations will become indistinguishable from transactions between organizations; organizational boundaries will become much less important, and business processes will cross the boundaries between once proprietary enterprises, which means more cooperation is demanded. With the increasing business scale, companies are encountering more complex situations, such as globalized sales and sourcing markets, shortened product life cycles, and innovative pressure on products, services and processes [7]. In order to stay in and win the game in the competitive business world, enterprises need to adapt to the fast changing market quickly and expand their cooperative relationships with their business partners.

Regarding the globalized and drastic business competition, flexibility and interoperability are crucial factors for enterprises. Flexibility promises the enterprise an agile and quick adjustment to both organizational and technical structure, and releases the constraints from traditional stereotypes, complying with certain business strategies of the enterprise. Interoperability, from the perspective of information technology, enables communication among diverse systems and organizations, which provides the possibility to increase the effectiveness of applications.

Since the first use of information technology for business applications a half-century ago, the business value and impact of information technology on organizations has been increasing, as reviewed by Mooney [16], “the business value of information technology is a joint of technology organization phenomenon, and it requires theoretical perspectives of both technology and organizations, and their interaction”. Thus while developing information systems for enterprises, it is necessary to have a suitable methodology for modeling the business domain. The fact of modeling the business domain implies that the information systems need to be modeled at a high-level of information abstraction, which can be understood by business people who define the organizational requirements. The concept of reusable and marketable business components is proposed for building adaptive and agile information systems for enterprises [22]. The use of business components in deploying high level information systems has been proved valuable by [6], since they “directly model and implement the business logic, rules and constraints that are typical, recurrent and comprehensive notions characterizing a domain or business area”.

Proposed in [5], Business Component Identification (BCI) is the first and a crucial step in the business component modeling process. It starts by generating a set of essential elements from the business domain, preferably on the basis of a high-level business domain model, which abstracts the organizational activities. The appropriateness and the quality of the business domain model are vital to the information elicited from the business domain. Dietz proposes some quality criteria regarding a business domain model in [9], which are *coherence* (the domain models constitute a logical and truly integral whole), *comprehensiveness* (complete coverage of all relevant issues), *consistency* (the domain models are free of contradictions or irregularities), *conciseness* (all relevant models are compact and succinct), and *essence* (the domain model should only show the essence of the enterprise). In addition, he also elaborates a well-defined methodology called DEMO (Design and Engineering Methodology for Organizations), which satisfies all of the mentioned quality criteria.

DEMO is a promising modeling paradigm to represent the essence of an organization. Compared with other existing process modeling techniques, such as Petri Net, Event Driven Process Chains (EPC) and Activity Diagrams, which do not define the business process well and do not distinguish the business and informational actions [4], DEMO analyzes the organizational activities at a high level of abstraction, purely from the business domain, and distinguishes between business, informational and documental actions. DEMO provides four related aspect models, which are the construction model (represents the organizational construction), the process model (represents the interrelations within and between the transactions), the action model (includes all the action rules), and the state model (represents the allowable states of the production world and the coordination world of the enterprise). By providing deep insights into the essential structure of the business process within an enterprise, DEMO can be chosen for producing the business domain models of an enterprise, in preparation for the BCI.

One of the methods for identifying the business component is *the three dimensional method for business components identification* (BCI-3D), which "aims at grouping business tasks and their corresponding information objects into business components" [4]. Choosing DEMO methodology for producing the business domain models, the information required by BCI-3D about the organizational activities is all elicited from the DEMO aspect models.

As introduced in [5], the BCI-3D method uses information objects and process steps from the DEMO aspect models, including their relationships. However the tool for implementing the BCI-3D method cannot generate the mentioned information directly from the graphical DEMO aspect models. Currently, all the information for identifying the business component is generated manually, due to the inability of communication between different platforms of the BCI-3D tool and the DEMO models. The DEMO aspect models are high-level conceptual models, and its modeling procedure does not take any implementation related question into consideration. Being separated from specific applications guarantees that DEMO models are platform independent, but also makes a gap to the specific platform applications. After establishing the DEMO models, it requires extra human effort to transfer the platform independent model information into another format which can be used by the BCI-3D tool. Therefore, an automatic identification of business components is demanded, which is "without the need of manually transforming the model information into the representation needed in order to apply the BCI-3D method" [5]. More specifically speaking, an automatic transformation of the DEMO model information, from the original graphical notations to an exchangeable format which can be accessed by other applications, such as the BCI-3D tool, is initiated.

1.2 Research Questions

As already mentioned in the problem statement (Chapter 1.1), the conceptual DEMO models are separated from any specific platform. The gap between the platform independent DEMO models and the platform specific programmable models complicates the procedure of identifying the business components. It is required to bridge the gap with smooth transformation from the graphical DEMO models into the ones that can be applied in a specific platform. Therefore this raises the main research question of this graduation project as below:

How can graphical DEMO models be transformed into an exchangeable format, without information loss, that is readable by third party applications?

With the purpose of answering the above-mentioned research question, some prerequisite work has been done in [27]. The Object Management Group (OMG) proposes a design approach for developing software systems, Model Driven Architecture (MDA), in which a model transformation process is defined. The transformation is from the Platform Independent Model (PIM) to Platform Specific Model (PSM), under the guidance of some transformation rules. The metamodel, which represents the models, and the metamodeling, which is the act and science of engineering a metamodel, are also introduced by the OMG and play an important role in MDA model transformation. By using metamodels, the MDA model transformation is elaborated at a higher level of abstraction, the level of metamodel [17].

Resulting from our previous research [27], in which several existing modeling paradigms have been studied, we choose the XML Schema language [24] as the target format for the transformed model. Inspired by MDA model transformation and metamodeling, we will seek a probable way of transforming the DEMO models by applying the MDA model transformation approach.

In the expected transformation of the DEMO models, by applying the MDA model transformation approach, the DEMO aspect models are the source PIM, and the target PSM will be in the XML based format. The actual transformation will be elaborated at the level of metamodel, which implies that the metamodel of the DEMO models is the crucial factor in our research. The expected transformation should be conducted under guidance, so that the transformation rules need to be defined for the entire transformation. Since we choose the XML Schema as the target format for the PSM, there should be a corresponding metamodel for the PSM in XML Schema, and the concrete transformation work should bridge the gap between the PIM (DEMO models) and the PSM (XML based model). As completion of the transformation, verifying the transformation result is also necessary. In addition, an example of the utilizing the transformation result by other applications would be convincing evidence to show that the whole transformation would be helpful to the existing problem mentioned in Chapter 1.1.

Regarding the outline of the expected transformation mentioned above, we divide the main research question into several sub-questions, and the answers to those questions are intended to realize the expected model transformation step by step. The sub-questions and corresponding tasks are listed in Table 1.1. The explanations for each subtask will be made in Section 1.3.

Table 1.1: Research questions and corresponding tasks

	Research Questions	Tasks
1	How would the DEMO metamodel be constructed?	Understand and analyze DEMO metamodel.
2	How can the transformation rules be defined?	Define the transformation rules with an understanding of the demands of transforming the DEMO and the knowledge of the DEMO metamodel.
3	How can we transform the graphical model information into a syntax-based format?	Design XML schema for DEMO metamodel.
4	What would the transformed model look like?	Produce the instance XML documents with model information.
5	How can we verify the result of the transformation?	Compare the produced instance XML documents with the original diagrams.
6	How can the transformed model be used by other applications?	Construct the Create / Use Table.

1.3 Research Approach

The graduation project covers a period of 11 months, starting in May, 2008 (excluding the seven-week literature research), and was completed in April, 2009. The research approach is proposed in Figure 1.1, and the research steps in this approach correspond to the research questions and tasks mentioned in Table 1.1. The research contents contained in this approach are explained below.

1. DEMO metamodel construction
The research starts with acknowledging the metamodeling approach in model transformation, and analyzing the DEMO metamodel [1]. As the outcome of this step, an adequate knowledge about the role of metamodeling in model transformation is obtained by analyzing the DEMO metamodel. The acknowledgement of metamodeling is considered a requisite foundation to steer the whole research. A full understanding of the DEMO metamodel prepares us with the clear perception of the essential structure of the DEMO aspect models. By accomplishing this step, it is supposed to answer the first research question (see Table 1.1), "How will the DEMO metamodel be constructed?"

2. Transformation analysis
Specifically regarding the expected transformation in this research, mentioned in Chapter 1.2, analysis of the demands and requirements to transforming the DEMO models is made in this step. We will address the reasons that raise the demands to the transformation and the desired transformation result. In addition, some transformation rules will be defined based on the mentioned analysis and our knowledge

[1] The DEMO metamodel is provided by Prof. Dr. Ir. Jan.L.G.Dietz.

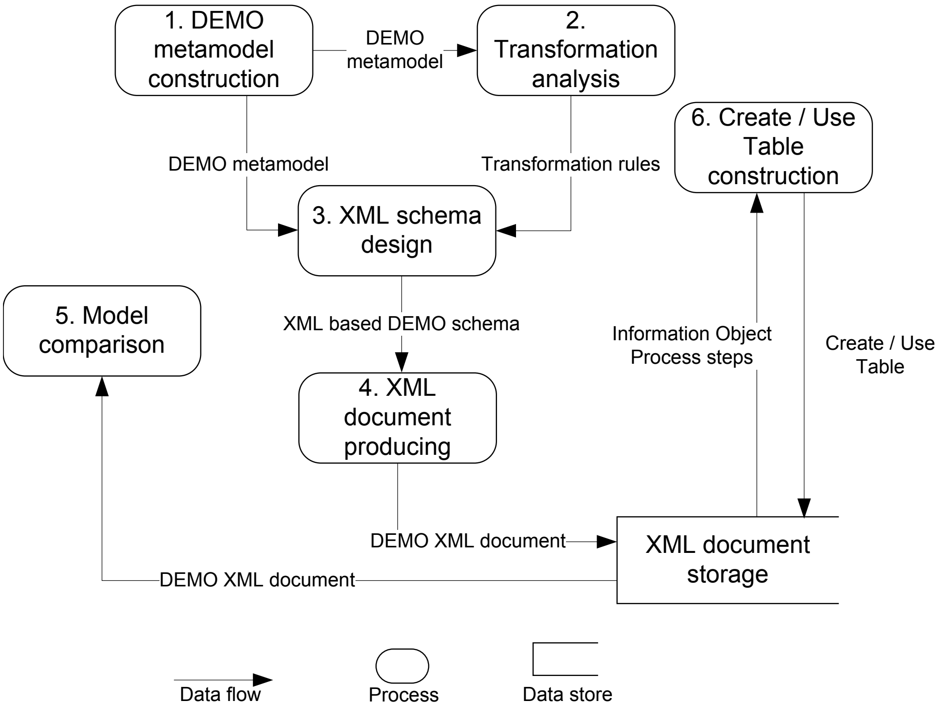


Figure 1.1: Thesis project research approach

about the DEMO metamodel. By accomplishing this step, we answer the second research question (see Table 1.1), "How can the transformation rules be defined?"

3. XML schema design

A schema for the DEMO metamodel is designed in XML schema language in this step (the specific schema language is chosen in [27]). The schema is the interpretation of the DEMO metamodel, and will cover the essential structure of DEMO models. The transformation rules will be used in the schema design. By accomplishing this step, we answer the third research question (see Table 1.1), "How can we transform the graphical model information into a syntax-based format".

4. XML document producing

After accomplishing the schema design for the DEMO metamodel, a set of instance XML documents are produced in this step, against the designed XML schema. The content of the produced instance XML documents are from the DEMO aspect model information from a case study. The elaboration of producing the XML documents is supported by an application (see Appendix E), and indicates that the graphical DEMO models are transformed into a syntax-based format. By accomplishing this step, we answer the fourth research question (see Table 1.1), "What would the transformed model look like?"

5. Model comparison
This step will verify the results of the expected transformation. We will compare the produced instance XML documents, which are the transformed DEMO models from the previous step, with the original DEMO models. The information contained in both models will be mapped with each other in order to see the information completeness and correctness during the transformation. The result of this comparison is used in our evaluation of the model transformation. By accomplishing this step, we answer the fifth research question (see Table 1.1), "How can we verify the result of the transformation?"

6. Create / Use Table construction
In this step, we will produce the Create / Use Table, which can be used by the BCI-3D tool to identify the business components. The information used to create this table is directly obtained from the instance XML documents, which are produced as the result of our proposed DEMO model transformation. The elaboration of producing this table is supported by an application (see Appendix E). The producing of the Create / Use Table shows that the result of our proposed DEMO model transformation can be used by other applications and helps generate the information for BCI. By accomplishing this step, we answer the sixth research question (see Table 1.1), "How can the transformed model be used by other applications?"

1.4 Report Structure

An overview of this thesis is given in this section. In Chapter 2, background information on DEMO and business component is provided. In addition, the theory of metamodeling and model transformation is introduced, which illustrates the importance of metamodeling in model transformation. A short review of the characteristics of our chosen schema language for the transformed model, the XML Schema language, is made as well in this chapter.

A specification of the DEMO metamodel is provided in Chapter 3. The DEMO metamodel provides an essential graphical schema for DEMO models. The first research question is answered in this chapter.

In Chapter 4, we analyze the requirements to transform the DEMO models; with the analysis and knowledge on the DEMO metamodel, we define the transformation rules that are used throughout the entire transformation. The second research question is answered in this chapter.

Following the transformation rules, the corresponding XML specification of the DEMO metamodel is made in Chapter 5. The XML specification interprets the DEMO metamodel and defines the XML schema for DEMO models. The third research question is answered in this chapter.

In Chapter 6 we produce the instance XML documents based on the designed schema for the DEMO metamodel. The produced XML documents are consid-

ered the result of the proposed transformation in this research. The fourth research question is answered in this chapter.
In order to evaluate the entire DEMO transformation, we compare the produced instance XML documents with the original DEMO models, and produce the Create / Use Table in Chapter 7. The comparison between the XML documents and the original DEMO diagrams verifies the transformation results. The production of the Create / Use Table is an example to show how the transformation result could be used by third party applications. The last two research questions are answered in this chapter.
At last, Chapter 8 recaps the research content of this project, emphasizes the scientific foundation that the research is based on, and evaluates the project work from four aspects, which focus on the transformation contents, the transformation of the chosen contents, the verification of the transformation results and the added value of the transformation result. In addition, some relevant issues for future work are also discussed in this chapter.

Chapter 2

Background

This chapter aims to provide necessary background information concerning this graduation project. Business Component Modeling Process (BCMP) and DEMO methodology are explained in Chapter 2.1 and 2.2. The concepts of metamodeling and model transformation are introduced in Chapter 2.3, which provides the theoretical foundation that guide the research work. Chapter 2.4 recaps the characteristics of the XML Schema that prepares us with the knowledge of this schema language, in which the graphical DEMO models will be transformed.

2.1 Business Components Modeling Process

A component is a reusable, self-contained, and marketable piece of software, which provides services through a well-defined interface and which may be deployed in configurations unknown at the time of development. A business component is a component of an information system that supports directly the activities in an enterprise. In the process of Business Component Modeling (BCM), there are three phases, Component Based Domain Analysis, Domain Based Component Realization and Components Composition. It starts with the domain modeling and identification of business component, which are in the analysis phase. [3]

The domain modeling is intended to provide the information of an enterprise in the business domain, by making a reference model within defined scope. The identification of business component is based on the information provided by the domain model. There are three design principles forming the base for the identification:

- Within a business component, coherent domain functionality must be clustered.
- Business components must be coupled loosely.
- Related information objects must be managed within one business component.

The above principles form the base of the formal business components identification method, BCI-3D. It is required that information objects corresponding to certain business tasks must be grouped together, the functionality performed by the formed business component must be coherent and consistent. Each business component is self-contained, namely individual, but should have well-defined interface to the environment.

The above requirements imply that the crucial measurement for grouping those information objects are relationships between them. The kinds of relationships are influenced by the process steps in which the information objects are involved. The more coherent process steps are, the stronger relationship there is. Different relationships indicate the relevant information objects and process steps within a component. These information objects, process steps and the relationships must be generated from an abstraction of an enterprise in a valid level, comprehensively and correctly.

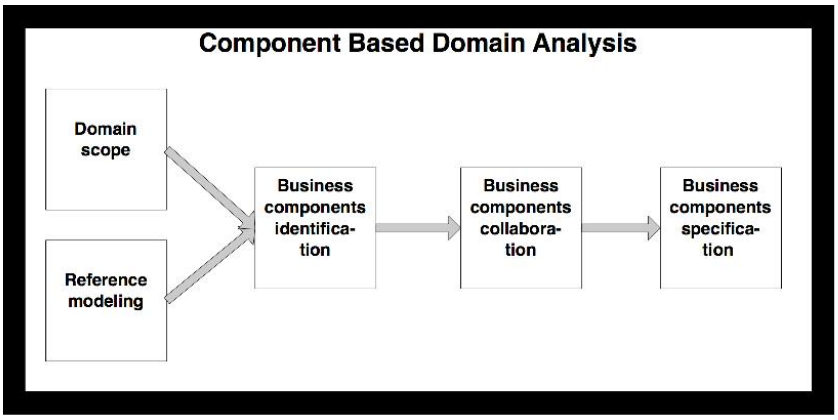

Component Based Domain Analysis
Domain scope
Reference modeling
Business components identification
Business components collaboration
Business components specification

Figure 2.1: Component Based Domain Analysis [3]

2.2 DEMO Methodology

DEMO methodology provides a way of representing the essence of an enterprise on an ontological level. The essential information is visualized in a set of models, each of which expresses different aspects of the certain enterprise. The models, which are Construction Model (CM), Process Model (PM), Action Model (AM) and State Model (SM), are built correlated with each other, containing coherent information in a platform-independent way.

CM specifies the construction of the organization, it declares the actor roles and the transactions between them, the information links between the actor roles and the information banks. Due to the activeness and the passiveness of the influence

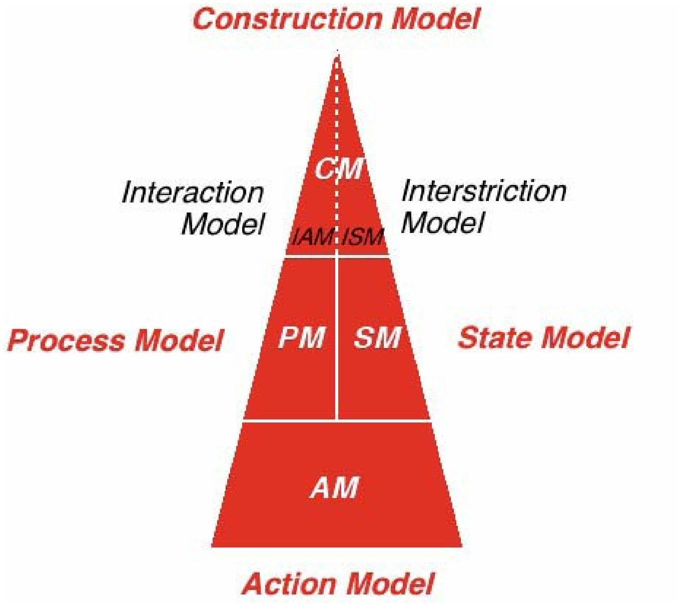


Figure 2.2: The ontological aspect models

between actor roles, the CM is divided into interaction model (IAM) and interstriction model (ISM). IAM shows the interaction structure of the organization, which consists of the transaction types and the partaking actor roles, initiator and executor. ISM contains the internal and external information banks, as well as the information links between the actor roles and the information banks.
PM contains the specific transaction pattern for each transaction in CM. The contained transaction pattern is comprehensive, including basic pattern, cancellation pattern, as well as the causal and conditional relationships between transactions.
AM is the rule base that serves as guidelines for the actors in dealing with their agenda. The action rules contain all the information from CM, PM, and SM, which means all the information about the enterprise's business is covered in this model.
SM specifies the object classes, fact types, result types, and the ontological coexistence rules. This model is constructed around the main object types, which are the variables in the result types of the transaction types. [9]

The information contained in the aspect models is valid and available for perceiving the architecture of the certain enterprise. All the above models constitute an essential representation of an enterprise in the business domain, with specifying the information objects and their relationships successfully. DEMO is a well-defined methodology in the business domain.

2.3 Metamodeling and Model Transformation

2.3.1 Metamodeling

Metamodeling, similar with modeling, is the act and science of engineering metamodels. Metamodel is a specific kind of model. With the prefix “meta-”, which means “higher” or “posterior” in Greek, and here we use it to mean “about its own category”, the metamodel is built to represent other models. [8]
Regarding to the characteristics of models, a model is to represent a part of the real-world in an abstract, understandable, accurate, predictive and inexpensive way [20]. As metamodel is a kind of model, it implies that the five characteristics also apply to the metamodel, just with different levels of information abstraction due to the contents modeled in metamodel.
The subject, the metamodel represents, is models. More explicitly, the metamodel states what can be expressed and how it should be structured in models. It specifies the concepts, rules and syntax that are used in models. The metamodel is concerning nothing about the contents in instance models, but the essential construction and modeling language of the models.

The relationship between a model and its metamodel is always “instance-of” [8], which means the model is built upon its metamodel, and embodying the structure, defined in metamodel, with concrete contents. The structures the model construction follows, the language used in building the model, even the syntax adopted by the modeling are all included in the metamodel and specified during the metamodeling.

Let us take an example to explain the relationship between the metamodel and model in a more understandable way. The real world is various and comprehensive. The models, which abstract and represent a part of the real world, are diverse in either type or content. Models of the cars are totally different from models of the pianos. Even within the car models, the model of a sports car differs from the model of a pony car. The instance models are quite miscellaneous items with concrete contents.
However, regardless of the diversity in instance models, there are some generic rules that could be generated to define the same sort of models. The cars, no matter what specific type they are, are made based on similar components and principles of work. When learning the principle and components used for making a car, it does not refer to any specific type of cars. With deepening the knowledge, it would detail the classification and embody the representation with concrete and specific features of cars. Thus the car models are the artifacts resulted from instantiating the car metamodel.

The classical metamodeling architecture, proposed by Object Management Group (OMG), is a layered hierarchy. Depicted in Figure 2.3, it is formed with four layers with different levels of information abstraction. M0 contains the data from the real-world. M1 is the model that represents the data of the real-world in layer M0. The metamodel in M2 specifies the concepts and constraints of M1. The concepts used in M2 are defined in M3. Within M3, the Meta-Object Facility (MOF) is the metamodel of the metamodel in M2. There is no more layer above M3, which indicates that there is no higher level of information abstraction.

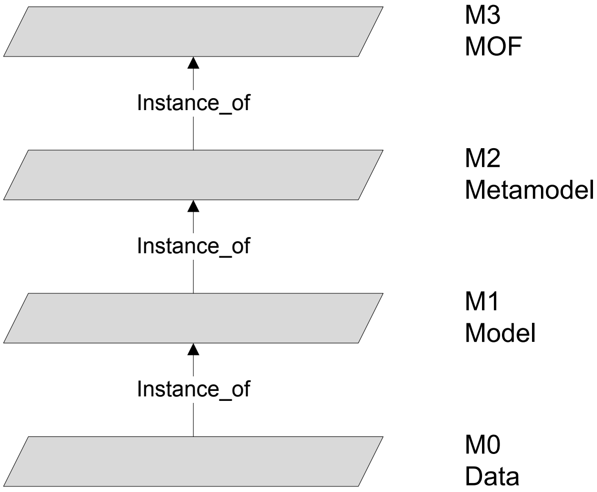


Figure 2.3: The OMG four-layer hierarchy [8]

The relationship "instance-of", according to [8], implicates that every entity in any layer Mx is instantiated from the entity in its above layer Mx+1. One entity in the higher layer could have numerous instance models in the lower layer. No matter in which layer, the one in a higher layer is the metamodel of the one in a lower layer. This relationship ties the models and the metamodels up with each other.

In principle, the layering of this architecture could be continued to infinity, as long as the concepts used in layer Mn are defined in layer Mn+1. However we need it to stop at some point to make more sense in practice. The OMG architecture stops at the level which is defined by its own concepts, namely metacircular [10]. It does not require additional concepts, but just uses existing concepts to express the models in each layer, including themselves. [12] argues about the number of layers in the OMG four layer architecture, and proves that the metamodel in M3 is the same as the metamodel in M2, since the metamodel is self-defined already in M2.
The number of the layers is not the main concern in this chapter. Our attention is paid on the relationship between model and metamodel, and the idea of the layered metamodeling architecture. The "instance-of" relationship moves the focus of modeling from concrete models to higher abstracted metamodel. Metamodeling stands at a higher level of information abstraction, views and constructs the

essential concepts of modeling.

2.3.2 Model Transformation

When developing systems, there are several different viewpoints to vision a system [10], namely, we can model the system from different levels of information abstraction. In Model Deriven Architecture (MDA), it proposes three viewpoints [11], which are computation independent viewpoint, platform independent viewpoint and platform specific viewpoint.

The perspectives of these three viewpoints are from different levels of information abstraction. In the viewpoint of independent computation, it is about the environment of the system and the requirements for the system. The structure or processing details of the system does not emerged yet from this perspective.

The platform independent viewpoint focuses on the operation of a system, but without considering any particular platform. The statements or specification made from this viewpoint would not be different from one platform to another. A general modeling language is used in this viewpoint.

The core focus of platform specific viewpoint is similar with the one of platform independent viewpoint, but plus an additional attention on using a specific platform by a system.

The models corresponding to the three viewpoints, defined by MDA, are the Computational Independent Model (CIM), the Platform Independent Model (PIM), and the Platform Specific Model (PSM).

A CIM is made from the computation independent viewpoint. It does not show details of the structure of a system. In software engineering, the CIM is considered as a domain model, and specified by domain experts. It defines the function of the system without showing the constructional details. [11]

From the platform independent viewpoint, the PIM shows the specification of the whole system. It maintains the focus on an ontological level to describe the system construction. Staying at the ontological level means the PIM does not contain any implementation details.

With the viewpoint of specific platform, the PSM combines the specification in PIM with details of the particular platform. In other words, the PSM details the PIM with adding relevant technological elements. [11]

The levels of information abstraction increase from PSM to PIM and CIM, which shift our focus from a lower level to a higher level. It is advisable to use a high-level modeling language and generate the modeling information into a platform-specific details. The Figure 2.4 visualizes the process of MDA transformation, that the PIM is transformed into PSM based on some transformation rules. The transformation rules contain descriptions about how elements in PIM should be transformed into elements in PSM. There is no uniform standard for the transformation rules. It can be patterns, logic or any other terms as long as it fits the transformation require-

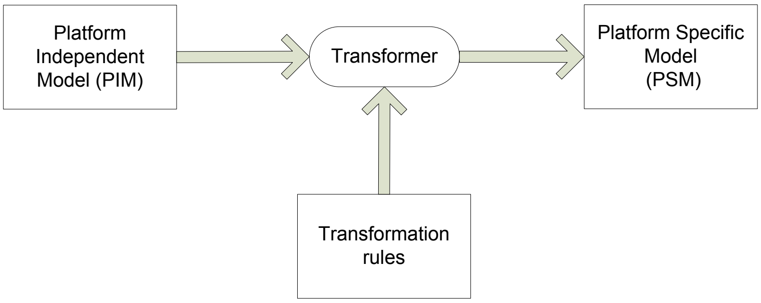

Platform Independent Model (PIM)
Transformer
Platform Specific Model (PSM)
Transformation rules

Figure 2.4: MDA model transformation process [11]

ment.
The relationship between PIM and PSM is that the PSM is made based on the PIM, which is in compliance with the meaning of transformation. Since almost no one codes in assembler languages, but prefer to work with a high-level language, the PSM is the result of a transformation from the PIM via several machine steps, from model building to coding [13].

We did not dig into any specific model transformation technique or language used in MDA in this chapter, because our focus is on the general theory of model transformation. Model transformation bridges the gap between PIM and the PSM. Initiated by OMG in terms of their MDA vision [17], metamodeling proposes an efficient approach to elaborate the transformation. This approach will be explicated in the following section.

2.3.3 Metamodeling for Model Transformation

There are several approaches to elaborate the model transformation in MDA [11]. One of them is using the metamodel, visualized in Figure 2.5. The transformation is from Platform Independent Model (PIM) to Platform Specific Model (PSM). The language used for expressing PIM is defined in platform independent metamodel. The platform specific metamodel defines the language that describes the PSM. These two languages are connected by the transformation specification. The duty of this specification is to map the source language and the target language, so that it can build a bridge from independent platform to specific platform. The specification is expressed in terms of a set of transformation rules that address the detailed guidance of the transformation. Thus the transformation procedure from PIM to PSM is accomplished by the metamodeling.

A good example of applying metamodeling for model transformation is the transformation between UML class diagram and XML Schema from [14]. Depicted in Figure 2.6, the UML model is considered as the platform independent model and the XML Schema is chosen as the specific platform language. The source UML

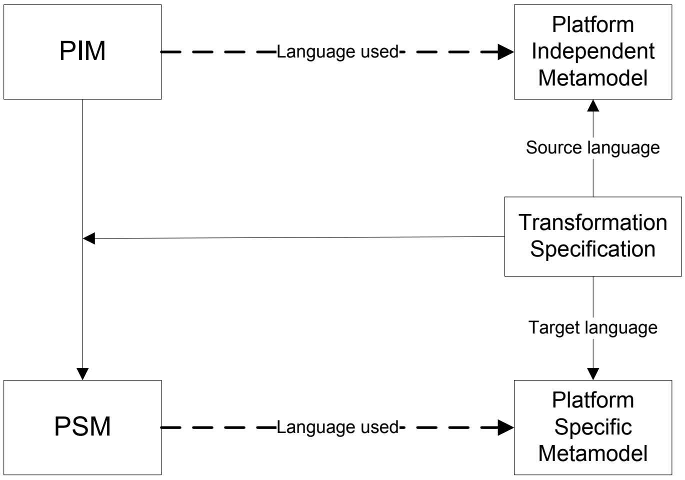

PIM
Language used
Platform Independent Metamodel
Source language
Transformation Specification
Target language
PSM
Language used
Platform Specific Metamodel

Figure 2.5: Metamodel transformation [11]

model and the target alternative XML Schema are situated in level M1, the source UML metamodel and the target XML Schema metamodel are situated in level M2. The source and target models are mapped with each other within the same level. For example, the class construct from UML metamodel, which is in level M2, is possible to be mapped either to element declaration or complexType definition construct from XML Schema metamodel, which is also in level M2. As instances of metamodel, the class from UML model (level M1), in the same principle of the metamodel mapping in level M2, is mappable either to an element or a complexType in alternative XML Schema.

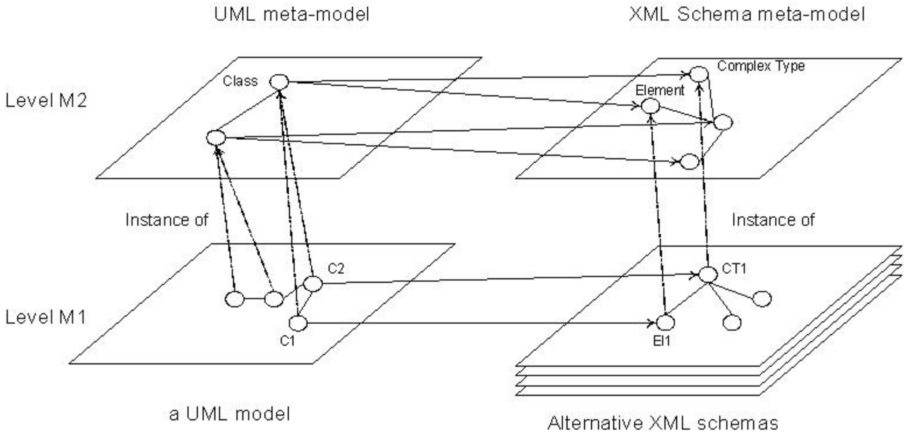


Figure 2.6: UML to XML Schema Transformation

When using the metamodeling in model transformation, the actual transformation in model transformation is executed within the modeling languages used in the metamodel of PIM and PSM, that it leaves the complex instance contents at the model level out of our sight during the transformation.

2.4 XML Schema

At the beginning of this section, a distinction needs to be declared between the terms of "XML Schema" and "XML schema". XML Schema (with capital letter S) is a specific schema definition language, recommended by World Wide Web Consortium (W3C), also called XSD. XML schema is the general term for schema defined in XML format. In the rest of this report, we will use XSD for short instead of XML Schema, when we mention the language used during transformation, in order to be distinguished from XML schema.

With the knowledge of applying metamodeling in model transformation (Figure 2.5), the transformation is carried out between the languages used for platform independent metamodel and platform specific metamodel. The source DEMO metamodel is expressed in World Ontology Specification Language (WOSL) [9]; XSD has been chosen as the language of target metamodel [27].

XSD is the W3C-recommended schema definition language, expressed in XML 1.0 syntax, which is intended to describe the structure and constrains the content of documents written in XML. With XSD, it is possible to exchange information between applications with greater confidence and less custom programming to test and confirm the structure of an instance document, or to confirm that the data in a particular part of the document is of a particular data type [24].

XSD performs well in perspectives of the main building block, the supported data type, to what extent the schema could be specified, and how much room left for adoptions by other languages or systems.

Building Block
In XSD, items could be defined as either an element or a type. Simple elements in XSD could be empty or contain only text. Complex elements contain other elements / attributes or a combination of elements and text. The concept of type is one big merit of XSD, that could be reused by other elements and it is always possible to create an element from a type [2].

Data Type
Data types in XSD are in a wide range. The most basic programming types are included, such as String, Date, Numeric and other miscellaneous data types. The String data type can contain characters, line feeds, carriage returns, and tab characters. Date and time data could be defined in XSD by using the Date / Time data

type. The specific date, time, time zones, time intervals are all available. The numeric data could be decimal, integer, byte and so on. Other data types, such as Boolean, is also used validly. A full list can be found at [25].

Structural Capacity[1]
XSD provides a powerful XML document structure and restrictions, even though it could be very verbose [15]. XSD is namespace aware and has good capability by using both element and type. The concept of type provides more possibility in creating more elements and expanding the schema structures [2]. The feature of reusing types in XSD makes the structure compact and flexible, which reduces the redundancy and enhances the possibility of interactions with other structures.

Compatibility[2]
XSD is written in XML and has a schema of its own. When validating the XML document in such schema language, any node of this XML document is expressed in terms of the schema itself, which transform the document into a hierarchy of typed objects. The resulted objects can be accessed in a programming language through a neutral interface.

The above recaps the background information about XSD. For the lists of the elements and data types provided and supported in XSD, a quick reference is made available in Appendix A.

2.5 Conclusion

This chapter prepares us with background information on DEMO methodology and Business Component Modeling. The notion of business component has proved valuable for developing the information system. There are three phases in the process of BCM, which are: Component Based Domain Analysis, Domain Based Component Realization, and Components Composition. The beginning step in this process is domain modeling and the identification of the business component.
DEMO is a well-defined methodology for representing the essence of an organization, which provides a holistic picture of the entire organization in the business domain. Built on the existing DEMO methodology, there is a set of aspect models that represents the ontological knowledge of the organization. It covers issues of the organization's construction, the business process steps, the business rules, and the information objects.

The concepts of metamodeling and model transformation are also introduced in

[1] Structural capacity measures to what extent of specification the schema language can be used to define.

[2] Compatibility measures the schema languages extensive capacity with other languages or systems.

this chapter. A metamodel is a specific kind of model, which represents other models at a higher level of information abstraction. Any model is an instance of its metamodel.
MDA proposes three viewpoints in visioning a system, the platform specific viewpoint, the platform independent viewpoint, and the computation independent viewpoint. Our focus is shifted from a lower level to a higher level of information abstraction in the mentioned order of these three viewpoints. Models corresponding to these viewpoints are transformed based on transformation rules.
Metamodeling is one approach to elaborate the model transformation in MDA. It conducts the transformation at the level of metamodel, and moves the focus away from the instance models. We will follow this approach (Figure 2.5) to transform the DEMO models in this project.

In addition, we create a short recap of the characteristics of XSD from the research assignment [27]. The format of the transformed DEMO model information is XSD. We made this decision due to the good performance of XSD in the building blocks, the data type, the structural capacity, and the compatibility.

Next Task
With the above background information, we will research and answer the research questions, mentioned in Chapter 1.2, in the following chapters. Regarding the fact that the metamodel is the main modeling material in the metamodeling, and we intend to use metamodeling as the model transformation approach in this project, we will start with understanding and analyzing the DEMO metamodel in Chapter 3.

Chapter 3

DEMO Metamodel

In this chapter, a specification of DEMO metamodel is presented. The graphical DEMO metamodel[1] is showed in Figure 3.1. Firstly, some factual knowledge is recollected in Chapter 3.1, in order to understand the foundation of constructing the metamodel. The description of the metamodel is divided in five parts: Chapter 3.2 the specification of Meta Construction Model (MCM), Chapter 3.3 the specification of Meta Process Model (MPM), Chapter 3.4 the specification of Meta Action Model (MAM) and Chapter 3.5 the specification of Meta State Model (MSM), Chapter 3.6, the metamodel for three cross-model tables, namely Transaction Result Table (TRT), Bank Contents Table (BCT), and Information Use Table (IUT). It is intended to provide a clear insight into the structure of DEMO models as well as the interrelationships between them. At the end of this chapter, a short conclusion and answers to the first research question are provided.

[1] This metamodel of DEMO model is provided and owned by Prof. Dr. Ir. Jan.L.G.Dietz.

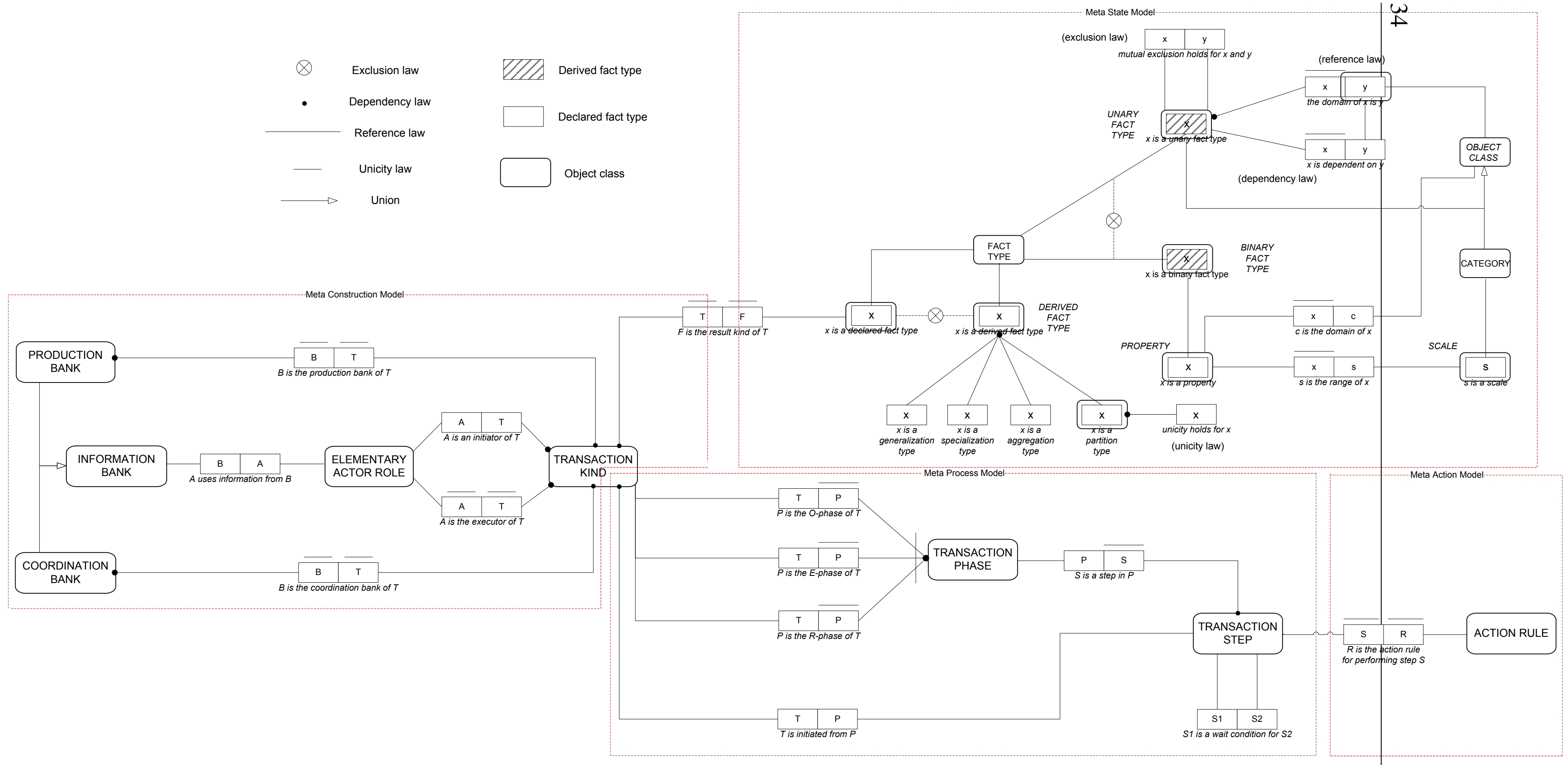


Figure 3.1: The metamodel of DEMO model

3.1 Factual knowledge

Before we start specifying the metamodel, some factual knowledge needs to be recollected. The following knowledge is the foundation for describing the DEMO metamodel. The core notions in the basic ontological parallelogram (Chapter 3.1.1), and the relationships between those notions are the base to construct the MSM. The content of state space and transition space (Chapter 3.1.2), prepares us with the essential content of the SM and PM, which would help understanding the procedure of constructing MSM and MPM.

This section only pick up the knowledge from [9] for analyzing the DEMO metamodel . For the other fundamental concepts in DEMO methodology, such as ontology, production world (P-world), coordination world (C-world), acts, and facts, an explicit explanation could be found in [9].

3.1.1 Ontological Parallelogram

Figure 3.2 shows the four core notions in ontology and their relations: concept, type, object, class. The notions of concept and type are considered to be a subjective notion, whereas object and class are considered to be objective notions.

> Subjective notions are thoughts inside the human mind. A *concept* is a consequence of how the human mind works about the objective world. Concepts are the human thoughts that relate to objects in the world, which includes concrete concepts and abstract concepts. For instance, the mental picture people may have of a laptop is a concrete concept; the concept of number three is absolutely an abstract one. A concept is always a concept of a type.
>
> A *type* is a generic concept, such as: the type laptop, the type person. A type prescribes a collection of properties of an object. It may be that one object conforms to one or more types, due to the different collections of properties, for instance one object could be referred by its color, as well as by its shape, which are different types.

The relationship between a concept and a type is *instantiation*, which means every concept is an instantiation of a type. Examples: the person Salvador Dali is an instantiation of the type person.

The notion of *object* in this ontological parallelogram includes the notions of concrete objects and abstract objects. Concrete objects are the ones which are observable by human beings. Abstract objects are the ones which can not be observed. The number three is absolutely an abstract object, and a car is a concrete object obviously.

> A *class* is a collection of objects. The objects, which conform to the associated type, belong to a class. Let us take persons as an example again. The class of persons contains all the objects that have some shared properties which make them conform to the type person. *Population* is the relationship between an object and a class. Simply, it is said that the object is a member of the class.

The relationship between a concept and an object is *Reference*, which means a concept is referred to an object. By object here we mean both concrete and abstract objects in the world. *Conformity* is the relationship between an object and a type. An object conforms to a type.

A type is extended to a class. The relationship *Extension* is held between a type and a class. Examples: the class persons is the extension of the type person.

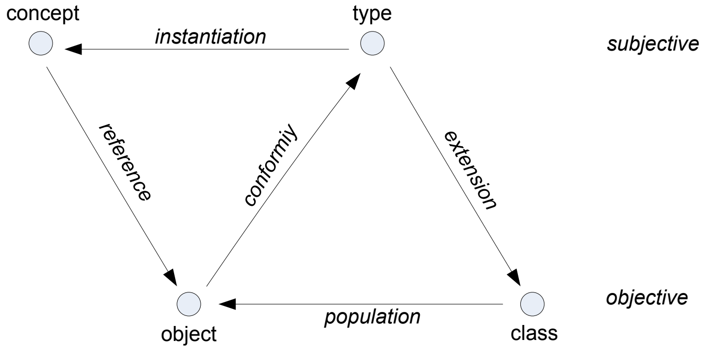


Figure 3.2: The ontological parallelogram

It is advisable to bear in mind this ontological parallelogram, especially the relationships held by different pair core notions. These relationships are one of the foundations in understanding the MSM. It will be explained in section 3.5.

3.1.2 State and Transition Space

> The elementary state elements are called facts. Two kinds of facts are distinguished. A fact is either a statum (plural: stata) or a factum (plural: facta). A *statum* is something that is the case, has always been the case, and will always be the case. It is an inherent property of a thing or an inherent relationship between things. One example of stata in the context of library is: “the author of book title T is A” (in which the variables, expressed in the capital letters, are the placeholders of object instances).

> The existence of stata is timeless, which means, for example, if it is the case that a particular book title has one or more particular author(s), it will forever be the case. Even before the book was written, it was still the case, but just was not knowable yet.
>
> Some stata come into existence only when there is necessary and sufficient condition. These kind stata is called *derived stata*. Those necessary and sufficient conditions are the rules for derivation. There are four types[2] of derived stata, which are the generalization type, the specialization type, the aggregation type and the partition type.
>
> A *factum* is the result or the effect of an act. One example in the case of library is: “membership X has been started”. An *event* is the becoming existent of a factum. Events can be conceived as status changes of a concept of some type; an event has a time stamp.
>
> The world is in a particular *state* at any moment. The state is defined as a set of facts, which are said to be current during the time that the state prevails. A *state space* is understood as the set of lawful or allowed states. It is specified by means of the state base and the existence laws. The state base contains a set of statum types, and the existence laws determine the inclusion or exclusion of the coexistence of stata.
>
> A state change is called a *transition*. A transition is always from one state to another state. A *transition space* is understood as the set of allowed or lawful sequences of transitions. It is specified by the transition base and the occurrence laws. The transition base is a set of factum types, and the occurrence laws determine the order in which facta are required or allowed to occur.

This section mainly focus on the contents of state space and transition space. Distinguishing the different kinds of stata and facta, helps clarifying the composition of the state space and transition space. Furthermore, it prepares us with fundamental knowledge for eliciting the structure of PM and SM. It will be explained later in Chapter 3.3 and Chapter 3.5.

3.2 Specification of the Meta Construction Model

The definition and function of Construction Model (CM) is explained clearly in DEMO methodology [9].

[2] For detailed explanation about the derived stata, please check section 5.3 in [9].

> *The CM specifies the identified transaction*[3] *types and the associated actor roles, as well as the information links between the actor roles and the information banks; in short the CM specifies the construction of the organization.*

CM is about the organization's composition, environment and its structure, according to chapter 6 in [9], where the *composition* is a set of elements of some category (physical, social, biological, etc.); the *environment* is a set of elements of the same category; the *structure* is a set of influence bonds among the elements in the composition, and between them and the elements in the environment. At model level, the environmental actor roles are drew as composite actor roles, since it is not always clear that an environmental actor role is elementary or composite. Only the actor roles within the kernel are elementary actor roles.

Differentiated from the level of model, at metamodel level, it does not distinguish composition and environment of any specific organization. It just abstracts the pure relationships between transaction types, elementary actor roles and information banks, namely the *interaction structure*. There is no need to specify the concept of *boundary* at the meta level, since the structure constructed here is not related to any specific construction of organization.

In the MCM, it specifies the correlation between the transaction types, actor roles and the information banks in higher abstraction. As a meta schema for CM, the metamodel does not relate to any specific example, but comprehends all the possible conditions that occur in practice. In other words, any instance CM could be instantiated based on this meta schema. Being outlined in figure 3.1, transaction kind, elementary actor role and information banks including the production bank and coordination bank are the core object classes within the MCM.

For describing the DEMO metamodel we adopt the next convention (we take the type elementary actor role as an example): if there is a fact a, and the type of a is the elementary actor role, we would say that a is an elementary actor role or elementary actor role a, which is equivalent to saying that elementary actor role (a) holds. This kind of expression will be used throughout the entire specification of the DEMO metamodel in this chapter.

Let us first look into the correlation between the ELEMENTARY ACTOR ROLE and TRANSACTION KIND, the right side links of ELEMENTARY ACTOR ROLE. The capital letters are used to represent the object class. As known from the model level, there are two kinds of elementary actor roles in the CM, namely the initiator and the executor. If there is an elementary actor role a is an initiator of transaction kind t, there must be a transaction kind (t) holds. Meanwhile, constrained by the dependency law, for every transaction kind, there must be an elementary actor role

[3] The concept of transaction is defined in chapter 10 of [9]

a, that *a* is an initiator of *t* holds, *t* is the transaction kind.

If there is an elementary actor role *a* is the executor of transaction kind *t*, there must be a transaction kind (*t*) holds. Meanwhile, constrained by the dependency law, for every transaction kind, there must be an elementary actor role *a*, that *a* is the executor of *t* holds, *t* is the transaction kind. Note that there are unicity laws hold for both *a* and *t* in the lawful binary fact type, that means every transaction kind and every elementary actor role cannot occur more than once in the lawful binary fact type. Thus it implicates that there is a strict one-to-one relationship between the transaction and its executor.

On the left side of ELEMENTARY ACTOR ROLE, it states that if for some facts *a* and *b* that *a* uses information from *b* holds, then it is necessary that elementary actor role (*a*) and information bank (*b*) also hold. Conversely, if for some elementary actor role (*a*) or information bank (*b*) holds, then it may be the case that the predication *a* uses information from *b* holds, but not necessarily.

The object class INOFRMATION BANK is the union of object classes PRODUCTION BANK and COORDINATION BANK. When an actor role uses information from information banks, it could be either from production bank or coordination bank.

The correlation between transaction kinds and information banks are specified as below. If for some facts *b* and *t* that *b* is the production bank of *t* holds, then it is necessary that transaction kind (*t*) and production bank (*b*) also hold. Conversely, if there is some transaction kind (*t*) or production bank (*b*)holds, then it is also necessary that the predication *b* is the production bank of *t* holds. Note that there are unicity laws hold for both *b* and *t* in the lawful binary fact type, that means every transaction kind and the every production bank cannot occur more than once in the lawful binary fact type. Thus it implicates that there is a strict one-to-one relationship between the transaction and its production bank.

The similar correlations apply to the one between the transaction kind and coordination bank. Dependency laws hold for both transaction kind and coordination bank, and there is also a strict one-to-one relationship between the transaction and its coordination bank, thus the two unicity laws hold for both *b* and *t* in the lawful binary fact type.

Note that there are dependency laws for both transaction kind and the two kinds of information banks respectively in their lawful binary fact type. The reason is that every transaction kind must have its coordination bank and production bank, since there are always c-facts and p-facts produced as the results of the c-acts and p-acts which are performed during transaction process steps. And vise versa, the coordination bank and production bank cannot be without the corresponding transaction kind. Thus, the dependency laws are necessary in these two binary fact type.

The result types are the connections with SM at the model level[4], and the same connections hold at the metamodel level. The transaction kind plays the crucial role in connecting MCM to MSM. The connection part is defined in the metamodel like this[5]: if transaction kind (t) holds, there must be a declared fact type (f) such that f is the result kind of t holds. Unicity laws hold for both f and t. The connection between the transaction kind and its result kind is strictly one-to-one relationship.

Figure 3.3 shows an instance CM of the library[6]. The whole instance CM contains ten business transactions; we will just take transaction T01 as example to explain how the instance model is structured upon the MCM.

In transaction T01, membership registration, the initiator of T01 is composite actor role CA02 aspirant member; the executor of T01 is elementary actor role A01 registary. The composite actor role could be an elementary actor role or a composite actor role. For T01, there must be a result listed in the transaction result table (TRT). In table 3.1, the result type for T01 is R01. Thus the relationship between actor roles and transaction in the instance model is fully consistent with the corresponding structure in the metamodel.

The transaction symbol has two meanings, one is the transaction T01, the other one is the combination of production bank PB01 and coordination bank CB01 that belong to transaction T01. The elementary actor role A01 uses information from composite production bank CPB11 personal data. This structure is stated in MCM.

Table 3.1: The TRT of the library

transaction type	**result type**
T01 membership registration	R01 membership M has been started
T02 membership fee payment	R02 the fee for membership M in year Y has been paid
T03 reduced fee approval	R03 the reduced fee for M in year Y is approved
T04 loan start	R04 loan L has been started
T05 book return	R05 book copy C has been returned
T06 loan end	R06 loan L has been ended
T07 return fine payment	R07 the late return fine fee for loan L has been paid
T08 book shipment	R08 shipment S has been performed
T09 stock control	R09 the stock control for month M has been done
T10 annual fee control	R10 the annual fee control for year Y has been done

[4] The result types are produced by transactions in CM. In SM, the result types are used to denote the different states of the object classes.

[5] This connection is the construct for Transaction Result Table (TRT) (section 3.6), we also present it here in order to show the central position of transaction kind in DEMO metamodel.

[6] The description of the library case can be found in the appendix of the book [9].

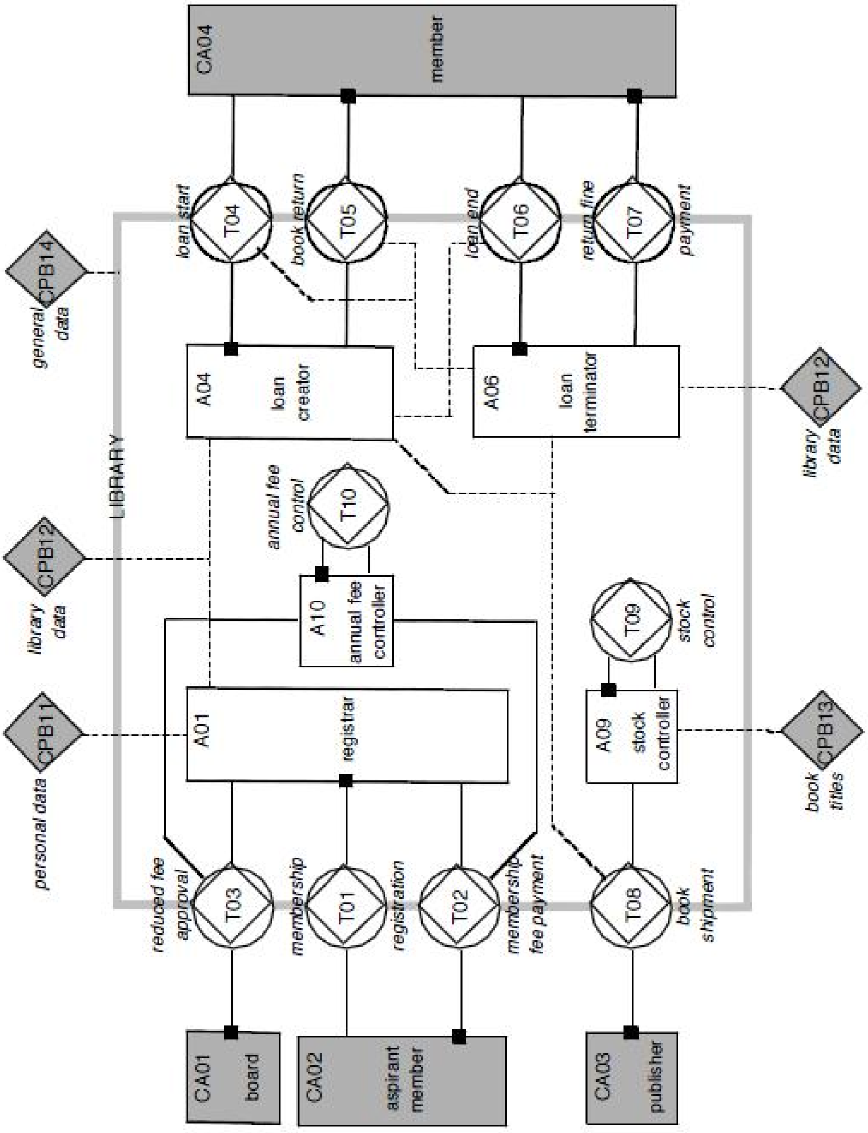


Figure 3.3: OCD of the library

3.3 Specification of the Meta Process Model

The definition and function of Process Model (PM) is explained clearly in DEMO methodology [9].

> *The PM of an organization is the specification of the state space and the transition space of the C-world; thus, the set of lawful or possible or allowed sequences of states in the C-world.*

Simply speaking, PM specifies the *transaction steps* for every transaction. In addition, for every transaction step, the information used to perform the step is also included in PM, so does the *responsibility areas*. The *responsibility areas* are generated from CM that states which actor roles perform which process steps. Apparently, transactions play the role of being the connection between CM and PM, since PM details the transaction types of CM into a sequence of process steps.

Let us look into the composition of a transaction's structure. A transaction evolved in three phases: the *order* phase (O-phase for short), the *execution* phase (E-phase for short), and the *result* phase (R-phase for short). The two partaking actor roles are called the initiator and executor of the transaction.

In the order phase, the initiator *requests* for the intended result, and the executor agrees and *promises* to reach the intended result. The intended result is the production fact of the transaction. In the execution phase, the agreed production fact is brought out by the executor. In the result phase, the executor *states* that the production fact is produced, and the production fact come into existence when the initiator *accepts* the result. This sequence of transaction steps forms the basic pattern of a transaction. By basic it means the initiator and executor keep consenting to each other's acts.

Besides the basic pattern, in a transaction, there may be more acts performed by either initiator or executor and more facts resulted from those acts. When the two actor roles have dissent about the each other, more acts are performed. In the standard transaction pattern, the executor may *decline* the request, instead of promising it; the initiator may *reject* the statement, instead of accepting it.

In addition to those acts already mentioned above, it is also possible for both initiator and executor to revoke their C-acts. Thus, there are patterns started with *cancel* a request, cancel a promise, cancel a statement or cancel an acceptance. An explicit explanation about the different transaction patterns is elaborated in chapter 10 of the book [9].

The transaction pattern outlines the particular structures of the clustered transaction steps. No matter which pattern the transaction steps follow, they are all conducted within the three transaction phases. Any transaction step is taken in one of the three phases. And in which transaction phase the transaction steps should be, it regards to the specific type of the transaction step, which is not a question for this MPM. It will be explained later this section.

The structure of PM is specified in the MPM, outlined in Figure 3.1. TRANSACTION PHASE and TRANSACTION STEP are the two core object classes. An instance of TRANSACTION PHASE can only be one of the three transaction phases. For every transaction phase *p*, there must be a transaction kind *t* that *p* is the transaction phase of *t* holds. And if for some facts *p* and *t* that *p* is the one of the transaction phases of transaction kind *t* holds, there must be a transaction phase (*p*) holds. In addition, transaction phase *p* could only be one of the three transaction phases at one time and occur exactly once in a transaction kind.

The correlation between TRANSACTION PHASE and TRANSACTION STEP is: for every transaction step, there must be a transaction step *s* that *s* is a step in transaction phase *p* holds. Conversely, if for some fact *s* that *s* is a step in transaction phase fact *p* holds, there must be a transaction step (*s*) holds. Note that there is a unicity law hold for *s* in the binary fact type, that means every transaction step occurs exactly once in this binary fact type. Interpreted at the model level, this correlation means every transaction step must be and only occur once in one of the transaction phases.

As we know, stated in chapter 11 of [9], that every transaction is enclosed in some other transaction, or is a customer transaction of the organization under consideration, or is a self-activation transaction. It shows that the transaction steps are interrelated with each other in a causal way; the starting step is either a request performed by external actor role (*external activation*) or a request performed by an internal actor role to itself (*self-activation*). For example, in Figure 3.4, transaction T1 is initiated by self-activation, while transaction T2 is initiated by transaction T1. Transaction T2 is an enclosed transaction T1.

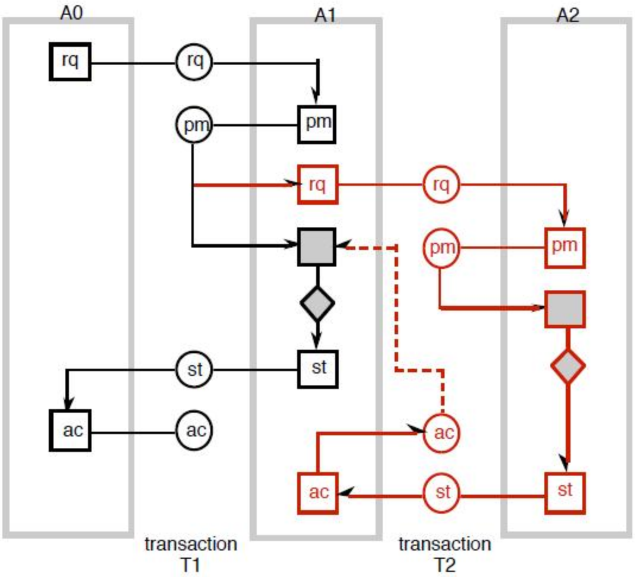


Figure 3.4: The structure of enclosing a transaction

The above enclosing structure of a transaction, showed in Figure 3.4 is expressed in the MPM like this[7]: for every transaction kind, there must be a transaction kind *t* that *t* is initiated from transaction step *p* holds. Conversely, if for some facts *t* and *p* that *t* is initiated from *p* holds, then there must be transaction kind (*t*) holds. It means, every transaction is initiated from a transaction step; this transaction step could be a request from another transaction, but also could be a request from itself. It does not mention any specific step or specific transaction in the metamodel, but just presents the generic initiation condition here.

There is another kind of relationship between transaction steps, which is called wait condition. In Figure 3.4, the dashed link between step accept of T2 and step execute of T1 means there is a wait condition. The execution of T1 is waiting for the completion of T2.

The wait condition between transaction steps is expressed in the MPM as below. For some facts *s1* and *s2*, if the predication that *s1* is a wait condition for *s2* holds, there must be transaction step (*s1*) and transaction step (*s2*) also hold. In an instance PM, let us assume s1 and s2 are two transaction steps, it means that dealing with the result from step s2 has to wait until the result from step s1 has been created.

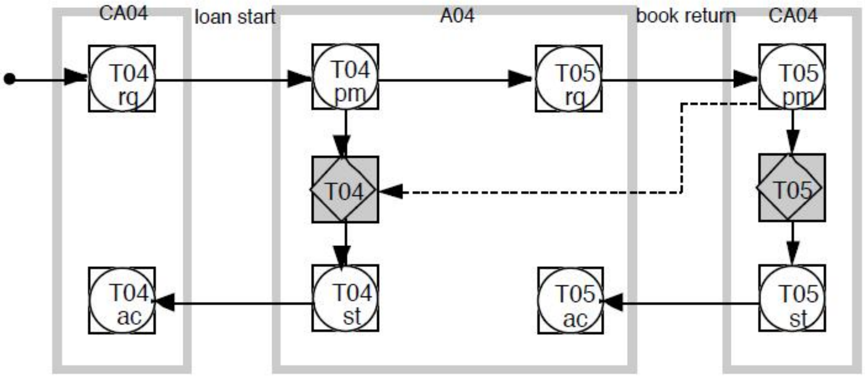


Figure 3.5: PSD of a business process of the library

There is no need to draw the transaction patterns completely in the metamodel. It is true that the transaction pattern contains the structure of PM, and a meta schema should provide the structure for the instance model. However at a higher level of abstraction, the pure relationship kinds between those specific transaction steps are the ones we are concerned about. The transaction steps do not need to be entitled with specific types. That is why we only present the generic relationship between transaction step and transaction phase, the generic initiation condition be-

[7] The enclosing structure we are concerned about is within the kernel of the system, so the acts performed by external actor role is not expressed in the metamodel.

tween transaction kind and transaction step, and the generic wait condition between transaction steps.

Actually, different patterns could be instantiated based on the initiation condition and wait condition. Every transaction step is connected with another transaction step. The start of every transaction step is triggered by either another transaction step or itself. So the MPM covers all the possible conditions that may occur in the instance model.

Let us see how the transaction steps structure in MPM is instantiated in PM. Figure 3.5 shows an exemplary PSD of library. It details the transactions T04 and T05 in Figure 3.3. Actor role CA04, as initiator of T04, performs two acts: T04\rq and T04\ac. Actor role A04, as executor of T04, performs three acts: T04\pm, T04\ex, and T04\st. Actor role A04, as initiator of T05, performs two acts: T05\rq and T05\ac. Actor role CA04, as executor of T04, performs three acts: T05\pm, T05\ex and T05\st.

In the process of T04, the transaction step T04\rq is initiated from external activation, which is out of the kernel of concern. The sequential transaction steps T04\pm, T04\ex, T04\st and T04\ac are initiated by its previous step. It can also be considered that the start of these steps waits for the completion of its previous step. The step T04\ex has one more wait condition, which is from the completion of transaction step T05\pm. The transaction T05 is initiated from transaction step T04\pm. The rest steps in T05 follows the same sequencing as the one in T04.

It does not specify the responsibility area in the MPM. The reason is that the information about responsibility area is generated from the specific transaction in CM. The elementary actor roles that determine the responsibility area are already specified in the MCM, related to the transaction kind. The MPM contains connection with transaction kind already, which means it is possible to generate the information of actor roles in its instance model. So there is no need to duplicate a part of the structure in this essential metamodel.

3.4 Specification of the Meta Action Model

The definition and function of Action Model (AM) is explained clearly in DEMO methodology [9].

> *The AM of an organization is the specification of the action rules that serve as guidelines for the actors in dealing with their agenda.*

The action rule is the only construct in AM[8], thus in MAM, the structure is simple. Outlined in figure 3.1, the object class ACTION RULE is the set of all the instance

[8] It is explained in chapter 18 of [9].

action rules. If there are objects *s* and *r* that *r* is the action rule for performing step *s* holds, then it is necessary that action rule (*r*) and transaction step (*s*) also hold. Conversely, if for some action rule (*r*) or transaction step (*s*) holds, then it may be the case that the predication *r* is the action rule for performing step *s* holds, but not necessarily. Note that the unicity laws hold for both *s* and *r* in the lawful binary fact type, that means there is a strict one-to-one relationship between the transaction step and its action rule.

Based on the presented structure in MAM, the action rules are concerning with transaction steps, but not every transaction step necessarily needs action rules, for instance: transaction step accept.
Every instance action rule belongs to the object class ACTION RULE. We do not dig into the various possible conditions in concrete instance rules, but just group them into the set of action rules at a higher level of abstraction.

3.5 Specification of the Meta State Model

The definition and function of State Model (SM) is explained clearly in DEMO methodology [9].

> *The SM of an organization is the specification of the state space of the P-world. It consists of specifying the object classes, the fact types, and the result types, as well as the existential laws that hold.*

According to the factual knowledge mentioned earlier in Chapter 3.1, the relationships between types and classes, as well as the concept of state space are obtained, which are the premise to elaborate the specification of the MSM. Being aware of the distinction between statum types and factum types, we conclude that the SM contains both stata and facta. The object fact types in SM are statum types, and the result types are factum types. At meta level, the concepts of all the fact types and their existential laws are defined in the MSM.

In the chapter 5 of the book [9], it presents a language for the specification of world ontology, which is called World Ontology Specification Language (WOSL). It is used for specifying SM, and contains the constructs that are needed for SM. The graphical notations used in WOSL are adopted from Object Role Modeling (ORM) [19], which is one of the fact oriented conceptual modeling languages.

In the MSM, the graphical notations are adopted from ORM as well, in consistency with the one used in the SM. This metamodel explains the fact types, object classes and the existence laws at a higher level of abstraction, compared with the specifications of SM in the book [9]. In the following sections, it is going to present the specification of MSM in a number of figures.

3.5.1 Fact types and object classes

The declaration of fact[9] types are specified in the MSM. Outlined in Figure 3.1, we use capital letter FACT TYPE to refer the set of fact types, any fact type is an instance of the object class FACT TYPE. A unary fact type is a member of FACT TYPE. The declared fact type is symbolized by a rectangle, denoted with lowercase letters. The derived fact type is symbolized by a rectangle filled with slash pattern, denoted with lowercase letters. The object class is symbolized by a round rectangle, denoted by capital letters.

Objects belong to FACT TYPE can either be declared fact types or derived fact types. The declared fact type is the one that is the case, has always been the case, and will always be the case. The derived fact type is the one that its existence know by people is determined by some conditions. Only under those certain conditions, the derived fact type is also the one that is the case, has always been the case, and will always be the case. Thus, an exclusion law is hold between the two types.

As already mentioned in 3.1.2, derived fact types come to exist by the derivation rules. There are four kinds of derivation: generalization, specialization, aggregation and partition. The object class DERIVED FACT TYPE is formed with objects, which are one of the four types. The specification of DERIVED FACT TYPE in MSM is shown in figure 3.6.

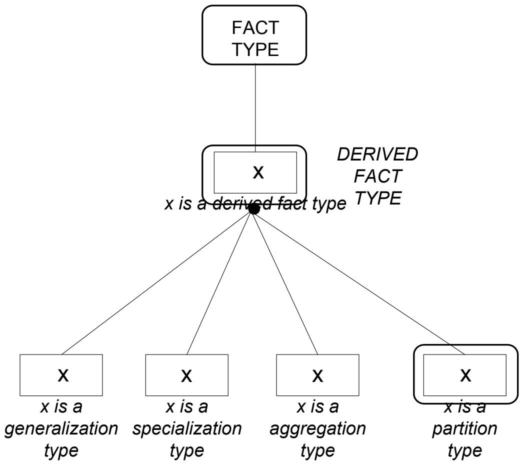


Figure 3.6: The specification of DERIVED FACT TYPE in MSM

With the knowledge obtained from the ontologcial parallelogram in Chapter 3.1.1, the notion of a class is the extension of a fact type. A category is a primal type, and is not a derived type. Explained in the book [9], any other class is the extension

[9] The more common term “fact” is used to refer the elementary object in the world, it includes both stata and facta.

of a statum type that is defined on the basis of one or more other classes, including categories, by means of reference law. Object class UNARY FACT TYPE is a set of unary fact types. In MSM, OBJECT CLASS is defined as the union of the extension of unary fact type and CATEGORY (Figure 3.7).

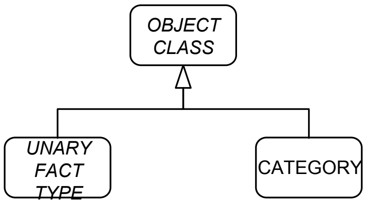


Figure 3.7: The specification of OBJECT CLASS in MSM

3.5.2 Existence laws

The exclusion law is described in the MSM, marked by a bracketed exclusion law (Figure 3.8): for two unary fact types x and y, which belong to the class UNARY FACT TYPE, a predication that mutual exclusion holds for x and y is declared. If there is an exclusion law between two fact types, there is no one object that would be instance of both these two fact types. The graphical notation for an exclusion law could be seen between declared fact type and derived fact type in Figure 3.1.

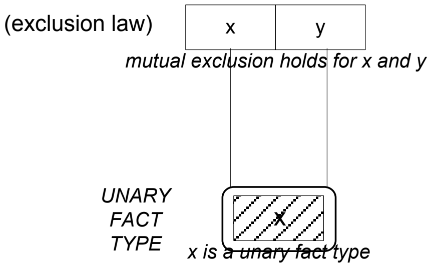


Figure 3.8: The specification of exclusion law in MSM

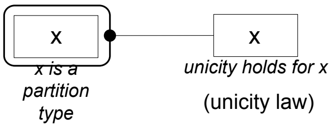


Figure 3.9: The specification of unicity law in MSM

The unicity law is defined by means of that, for a unary fact type x, which belongs to the derived partition type, a predication that unicity holds for x is declared. In the metamodel, this definition is marked by a bracketed unicity law (Figure 3.9). In graphical notations, a line above a unary fact type indicates that a unicity law is held for the unary fact type. The reference law and dependency law are specified coherently in the MSM. Let us first recollect the specification of reference law and dependency law at the model level. An example of a reference law from the book [9] is showed in Figure 3.10. It states that if for some object x **student**(x) holds, then it is necessary that **person**(x) also holds. It is equivalent to saying that x ∈ PERSON must hold. Conversely, it is not necessarily be the case that if some object x **person**(x) holds, then **student**(x) holds also.

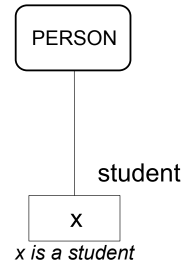


Figure 3.10: Example of a reference law

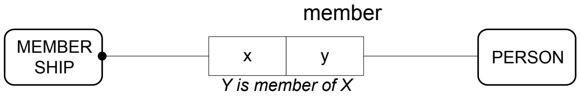


Figure 3.11: Example of a dependency law

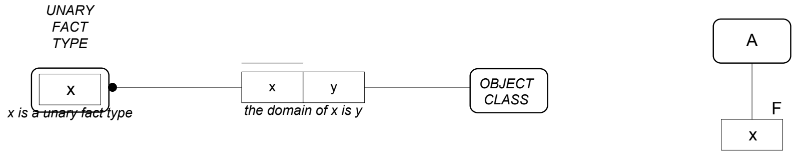
An example of a dependency law is showed in Figure 3.11. First concerning the reference laws in it, if some x and y **member**(x,y) holds, then **membership**(x) and **person**(y) must hold also. With the dependency law, for every x ∈ MEMERSHIP, there must be a y ∈ PERSON such that **member**(x,y) holds. Holding the dependency law means that the coexistence of a member object and a membership is dependent on each other.

At metamodel level, in a binary fact type, object x and y, x belongs to class UNARY FACT TYPE, y belongs to class OBJECT CLASS, a predication, that the domain of x is y, is declared. The declaration is visualized as the normal link without a dot at any end of the line, namely reference law. The expression of reference law is the declaration of relationship “belong to”. This definition is marked by a bracketed reference law in the metamodel.

For object x and y, x belongs to class UNARY FACT TYPE, y belongs to an extension of unary fact type, a predication, that x is dependent on y, is declared. This declaration is visualized as a dot, which is used at the end of a reference law line, namely dependency law. This definition is marked by a bracketed dependency law in the metamodel.

3.5.3 Basic construct

The MSM defines the concepts of fact types, object classes, existence laws that used in the SM. In addition, it also defines the basic construct of the SM. An instance SM is instantiated in terms of a lawful set of basic constructs.

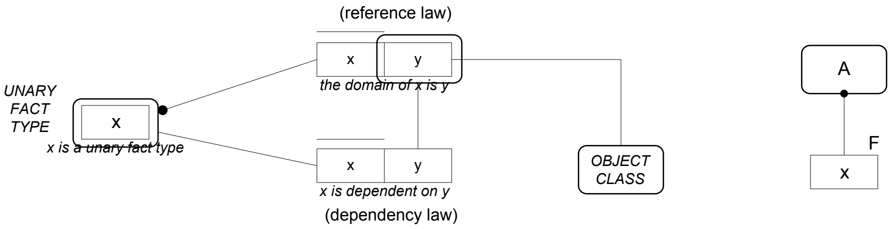


(a) one simple construct in meta State Model

(b) one simple construct in State Model

Figure 3.12: Basic construct at both metamodel level and model level (without dependency law)

Figure 3.12 shows how the MSM specifies a basic construct without a dependency law in SM. The basic construct in SM states that the role x in fact type F has a domain A. A reference law is held between the object class and the fact type. The corresponding construct at the meta level is expressed like this: for every unary fact type there must be one and only one instance fact type x in the binary fact type <x, y>, the domain of fact type x is the object class y. This meta construct contains the definition of the reference law, which is compliant with the exemplary construct in SM.



(a) one simple construct in meta State Model

(b) one simple construct in State Model

Figure 3.13: Basic construct at both metamodel level and model level (with dependency law)

Figure 3.13 shows how the MSM specifies a basic construct with a dependency law in SM. The basic construct in SM states that the role x in fact type F has a domain A. There is a dependency law for A, which means, for every a ∈ A there must be a tuple <a>in F. Thus in MSM, we add an extra part for defining the dependency law. Besides the same construct with the one in Figure 3.12, this extra part states that object x is dependent on object y, x belongs to class UNARY FACT TYPE, y belongs to an extension of unary fact type.

The above construct (Figure 3.12 and Figure 3.13) presents the core object classes and fact types, plus the existential laws. For other fact types that are pure properties, the MSM also defines relevant construct. Let us first recap how the property type is displayed in SM.

In OPL, the property type, object class and scale are listed in terms of table, the object class and scale are mapped with each other by mathematical functions. In WOSL, the property fact is a binary fact type, showed in Figure 3.14. The domain of role a of property P is object class A; the range of role s of property P is scale S. For every a ∈ A there must be a tuple <a,->in P, and the tuple <a,->can only appear once in a population of P.

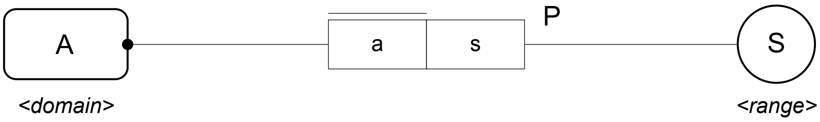


Figure 3.14: Basic construct of Object Property in SM

An example of the library explains the structure more explicitly. In the property type #books_in_loan (Figure 3.15), the domain of this property type is object class MEMBERSHIP; the range of this property type is absolute scale NUMBER. It is clearly to see that for every membership there is a property states the number of books which are in loan. This property type is always there once the membership holds, which is in compliance with the dependency law in the construct. For the same membership, it is not possible to have more than one of the same property type, since it makes no sense to state the number of books in loan in more than one property type. In the construct, this condition is constrained by the unicity law.

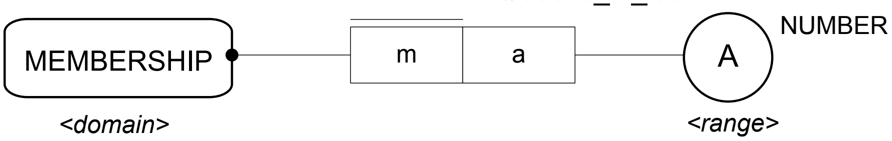


Figure 3.15: An example of Object Property of the library

As already seen how the property type is expressed in a binary fact type, we can find the corresponding structure in MSM. The structure of the property type is defined in two separate parts in MSM. We can divide the construct in Figure 3.14 from the middle, the left side includes role a of property P and the domain of role a, the role s of property P and the range of role s belong to the right side.

To be distinguished from the unary fact type, there is an exclusion law hold between the unary fact type and binary fact type in MSM (Figure 3.1). Object class BINARY FACT TYPE is a set of binary fact types. Some objects that belong to BINARY FACT TYPE are properties. The object class PROPERTY is defined as a set of properties. Object class SCALE is derived from a set of objects. Those objects are members of class CATEGORY, and with a sentence that predicates each of them is a scale.

The left side of the property type, how the property is associated with the object class, is defined in Figure 3.16. If for some objects *x* and *c* that *c* is the domain of *x* holds, then it is necessary that property (*x*) and domain (*c*) also hold. Conversely, if for some property (*x*) or domain (*c*) holds, then it is not necessary that the predication *c* is the domain of *x* holds. For each property, it cannot occur more than once in a lawful population of the fact type that *c* is the domain of *x*.

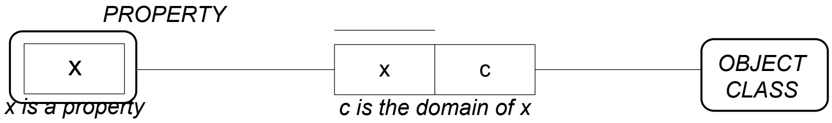


Figure 3.16: The definition of the left side of an Object Property

The structure of the right side of the property type is defined in Figure 3.17. This construct specifies that how the property is associated with the scale. If for some objects *x* and *s* that *s* is the range of *x* holds, then it is necessary that property (*x*) and scale (*s*) also hold. Conversely, if for some property (*x*) or scale (*s*) holds, then it may be the case that the predication *s* is the range of *x* holds, but not necessarily. For each property, it cannot occur more than once in a lawful population of the fact type that *s* is the range of *x*.

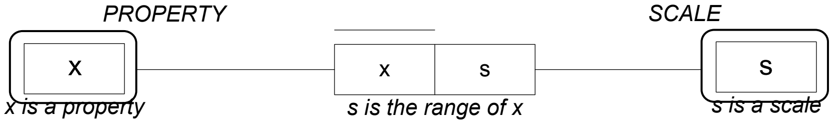


Figure 3.17: The definition of the right side of an Object Property

According to the contents defined in the MSM, this metamodel could be considered from two perspectives, namely it fulfills two roles. The first one is the metamodel of SM. Being this role, the MSM is at the same level as the other two metamodel, MCM and MPM, that it defines the basic construct used in the instance SM. Apparently, the whole metamodel, including the MCM, MPM, MAM and MSM, could be regarded as a big SM, since they are correlated with each other as a whole and expressed in the same language, WOSL. The whole DEMO metamodel uses fact types, object classes and existence laws to present structures of DEMO models. The meanings of those symbols are the same as the ones used in SM. At this point, another role of MSM is being the metamodel of the DEMO metamodel. We call the metamodel of DEMO metamodel as meta schema. The MSM defines the meta schema of the whole DEMO metamodel, and specifies the concepts of the symbols that used in DEMO metamodel.

3.6 Specification of the metamodel for cross-model tables

In DEMO, there are three cross-model tables (Figure 3.20) that containing information from more than one aspect model. Due to the fact that these tables cross different models, the object classes in their metamodel belong to different metamodel of aspect models. For the convenience of reading, we present the metamodel of the cross-model tables in Figure 3.18, separately from the main DEMO metamodel.

TRT contains the transactions and their corresponding transaction results. The metamodel for TRT[10] is: if transaction kind (*t*) holds, there must be a declared fact type (*f*) such that *f* is the result kind of *t* holds. Unicity laws hold for both *f* and *t*. The connection between the transaction kind and its result kind is strictly one-to-one relationship. The object class TRANSACTION KIND belongs to MCM; the object class DECLARED FACT TYPE belongs to the MSM.

In Chapter 20 of [9], it states that BCT specifies the fact banks in which the elements of object classes, and the instances of fact types and result types from the SM are contained. We specify the metamodel for BCT like this: if fact type (*t*) holds, there must be an information bank (*b*) such that *b* is the information bank of *f* holds. If there is an information bank (*b*) holds, there also must be some fact type (*t*) so that *b* is the information bank of *f* holds. FACT TYPE is the general object class that including declared fact types and derived fact types. Unicity law holds for *f*, which means every member of class FACT TYPE can only have one information bank. There are both dependency laws hold for FACT TYPE and INFORMATION BANK, that implicates every member of FACT TYPE must belong to an information bank; conversely it is also necessary for information banks to have some facts, otherwise the information banks cannot exist.

[10] This construct is also presented in Chapter 3.2.

IUT specifies the transaction steps in which the elements of object classes, and the instances of fact types and result types from the SM are used. The metamodel for IUT is: if for some facts f and s that f is used in s holds, then it is necessary that fact type (f) and transaction step (s) also hold. There is no other constraint hold for this construct, which means a fact type could be used in more than one transaction steps, and a transaction step could used more than one fact types, however it is not necessarily to hold fact type (f) or transaction step (s) all the time. Examples of

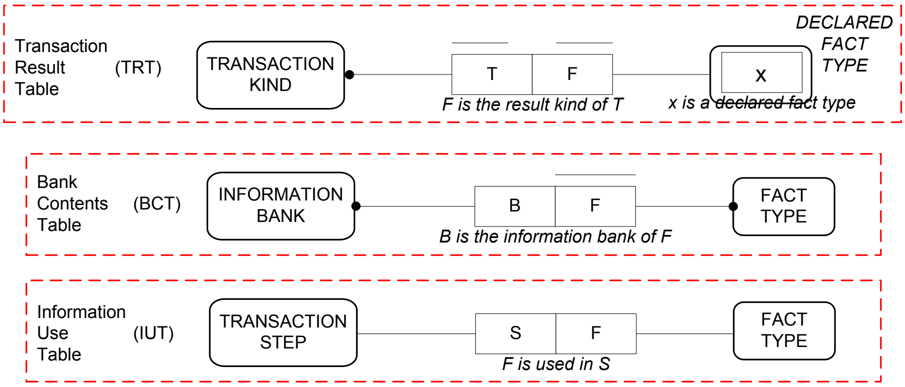


Figure 3.18: The metamodel of cross-model tables

TRT has been discussed in Chapter 3.2. Table 3.2 shows part of the instance BCT of the library case[11]. Information banks PB01 and PB02 are instances of INFORMATION BANK, the items in the first column are the instances of FACT TYPE. PB01 includes the first three instances of fact type, while the other one belongs to PB02. Note that every instance of FACT TYPE could only belong to one information bank, which is constrained in the metamodel.

Table 3.3 provides part of the instance IUT of the library case. The items in first column are the instances of FACT TYPE; the instances of TRANSACTION STEP are listed in the second column. It is obviously to see that, in appliance with its metamodel, there might be more than one transaction steps using the same instance of fact type; and in one transaction step it is possible to use more than one instances of fact type.

Table 3.2: Example BCT of the library

object class, fact type, or result type	**P-bank**
MEMBERSHIP	PB01
P is the member in M	
membership M has been started	
the fee for member ship M in year Y has been paid	PB02

[11] Note: if the bank in which instances of a type are contained is the same as the one in the previous entry of the table, the bank number is not repeated. [9]

Table 3.3: Example IUT of the library

object class, fact type, or result type	**process steps**
MEMBERSHIP	T01/rq T01/pm T04/rq T10/pm
P is the member in M	T01/rq

3.7 Conclusion

DEMO metamodel summary
DEMO metamodel specifies the basic constructs and relationships in CM, PM, AM and SM. The language used to express the metamodel is the same one adopted in SM. The whole metamodel could be considered as a big SM. The transaction kind has the central position in this big SM, since it connects MCM with MPM and MSM respectively.

MCM and MPM provide the essential structure for CM and PM, The structure for CM includes the transaction kind, elementary actor role, information banks and the correlations between them. It indicates the constraints to those items on their occurrence in CM.

The structure for PM focuses on the relations among transaction steps and transaction phases. In MPM, it does not consider any specific transaction pattern used in PM, but concentrates on the pure relation between every single transaction step. By instantiating the relations between transaction steps, every possible instance process steps could be obtained in different transaction patterns.

MCM and MPM are connected via transaction kind. Transaction phase and transaction step in MPM have linkage with transaction kind respectively. It is also the reason that MPM does not address anything about the responsibility area in PM, since the information of identifying the responsibility area could be generated via this link in MCM; it is better to avoid this information duplication.

Connected with transaction step in MPM, MAM provides the only construct in AM. It only specifies the interrelationship between action rules and transaction steps, but not digs into any concrete condition in instance action rules.

MSM plays two roles, namely the metamodel of SM and the meta schema of the whole DEMO metamodel. Being the metamodel for SM, it defines the basic constructs used in MS, such as basic construct with/without dependency law (Figure 3.10, Figure 3.11), construct of object property (Figures 3.14 and 3.17).

As the meta schema of the DEMO metamodel, it gives definitions to concepts of declared fact type, derived fact type, unary fact type, binary fact type, object class, scale and category. In addition, existence laws, such as exclusion law, unicity law, reference law and dependency law, are also specified in the meta schema.

The connection with MCM is the link between the declared fact type in MSM and the transaction kind in MCM. The instance of declared fact type in DEMO models is the result type, which is the fact produced by transaction in CM and also is included in SM.

Reflection of metamodeling

The specification of DEMO metamodel is elaborated in five separate sections, according to the metamodel of the CM, PM, AM, SM and the cross-model tables. In each section, the metamodel visualizes the schema that can be instantiated into DEMO models. Applying the knowledge of metamodeling mentioned in Chapter 2.3.1, this is in compliance with the "instance-of" relationship that the DEMO aspect models are instances of DEMO metamodel. Therefore we can say that the information abstraction of DEMO metamodel is equivalent to the metamodel level in OMG's architecture (figure 2.3), while the DEMO aspect models are of course equivalent to the model level.

Regarding the characteristic of MSM and the fact that MSM is part of the whole DEMO metamodel, it leads to the conclusion that DEMO metamodel is self-defined, which implicates that there is no higher level of information abstraction above this DEMO metamodel. Considering the stop point in OMG's opinion[12], this DEMO metamodel is already sufficient in metamodeling.

Completeness of the DEMO metamodel

DEMO metamodel includes the metamodels of all the diagrams. MCM presents the structure of Organization Constructtion Diagram (OCD), which combines the Actor Transaction Diagram (ATD) and Actor Bank Diagram (ABD). MPM, MAM and MSM describe the other three diagrams' (Figure 3.19) structure respectively.

Besides the diagrams, there are also three cross-model tables in DEMO aspect models. In Figure 3.18, we also provide the metamodel for Transaction Result Table (TRT), Bank Contents Table (BCT) and Information Use Table (IUT).

Thus we conclude that DEMO metamodel is a complete model of all the aspect models, including all the diagrams and cross-model tables. Next, we are going to design XML schema for DEMO models in Chapter 5, based on the mentioned metamodel.

Answers to the research question

In this chapter, the first research question "How would the DEMO metamodel be constructed?" has been answered. The DEMO metamodel represents the DEMO aspect models. It abstracts the essential structure in DMEO aspect models, including the concepts used in building those models, as well as the relationships and the constrains among them. The DEMO metamodel is specified in WOSL and self-defined, so that there is no higher level of information abstraction in DEMO metamodeling.

Next Task

Having a clear structure of the DEMO aspect models from the metamodel that has been specified earlier in this chapter, Chapter 4 will firstly illustrate the require-

[12] OMG's opinion about stop layering the metamodeling architecture is mentioned in Chapter 2.3.1.

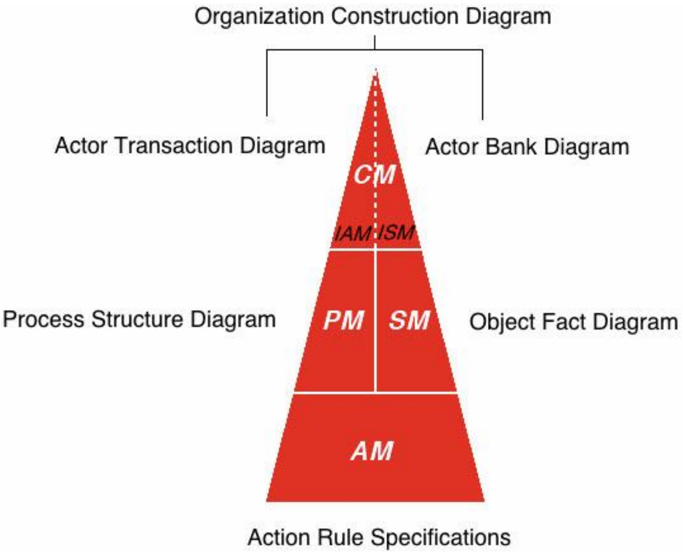

Organization Construction Diagram
Actor Transaction Diagram
Actor Bank Diagram
CM
IAM
ISM
Process Structure Diagram
PM
SM
Object Fact Diagram
AM
Action Rule Specifications

Figure 3.19: The diagrams

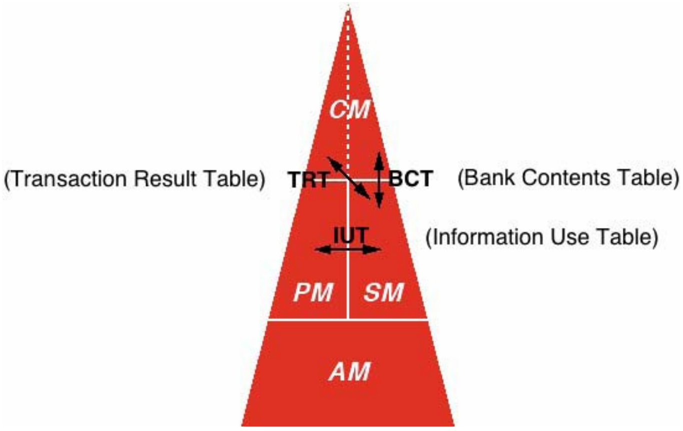


Figure 3.20: The cross-model tables

ments to transform the DEMO models, and then define the rules that will provide the guidance and restriction to the expected transformation from the DEMO models to the XML format.

Chapter 4

Transformation Analysis

This project was initiated by the demands to bridge the gap between graphical DEMO models and platform specific models (Chapter 1.1). After obtaining the background information about the DEMO methodology and BCMP, we will analyze this gap in more detail, and find the reasons which cause the complexities in the gap. Chapter 4.1 illustrates the demands to transform DEMO models, as well as the desired result from the transformation. Another part of the analysis in this chapter (Chapter 4.2) is the definition of the transformation rules. These rules are defined based on our knowledge about the DEMO metamodel and also with consideration of the requirements from the first section in this chapter.

4.1 Requirements for DEMO Transformation

DEMO models[1], as a high-level conceptual model, are created and understood by humans. They visualize the business organization in terms of either graphical diagrams, such as the Organization Construction Diagram (OCD) in CM, Process Structure Diagram (PSD) in PM or Object Fact Diagram (OFD) in SM, or in expressive format, such as the Action Rule Specifications in AM and the cross-model tables TRT, BCT and IUT [9].
As a conceptual model, DEMO models are not intended for direct application in information systems. They cover issues in an enterprise such as: the organization's construction, the business processes, the business rulesm, and the information objects on the business level. This leads the abstracted essence of the organization to stay away from the concrete application platform.

The desire to transform conceptual DEMO models is raised when the focus is moved to bridge the gap between conceptual DEMO models and platform specific models. Business Component Modeling is a good example that will benefit from an automatic and effective transformation [4] of the DEMO models. The information used for identifying business components in the BCI-3D method includes the

[1] DEMO models are explained with examples in chapter 16-20 in [9].

business process steps from PM, and the information objects from SM. In addition, the relationship among these types of information is also required for BCI. However, the relationships are not directly shown in any one of DEMO aspect models, which leads us to generate them out of our own knowledge and understanding [4].

The complexity of information generation for BCI is twofold. First is the diversity of information resources. DEMO separates concerns into several aspect models. CM focuses on the organization construction, PM specifies the business process, AM comprehends the action rules of the entire organization, SM concentrates on the existence of information objects within the organization's business. Besides clarifying the essence of each aspect, this separation also group the information into different sets, which belong to different aspect models. Thus when generating information for BCI, required information has to be obtained from different resources. Even though there are three cross-model tables in DEMO[2] to collect information from different models, the information provided by these tables is not complete and suitable for direct usage.

Another complexity comes from the indirect information. By indirect information, we mean the relationships among the information objects and process steps, because those relationships are not directly shown in any of the DEMO aspect models, even the cross-model tables cannot provide sufficient information. The demanding relationships are in three different types, which are the relationship between the process steps and information objects, the relationship between information objects, and the relationship between process steps [3].
The relationship between information objects needs to state three types of relations, which are "relate-to", "part-of" and "state-of". "Relate-to" indicates which information objects are connected with each other, "part-of" points out which information object is part of another one, "state-of" marks different states of information objects. All the items of information objects are contained in SM, and those relations are diagrammed in Object Fact Diagram (OFD). The OFD diagram is readable if we have knowledge of the modeling language WOSL[3]; however the relationship contained in this diagram can not be generated directly by machine. It requires additional translation of this relationship into machine understandable format.
All the process steps are shown in PM. Generally, the process steps from the same transaction are grouped in fixed sequence, to which we assign a "standard" relation. There are also some optional connections between some process steps, to which we assign "optional" relation, due to the connection between different transactions[4]. By optional, we mean the following step might not be executed every time. Besides the standard and optional conditions, some process steps may be

[2] The cross-model tables are mentioned first in chapter 15 of [9]

[3] A short explanation about the modeling language in SM is provided in chapter 3.

[4] More details are explained in the Composition Axiom, chapter 11 of [9].

executed more than once, to which we assign the term “main” relation with their followed process step. For these process steps the three kinds of relations between process steps, namely “standard”, “optional” and “main” are requisite relationships between business process steps for BCI.
The most complex and important is the relationship between information objects and business process steps. BCI asks for a relationship that states which process step creates the information objects and in which steps the information objects are used [3]. The cross-model table Information Use Table (IUT) collects the information objects and process steps from SM and PM respectively, and assigns the information objects to the relevant steps. But the relation of “use” in IUT does not distinguish the relations of “create” and “use” which are demanded in BCI. Thus the IUT cannot provide sufficient information, which implies that additional division of the relations between information objects and process steps needs to be done over IUT.

Considering the complexity in generating the information from DEMO aspect models for BCI, the transformation of DEMO models ought to be able to relieve people of the need for massive information collection and analysis. The requisite information should be gathered from various resources automatically, and reduce the complexity of generating the relationships among the information as much as possible. Practically and ideally, a successful transformation will avoid any information loss, and expand the usage of the transformed information in a wide range of fields.

4.2 Transformation Rules

We need rules to define the contents of the DEMO transformation and the way that the transformation should be elaborated. In this section, we specify the rules that have been followed during the entire transformation procedure, which are selection rules, structuring rules and mapping rules. Selection rules and structuring rules are used during the DEMO transformation, while mapping rules are sued to verify the transformation results.

Selection rules
The core information in the DEMO metamodel must be transformed into XML schema completely. Selection rules aim to define the complete and essential information objects for the transformation. Regarding the fact that the DEMO metamodel is a big State Model (SM)[5], we select information in the same way as the one used in [9] for constructing the Information Use Table (IUT). Thus the target information to be selected includes the object class and fact type.
Tables 4.1, 4.2 and 4.3 list all the information objects chosen from the Meta Construction Model (MCM), Meta Process Model (MPM), Meta Action Model (MAM),

[5] We illustrated it in Chapter 3.5.3.

Meta State Model (MSM) and the metamodels of three cross-model tables. These information objects are either object classes or fact types contained in those metamodels.

Considering the unique characteristics of MSM[6], its being the meta schema of the DEMO metamodel, we make an additional rule when defining the to-be transformed information objects in MSM. It is: we only pick up the object classes and fact types that are directly used in constructing the instance models.
The interpretation of this rule is: the definitions of fact type and object class are exclusive in our selection. We only pick up the object classes and fact types that are used in the basic constructs mentioned in Chapter 3.5.3. In addition, the definitions of the existence laws are also taken into our account, since they are directly used in building the instance models by the other three metamodels (MCM, MPM and MAM).

Table 4.1: The selected information from the DEMO metamodel (MCM and MPM)

Meta Construction Model	**Meta Process Model**
TRANSACTION KIND	TRANSACTION PHASE
A is an initiator of T	P is the O-phase of T
A is the executor of T	P is the E-phase of T
ELEMENTARY ACTOR ROLE	P is the R-phase of T
A uses information from B	TRANSACTION STEP
INFORMATION BANK	S is a step in P
PRODUCTION BANK	T is initiated from P
B is the production bank of T	S1 is a wait condition for S2
COORDINATION BANK	
B is the coordination bank of T	
F is the result kind of T	

The selected information is the target information to be structured in the XML schema files in Chapter 5.

Structuring rules
The chosen contents from the DEMO metamodel must be structured hierarchically in the XML schema files. It implies that those information objects must be clustered as elements, complex types or attributes in XSD, regarding their different features and priorities in DEMO metamodel.

We define those which have the central position in the metamodel as the root element in the schema. By central position, we may mean several cases, such as: the one which holds the most dependency laws, or the one which is the most generically constructed in the metamodel. For instance, the transaction kind has the central position in MCM or even the entire DEMO metamodel, since in DEMO aspect

[6] It has been discussed in chapter 3.7.

Table 4.2: The selected information from the DEMO metamodel (MAM and MSM)

Meta Action Model	**Meta State Model**
ACTION RULE	FACT TYPE
R is the action rule for performing step S	OBJECT CLASS
	the domain of x is y
	x is a declared fact type
	SCALE
	PROPERTY
	c is the domain of x
	s is the range of x
	mutual exclusion holds for x and y
	x is dependent on y
	unicity holds for x

Table 4.3: The selected information from the DEMO metamodel (TRT, BCT and IUT)

Metamodel of the TRT	**Metamodel of the BCT**	**Metamodel of the IUT**
TRANSACTION KIND	INFORMATION BANK	TRANSACTION STEP
F is the result kind of T	B is the information bank of F	F is used in S
DECLARED FACT TYPE	FACT TYPE	FACT TYPE

models, CM is the most concise model, while the other models detail part of the CM; in MCM the transaction kind holds the most dependency laws. In MPM, the transaction phases are more generic than the transaction steps, since every transaction step belongs to one and only one transaction phases. In MAM, which is obvious, the ACTION RULE is the root element. The basic constructs in MSM occupy the central positions in the schema for the SM.

The other more detailed branch information, compared with the root elements, is defined as complex types in order to detail the root element. The existential constraints contained in the metamodels are defined as attributes of either elements or complex types. An example of the ruled structure is shown in Listing 4.1.

Listing 4.1: An example of defining the elements and complex types

```
<xsd:element name="ELEMENT" type="COMPLEX_TYPE"/>

<xsd:complexType name="COMPLEX_TYPE">
        ...
<xsd:attribute name="attribute" type="type" use="required/optional"/>
</xsd:complexType>
```

Besides considering the hierarchy in the chosen contents, we should also pay attention to the structure of the instance XML files. Sometimes we need to add an additional root element to have the structured model information nesting in the root element.

An example of the root element in the schema of the CM explains itself (Listing 5.1). The element <Transaction>and its complex type content "TRANSACTION_KIND" comprehends all the model information[7] within the transaction. However we set another element <Transactions>as the root element, because in CM there is usually more than one business transaction, so that we need the element <Transactions>to include all the <Transaction>elements in the instance XML file.

The structuring rules are applied to the construction of the XML schema for DEMO models in Chapter 5.

Mapping rules
This rule is made to guarantee the information completeness and correctness in the transformation results, in order to verify the transformation results. The mapping is made from two perspectives, which are the precision of the constraints in the XML schema files and the completeness of the information objects in the instance XML documents, compared with the original DEMO metamodel. Therefore, for each information object in the original DEMO metamodel, there should be a corresponding interpretation in the XML schema files in terms of either an element or a complex type; for every necessary constraint, there should also be an equivalent part in the schema files, in terms of an attribute of an element or a complex type.

The mapping rules are applied in the comparison between the produced instance XML documents and the original DEMO diagrams in Chapter 7.1.

4.3 Conclusion

In Chapter 1.1 the gap between the graphical DEMO models and the platform specific models was already mentioned. We described this gap in the first section of this chapter in more detail, in that it causes the complexities in the information generation from the DEMO models to the BCI in the BCI-3D tool. Two reasons for the complexities are found in our analysis, which are the diverse information resources and the indirect information. There are demands for easing the information generation procedure for these two reasons.

With a clear picture of the DEMO metamodel and the complexities in the current information generation procedure from the DEMO models to the BCI-3D tool, we define three transformation rules as guidance for the model transformation that will be done in this project. Following the selection rules, we decide which information should be chosen to be transformed. The structuring rules give us how the selected information should be structured in the XML schema. The mapping rules guarantee information completeness during the entire transformation process.

[7] The rest of the definition of the CM schema file is explained in Chapter 5.1.

Answers to the research question

In this chapter, the research question "How can the transformation rules be defined?" has been answered. We first analyzed the reasons for the complexities in generating the information from the DEMO models for BCI, and clarified the expectation of the DEMO model transformation. Combining the requirements with the expected DEMO transformation and the knowledge about the DEMO metamodel, we defined three rules in selecting information items from the metamodel, structuring the selected information in XSD, and guaranteeing information completeness and correctness in the transformation results.

Next Task

In Chapter 5 the interpretation of the DEMO metamodel will be made in XSD, under the direction of the transformation rules. The interpretation will transform the requisite information selected from the DEMO metamodel into a syntax-based format, and be structured hierarchically following the structuring rules. The selected information and the constructed elements or complex types in XSD should be able to be mapped with each other.

Chapter 5

DEMO Model schema

In Chapter 3, the DEMO metamodel showed the essential structure of DEMO models, including the MCM, MPM, MAM and MSM, as well as the metamodel for three cross-model tables, the TRT, BCT, and IUT. In this chapter, we will present the corresponding structure of the MCM, MPM, MAM, MSM, TRT, BCT, and IUT in XSD, including the interrelationships within them. By means of that, the graphical DEMO metamodel will be transformed into an exchangeable format, which can be used by third party applications.

Conforming with the way that we specified the DEMO metamodel in Chapter 3, we separately provide the specification about the XML schema for DEMO metamodel in five sections, namely the CM schema (Section 5.1), PM schema (Section 5.2), AM schema (Section 5.3), SM schema (Section 5.4) and the cross-model tables schema (Section 5.5). In each section, the schema will be explained in detail, and some design issues are discussed following the schema specification.

A short summary about the schema design is given at the end of this chapter, and answers to the second research question are provided. A quick reference about the elements, data types and attributes used in the designed XML schema can be found in Appendix A. The complete schema codes for CM, PM, AM, SM, TRT, BCT, and IUT can be viewed in Appendix B, with their corresponding graphical representation of the schema.

5.1 Construction Model schema

The Construction Model (CM) consists of a sequence of transactions that elementary actor roles and information access to information banks are involved. Regarding to the dependency laws held by class TRANSACTION KIND in the MCM, the way that represents the structure of CM in XML Schema should be transaction-centered. By transaction-centered, the global element in the instance CM XML document is <Transactions>.

The schema for the global element <Transactions>is expressed in Listing 5.1. The <Transactions>element content is defined as a sequence of <Transaction>element. Each <Transaction>element has complex type content “TRANSACTION_KIND”. The maximal time of <Transaction>element occurrence is unlimited in the sequence.

Listing 5.1: XML schema for global element <Transactions>

```
<xsd:element name=”Transactions”>
        <xsd:complexType>
                <xsd:sequence>
                        <xsd:element name=”Transaction” type=”
                            TRANSACTION_KIND” maxOccurs=”unbounded”/>
                </xsd:sequence>
        </xsd:complexType>
</xsd:element>
```

The <xsd:complexType>element contains the information that defines the complex type named “TRANSACTION_KIND” in Listing 5.2. It is formed up with a sequence of elements with element type names of Tname, Initiator, Executor, ProductionBank, CoordinationBank and Result. In addition, an attribute, TransactionID, is given in this complex type.
<Tname>is used to present the transaction name, in simple type of <xsd:string>. Elements <Initiator>and <Executor>are with complex type content, “ELEMENTARY_ACTOR_ROLE”. One transaction must have at least one initiator, and one and only one executor. So in every <Transaction>element, the number of occurrence of <Initiator>is minimal once and maximal unbounded; the number of occurrence of <Executor>is restrictly set to one for both attributes minOccurs and maxOccurs. Elements <UseInformation>is with complex type content, “INFORMATION_BANK”. The information bank used here we mean the combination of the production bank and the coordination bank that belong to a transaction, because, in the instance ISM, we interpret the transaction symbol as the combination of the production bank and the coordination bank that belong to a transaction [9]. We specify one for their attribute minOccurs and maxOcurs, since it is necessary to have the information bank in the parent element <Transaction>, and the interrelationship between transaction kind and information banks are strict one-to-one relationship. Element <Result>is with complex type content, “DeclaredFactType”. It must occur once and exactly once in its parent element, so its number of occurrence is restricted to one, for both attributes minOccurs and maxOccurs.
Every transaction should have an uniquely identified ID. The <xsd:attribtue>element named “TransactionID” is added to the complex type content, in simple type of <xsd:string>. The use of this attribute is required.

Listing 5.2: XML schema for complex type named TRANSACTION_KIND

```
<xsd:complexType name=”TRANSACTION_KIND”>
        <xsd:sequence>
                <xsd:element name=”Tname” type=”xsd:string”/>
                <xsd:element name=”Initiator” type=”ELEMENTARY_ACTOR_ROLE”
                     minOccurs=”1” maxOccurs=”unbounded”/>
                <xsd:element name=”Executor” type=”ELEMENTARY_ACTOR_ROLE”
                    minOccurs=”1” maxOccurs=”1”/>
                <xsd:element name=”UseInformation” type=”INFORMATION_BANK”
                     minOccurs=”1” maxOccurs=”1”/>
                <xsd:element name=”Result” type=”DeclaredFactType”
                    minOccurs=”1” maxOccurs=”1”/>
        </xsd:sequence>
        <xsd:attribute name=”TransactionID” type=”xsd:string” use=”
             required”/>
</xsd:complexType>
```

In Listing 5.3, the content of complex type “ELEMENTARY_ACTOR_ROLE” is defined. It consists of a sequence of elements with element type names of name and UseInformation, plus an attribute with name of ActorID.
<name>is used to present the name of the actor role, in simple type of <xsd:string>.
Element <UseInformation>is with complex type content, “INFORMATION_BANK”.
The number of the occurrence of <UseInformation>is not limited, so we set zero for attribute minOccurs and unbounded for attribute maxOccurs.
An unique actor role identification is assigned to every elementary actor role. It is set by the attribute “ActorID”, in simply type <xsd:string>. The use of this attribute is required.

Listing 5.3: XML schema for complex type named ELEMENTARY_ACTOR_ROLE

```
<xsd:complexType name=”ELEMENTARY_ACTOR_ROLE”>
        <xsd:sequence>
                <xsd:element name=”name” type=”xsd:string”/>
                <xsd:element name=”UseInformation” type=”INFORMATION_BANK”
                     minOccurs=”0” maxOccurs=”unbounded”/>
        </xsd:sequence>
        <xsd:attribute name=”ActorID” type=”xsd:string” use=”required”/>
</xsd:complexType>
```

The content of complex type “INFORMATION_BANK” is defined in Listing 5.4. This complex type has simple content, which is extended by adding two attributes. These two attributes are nested within the <xsd:extension>element, and based on simple type <xsd:string>.
The type of the information bank is set by the attribute “BankType”. The choices of this attribute is restricted to production and coordination. This restriction is accomplished by using the <xsd:restriction>element. Two values, Prodcution and Coordination, in simple type <xsd:string>can be chose as the value of the attribute “BankType”.
The attribute “BankID” is used to assign a unique identification to every information bank; the “BankID” is with simple type content <xsd:string>.

Listing 5.4: XML schema for complex type named INFORMATION_BANK

```
<xsd:complexType name=”INFORMATION_BANK”>
        <xsd:simpleContent>
                <xsd:extension base=”xsd:string”>
```

```
                        <xsd:attribute name=”BankType” use=”optional”>
                            <xsd:simpleType>
                                <xsd:restriction base=”xsd:string
                                    ”>
                                        <xsd:enumeration value=”
                                            Production”/>
                                        <xsd:enumeration value=”
                                            Coordination”/>
                                </xsd:restriction>
                            </xsd:simpleType>
                        </xsd:attribute>
                        <xsd:attribute name=”BankID” type=”xsd:string” use
                            =”required”/>
                    </xsd:extension>
        </xsd:simpleContent>
</xsd:complexType>
```

The last complex type content definition is for complex type “DeclaredFactType”. Shown in Listing 5.5, its simple content is extended by adding one attribute. The attribute “ResultID” is required to use in this complex type, which content is to identify a unique transaction result in simple type <xsd:string>.

Listing 5.5: XML schema for complex type named DeclaredFactType

```
<xsd:complexType name=”DeclaredFactType”>
        <xsd:simpleContent>
                <xsd:extension base=”xsd:string”>
                        <xsd:attribute name=”ResultID” type=”xsd:string”
                            use=”required”/>
                </xsd:extension>
        </xsd:simpleContent>
</xsd:complexType>
```

It is apparently to see the convenience of the transaction-centered design. The initiator and executor are vital elements in the complex type of TRANSACTION_KIND, since there are dependency laws hold between transaction kind and them respectively in MCM. In addition, the transaction kind is the connection between MCM and MPM that it is easy to search for the relevant information if setting the transaction as root element.

5.2 Process Model schema

The Process Model (PM) consists of a sequence of transaction phases for each transaction kind. Each transaction phase contains a sequence of transaction steps. They together form up the transaction pattern in PM. From analyzing the MPM (Chapter 3.3), we are aware of that those transaction phases and transaction steps covers all the situations that may occur in PM, namely all the transaction patterns can be achieved based on the structure introduced in MPM.

In the schema for PM, we define the transaction pattern as the global element, since it includes the transaction phases and transaction steps. Listing 5.6 shows

the definition of element <TransactionPattern>in XML schema. This global element is defined as a sequence of <Transaction>element. There must be at least one <Transaction>element in <TransactionPattern>, and there is no constraint for the occurrence of <Transaction>. Thus the attributes minOccurs and maxOccurs of <Transaction>are set to one and unbounded respectively.
The element <Transaction>is composed a sequence of <TransactionPhase>element, and two attributes "TransactionID" and "name". The element <TransactionPhase>is with complex type content, "TRANSACTION_PHASE". For every transaction, it is not necessary to have a transaction phase, regarding to the MPM. But the maximal number of occurrenc of transaction phase in one transaction can not be greater than 3. So we set zero for attribute minOccurs, and three for attribute maxOccurs. The attribute "TransactionID" for <Transaction>is required. A unique value should be assigned to this attribute in simple type <xsd:string>. The attribute "name" for <Transaction>is used to present the description of the transaction, in simple type <xsd:string>.

Listing 5.6: XML schema for global element <TransactionPattern>

```
<xsd:element name=”TransactionPattern”>
        <xsd:complexType>
                <xsd:sequence>
                        <xsd:element name=”Transaction” minOccurs=”1”
                            maxOccurs=”unbounded”>
                                <xsd:complexType>
                                        <xsd:sequence>
                                                <xsd:element name=”
                                                     TransactionPhase” type
                                                     =”TRANSACTION_PHASE”
                                                     minOccurs=”0”
                                                     maxOccurs=”3”/>
                                        </xsd:sequence>
                                        <xsd:attribute name=”TransactionID
                                             ” type=”xsd:string” use=”
                                             required”/>
                                        <xsd:attribute name=”name” type=”
                                             xsd:string”/>
                                </xsd:complexType>
                        </xsd:element>
                </xsd:sequence>
        </xsd:complexType>
</xsd:element>
```

The schema for complex type "TRANSACTION_PHASE" is showed in Listing 5.7. This <xsd:complexType>element contains a sequence of unlimited child element <step>, which presents the transaction step in PM. A required attribute is used to assign the name of the transaction phase.
It is possible that there is no transaction step in some transaction phase, regarding to the fact that a transaction may be terminated in early transaction phase. The reason to the unlimited occurrence of transaction step in one transaction phase is that it is unknown which transaction pattern is being followed; the acts performed by actor roles are different regarding to various situations. So the attributes minOccurs and maxOccurs are set with zero and unbounded respectively. The content of

<step>is defined in complex type “TRANSACTION_STEP”.
The name of the transaction phase is restricted to Order, Execution or Result, which is done by the element <xsd:restriction>in attribute “name”. The values of these three options are in simple type <xsd:string>.

Listing 5.7: XML schema for complex type named TRANSACTION_PHASE

```
<xsd:complexType name=”TRANSACTION_PHASE”>
        <xsd:sequence>
                <xsd:element name=”step” type=”TRANSACTION_STEP” minOccurs
                    =”0” maxOccurs=”unbounded”/>
        </xsd:sequence>
        <xsd:attribute name=”name” use=”required”>
                <xsd:simpleType>
                        <xsd:restriction base=”xsd:string”>
                                <xsd:enumeration value=”Order”/>
                                <xsd:enumeration value=”Execution”/>
                                <xsd:enumeration value=”Result”/>
                        </xsd:restriction>
                </xsd:simpleType>
        </xsd:attribute>
</xsd:complexType>
```

The schema of the last complex type content in MPM, “TRANSACTION_STEP”, is defined in Listing 5.8. This complex type presents the structure of the transaction step. The content of this complex type element includes a sequence of elements, which are <Name>, <InitiatedFrom>, <InitiationTo>, <IsWaitConditionOf>and <WaitingFor>. In addition, an attribute “TransactionID” is given to “TRANSACTION_STEP”.
<Name>is used to denote which transaction step it is. It is expressed in simple type <xsd:string>. And there is only one name for one transaction step, thus the attributes minOccurs and maxOccurs are set with one.
<InitiatedFrom>and <InitiationTo>describe the initiation condition between transaction steps. Note that the conditions we mean here is among different transactions, the one within the same transactions is not taken into account. <InititiatedFrom>is needed when the parent element <step>is initiated from a step in another transaction. The content of <InitiatedFrom>is in complex type “TRANSACTION_STEP”, which means the initiation of one transaction step is another transaction step. This element is only necessary when there is such activation from another transaction, and it is possible that the activation is from more than one transaction. Thus zero is set to attribtue minOccurs and unbounded is set to attribute maxOccurs.
Contrary to <InitiatedFrom>, <InitiationTo>is used when the parent transaction step initiates some transaction steps in other transactions. Similar with <InitiatedFrom>, the content of <InitiationTo>is in complex type “TRANSACTION_STEP”; its attributes minOccurs and maxOccurs are set with zero and unbounded, since not every transaction step initiates steps in other transactions, but once it does, it can activate the start of more than one step in different transactions.
<IsWaitConditionOf>and <WaitingFor>are two elements used to describe the wait condition between transaction steps. As same as <InitiatedFrom>and <Ini-

tiationTo>, it only applies to conditions between different transactions. When the start of one transaction step needs to wait for the completion of another transaction's step, the awaiting transaction step needs to be marked by using <WaitingFor>element, with noting which transaction step it is waiting for. The element <IsWaitConditionOf>is denoted in the other transaction step, with noting of which transaction step it is the wait condition. For both of these two elements, there is no constraint for either attribute minOccurs or attribute maxOccurs.
Note that the occurrence of elements <InitiatedFrom>and <InitiationTo>in instance PM should be consistent, so do the elements <IsWaitingConditionOf>and <WaitingFor>. Because these two pairs of elements specify the same interrelationship respectively. Once transaction step *s1* initiates transaction step *s2*, *s2* is initiated from *s1*. Once transaction step *s1* is the waiting condition of transaction step *s2*, *s2* waits for the completion of *s1*.
The attribute, "TransactionID", is optional for the complex type "TRANSACTION_STEP". Only when one of the child elements in this complex type content refers to a transaction step of another transaction, does the value of the transaction ID need to be declared in this attribute. If a transaction step does not have any connection with another transaction, this attribute does not make any sense, since the ID for the step's belonged transaction is already given in its ancestor element <Transaction>.

Listing 5.8: XML schema for complex type named TRANSACTION_STEP

```
<xsd:complexType name=”TRANSACTION_STEP”>
        <xsd:sequence>
                <xsd:element name=”Name” type=”xsd:string” minOccurs=”1”
                    maxOccurs=”1”/>
                <xsd:element name=”InitiatedFrom” type=”TRANSACTION_STEP”
                    minOccurs=”0” maxOccurs=”unbounded”/>
                <xsd:element name=”InitiationTo” type=”TRANSACTION_STEP”
                    minOccurs=”0” maxOccurs=”unbounded”/>
                <xsd:element name=”IsWaitConditionOf” type=”
                    TRANSACTION_STEP” minOccurs=”0” maxOccurs=”unbounded
                    ”/>
                <xsd:element name=”WaitingFor” type=”TRANSACTION_STEP”
                    minOccurs=”0” maxOccurs=”unbounded”/>
        </xsd:sequence>
        <xsd:attribute name=”TransactionID” type=”xsd:string” use=”
            optional”/>
</xsd:complexType>
```

5.3 Action Model schema

The Action Model (AM) contains a set of action rules of an organization. The global element in the schema for AM is the action rules, which includes the specific rules. Listing 5.9 defines the element <ActionRules>in XML schema.
The content of this global element is a sequence of <Rule>element, with complex type content "ACTION_RULE". The numbers of the rules are not limited, so the attribute minOccurs is set with zero, and the attribute maxOccurs is set unbounded.

Listing 5.9: XML schema for global element <ActionRules>

```
<xsd:element name=”ActionRules”>
        <xsd:complexType>
                <xsd:sequence>
                        <xsd:element name=”Rule” type=”ACTION_RULE”
                             minOccurs=”0” maxOccurs=”unbounded”/>
                </xsd:sequence>
        </xsd:complexType>
</xsd:element>
```

The content of complex type “ACTION_RULE” is defined in Listing 5.10. This complex type contains simple content with two attributes. Based on the simple type <xsd:string>, this simple content is extended by adding attributes “TransactionID” and “step”. These two attributes are required to be used, for indicating to which transaction step in which transaction the action rule is concerning about.

Listing 5.10: XML schema for complex type named ACTION_RULE

```
<xsd:complexType name=”ACTION_RULE”>
        <xsd:simpleContent>
                <xsd:extension base=”xsd:string”>
                        <xsd:attribute name=”TransactionID” type=”xsd:
                             string” use=”required”/>
                        <xsd:attribute name=”step” type=”xsd:string” use=”
                             required”/>
                </xsd:extension>
        </xsd:simpleContent>
</xsd:complexType>
```

5.4 State Model schema

In the light of the two roles the MSM has (illustrated in Chapter 3.5), the metamodel for SM and the meta schema for the whole DEMO metamodel, a filtration needs to be made before we design the schema for SM, by distinguishing the definitions for basic concepts and the ones for basic constructs. The basic constructs are the ones directly used to structure the instance SM, such as a binary fact type. The basic concepts are the ones used to form up the basic construct, such as the fact types, object classes, and existence laws. It is advisable to bear in mind that SM is made up of a set of basic constructs, which are made of object classes, fact types, and existence laws.

The global element set for SM in instance document is <ObjectFact>. Its content is performed by a set of basic constructs which are contained in the global element as element <BasicConstruct>. The content of <BasicConstruct>is in complex type “BasicConstruct”, which will be discussed later in this chapter. There is no constraint about the occurrence of this element, attributes minOccurs and maxOccurs are set with zero and unbounded respectively. Listing 5.11 shows the schema of the global element <ObjectFact>.

Listing 5.11: XML schema for global element <ObjectFact>

```
<xsd:element name=”ObjectFact”>
        <xsd:complexType>
                <xsd:sequence>
                        <xsd:element name=”BasicConstruct” type=”
                             BasicConstruct” minOccurs=”0” maxOccurs=”
                             unbounded”/>
                </xsd:sequence>
        </xsd:complexType>
</xsd:element>
```

Being aware with that the definitions of basic concepts, such as definitions for object class, fact type and existence laws, are not considered in the schema for basic construct in SM, the issues taken into account are the roles of a fact type, the domains or scales of those roles and the types of the existence laws held in the basic construct. It is going to explain how these issues are structured in the schema for SM.

There might be different approaches to proceed the design for the schema of basic construct. Here we choose the instance of FACT TYPE as the fundamental composition in the basic construct, since the fact type includes one or more roles regarding to the arity of the certain fact type. In the <xsd:complexType>element in Listing 5.12, the content of “BasicConstruct” is a sequence of element <role>, with comlex type content “FACT_TYPE”. The attributes minOccurs and maxOccurs of <role>is set with zero and unbounded respectively, which means the basic construct is allowed to be empty and without constraint on the maximal number of <role>in one basic construct. A formulation is set to the basic construct by attribute “formulation” in simple type <xsd:string>. Another attribute of the complex type “BasicConstruct” is “FactTypeID”, which is used to assign a unique identification to the construct. This attribute is required and with value in simple type <xsd:string>.

Listing 5.12: XML schema for complex type named BsicConstruct

```
<xsd:complexType name=”BasicConstruct”>
                <xsd:sequence>
                        <xsd:element name=”role” type=”FACT_TYPE”
                             minOccurs=”0” maxOccurs=”unbounded”/>
                </xsd:sequence>
                <xsd:attribute name=”Formulation” type=”xsd:string”/>
                <xsd:attribute name=”FactTypeID” type=”xsd:string” use=”
                     required”/>
</xsd:complexType>
```

The conditions that a fact type can have are the issues need to be declared next. According to the basic construct discussed in Chapter 3.5.3, the role of a fact type can have an object class as its domain, or have a type of scale as its range[1]. It can also hold a unicity law to constrain its appearance in the same instance fact type if necessarily. Another condition that need to be concerned is the extension of the

[1] Scale is chosen when object property is the case. Object property is a binary fact type.

fact type. Some role of a fact type can be extended to a new type of object class. In Listing 5.13, these conditions are defined as below:
The conditions of having a domain or having a range, and being extended to a new object class are defined in a sequence of elements. <HasDomain>and <IsExtendedTo>are with complex type content “OBJECT_CLASS”, while <HasRange>is with complex type content “Scale”. Regarding to the constraint that the same role of a fact type cannot have a domain and a range at the same time, the elements <HasDomain>and <HasRange>are made as child elements of element <xsd:choice>. <xsd:choice>enables only one of its child elements to appear in the instance document, which means either <HasDomain>or <HasRange>will be chose in the instance fact type at one time. Not every fact type can be extended to a new object class, thus the attribute minOccurs of <IsExtendedTo>is set with zero.
The unicity law is defined as an attribute of the <xsd:complexType>element, “HoldUnicity”. The choice of this attribute is restricted to “Yes” and “No”, indicating the unicity law is held or not. This restriction is accomplished by using the element <xsd:restriction>, with two values of “Yes” and “No” in simple type <xsd:string>.
Another attribute of the complex type “FACT_TYPE” is “name”, which is used to assign a unique name to each role in the fact type. This attribute is required and with value in simple type <xsd:string>.

Listing 5.13: XML schema for complex type named FACT_TYPE

```
<xsd:complexType name=”FACT_TYPE”>
                <xsd:sequence>
                        <xsd:choice>
                                <xsd:element name=”HasDomain” type=”
                                    OBJECT_CLASS”/>
                                <xsd:element name=”HasRange” type=”Scale
                                    ”/>
                         </xsd:choice>
                        <xsd:element name=”IsExtendedTo” type=”
                            OBJECT_CLASS” minOccurs=”0” maxOccurs=”
                            unbounded”/>
                 </xsd:sequence>
                <xsd:attribute name=”HoldUnicity” default=”Yes”>
                        <xsd:simpleType>
                                <xsd:restriction base=”xsd:string”>
                                        <xsd:enumeration value=”Yes”/>
                                        <xsd:enumeration value=”No”/>
                                 </xsd:restriction>
                         </xsd:simpleType>
                 </xsd:attribute>
                <xsd:attribute name=”name” type=”xsd:string” use=”required
                    ”/>
</xsd:complexType>
```

In the next Listing 5.14, the schema of the complex type “OBJECT_CLASS” is described. There is only simple content in this complex type, and three attributes are nested in the simple content within <xsd:extension>element.
It is necessary to state that what kind of existence law is connecting the object

class and the fact type. This task is accomplished by the attribute “LawType”. The available existence laws for this attribute are reference law and dependency law. Actually the reference law must be obtained, otherwise there is no connection between the fact type and its corresponding domain[2]. The difference is only with or without dependency law due to different instance conditions. The element <xsd:restriction>constrains the values of the attribute to “Reference” and “Dependency”. By “Reference” it means the reference law is held between this object class and the fact type, by “Dependency” it means besides the referent law is held, a dependency law is also held for this object class.
Like fact type, every object class must has its own identification to be distinguished from each other. The attribute “ObjectClassID” presents this value in simple type <xsd:string>. The usage of this attribute is required.

On some occasions, there are result types declared on object classes. An addition attribute “ResultID” is used to denote the ID of the result type, which corresponding result type can be found in the relevant CM. The usage of this attribute is optional.

Listing 5.14: XML schema for complex type named OBJECT_CLASS

```
<xsd:complexType name=”OBJECT_CLASS”>
        <xsd:simpleContent>
                <xsd:extension base=”xsd:string”>
                <xsd:attribute name=”LawType” use=”optional” default=”
                     Reference”>
                <xsd:simpleType>
                        <xsd:restriction base=”xsd:string”>
                                <xsd:enumeration value=”Reference”/>
                                <xsd:enumeration value=”Dependency”/>
                        </xsd:restriction >
                </xsd:simpleType>
                </xsd:attribute >
                <xsd:attribute name=”ObjectClassID” type=”xsd:string” use
                     =”required”/>
                <xsd:attribute name=”ResultID” type=”xsd:string” use=”
                     optional”/>
                </xsd:extension >
        </xsd:simpleContent>
</xsd:complexType>
```

The last schema design is for the complex type content “Scale” (Listing 5.15). There is no need to specify the type of existence law held for the scale, according to the basic construct of object property in Figure 3.14, the reference law is the only type of connection between the object property and its corresponding scale. The variety in this <xsd:complexType>element is the types of the scale. There are five types of scale: absolute, ratio, interval, ordinal and categorial. The choice is realised by element <xsd:attribute>, the values of thses five types are represented by capital letters A, R, I, O and C in simple type <xsd:string>. As an absolutely necessary attribute, “ScaleID” is given in the content of this complex type “Scale”.

[2] Examples of the basic construct with different laws are discussed in Chapter 3.5.3

Listing 5.15: XML schema for complex type named Scale

```
<xsd:complexType name=”Scale”>
        <xsd:simpleContent>
                <xsd:extension base=”xsd:string”>
                <xsd:attribute name=”ScaleType” use=”optional” default=”A
                    ”>
                <xsd:simpleType>
                        <xsd:restriction base=”xsd:string”>
                                <xsd:enumeration value=”A”/>
                                <xsd:enumeration value=”R”/>
                                <xsd:enumeration value=”I”/>
                                <xsd:enumeration value=”O”/>
                                <xsd:enumeration value=”C”/>
                        </xsd:restriction>
                </xsd:simpleType>
                </xsd:attribute>
                <xsd:attribute name=”ScaleID” type=”xsd:string” use=”
                    required”/>
                </xsd:extension>
        </xsd:simpleContent>
</xsd:complexType>
```

5.5 Cross-model tables schema

In cross-model tables, the TRT, BCT and IUT, the information is in terms of tabular data. In each instance table, there are two columns listing the information, which is from different DEMO aspect models. In every row of the table, the information from different columns is associated with each other, due to the different relationships titled in these three tables. As already seen in Chapter 3.6, the metamodels for these cross-model tables are with similar structures, namely a binary fact type, however the information objects and existence laws in those binary fact types are different from each other. Regarding their simple structures and the transformation rules 4.2, we design the XML schema for them in a similar way.

Listing 5.16 shows the schema for the TRT. The global element <TransactionResults>is with complex type content, which is performed by a set of transaction results. The information of a transaction result is contained in the element <TransactionResult>, which has the complex type content “TRT”. We do not restrict the occurrence of <TransactionResult>, since a table can be empty or has numerous rows. So we set zero and unbounded to its attribute “minOccurs” and “maxOccurs” respectively.
The content of the complex type “TRT” is performed by a sequence of elements <TransactionType>and <ResultType>, which are in simple type <xsd:string>. Regarding the existence laws they hold in the metamodel, shown in Figure **??**, we set the attribute “maxOccurs” with one for each of them, since they both hold a unicity law in the lawful binary fact type. We set the attribute “minOccurs” with one for <TransactionType>, and the attribute “minOccurs” with zero for <ResultType>, because a dependency law holds on the transaction type, but not on the

result type.

Listing 5.16: XML schema for Transaction Result Table

```
<xsd:element name=”TransactionResults”>
        <xsd:complexType>
                <xsd:sequence>
                        <xsd:element name=”TransactionResult” type=”TRT”
                             minOccurs=”0” maxOccurs=”unbounded”/>
                </xsd:sequence>
        </xsd:complexType>
</xsd:element>
<xsd:complexType name=”TRT”>
        <xsd:sequence>
                <xsd:element name=”TransactionType” type=”xsd:string”
                     minOccurs=”1” maxOccurs=”1”/>
                <xsd:element name=”ResultType” type=”xsd:string” minOccurs
                     =”0” maxOccurs=”1”/>
        </xsd:sequence>
</xsd:complexType>
```

In Listing 5.17 and 5.18, we present the schema for the BCT and IUT respectively. As mentioned at the beginning of this section, the only differences in the metamodels of these tables are the information objects and the existence laws.

In the schema for the BCT (Listing 5.17), the global element is <BankContents>, which is made up of a sequence of <BankContent>. In the complex type content of element <BankContent>, two elements are <InformationBank>and <FactType>. There are dependency laws hold for both the information bank and the fact type in the metamodel of the BCT, so we set the attribute “minOccurs” with one for them. There is a unicity law holds to restrict the occurrence of the fact type in the binary fact type. Note that, in the metamodel of the BCT, holding the unicity law means that every fact type is not allowed to belong to more than one information bank, and every information bank can includes more than one fact types. The attributes “minOccurs” and “maxOccurs” restrict the occurrence of the element in the BCT, so we set the attributes “maxOccurs” with one for <InformationBank>, and the attribute “maxOccurs” with unbounded for <FactType>.

Listing 5.17: XML schema for Bank Contents Table

```
<xsd:element name=”BankContents”>
        <xsd:complexType>
                <xsd:sequence>
                        <xsd:element name=”BankContent” type=”BCT”
                             minOccurs=”0” maxOccurs=”unbounded”/>
                </xsd:sequence>
        </xsd:complexType>
</xsd:element>
<xsd:complexType name=”BCT”>
        <xsd:sequence>
                <xsd:element name=”InformationBank” type=”xsd:string”
                     minOccurs=”1” maxOccurs=”1”/>
                <xsd:element name=”FactType” type=”xsd:string” minOccurs
                     =”1” maxOccurs=”unbounded”/>
        </xsd:sequence>
</xsd:complexType>
```

In the schema for the IUT (Listing 5.18), the global element is <InformationUsage>, which is made up of a sequence of <InformationUse>. The content of <InformationUse>is performed by two elements, namely <InformationObject>and <TransactionStep>. Note that, in the metamodel of the IUT, there is no unicity law and dependency law hold for neither the information object nor the transaction step. We set unbounded to the attribute “maxOccurs”, and set zero to the attribute “minOccurs” for both <InformationObject>and <TransactionStep>.

Listing 5.18: XML schema for Information Use Table

```
<xsd:element name=”InformationUsage”>
        <xsd:complexType>
                <xsd:sequence>
                        <xsd:element name=”InformationUse” type=”IUT”
                             minOccurs=”0” maxOccurs=”unbounded”/>
                </xsd:sequence>
        </xsd:complexType>
</xsd:element>
<xsd:complexType name=”IUT”>
        <xsd:sequence>
                <xsd:element name=”InformationObject” type=”xsd:string”
                     minOccurs=”0” maxOccurs=”unbounded”/>
                <xsd:element name=”TransactionStep” type=”xsd:string”
                     minOccurs=”0” maxOccurs=”unbounded”/>
        </xsd:sequence>
</xsd:complexType>
```

5.6 Conclusion

Schema design summary

On the basis of analyzing the DEMO metamodel, this chapter presents the schema for DEMO models in XSD. The schema design is elaborated in seven separate XSD files, which comprehend the structures for all possible conditions in instance DEMO models, including the four graphical DEMO diagrams and three cross-model tables. We follow the structuring rules defined in Chapter 4.2 during the whole design procedure.

The schema for CM is transaction-center designed. It emphasizes the importance of business transactions in an organization’s construction, and its core position in the whole DEMO metamodel. Different actor roles are played around the transactions and the usage of information banks also depends on the demands from specific transactions.

The essential interrelationships in the PM, especially interrelationships among the transaction steps, constitute the schema for PM. It focuses on the internal sequence of a transaction, in terms of three transaction phases. Transaction steps are contained in transaction phase, with specification about the the four possible connections with step in other transactions. Transaction phases and transaction steps belong to the same transaction are associated with each other, and identified by the transaction ID, which also demonstrates the central position of transaction kind in

the structure of DEMO metamodel.
The schema for AM is the simplest one, only with one basic construct, namely action rules. Every action rule does not have any constraints about its content, but is strictly connected with a transaction step.
In the schema for SM, fact type is chosen as the starting point to proceed the basic construct of SM. It covers unlimited roles to constitute unary fact type, binary fact type, property or other possible constructs in SM. The specification about associations with object classes and scales are declared clearly, so are the existence laws held between them. Some object classes may have result types declared on them, that the result is from relevant transactions in CM.
We also provide the schema for three cross-model tables, the TRT, BCT, and IUT. Regarding their metamodels (Chapter 3.6), which are binary fact types, the schema for those models is in a similar structure. The differences between their schema are the elements, which represent the information objects in the metamodel, and the allowed occurrence of those elements, which represent the existence laws in the metamodel.

The XML schema for DEMO models may be various. The one we proposed in this chapter interpret the graphical DEMO metamodel into XSD in a concise, coherent and comprehensive way. We tried to reduce the duplication of the modeling information in the schema as much as possible, while maintained necessary connections with other relevant information. The instance XML documents based on the designed schema are intended to carry all the modeling information from the instance DEMO models. And those graphical modeling information is transformed into the XML based format, and stored in a structural way in terms of XML documents.

During the schema design procedure, we applied the transformation rules that have been defined in Chapter 4.2, to guide the transformation from DEMO metamodel to XML schema.

Completeness of the DEMO model schema
In compliance with the DEMO metamodel, which is introduced in Chapter 3, we provide the XML schema for the MCM, MPM, MAM, MSM, and metamodels of cross-model tables (TRT, BCT, and IUT). The designed XML schema is the interpretation of the DEMO metamodel. Even the three cross-model tables are displayed in tabular format, which is already an exchangeable format, we still present the XML schema for them, so that the DEMO metamodel interpretation is complete.

Answers to research question
In this chapter, research question “How can we transform the graphical model information into a syntax-based format?” has been answered. Looking back at the transformation approach in Figure 2.4, the model transformation is done at the level

of metamodel due to applying the metamodeling. WOSL is the source language used by DEMO metamodel, and XML Schema has been chosen as the modeling language of the target model. The transformation is conducted by specifying the DEMO metamodel in terms of XSD. Referring to each information object in the DEMO metamodel, there is corresponding XML specification in the designed schema. Those information objects are transformed into either elements or complex types, according to the transformation rules (Chapter 4.2). The mapping between those information objects and elements or complex types guarantees the completeness of the transformation.

Next Task

Now we have the XML schema for CM, PM, AM, SM and cross-model tables TRT, BCT and IUT, which are the structures of the instance XML documents. In the following chapter, we will provide some instance XML documents against the designed schema. Those instance XML documents have the model information from the library case in [9], and are considered as the transformed DEMO models.

Chapter 6

Producing XML Documents

The XML documents for CM, PM, AM, SM and three cross-model tables (the TRT, BCT, and IUT) are produced separately in this chapter. We built an application to collect the model information, and create the XML files for each DEMO aspect model and cross-model table. The core information of each DEMO aspect model and cross-model table is required to be filled in different forms, each of which corresponds to different DEMO aspect models and cross-model tables. The XML documents are made by structuring those information items against the designed schema. In this chapter we separately provide the specification about the core information items required in our application, and the exemplary XML documents for each DEMO aspect model and cross-model table. For more information about this application, please see the manual in Appendix E. The screenshots of the information forms in the application for each DEMO aspect model can be found in Appendix C. The complete exemplary XML documents are attached in Appendix D.

6.1 Construction Model

As already mentioned in Chapter 3.2, the construction of CM is made up of the transaction types, the associated actor roles, and the information links between the actor roles and the information banks. Therefore the information items that need to be provided to construct the CM should include the transaction, the initiator and executor of the transaction, and the relevant information banks, as well as the interrelationships among them. Applying the selection rules, the mentioned information items for constructing CM have all been included in Table 4.1.

Figure C.1 shows a screenshot of the form of collecting the information for producing the XML document for CM. One transaction is produced by filling in the required information items, which includes the transaction name, its initiator and the information banks used by the initiator, its executor and the information banks used by the executor, and its transaction result. Note that compared with the selected information items in Table 4.1, in this form, we do not ask for the information bank that belong to the transaction. Since according to the MCM (Chapter 3.2), for every transaction kind its coordination bank and production bank are necessarily produced, our application automatically creates the information bank that belongs to the transaction for each transaction.

The XML document for CM is produced with the input information items. Figure C.2 is one exemplary XML file with only one transaction from the library case[1]. Each input items is displayed as either an element attribute or element content. The elements and their content in this XML document are structured against the designed XML schema, which has already been described in Chapter 5.1. The complete XML schema for CM could be found in Appendix B.1.

6.2 Process Model

The PM XML document should provide the same information as the PM does. PM details the business process steps of each transaction that turns out to be a sequence of process steps and responsibility areas ([9] and Chapter 3.3). Selected from the MPM (Table 4.1), the information items for constructing the PM XML document include the transaction steps, the transaction phases that those steps belong to, the interrelationships between transaction steps, and those between transaction steps and the transaction kind. We did not directly provide the information about the responsibility areas in MPM, since it could be generated from MCM that states which actor role performs which process steps.

For one transaction step, the interrelationship with the transaction kind and another transaction step could be interpreted into four kinds of conditions: it is initiated from another transaction step, it initiates another transaction step, it is the wait condition of another transaction step, it waits for the completion of another transaction step. There are three transaction phases, namely *order* phase (O-phase), *execution* phase (E-phase) and *result* phase (R-phase). Every transaction step must belong to one and only one of these transaction phases.
We provide the screenshot for the form, which collect the information items for creating PM XML document, in Figure C.3. Since the basic transaction pattern[2] is the common one that we often have, the button “Basic Pattern” provides a direct way to produce the XML document with just the basic transaction steps. The only required information is the transaction information, and the steps in the basic pattern are added to the transaction automatically in our application. However, for any other additional condition in the transaction process, the rest of the items in this form take care of that. In this form, we do not ask for input the information about the transaction phases, because as we already mentioned in Chapter 3.3 that every

[1] The description of the library case can be found in the appendix of the book [9].

[2] See chapter 10 of [9].

specific transaction step belongs to one of the three transaction phases, we set the transaction steps to their corresponding transaction phase automatically. Thus we only ask for the information about the transaction steps. In Appendix E, there is more about this application.

Figure C.4 presents one exemplary transaction process from the library case. It lists all the basic transaction steps of T04 that the steps request and promise belong to the O-phase; the step execute belongs to the E-phase; the steps state and accept belong to the R-phase. In addition, it also indicates the interrelationship with another transaction T05, that T04/pm initiates T05/re and T04/ex wait for the completion of T05/pm. The designed XML schema is assigned to the instance document. The complete XML schema for PM can be found in Appendix B.2, and has been described in Chapter 5.2.

6.3 Action Model

The content in the AM is the specification of the action rules for the entire organization. The rules are the main information items required for constructing the XML document for AM, denoting the specific transaction steps the rules are defined for. As seen in Table 4.2, we created the corresponding form in Figure C.5. The rule content is considered as one single entity; a specific transaction step is assigned to every rule. Figure C.6 shows one simple instance XML document for the AM, which is against the designed schema for AM (Chapter 5.3).

6.4 State Model

The SM specifies the object classes, fact types and result types, as well as the existential laws that hold. [9] The MSM generates the basic construct of the SM and provides the essential information items for building the SM. Following the selection rule and the structuring rules in Chapter 4.2, we pick up the information items that are directly used in the basic constructs, and structure them in the form of Figure C.7.

Every basic construct has the formulation to state the meaning of the certain fact type, while an unique fact type ID is set for identification. The information about the roles in the fact type is required to be input individually, specifying its name, its domain or scale, the relevant information regarding its domain or scale, the existential laws hold on this role, and the extension information of this role if necessary. In this way, the SM diagram is displayed in the XML documents in terms of a set of the basic constructs.

In the XML document shown in Figure C.8, part of the SM of the library case is represented in a few basic constructs, and semantically described with the input information. The structure of this XML document is against the designed schema for SM, which is specified in Chapter 5.4 and the complete version could be found in Appendix B.4.

6.5 Cross-Model Table

Transaction Result Table
TRT lists the transaction types and their corresponding transaction results, which are also required in producing the XML document for CM in Figure C.1. For convenience, our application automatically produces the XML document for TRT when the information is filled in the form for producing the XML document for the CM. The exemplary XML document for TRT is shown in Figure C.13. The complete XML schema for TRT XML document is in Listing B.5, in Appendix B.5.

Bank Contents Table
The information items required for producing the XML document for BCT are the information bank and the fact type, regarding the selected items in Table 4.3. The fact types are from the instance SM, which are instantiated into object classes, fact types or result types. In the form (Figure C.10) for producing the XML document for BCT, we ask for the name of the information bank, and the information objects. Figure C.11 shows an exemplary XML document for BCT, and the complete XML schema for this XML document is in Listing B.6, in Appendix B.5.

Information Use Table
The information items required for producing the XML document for IUT are the transaction step and the fact type, including object classes, fact types or result types, regarding the selected items in Table 4.3. In the form shown in Figure C.12, we import the information objects from the XML document for SM, the transaction steps from the XML document for PM. An exemplary XML document for IUT can be seen in Figure C.13. The complete XML schema for IUT is in Listing B.7, in Appendix B.5.

6.6 Conclusion

We present the instance XML documents of the library case for CM, PM, AM, SM, TRT, BCT, and IUT in this chapter. The structures of these XML documents are the schema which has been designed in Chapter 5. A simple application is built for collecting the model information and creating the XML files.

The required information items for creating the XML files are selected from the DEMO metamodel following the selection rules which are defined in Chapter 4.2.

In our application, the idea of producing the XML documents involves first collecting the required information items with concrete model information and then structuring them in the XML documents against our designed schema for each DEMO aspect model.
However, not all the information items are required to be filled in our application forms, for instance, the coordination bank and the production bank that belong to the transaction in CM, and the process steps within a transaction when it follows the basic pattern in PM. This information will be automatically created in the instance XML documents according to the transaction information.
In addition, most information for producing cross-model tables is from the DEMO aspect models, which does not need to be provided repeatedly. For instance, the TRT is automatically produced when the transaction type and result type is provided in the CM. The information objects used in the BCT and IUT are imported from the SM.

The exemplary XML documents are shown in Appendix D, from which we can see all the model information is included and structured in an XML based format. As we know from the background information (Chapter 2.4), the XML data can be accessed by other platform specific applications. Therefore the information in the DEMO models is transformed into an exchangeable format, and these exemplary XML documents are the transformation results, namely the expected platform specific models.

Answers to the research question
The research question “What would the transformed model look like?” has been answered by showing the exemplary XML documents in this chapter. These instance XML documents are the transformation model, in which the model information of the library case is structured in XML based format. Differentiated from the original DEMO diagrams, the aspect models are stored in terms of a set of elements in the XML documents respectively. The unit elements in the XML documents of CM, PM, AM and SM are <Transaction>, <Transaction>, <Rule>and <BasicConstruct>. The unit elements in the XML documents of TRT, BCT and IUT are <TransactionResult>, <BankContent>and <InformationUse>. Note that the <Transaction>in the XML documents for CM and PM has different contents. In the CM file the <Transaction>contains the information about the transaction, the actor roles and the information banks used by the actor roles. In the PM file, it contains the process steps within the transaction. Every <Rule>in the XML file of AM contains one action rule in the AM. The information in the SM is stored in a set of <BasicConstruct>in the XML file of SM. The information in three cross-model tables is stored in a set of elements, which contains a pair of elements representing the tabular data, in the XML documents for each table.

Next task
Till this chapter, we analyzed the DEMO metamodel (Chapter 3) and the demands

to transform the DEMO models (Chapter 4.1), defined the transformation rules (Chapter 4.2), designed the XML schema for the metamodel (Chapter 5), and transformed the original DEMO models into an XML based format (Chapter 6). In chapter 7, we will evaluate the whole transformation by verifying the transformation results and discussing the utilization of the transformation results.

Chapter 7

Transformation Evaluation

The instance XML documents that have been produced in Chapter 6 are the expected results of the DEMO models transformation, which is from a graphical format to an XML based format. Till now the DEMO models have been transformed from graphical diagrams into the XML based syntax. In this chapter, we will evaluate this transformation from two aspects. Firstly, the transformation has to be verified to see if the information is completely and correctly maintained during the transformation procedure. We provide a comparison between the XML documents and the original DEMO diagrams in Chapter 7.1 as the verification. Secondly, there should be some added value from the transformation result. We introduce an application of building the Create / Use Table in Chapter 7.2, as an example of the utilization of the transformation result in simplifying the information generation for Business Component Identification (BCI) from the DEMO models.

7.1 Transformation Verification

The exemplary XML documents provided in Chapter 6 are the target Platform Specific Models (PSMs) which are the results of the proposed DEMO transformation. In order to achieve such result in the transformation, we take the steps of specifying the metamodel of the original DEMO aspect models (Chapter 3), interpreting the metamodel into an XML based schema format (Chapter 5), and constructing the XML documents based on the schema files (Chapter 6). Next, verifying the transformation results is of significant importance to the feasibility of this proposed transformation approach. In the rest of this section, we will present a comparison of the XML documents and their corresponding original DEMO models.

We take the XML documents of the library case as the transformed models for comparison. The original DEMO models are provided in [9]. The comparison aims to map the information contained in the XML documents with that in the original models, to see if the information is completely and precisely maintained during the transformation. The comparison is conducted with the four aspect models respectively.

7.1.1 Construction Model

The comparison between the XML document and the original diagram of CM is carried out in the unit of the business transaction. We will take the first three out of ten transactions from the library case for detailed comparison (Figure 7.1). The comparison contents include the actor roles that participate in the transaction and the information banks belonging to or used by particular transaction (Figure 3.3). The complete XML documents can be found in Appendix D.1. Reading from these diagrams, the semantic description of the first three transactions is obtained as below:

T01 membership registration

CA02 (aspirant member) initiates T01 (membership registration), of which the executor is numbered A01 (register). The result type of T01 is that R01 membership M has been started. The coordination and production bank that belong to T01 are combined in the information bank PB01. In addition, internal actor role A01 gets to know some general information from CPB14, the personal information about the aspirant member from CPB11.

T02 membership fee payment

The initiator of T02 (membership fee payment) is A01 (register) and A10 (annual fee controller). This transaction is executed by CA02 (aspirant member), and the result R02 indicate that the fee membership M in year Y has been paid. The combination of the coordination bank and the production bank that belong to T02 is the information bank PB02. Internal actor role A01 and A10 get some general information from CPB14.

T03 reduced fee approval

A01 (register) and A10 (annual fee controller) initiate T03 (reduced fee approval); the reduced fee is approved by CA01 (board). The result R03 that the reduced fee for M in year Y is approved. Information bank PB03 is the combination of the coordination bank and the production bank that belong to the transaction T03. The general information is known by A01 and A10 from CPB14. A01 also uses library data from the information bank CPB12 in this transaction.

The core information in the above description can be precisely mapped with the elements and the complex types in the corresponding XML documents. We present the mapping in Table 7.1, Table 7.2 and Table 7.3.

7.1.2 Process Model

In order to be coherent and consistent with the comparison we did for the CM, we still choose the PM diagrams of the first three transactions of the library case.

As seen in the Figure 7.2, the process steps in a transaction are connected to each

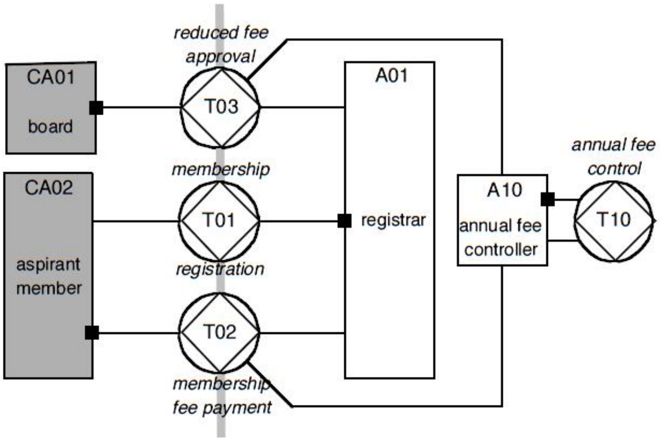


Figure 7.1: First part of the detailed ATD of the Library

Table 7.1: Mapping of T01 membership registration

T01 membership registration	<Transaction TransactionID=”T01”> <Tname>membership registration</Tname>
CA02 (aspirant member) initiates T01 information bank PB01 belong to transaction T01	<Initiator ActorID=”CA02”> <name>aspirant member</name> <UseInformation BankID=”PB01”>PB01</UseInformation> </Initiator>
the executor is A01 (register)	<Executor ActorID=”A01”> <name>registrar</name>
information bank PB01 belong to transaction T01	<UseInformation BankID=”PB01”>PB01</UseInformation>
A01 gets general information from CPB14	<UseInformation BankID=”CPB14” BankType=”Production”>general data</UseInformation>
A01 gets personal information from CPB11	<UseInformation BankID=”CPB11” BankType=”Production”>personal data</UseInformation> </Executor>
information bank PB01 belong to transaction T01	<UseInformation BankID=”PB01”>PB01</UseInformation>
result type of T01 is R01	<Result ResultID=”R01”>membership M has been started</Result> </Transaction>

other in a basic consequence by causal links. The causal link to T01/rq is an external one; the ones to T02/rq and T03/rq are internal, initiated from the C-result T01/pm. The conditional link from T03/ac to T02/rq means that dealing with the C-result T02/rq has to wait until the C-result T03/ac has been created, if there is a corresponding T03. A similar conditional link holds for T01/ex, the action of T01/ex can be taken until T02 is successfully completed. The responsibility areas are in compliance with the actor roles that participate in each transaction in the CM.

We present the XML document of the described process in Appendix D.2. For each transaction, the basic pattern is specified in a standard structure in an XML based format. Figure 7.3 shows the comparison of the basic pattern in the DEMO

Table 7.2: Mapping of T02 membership fee payment

T02 membership fee payment	<Transaction TransactionID=”T02”> <Tname>membership fee payment</Tname>
The initiator of T02 is A01 and A10	<Initiator ActorID=”A01”> <name>register</name>
information bank PB02 belong to T02	<UseInformation BankID=”PB02”>PB02</UseInformation>
A01 gets general information from CPB14	<UseInformation BankID=”CPB14” BankType=”Production”>general data</UseInformation> </Initiator>
	<Initiator ActorID=”A10”> <name>annual fee controller</name>
information bank PB02 belong to T02	<UseInformation BankID=”PB02”>PB02</UseInformation>
A10 gets general information from CPB14	<UseInformation BankID=”CPB14” BankType=”Production”>general data</UseInformation> </Initiator>
T02 is executed by CA02	<Executor ActorID=”CA02”> <name>aspirant member</name>
information bank PB02 belong to T02	<UseInformation BankID=”PB02”>PB02</UseInformation> </Executor>
information bank PB02 belong to T02	<UseInformation BankID=”PB02”>PB02</UseInformation>
T02 results R02	<Result ResultID=”R02”>the fee for membership M in year Y has been paid</Result> </Transaction>

Table 7.3: Mapping of T03 reduced fee approval

T03 reduced fee approval	<Transaction TransactionID=”T03”> <Tname>reduced fee approval</Tname>
A01 initiates T03	<Initiator ActorID=”A01”> <name>register</name>
information bank PB03 belong to T03	<UseInformation BankID=”PB03”>PB03</UseInformation>
A01 uses library data from CPB12	<UseInformation BankID=”CPB12” BankType=”Production”>library data</UseInformation>
A01 gets general information from CPB14	<UseInformation BankID=”CPB14” BankType=”Production”>general data</UseInformation> </Initiator>
A10 also initiates T03	<Initiator ActorID=”A10”> <name>annual fee controller</name>
information bank PB03 belong to T03	<UseInformation BankID=”PB03”>PB03</UseInformation>
A10 gets general information from CPB14	<UseInformation BankID=”CPB14” BankType=”Production”>general data</UseInformation> </Initiator>
the reduced fee is approved by CA01	<Executor ActorID=”CA01”> <name>board</name>
information bank PB03 belong to T03	<UseInformation BankID=”PB03”>PB03</UseInformation> </Executor>
information bank PB03 belong to T03	<UseInformation BankID=”PB03”>PB03</UseInformation>
T03 results R03	<Result ResultID=”R03”>the reduced fee for M in year Y is approved</Result> </Transaction>

diagram and in the XML based format.

The specification about the causal link from T01/pm to T03/rq is added to both T01/pm and T03/rq, the one from T01/pm to T02/rq is added to both T01/pm and T02/rq, the one about the conditional link from T03/ac to T02/rq is added to both T03/ac and T02/rq, the one from T02/ac to T01/ex is added to both T02/ac and

T01/ex. The XML specification about the mentioned special causal and conditional link is particularly picked up in Listings 7.1, 7.2, and 7.3.

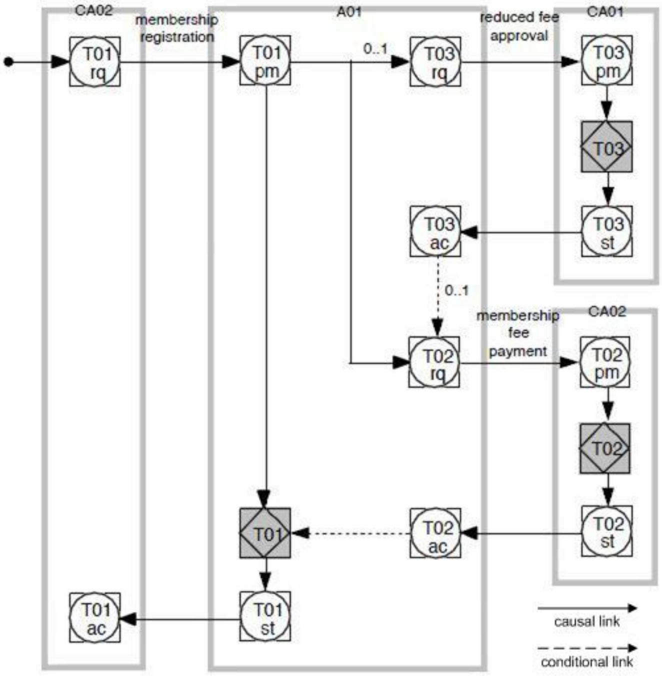
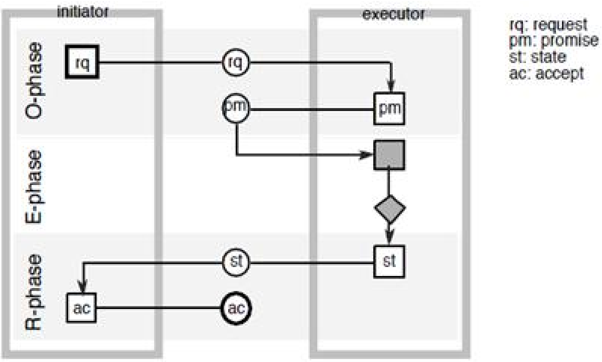


Figure 7.2: PSD of business process 1 of the Library



```
<TransactionPhase name="Order">
         <step>
            <Name>Request</Name>
         </step>
         <step>
            <Name>Promise</Name>
         </step>
</TransactionPhase>
<TransactionPhase name="Execution">
         <step>
            <Name>Execute</Name>
         </step>
</TransactionPhase>
<TransactionPhase name="Result">
         <step>
            <Name>State</Name>
         </step>
         <step>
            <Name>Accept</Name>
         </step>
</TransactionPhase>
```

Figure 7.3: The basic pattern in diagram and XML

Listing 7.1: The causal/conditional link in T01

```
<Transaction TransactionID=”T01” name=”membership registration”>
        <TransactionPhase name=”Order”>
                        ... ...
            <step>
                <Name>Promise </Name>
                <InitiationTo TransactionID=”T03”>
                    <Name>Request </Name>
                </InitiationTo>
                <InitiationTo TransactionID=”T02”>
                    <Name>Request </Name>
                </InitiationTo>
            </step>
        </TransactionPhase>
        <TransactionPhase name=”Execution”>
            <step>
                <Name>Execute </Name>
                <WaitingFor TransactionID=”T02”>
                    <Name>Accept </Name>
                </WaitingFor>
            </step>
        </TransactionPhase>
        ... ...
</Transaction>
```

Listing 7.2: The causal/conditional link in T02

```
<Transaction TransactionID=”T02” name=”membership fee payment”>
        <TransactionPhase name=”Order”>
            <step>
                <Name>Request </Name>
                <InitiatedFrom TransactionID=”T01”>
                    <Name>Promise </Name>
                </InitiatedFrom>
            </step>
            ... ...
        </TransactionPhase>
                        ... ...
        <TransactionPhase name=”Result”>
            ... ...
            <step>
                <Name>Accept </Name>
                <IsWaitConditionOf TransactionID=”T01”>
                    <Name>Execute </Name>
                </IsWaitConditionOf>
            </step>
        </TransactionPhase>
</Transaction>
```

Listing 7.3: The causal/conditional link in T03

```
<Transaction TransactionID=”T03” name=”reduced fee approval”>
        <TransactionPhase name=”Order”>
            <step>
                <Name>Request </Name>
                <InitiatedFrom TransactionID=”T01”>
                    <Name>Promise </Name>
                </InitiatedFrom>
            </step>
            ... ...
        </TransactionPhase>
```

```
                    ... ...
        <TransactionPhase name="Result">
            ... ...
            <step>
                <Name>Accept </Name>
                <IsWaitConditionOf TransactionID="T02">
                    <Name>Request </Name>
                </IsWaitConditionOf>
            </step>
        </TransactionPhase>
</Transaction>
```

7.1.3 Action Model

Because the single construct in the MAM is the action rules, the expression of the action rule in the XML documents is semantically the same as the original ones in [9]; the comparison of the XML document and the original AM is abbreviated here.

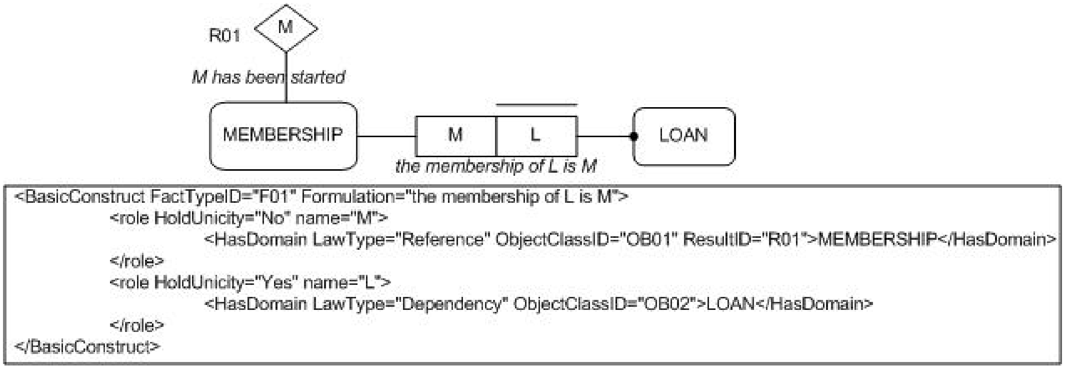
7.1.4 State Model

As explained in Chapter 6.4, the XML documents for SM contain a set of basic constructs in the instance SM. Taking the library case for instance, we divide the whole OFD into several basic constructs for comparison with the XML documents.

The XML document of the SM is attached in Appendix D.4. The content of the first <BasicConstruct>element in this XML file is mapped to the original OFD in Figure 7.4. The fact type states that *the membership of L is M*. The domain of role M of this fact type is object class MEMBERSHIP. The domain of role L of this fact type is object class LOAN. There is a unicity law and a dependency law for role L, which indicates that every loan L must appear once and only once in a population of this fact type. The result type R01 *M has been started* is declared on MEMBERSHIP.

The correspondence of this fact type in the XML document declares the formulation of this fact type as an attribute of element <BasicConstruct>, and assigns a unique identification to it. The two roles in this fact type are described separately. For each role, the unicity law is set as an attribute "HoldUnicity" of the role; its domain information is contained in the element <Hasdomain>, which is nested in element <role>. Thus for role M, the attribute "HoldUnicity" is set with "No"; its domain, MEMBERSHIP, is set in its child element <Hasdomain>, specifying the reference law that holds between role M and object class MEMBERSHIP, as well as the result type R01 that is declared on this object class. Note that here we only mention the unique ID of the result type, it is because this result type ID is consistent with the result type ID in the CM, where the content of the specific result type is described. The information about role L is stored in the same way within the same element <BasicConstruct>. The attribute "ObjectClassID" is an additional bit of information in the XML document, which is just used for managing the structured model information but has nothing to do with the content of the OFD.

Another condition in OFD is shown in Figure 7.5 together with its corresponding XML codes. In this fact type, *copies of B are delivered in S*, the domain of role S is object class SHIPMENT, which holds a dependency law and has the result type R08 *S has been performed* declared on itself; the domain of role B is object class BOOK. For the books that are delivered in S, a new derived object class *LIBRARY BOOK* is defined as the set.
In the XML codes, the indication of this fact type is stored in the attribute "Formulation". The information concerning the role S, including the name of this role, the unicity law condition, the domain of this role, the type of law holds on its domain and the result type declared on its domain, is stored within the element <role>. The relevant information about role B in this fact type is specified in the other <role>element. Note that the new condition, that there is a derived object class defined on role B, is described in element <IsExtendedTo>. The name of the new object class *LIBRARY BOOK* is the content of this element within role B.



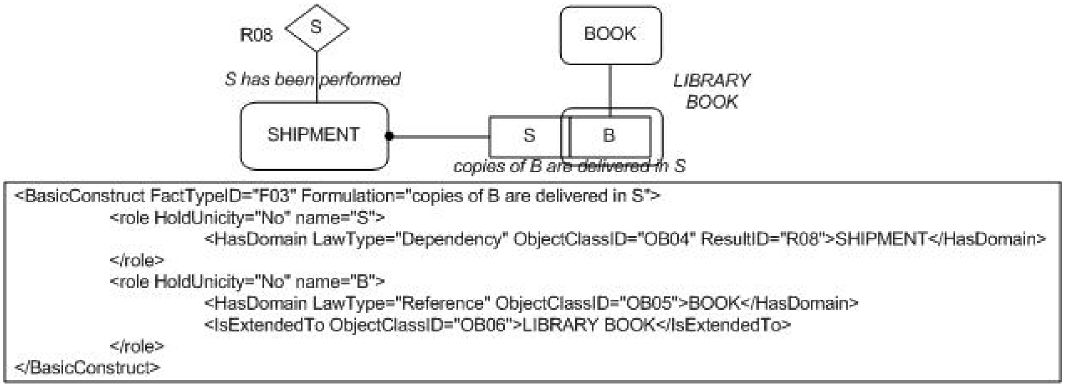
```
<BasicConstruct FactTypeID="F01" Formulation="the membership of L is M">
        <role HoldUnicity="No" name="M">
                <HasDomain LawType="Reference" ObjectClassID="OB01" ResultID="R01">MEMBERSHIP</HasDomain>
        </role>
        <role HoldUnicity="Yes" name="L">
                <HasDomain LawType="Dependency" ObjectClassID="OB02">LOAN</HasDomain>
        </role>
</BasicConstruct>
```

Figure 7.4: The comparison of the basic construct in SM - 1

7.1.5 Cross-Model Tables

The information items contained in three cross-model tables, namely the TRT, BCT and IUT, are in terms of the tabular data, which implies that they are easily accessible. The XML documents for these cross-model tables are in Appendix D.5, in which the information from the Library case is taken again.

Table 7.4 shows the mapping of the TRT, between the original tabular format and the XML based format. Each row in TRT is interpreted as an element <TransactionResult>in the XML document for TRT. The columns "transaction type" and "result type" are interpreted as elements <TransactionType>and <ResultType>,



```
<BasicConstruct FactTypeID="F03" Formulation="copies of B are delivered in S">
        <role HoldUnicity="No" name="S">
                <HasDomain LawType="Dependency" ObjectClassID="OB04" ResultID="R08">SHIPMENT</HasDomain>
        </role>
        <role HoldUnicity="No" name="B">
                <HasDomain LawType="Reference" ObjectClassID="OB05">BOOK</HasDomain>
                <IsExtendedTo ObjectClassID="OB06">LIBRARY BOOK</IsExtendedTo>
        </role>
</BasicConstruct>
```

Figure 7.5: The comparison of the basic construct in SM - 2

which are nested in the element <TransactionResult>. The information items contained in these two columns in TRT are stored in the corresponding elements in the XML document.
We also provide mappings for BCT and IUT in Tables 7.5 and 7.6 respectively.

Table 7.4: Mapping of TRT

transaction type	**result type**
T01 membership registration	*R01 membership M has been started*
<TransactionResult> <TransactionType>T01membership registration</TransactionType> <ResultType>R01membership M has been started</ResultType> </TransactionResult>	

Table 7.5: Mapping of BCT

object class, fact type, or result type	**P-bank**
MEMBERSHIP	PB01
P is the member in M	
membership M has been started	
<BankContent> <InformationBank>PB01</InformationBank> <FactType>MEMBERSHIP,P is the member in M,membership M has been started,</FactType> </BankContent>	

Table 7.6: Mapping of IUT

object class, fact type, or result type	**proces steps**
MEMBERSHIP	T01/rq T01/pm T04/rq
<InformationUse> <InformationObject>MEMBERSHIP</InformationObject> <TransactionStep>T01request,T01promise,T04request</TransactionStep> </InformationUse>	

7.2 Transformation Result Utilization

The transformation result, the instance XML documents of the DEMO aspect models, should have some added value to the current Business Component Modeling Process (BCMP). As expected in Chapter 4.1, it should help ease the complexity in generating the information for BCI. In this section, we will show a simple application built with the transformation result, which is in order to simplify the information generation procedure and create the required Create / Use Table for BCI.

Business Component Identification (BCI) requires diverse model information from DEMO aspect models. According to the contents of the information tables used by the BCI, the information objects are generated from the XML document of SM, and the process steps are from the XML document of PM. A Create / Use Table is needed to specify the classified relationship between the information objects and the process steps. [4]
Since there is no automatic way to produce the Create / Use Table solely based on the PM and SM, as already explained in chapter 4.1, the results of the proposed DEMO transformation should solve the indirection problem during information generation. Thus we build this simple application to show how the information generation for BCI benefits from the transformation results. The screenshot of the application is shown in Figure C.14.

In order to ease the problems of information resource diversity and information indirection that are mentioned in Chapter 4.1, the transformed information should be easily accessed and clearly displayed in the required forms for the BCI application. Having all the model information structured in the XML based format, we can find the required information by its tags in different XML documents.
In the column “IO-Fun”, all the information objects are imported from the XML document of the SM. They are selected by the tags “Formulation” and “ObjectClass”, which denote all the information items of object classes, fact types and result type in SM. The column “Create” lists all the transaction steps stored in the XML document of the PM, so do the transaction steps in the column “Use”.

Therefore the information recourse diversity problem that all the required information for the BCI be imported into this form automatically from the XML documents is solved. Though the creation of the Create / Use Table still relies on our own knowledge of specific relationships between the information objects and their associated transaction process steps, it saves us effort when read the models, and it also organizes the complete model information in a structured way for easy access. The DEMO models are transformed into a format that can be read directly by the third party application.

7.3 Conclusion

With the purpose of evaluating the DEMO transformation achieved in previous chapters, we start the evaluation from two perspectives, which are the transformation verification and the utilization of the transformation result.

The transformation result is verified by comparing the transformed models, the instance XML documents, with the original DEMO models. The comparison is done for each aspect model separately, and the information in the original DEMO diagrams is mapped with the information items in the XML documents. Thus the information is completely and correctly maintained during the transformation procedure.

The construction of the Create / Use Table is one example that shows how the transformation result can be used for other applications. The structured information in the XML documents can be accessed and imported into our application, which is for building the Create / Use table. Our application provides an easier way to generate the model information required by BCI from the diverse DEMO aspect models, and saves the effort of analyzing the model information.

Answers to the research question
In this chapter, we answered two research questions. The first one is how to verify the transformation result, which is answered in Chapter 7.1. The verification of the transformation result is done by comparing the produced instance XML documents with the original DEMO models. By mapping the information items in the XML documents and the original diagram information, we conclude that the model information is completely and correctly maintained during the transformation procedure.

The second research question, answered in Chapter 7.2 is how the transformed model could be used by other applications. In order to answer this question, we build an application to construct the Create / Use Table for BCI. The required model information for building the Create / Use Table is automatically imported from the produced XML documents. It shows that the transformed model, which is in XML based format, is easily accessed, and can be adopted by the BCI-3D tool directly.

Chapter 8

Conclusion and Discussion

This chapter aims to conclude the entire graduation project and discuss some relevant issues for future work. It includes, in Chapter 8.1, a recap of the research work that has been done in this project, which will emphasize the main focus of this research and the core steps during the project elaboration.
Following the project reflection, in Chapter 8.2, it will discuss the scientific foundation on which the research is based, including the metamodeling and MDA model transformation. In Chapter 8.2, we conclude that our adoption of metamodeling in our proposed DEMO model transformation is successful, and the transformation satisfies the requirements for generating the information from DEMO models for BCI.
An evaluation of the outcome of the project is concluded in Chapter 8.3. The items under evaluation include the to-be transformed model contents, DEMO metamodel interpretation, information completeness during transformation and utilization of the transformed model. Overall, it judges the quality of the research work.
In Chapter 8.4, some future work regarding the relevant issues in this project is discussed.

8.1 Project Reflection

This graduation project is conducted with the goal of transforming graphical DEMO models into an exchangeable format, in order to make them usable by other applications. This idea is inspired by the demands to bridge the gap between DEMO graphical diagrams and platform specific models. The expected result of this research is a successful storage of DEMO models in an exchangeable format, which can be used by third party applications, for instance BCI-3D tool.

In compliance with the mentioned research goal, the main tasks in this project involve a grasp of the DEMO metamodel, the definition of the transformation rules, the design of the XML schema for the DEMO metamodel, the instantiation of the designed XML schema, a comparison of the produced XML documents and the original DEMO models, and the construction of Create / Use Table.

Before elaborating the research work, background information is gained (Chapter 2) on the Business Component Modeling Process (BCMP), DEMO methodology, the concepts of metamodeling and model transformation, and the characteristics of the XML Schema (XSD).
Using business components has been proved valuable in the development of information systems. The identification of business components is a crucial factor in the BCMP. The *three dimensional method for business components identification* (BCI-3D) is one of the methods for identifying the business components. Information used for BCI should be generated from a well-defined model in the business domain.
DEMO methodology provides a holistic picture of enterprises. It includes four models which represent the essence of an enterprise, from the aspects of organizational construction, interrelation within and between the transactions, action rules and allowable states of information objects. DEMO methodology is a well defined business domain modeling.
Model transformation bridges the gap between a high-level Platform Independent Model (PIM) and a machine-readable Platform Specific Model (PSM). Metamodeling moves the focus of software development from coding to model building and proposes an approach to proceed with the model transformation.
XSD is the schema language chosen as the target language for the platform independent metamodel in our proposed DEMO model transformation. XSD shows a good ability in building block, data type, structural capacity and compatibility, as the result of our previous research in [27].

The DEMO metamodel (Chapter 3), as the main research material, specifies the essential structure of DEMO aspect models, including the Meta Construction Model (MCM), the Meta Process Model (MPM), the Meta Action Model (MAM), the Meta State Model (MSM) and the metamodel for the cross-model tables in Chapter 3. The MCM states the relationship between the transaction kind and the elementary actor role, the relationship between the elementary actor role and the information bank, and the relationship between the transaction kind and the information bank.
The MPM specifies the pure relationship between transaction steps, the relationship between transaction steps and transaction phases, as well as the interrelationships between transaction kind, transaction phases and transaction steps respectively. This metamodel does not take into account any constraints on the boundary of the responsibility area, since the corresponding instance information in PM could be generated from CM, which structure could be instantiated via the connection between the MCM and MPM.
The MAM provides the construct of action rules, and associates them with the transaction steps from MPM.
The MSM is the core of the whole DEMO metamodel. It is not only the metamodel

for SM, but also plays the role of being the meta schema for the entire DEMO metamodel. Besides defining the basic constructs used in SM, it also gives definitions to the symbols adopted in the metamodel. Thus SM defines the concepts used in the metamodel and itself, which implies that the whole DEMO metamodel is self-defined.
We also present the metamodel for the three cross-model tables such that the DEMO metamodel completely includes the structures of both diagrams and cross-model tables in DEMO.

The demands to transform DEMO graphical models are raised by the complex procedure of information generation from DEMO to BCI (Chapter 4.1). The complexities come from the diverse information sources and the indirect information. The expected transformation should be able to relieve us of the massive information generation procedure and save model information in an exchangeable format without any information loss.
Three transformation rules are defined regarding the demands for DEMO transformation and our knowledge of the DEMO metamodel (Chapter 4.2). They are:

> **Selection rules**: all the information of object class,and fact type in the DEMO metamodel, which is used directly for defining the structure of DEMO aspect models, should be selected.
>
> **Structuring rules**: the information item which has the central position is defined as the root element in the XML schema; the other information items are defined as complex types, the existential constraints are defined as attributes of either elements or complex types.
>
> **Mapping rules**: each information object in the original DEMO metamodel has a corresponding interpretation in the XML schema files in terms of an element or a complex type; every necessary constraint in the original DEMO metamodel has an equivalent part in the schema files in terms of an attribute of either an element or a complex type.

XML Schema has been chosen as the schema language in the design for theDEMO metamodel. The XSD files display the structure of the MCM, MPM, MAM,MSM and the three cross-model tables in the XML-based format (Chapter 5). The designed schema is mapped exactly and compactly with the metamodel structure. It comprehends all the information objects, the interrelationships and constraints in the metamodel, while avoiding information duplication and redundancy as much as possible.

The designed XML schema for the DEMO metamodel is instantiated into a set of XML documents with concrete model information from an exemplary case (Chapter 6). These XML documents are the transformation result, which means they are the expected platform specific models.

The complete DEMO metamodel transformation process is depicted in Figure 8.1. This is the instance model transformation applying the metamodeling approach, compared with the one in Figure 2.5. The transformation from DEMO models to XML documents is realized via the transformation from DEMO metamodel to XML schema. In order to guarantee transformation quality, the DEMO metamodel and XML schema are exactly mapped by the transformation specification.

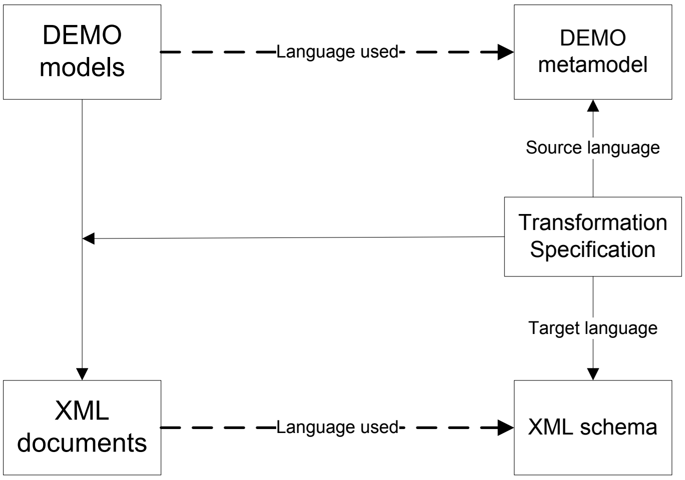


Figure 8.1: DEMO metamodel transformation

There might be more than one solution to the schema design for DEMO metamodel. However we verified the XML documents in Chapter 7.1, which are produced against our designed schema. The transformation rules guide and steer the transformation so that the right contents have been chosen and interpreted in the correct way. The mapping between the original DEMO diagrams and the XML documents verifies information completeness during the transformation process. With the transformation results, a Create / Use Table is built based on the information generated from the XML documents (Chapter 7.2). It provides an easier way to generate information for BCI, which is an example to show the utilization of the DEMO model transformation result.

8.2 Scientific foundation

Regarding the purpose of this project, to transform graphical DEMO models into an exchangeable format, two main theoretical foundations have been chosen and applied, which are the concept of metamodeling and the methodology of model transformation.

In recent years, metamodeling has been widely used in software engineering. Model transformation is a effective way to generating the source code from high-level models. Metamodeling provides an approach to proceed model transformation via metamodel at a higher level of information abstraction. In this specific project, we applied metamodeling to model transformation with our concrete research contents. As a result, the DEMO model transformation is accomplished under the direction of this chosen approach.

With the knowledge of model transformation and metamodeling (Figure 2.5), in Figure 8.1, high-level PIM is denoted as DEMO models, while PSM is denoted as XML documents. The metamodel transformation is carried out between the DEMO metamodel (Chapter 3) and the XML schema (Chapter 5). The rules we applied (Chapter 4.2) during the transformation align the contents of the DEMO metamodel and XML schema in consistent mappings, and adjust the transformed content in a reasonable structure. This leads the transformation to the result that the DEMO models are successfully transformed into an exchangeable format.

8.3 Project Evaluation

In order to evaluate the work that has been done in this project, we have four criteria to assess it, which are: the to-be transformed contents, interpretation of the DEMO metamodel, information completeness during transformation, and utilization of the transformed model. The first criterion is about what we need to transform in this project. The next two criteria focus on how we transformed the chosen contents from two aspects, one is the interpretation of the DEMO metamodel that was done during the transformation procedure, the other is the verification of the transformation result. The last criterion is about the added value of the transformation result.

To-be transformed contents
According to the main research question in Chapter 1.2, the source transformation contents are DEMO aspect models. The actual ones that have been transformed in this project are the diagrams in CM, PM, AM and SM, as well as three cross-model tables, namely the TRT, BCT and IUT. The essential structure of these aspect models is diagrammed in the DEMO metamodel (Figure 3.1 and Figure 3.18), and transformed into an exchangeable XML schema. In the instance XML documents, information that is contained in those diagrams is displayed in terms of XML based format.

DEMO metamodel interpretation
The designed schema files are the interpretation of the DEMO metamodel. During the procedure of constructing the XML schema, we apply the transformation rules (Chapter 4.2) to control the quality of the interpretation. The interpretation quality is guaranteed in aspects of the contents selection, the data structure and the information completeness. The object classes in the DEMO metamodel are precisely transformed into either elements or complex types in the schema file; the interrelationships between those object classes, as well as the constraints, are interpreted

as attributes of those elements or complex types. Therefore we can say that the interpretation of the DEMO metamodel is complete and correct.

Information completeness

Information loss must be avoided or reduced to the minimum in a successful model transformation. Our attention to information completeness is given throughout the entire transformation procedure. The DEMO metamodel comprehends exactly all the possible conditions in the instance aspect models. The interpretation of the DEMO metamodel is under the guidance of the selection rules and the structuring rules, to make sure the XML schema and the graphical metamodel are mapped with each other. The instance XML documents are built against the XML schema, which indicates that the structure and the allowed constraints in the instance documents are fully in compliance with the metamodel. Finally, the comparison of the transformed model (the instance XML documents for DEMO models) with the original model (the instance DEMO aspect models) semantically maps the information items. Thus the information completeness is guaranteed during the transformation.

Transformation utilization

Building the Create / Use Table is a requisite step for BCI. The application of building this table directly and automatically accesses the model information in the transformed DEMO models, the instance XML documents. The result of this project is a simplification of the procedure of generating the information from the DEMO models for BCI, and easing the existing problems, such as information resource diversity. Now the model information is structured completely in XML based format and can be read and used easily by the third party application.

8.4 To Be Discussed

Question 1: Is there a more specific way of describing the action rules in an exchangeable format?

Discussion: In DEMO, the action rules are specified in a pseudo-algorithmic language[1] in AM, such that the structure of those rules is quite expressive and difficult to merge into a structural format. What we have done in this project is to name an object class, ACTION RULE, that contains all the action rules for performing transaction steps, without considering the concrete structure of rule content. In the produced XML document for AM, we did not provide specific tags to the information objects or constraints mentioned in certain action rule, so that the rule content may only be used as a whole by other applications.

Nowadays, there is a growing interest in specifying business rules in an extensible format. For instance, in the development of business components, the required interrelationship between the information objects and the process steps is all con-

[1] The Action Model is explained in chapter 18 of [9].

tained in the action rules. Achieving the ultimate automation of the information generation from DEMO to business components relies on a detailed transformation of the action rules.
Therefore, describing the rules in an exchangeable format, in which the content of the rules can be used directly by other applications, is of significant usefulness. It may be one subject for future work.

Question 2: How can we optimize the XML schema for the DEMO metamodel?
Discussion: In this project, we designed an XML schema for the DEMO metamodel in XSD. The instance XML document based on the designed schema is mapped with the original DEMO models. The verification that has been done is mainly about information completeness during the transformation. However, there is lack of assessment about the efficiency of the schema design.
An item can be declared as an element or defined as a type, which are all valid building blocks in XSD. In addition, either elements or types can be declared global or local. "Global" means it is an immediate child of <schema>, "local" means it is nested within other elements or types [2]. There is no fixed principle about when it should be an element or a type, and when it should be declared global or local. It depends on the purpose and usage of the expected schema.
In order to achieve the minimal information redundant and maximal compactness in the XML interpretation of the DEMO metamodel, one of the relevant works could be constructing an assessment framework for the XML schema design.

Question 3: Regarding different specific purpose with DEMO models, whether our proposed DEMO metamodel is suitable for all of them?
Discussion: The application we introduced in this project that uses the transformation result is for BCI, which requires the information objects and the transaction process steps, as well as the numerous interrelationships among them. However, there may be other applications using the DEMO model information. Every third party application varies from every other, and different kinds of functions or information requirements to DEMO models may arise.
For instance, the application Xemod [2] supports the DEMO methodology and sets up DEMO models. Diagrams produced by Xemod can be mutually related. Regarding the specific requirements from Xemod, some alternative information objects or more detailed interrelationships are demanded, for instance composite actor roles, besides the ones already mentioned in our proposed DEMO metamodel.

The DEMO metamodel presented in this project illustrates the essential concepts that are used for constructing DEMO aspect models. However, this metamodel is concise and generic. More detailed specification about the DEMO metamodel could be possible.

[2] Xemod, stands for Xprise Enterprise Engineering Modeler, is a tool for modelling organizations and business processes, provided by Xprise. http://www.xprise.com/

Question 4: Is there an easy way to reproduce the DEMO diagram from the XML documents of the aspect models?
Discussion: In this report, we present a semantic mapping of the produced instance XML documents and the original DEMO diagrams in order to verify the transformation results. The semantic mapping successfully mapped the information contained in the instance XML documents and the original DEMO diagrams. However it would be a more direct verification if we could reproduce the DEMO diagrams from the instance XML documents.
In this project, we tried to reproduce the diagrams in Graphviz[3]. Two exemplary reproduced diagrams are shown in Figure 8.2 and Figure 8.3. Graphviz uses DOT language to generate the graph with a minimum of effort. Therefore, in order to reproduce the DEMO diagrams in Graphviz, we have to transform the original XML documents into DOT language.
We did not present the reproduction of the DEMO diagrams in the thesis as the verification of the transaction results, because the procedure of transforming the diagrams from the produced XML documents to the DOT files cannot be automatically done at the moment, but requires several steps to manually transform the information items from the XML documents, which arise another requirement for verifying the manual translation.
Even though we did not have enough time and knowledge to implement the reproduction of the DEMO diagrams, we can provide one possible approach of generating the graph from XML documents for further relevant research.
Firstly, transform the original XML documents into DotML documents by using XSLT. DotML is a XML based syntax for the input language of the 'Dot' graph drawing tool from Graphviz. It can be transformed to the native syntax of the “Dot” tool using XSLT[4] [1]. XSLT [26] is used to transform XML documents into other format.
Secondly, use the DotML package to generate the graph from the XML documents, which are in DotML syntax from previous step. [1]

An automatic reproduction of the DEMO diagrams from the XML documents would be very helpful for verifying the XML based format specification of the DEMO models.

[3] Graphviz, short for Graph Visualization Software, is a package of open source tools for drawing graphs specified in DOT language. http://www.graphviz.org/

[4] XSLT stands for Extensible Stylesheet Language Transformations.

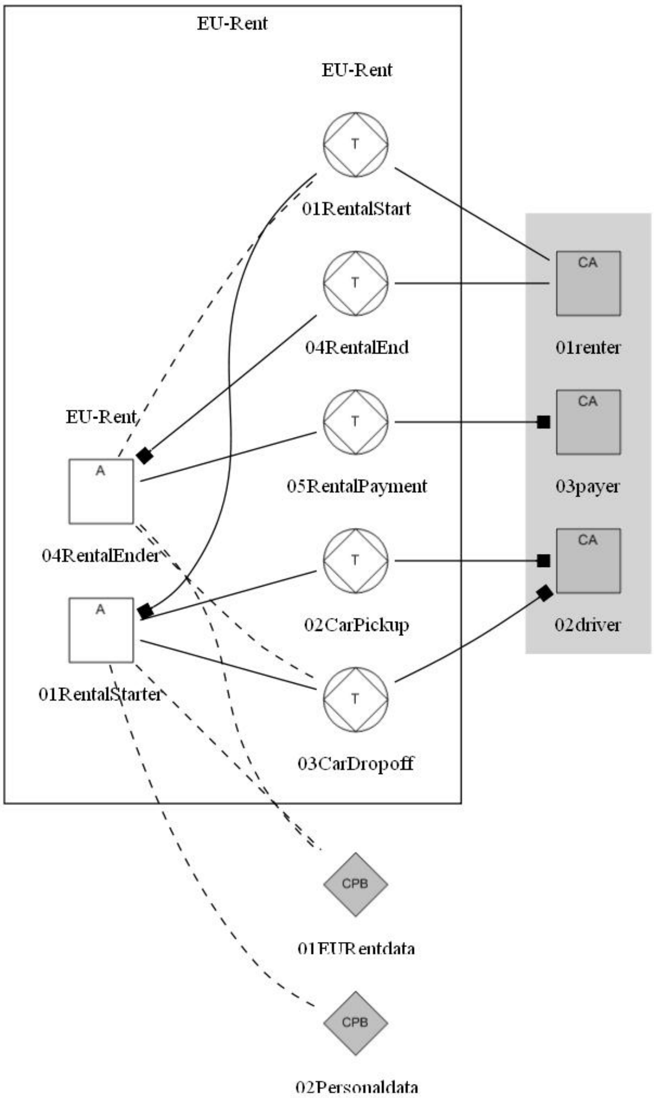


Figure 8.2: Exemplary reproduced CM diagram

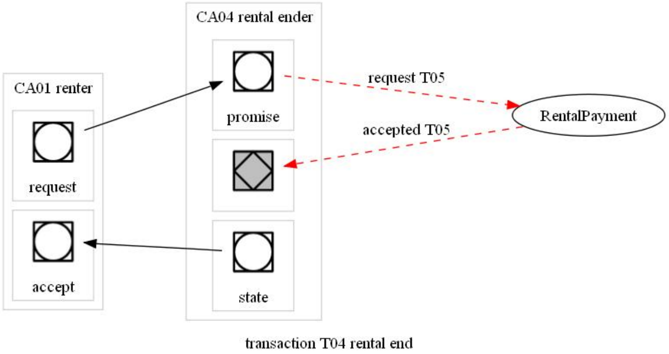


Figure 8.3: Exemplary reproduced PM diagram

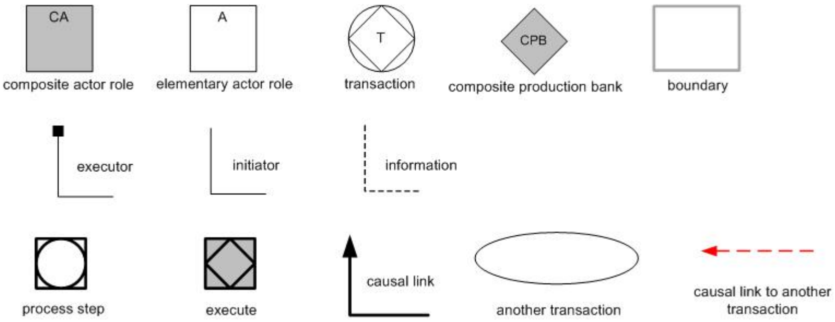


Figure 8.4: Legend of the reproduced diagrams

Abbreviations

AM	-	Action Model
BCI	-	Business Component Identification
BCI-3D	-	Three dimensional method for business components identification
BCM	-	Business Component Modeling
BCMP	-	Business Component Modeling Process
BCT	-	Bank Contents Table
CM	-	Construction Model
DEMO	-	Design and Engineering Methodology for Organizations
IUT	-	Information Use Table
MAM	-	Meta Action Model
MCM	-	Meta Construction Model
MDA	-	Model Driven Architecture
MPM	-	Meta Process Model
MSM	-	Meta State Model
OCD	-	Organization Construction Diagram
OFD	-	Object Fact Diagram
OMG	-	Object Management Group
ORM	-	Object-Role Modeling
OPL	-	Object Property List
PM	-	Process Model
PSD	-	Process Structure Diagram
SM	-	State Model
TRT	-	Transaction Result Table
XML	-	eXtensible Markup Language
XSD	-	XML Schema language (W3C)
XSLT	-	Extensible Stylesheet Language Transformation
W3C	-	World Wide Web Consortium
WOSL	-	World Ontology Specification Language

Bibliography

[1] The dot markup language. http://www.martin-loetzsch.de/DOTML/.

[2] Xml schemas: Best practices. http://www.xfront.com/BestPracticesHomepage.html.

[3] Antonia Albani. Enterprise ontology and business component. Course material, 2006.

[4] Jan L. G. Dietz Antonia Albani. The benefit of enterprise ontology in identifying business components. IFIP World Computer Congress (WCC 2006), 2006.

[5] Johannes Maria Zaha Antonia Albani, Jan L. G. Dietz. Identifying business components on the basis of an enterprise ontology. Interoperability of Enterprise Software and Applications, July 2006.

[6] F. Barbier and C. Atkinson. *Business components*. 2003.

[7] G. Hamel C. K. Prahalad. *The Core Competence of the Corporation*, pages 79–91. Springer Berlin Heidelberg, 2006.

[8] Brian Henderson-Sellers. Cesar Gonzalez-Perez. *Metamodelling for software engineering*. John Wiley and Sons, Ltd, 2008.

[9] Prof. Dr. Ir. J.L.G Dietz. *Enterprise Ontology Theory and Methodology*. Springer, 2006.

[10] Object Management Group. Mda specification. http://www.omg.org/mda/specs.htm.

[11] Object-Management Group. Mda guide. http://www.omg.org/docs/omg/03-06-01.pdf., 2003.

[12] Sjir Nijssen Inge Lemmens, Maurice Nijssen. A niam2007 conceptual analysis of the iso and omg mof four layer metadata architectures. *On the Move to Meaningful Internet Systems 2007: OTM 2007 Workshops*, Volume 4805/2007:613–623, 2007.

[13] ISO/IEC. Softerware engineering - metamodel for development methodologies. International Organization for standardization / International Electrotechnical Commission, 2007.

[14] Klaas van den Berg Ivan Kurtev and Mehmet Aksit. Uml to xml-schema transformation: a case study in managing alternative model transformations in mda. 2003.

[15] Michael Jervis. Xml dtds vs xml schema. http://www.sitepoint.com/print/xml-dtds-xml-schema.

[16] kenneth L.Kraemer John G.Mooney, Vijay Gurbaxani. A process oriented framework for assessing the business value of information technology. The Sixteenth Annual Intemational Conference on Information Systems, December 1995.

[17] Anneke G. Kleppe, Jos Warmer, and Wim Bast. *MDA Explained: The Model Driven Architecture: Practice and Promise*. Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA, 2003.

[18] Microsoft Developer Network. Xml standards reference. http://msdn.microsoft.com/en-us/library/ms256177.aspx.

[19] ORM. The orm foundation. http://www.ormfoundation.org/.

[20] Bran Selic. The pragmatics of model-driven development. *IEEE Softw.*, 20(5):19–25, 2003.

[21] DAVID O. STEPHENS. The globalization of information technology in multinational corporations. Information Management Journal, 1999.

[22] Zoran Stojanovic and Ajantha Dahanayake. *Service-oriented Software System Engineering Challenges And Practices*. IGI Publishing, Hershey, PA, USA, 2005.

[23] Robert J. Laubacher Thomas W. Malone. The dawn of the e-lance economy. *Harvard Business Review Article*, pages 151–152, Sep 1, 1998.

[24] W3C. W3c xml schema definition language 1.1 part 1: Structures. http://www.w3.org/TR/xmlschema11-1/.

[25] W3C. Xml schema data type. http://www.w3.org/TR/xmlschema-0/#simpleTypesTable.

[26] W3C. Xsl transformations (xslt) version 1.0. http://www.w3.org/TR/xslt.

[27] Yan Wang. The exploration of a proper demo model storage format for business component identification. Research Assignment, August 2008.

Appendices

Appendix A

XML Schema Quick Reference [18]

A.1 XML Schema Elements

XML Schema elements are grouped by their function: top level elements, particles, multiple XML documents and namespaces, identity constraints, attributes, named attributes, complex type definitions, and simple type definitions.

Top Level Elements
The elements in table A.1 appear at the top level of a schema document.

Particles
The elements in table A.2 can have minOccurs and maxOccurs attributes. Such elements always appear as part of a complex type definition or as part of a named model group.

Multiple XML Documents and Namespaces
The elements in table A.3 bring in schema elements from other namespaces or redefine schema elements in the same namespace.

Identity Constraints
The elements in table A.4 are related to identity constraints.

Attributes
The elements in table A.5 that define attributes in schemas.

Named Schema Objects
The elements in table A.6 define named constructs in schemas. Named constructs are referred to with a QName by other schema elements.

Complex Type Definitions
The elements in table A.7 create complex type definitions.

Simple Type Definitions
The elements in table A.8 create simple type definitions.

A.2 Attribute

The attribute element defines an attribute. Table A.9 shows the content of this element.

A.3 XML Schema Data Types

The World Wide Web Consortium (W3C) XML Schema Part 2: DataTypes is the specification for defining data types used in XML Schemas. This specification defines built-in primitive data types, derived data types, and facets.

Figure A.1 shows the type hierarchy.

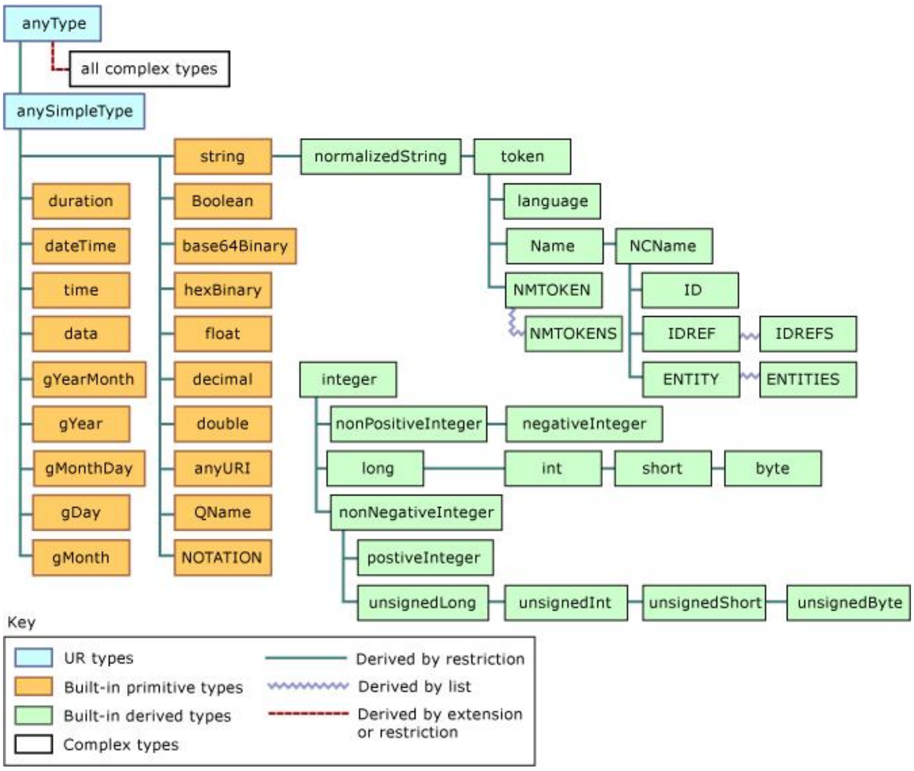


Figure A.1: XML Schema Data Type Hierarchy

Table A.1: Top Level Element

Element	**Description**
<xsd:annotation>	Defines an annotation.
<xsd:attribute>	Declares an attribute.
<xsd:attributeGroup>	Groups a set of attribute declarations so that they can be incorporated as a group for complex type definitions.
<xsd:complexType>	Defines a complex type, which determines the set of attributes and the content of an element.
<xsd:element>	Declares an element.
<xsd:group>	Groups a set of element declarations so that they can be incorporated as a group into complex type definitions.
<xsd:import>	Identifies a namespace whose schema components are referenced by the containing schema.
<xsd:include>	Includes the specified schema document in the target namespace of the containing schema.
<xsd:notation>	Contains the definition of a notation to describe the format of non-XML data within an XML document. An XML Schema notation declaration is a reconstruction of XML 1.0 NOTATION declarations.
<xsd:redefine>	Allows simple and complex types, groups, and attribute groups that are obtained from external schema files to be redefined in the current schema.
<xsd:simpleType>	Defines a simple type, which determines the constraints on and information about the values of attributes or elements with text-only content.

Table A.2: Particles

Element	**Description**
<xsd:all>	Allows the elements in the group to appear (or not appear) in any order in the containing element.
<xsd:any>	Enables any element from the specified namespace(s) to appear in the containing sequence or choice element.
<xsd:choice>	Allows one and only one of the elements contained in the selected group to be present within the containing element.
<xsd:element>	Declares an element.
<xsd:group>	Groups a set of element declarations so that they can be incorporated as a group into complex type definitions.
<xsd:sequence>	Requires the elements in the group to appear in the specified sequence within the containing element.

Table A.3: Multiple XML Documents and Namespaces

Element	**Description**
<xsd:import>	Identifies a namespace whose schema components are referenced by the containing schema.
<xsd:include>	Includes the specified schema document in the target namespace of the containing schema.
<xsd:redefine>	Allows simple and complex types, groups, and attribute groups that are obtained from external schema files to be redefined in the current schema.

Table A.4: Identity Constraints

Element	**Description**
<xsd:field>	Specifies an XML Path Language (XPath) expression that specifies the value (or one of the values) used to define an identity constraint (unique, key, and keyref elements).
<xsd:key>	Specifies that an attribute or element value (or set of values) must be a key within the specified scope. The scope of a key is the containing element in an instance document. A key must be unique, non-nillable, and always present.
<xsd:keyref>	Specifies that an attribute or element value (or set of values) correspond to those of the specified key or unique element.
<xsd:selector>	Specifies an XPath expression that selects a set of elements for an identity constraint (unique, key, and keyref elements).
<xsd:unique>	Specifies that an attribute or element value (or a combination of attribute or element values) must be unique within the specified scope. The value must be unique or nil.

Table A.5: Attributes

Element	**Description**
<xsd:anyAttribute>	Enables any attribute from the specified namespace(s) to appear in the containing complexType element or in the containing attributeGroup element.
<xsd:attribute>	Declares an attribute.
<xsd:attributeGroup>	Groups a set of attribute declarations so that they can be incorporated as a group for complex type definitions.

Table A.6: Named Schema Objects

Element	Description
<xsd:attribute>	Declares an attribute.
<xsd:attributeGroup>	Groups a set of attribute declarations so that they can be incorporated as a group for complex type definitions.
<xsd:complexType>	Defines a complex type, which determines the set of attributes and the content of an element.
<xsd:element>	Declares an element.
<xsd:group>	Groups a set of element declarations so that they can be incorporated as a group into complex type definitions.
<xsd:key>	Specifies that an attribute or element value (or set of values) must be a key within the specified scope. The scope of a key is the containing element in an instance document. A key must be unique, non-nillable, and always present.
<xsd:keyref>	Specifies that an attribute or element value (or set of values) correspond to those of the specified key or unique element.
<xsd:notation>	Contains the definition of a notation to describe the format of non-XML data within an XML document. An XML Schema notation declaration is a reconstruction of XML 1.0 NOTATION declarations.
<xsd:simpleType>	Defines a simple type, which determines the constraints on and information about the values of attributes or elements with text-only content.
<xsd:unique>	Specifies that an attribute or element value (or a combination of attribute or element values) must be unique within the specified scope. The value must be unique or nil.

Table A.7: Complex Type Definitions

Element	**Description**
<xsd:all>	Allows the elements in the group to appear (or not appear) in any order in the containing element.
<xsd:annotation>	Defines an annotation.
<xsd:any>	Enables any element from the specified namespace(s) to appear in the containing sequence or choice element.
<xsd:anyAttribute>	Enables any attribute from the specified namespace(s) to appear in the containing complexType element or in the containing attributeGroup element.
<xsd:appinfo>	Specifies information to be used by applications within an annotation element.
<xsd:attribute>	Declares an attribute.
<xsd:attributeGroup>	Groups a set of attribute declarations so that they can be incorporated as a group for complex type definitions.
<xsd:choice>	Allows one and only one of the elements contained in the selected group to be present within the containing element.
<xsd:complexContent>	Contains extensions or restrictions on a complex type that contains mixed content or elements only.
<xsd:documentation>	Specifies information to be read or used by users within an annotation element.
<xsd:element>	Declares an element.
<xsd:extension> (simpleContent)	Contains extensions on simpleContent. This extends a simple type or a complex type that has simple content by adding specified attribute(s), attribute groups(s) or anyAttribute.
<xsd:extension> (complexContent)	Contains extensions on complexContent.
<xsd:group>	Groups a set of element declarations so that they can be incorporated as a group into complex type definitions.
<xsd:restriction> (simpleContent)	Defines constraints on a simpleContent definition.
<xsd:restriction> (complexContent)	Defines constraints on a complexContent definition.
<xsd:sequence>	Requires the elements in the group to appear in the specified sequence within the containing element.
<xsd:simpleContent>	Contains extensions or restrictions on a complexType element with character data or a simpleType element as content and contains no elements.

Table A.8: Simple Type Definitions

Element	**Description**
<xsd:annotation>	Defines an annotation.
<xsd:appinfo>	Specifies information to be used by applications within an annotation element.
<xsd:documentation>	Specifies information to be read or used by users within an annotation element.
<xsd:element>	Declares an element.
<xsd:list>	Defines a collection of a single simpleType definition.
<xsd:restriction>	Defines constraints on a simpleType definition
<xsd:union>	Defines a collection of multiple simpleType definitions.

Table A.9: Attribute

Atribute	**Description**
default	Optional. Specifies a default value for the attribute. Default and fixed attributes cannot both be present
fixed	Optional. Specifies a fixed value for the attribute. Default and fixed attributes cannot both be present
form	Optional. Specifies the form for the attribute. The default value is the value of the attributeFormDefault attribute of the element containing the attribute. Can be set to one of the following:
id	
name	”qualified” - indicates that this attribute must be qualified with the namespace prefix and the no-colon-name (NCName) of the attribute
ref	unqualified - indicates that this attribute is not required to be qualified with the namespace prefix and is matched against the (NCName) of the attribute
type	Optional. Specifies a built-in data type or a simple type. The type attribute can only be present when the content does not contain a simpleType element
use	Optional. Specifies how the attribute is used. Can be one of the following values:

Appendix B

XML schema for DEMO metamodel

B.1 Meta Construction Model

Listing B.1: Schema code for Meta Construction Model

```
<xsd:schema xmlns:xsd=”http://www.w3.org/2001/XMLSchema”>
<xsd:element name=”Transactions”>
        <xsd:complexType>
                <xsd:sequence>
                        <xsd:element name=”Transaction” type=”
                             TRANSACTION_KIND” maxOccurs=”unbounded”/>
                </xsd:sequence>
        </xsd:complexType>
</xsd:element>
<xsd:complexType name=”TRANSACTION_KIND”>
        <xsd:sequence>
                <xsd:element name=”Tname” type=”xsd:string”/>
                <xsd:element name=”Initiator” type=”ELEMENTARY_ACTOR_ROLE”
                      minOccurs=”1” maxOccurs=”unbounded”/>
                <xsd:element name=”Executor” type=”ELEMENTARY_ACTOR_ROLE”
                     minOccurs=”1” maxOccurs=”1”/>
                <xsd:element name=”UseInformation” type=”INFORMATION_BANK”
                      minOccurs=”1” maxOccurs=”1”/>
                <xsd:element name=”Result” type=”DeclaredFactType”
                     minOccurs=”1” maxOccurs=”1”/>
        </xsd:sequence>
        <xsd:attribute name=”TransactionID” type=”xsd:string” use=”
            required”/>
</xsd:complexType>
<xsd:complexType name=”ELEMENTARY_ACTOR_ROLE”>
        <xsd:sequence>
                <xsd:element name=”name” type=”xsd:string”/>
                <xsd:element name=”UseInformation” type=”INFORMATION_BANK”
                      minOccurs=”0” maxOccurs=”unbounded”/>
        </xsd:sequence>
        <xsd:attribute name=”ActorID” type=”xsd:string” use=”required”/>
</xsd:complexType>
<xsd:complexType name=”INFORMATION_BANK”>
        <xsd:simpleContent>
                <xsd:extension base=”xsd:string”>
```

```
                        <xsd:attribute name=”BankType” use=”optional”>
                                <xsd:simpleType>
                                        <xsd:restriction base=”xsd:string
                                            ”>
                                                <xsd:enumeration value=”
                                                    Production”/>
                                                <xsd:enumeration value=”
                                                    Coordination”/>
                                        </xsd:restriction>
                                </xsd:simpleType>
                        </xsd:attribute>
                        <xsd:attribute name=”BankID” type=”xsd:string” use
                            =”required”/>
                </xsd:extension>
        </xsd:simpleContent>
</xsd:complexType>
<xsd:complexType name=”DeclaredFactType”>
        <xsd:simpleContent>
                <xsd:extension base=”xsd:string”>
                        <xsd:attribute name=”ResultID” type=”xsd:string”
                            use=”required”/>
                </xsd:extension>
        </xsd:simpleContent>
</xsd:complexType>
</xsd:schema>
```

B.2 Meta Process Model

Listing B.2: Schema code for Meta Process Model

```
<?xml version=”1.0” encoding=”UTF−8”?>
<xsd:element name=”TransactionPattern”>
        <xsd:complexType>
                <xsd:sequence>
                        <xsd:element name=”Transaction” minOccurs=”1”
                            maxOccurs=”unbounded”>
                                <xsd:complexType>
                                        <xsd:sequence>
                                                <xsd:element name=”
                                                    TransactionPhase” type
                                                    =”TRANSACTION_PHASE”
                                                    minOccurs=”0”
                                                    maxOccurs=”3”/>
                                        </xsd:sequence>
                                        <xsd:attribute name=”TransactionID
                                            ” type=”xsd:string” use=”
                                            required”/>
                                        <xsd:attribute name=”name” type=”
                                            xsd:string”/>
                                </xsd:complexType>
                        </xsd:element>
                </xsd:sequence>
        </xsd:complexType>
        <xsd:complexType name=”TRANSACTION_PHASE”>
        <xsd:sequence>
                <xsd:element name=”step” type=”TRANSACTION_STEP” minOccurs
                    =”0” maxOccurs=”unbounded”/>
        </xsd:sequence>
        <xsd:attribute name=”name” use=”required”>
```

```
                <xsd:simpleType>
                        <xsd:restriction base=”xsd:string”>
                                <xsd:enumeration value=”Order”/>
                                <xsd:enumeration value=”Execution”/>
                                <xsd:enumeration value=”Result”/>
                        </xsd:restriction>
                </xsd:simpleType>
        </xsd:attribute>
 </xsd:complexType>
 <xsd:complexType name=”TRANSACTION_STEP”>
        <xsd:sequence>
                <xsd:element name=”Name” type=”xsd:string” minOccurs=”1”
                     maxOccurs=”1”/>
                <xsd:element name=”InitiatedFrom” type=”TRANSACTION_STEP”
                     minOccurs=”0” maxOccurs=”unbounded”/>
                <xsd:element name=”InitiationTo” type=”TRANSACTION_STEP”
                     minOccurs=”0” maxOccurs=”unbounded”/>
                <xsd:element name=”IsWaitConditionOf” type=”
                     TRANSACTION_STEP” minOccurs=”0” maxOccurs=”unbounded
                     ”/>
                <xsd:element name=”WaitingFor” type=”TRANSACTION_STEP”
                     minOccurs=”0” maxOccurs=”unbounded”/>
        </xsd:sequence>
        <xsd:attribute name=”TransactionID” type=”xsd:string” use=”
             optional”/>
        </xsd:complexType>
</xsd:element>
```

B.3 Meta Action Model

Listing B.3: Schema code for Meta Action Model

```
<?xml version=”1.0” encoding=”UTF-8”?>
<xsd:schema xmlns:xsd=”http://www.w3.org/2001/XMLSchema”>
<xsd:element name=”ActionRules”>
        <xsd:complexType>
                <xsd:sequence>
                        <xsd:element name=”Rule” type=”ACTION_RULE”
                             minOccurs=”0” maxOccurs=”unbounded”/>
                </xsd:sequence>
        </xsd:complexType>
</xsd:element>
<xsd:complexType name=”ACTION_RULE”>
        <xsd:simpleContent>
                <xsd:extension base=”xsd:string”>
                        <xsd:attribute name=”TransactionID” type=”xsd:
                             string” use=”required”/>
                        <xsd:attribute name=”step” type=”xsd:string” use=”
                             required”/>
                </xsd:extension>
        </xsd:simpleContent>
</xsd:complexType>
</xsd:schema>
```

B.4 Meta State Model

Listing B.4: Schema code for Meta State Model

```
<?xml version=”1.0” encoding=”UTF−8”?>
<xsd:schema xmlns:xsd=”http://www.w3.org/2001/XMLSchema”>
<xsd:element name=”ObjectFact”>
    <xsd:complexType>
        <xsd:sequence>
            <xsd:element name=”BasicConstruct” type=”
                BasicConstruct” minOccurs=”0” maxOccurs=”
                unbounded”/>
        </xsd:sequence>
    </xsd:complexType>
</xsd:element>
<xsd:complexType name=”BasicConstruct”>
    <xsd:sequence>
        <xsd:element name=”role” type=”FACT_TYPE” minOccurs=”0”
            maxOccurs=”unbounded”/>
    </xsd:sequence>
    <xsd:attribute name=”Formulation” type=”xsd:string”/>
    <xsd:attribute name=”FactTypeID” type=”xsd:string” use=”required
        ”/>
</xsd:complexType>
<xsd:complexType name=”FACT_TYPE”>
    <xsd:sequence>
        <xsd:choice>
            <xsd:element name=”HasDomain” type=”OBJECT_CLASS
                ”/>
            <xsd:element name=”HasRange” type=”Scale”/>
        </xsd:choice>
        <xsd:element name=”IsExtendedTo” type=”OBJECT_CLASS”
            minOccurs=”0” maxOccurs=”unbounded”/>
    </xsd:sequence>
    <xsd:attribute name=”HoldUnicity” use=”optional” default=”Yes”>
        <xsd:simpleType>
            <xsd:restriction base=”xsd:string”>
                <xsd:enumeration value=”Yes”/>
                <xsd:enumeration value=”No”/>
            </xsd:restriction>
        </xsd:simpleType>
    </xsd:attribute>
    <xsd:attribute name=”name” type=”xsd:string” use=”required”/>
</xsd:complexType>
<xsd:complexType name=”OBJECT_CLASS”>
    <xsd:simpleContent>
        <xsd:extension base=”xsd:string”>
            <xsd:attribute name=”LawType” use=”optional”
                default=”Reference”>
                <xsd:simpleType>
                    <xsd:restriction base=”xsd:string
                        ”>
                        <xsd:enumeration value=”
                            Reference”/>
                        <xsd:enumeration value=”
                            Dependency”/>
                    </xsd:restriction>
                </xsd:simpleType>
            </xsd:attribute>
            <xsd:attribute name=”ObjectClassID” type=”xsd:
                string” use=”required”/>
            <xsd:attribute name=”ResultID” type=”xsd:string”
                use=”optional”/>
        </xsd:extension>
    </xsd:simpleContent>
```

```
</xsd:complexType>
<xsd:complexType name=”Scale”>
        <xsd:simpleContent>
                <xsd:extension base=”xsd:string”>
                        <xsd:attribute name=”ScaleType” use=”optional”
                             default=”A”>
                                <xsd:simpleType>
                                        <xsd:restriction base=”xsd:string
                                            ”>
                                                <xsd:enumeration value=”A
                                                     ”/>
                                                <xsd:enumeration value=”R
                                                     ”/>
                                                <xsd:enumeration value=”I
                                                     ”/>
                                                <xsd:enumeration value=”O
                                                     ”/>
                                                <xsd:enumeration value=”C
                                                     ”/>
                                        </xsd:restriction>
                                </xsd:simpleType>
                        </xsd:attribute>
                        <xsd:attribute name=”ScaleID” type=”xsd:string”
                             use=”required”/>
                </xsd:extension>
        </xsd:simpleContent>
</xsd:complexType>
</xsd:schema>
```

B.5 Metamodel of Cross-Model Tables

Listing B.5: XML schema for Transaction Result Table

```
<?xml version=”1.0” encoding=”UTF−8”?>
<xsd:schema xmlns:xsd=”http://www.w3.org/2001/XMLSchema”>
<xsd:element name=”TransactionResults”>
        <xsd:complexType>
                <xsd:sequence>
                        <xsd:element name=”TransactionResult” type=”TRT”
                             minOccurs=”0” maxOccurs=”unbounded”/>
                </xsd:sequence>
        </xsd:complexType>
</xsd:element>
<xsd:complexType name=”TRT”>
        <xsd:sequence>
                <xsd:element name=”TransactionType” type=”xsd:string”
                     minOccurs=”1” maxOccurs=”1”/>
                <xsd:element name=”ResultType” type=”xsd:string” minOccurs
                     =”0” maxOccurs=”1”/>
        </xsd:sequence>
</xsd:complexType>
</xsd:schema>
```

Listing B.6: XML schema for Bank Contents Table

```
<?xml version=”1.0” encoding=”UTF−8”?>
<xsd:schema xmlns:xsd=”http://www.w3.org/2001/XMLSchema”>
<xsd:element name=”BankContents”>
```

```
        <xsd:complexType>
                <xsd:sequence>
                        <xsd:element name="BankContent" type="BCT"
                             minOccurs="0" maxOccurs="unbounded"/>
                </xsd:sequence>
        </xsd:complexType>
</xsd:element>
<xsd:complexType name="BCT">
        <xsd:sequence>
                <xsd:element name="InformationBank" type="xsd:string"
                     minOccurs="1" maxOccurs="1"/>
                <xsd:element name="FactType" type="xsd:string" minOccurs
                     ="1" maxOccurs="unbounded"/>
        </xsd:sequence>
</xsd:complexType>
</xsd:schema>
```

Listing B.7: XML schema for Information Use Table

```
<?xml version="1.0" encoding="UTF-8"?>
<xsd:schema xmlns:xsd="http://www.w3.org/2001/XMLSchema">
<xsd:element name="InformationUsage">
        <xsd:complexType>
                <xsd:sequence>
                        <xsd:element name="InformationUse" type="IUT"
                             minOccurs="0" maxOccurs="unbounded"/>
                </xsd:sequence>
        </xsd:complexType>
</xsd:element>
<xsd:complexType name="IUT">
        <xsd:sequence>
                <xsd:element name="InformationObject" type="xsd:string"
                     minOccurs="0" maxOccurs="unbounded"/>
                <xsd:element name="TransactionStep" type="xsd:string"
                     minOccurs="0" maxOccurs="unbounded"/>
        </xsd:sequence>
</xsd:complexType>
</xsd:schema>
```

B.6 Graphical Representation of the schema

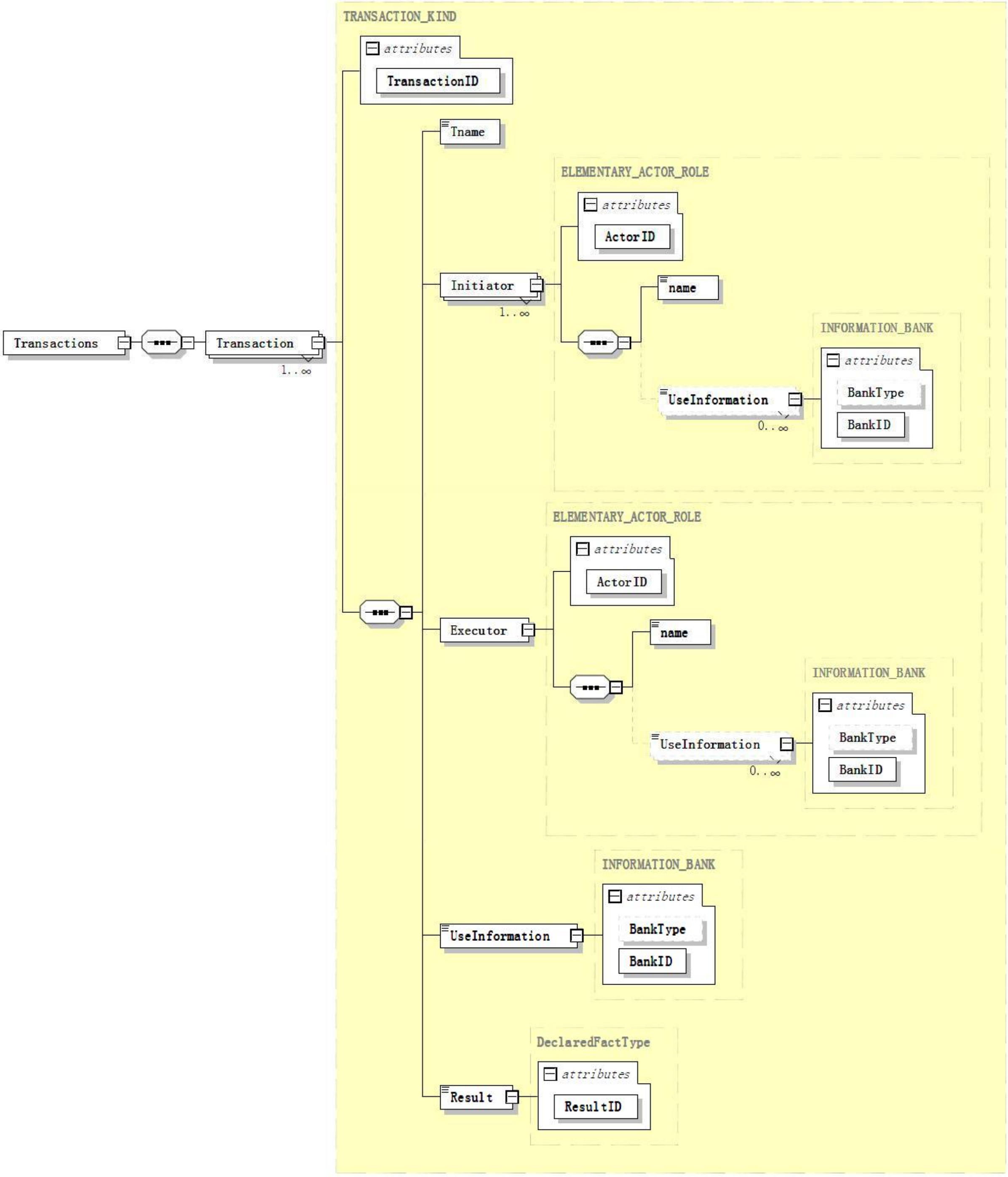


Figure B.1: The graphical representation of the schema for CM

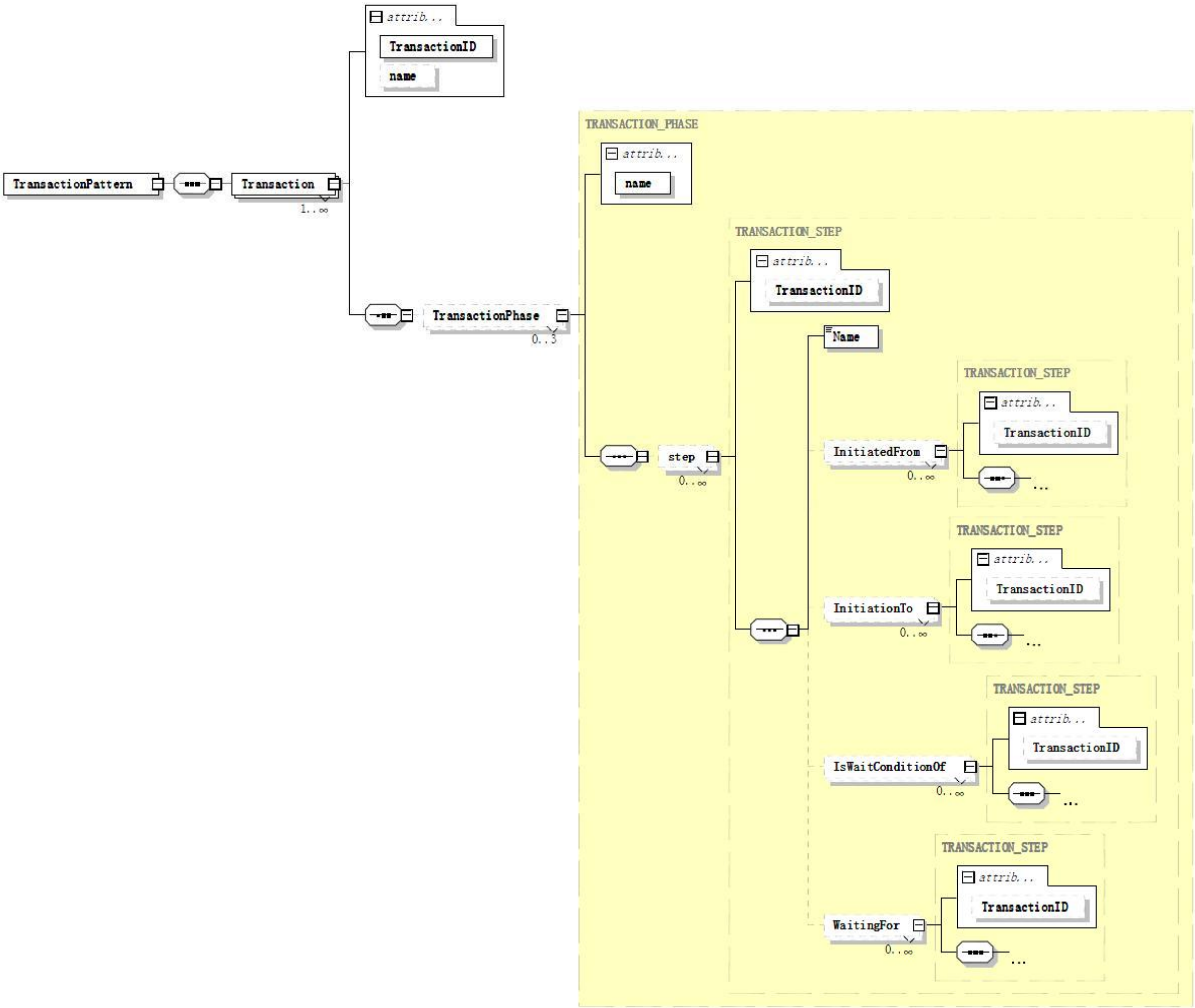


Figure B.2: The graphical representation of the schema for PM

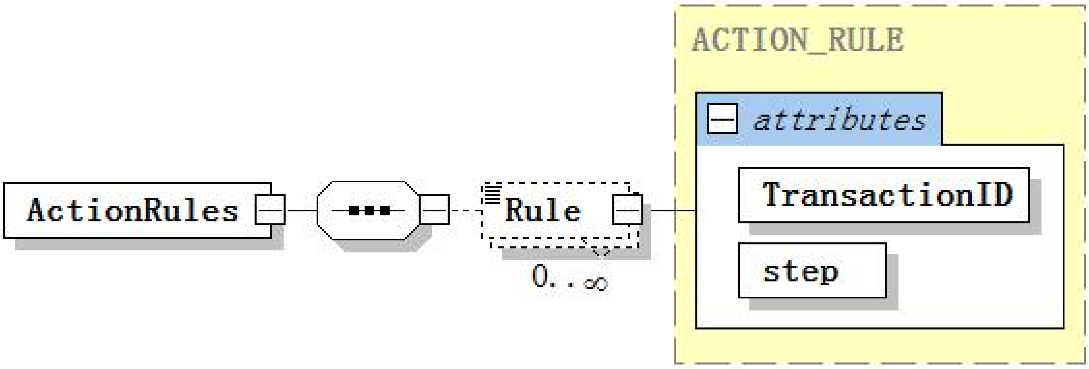


Figure B.3: The graphical representation of the schema for AM

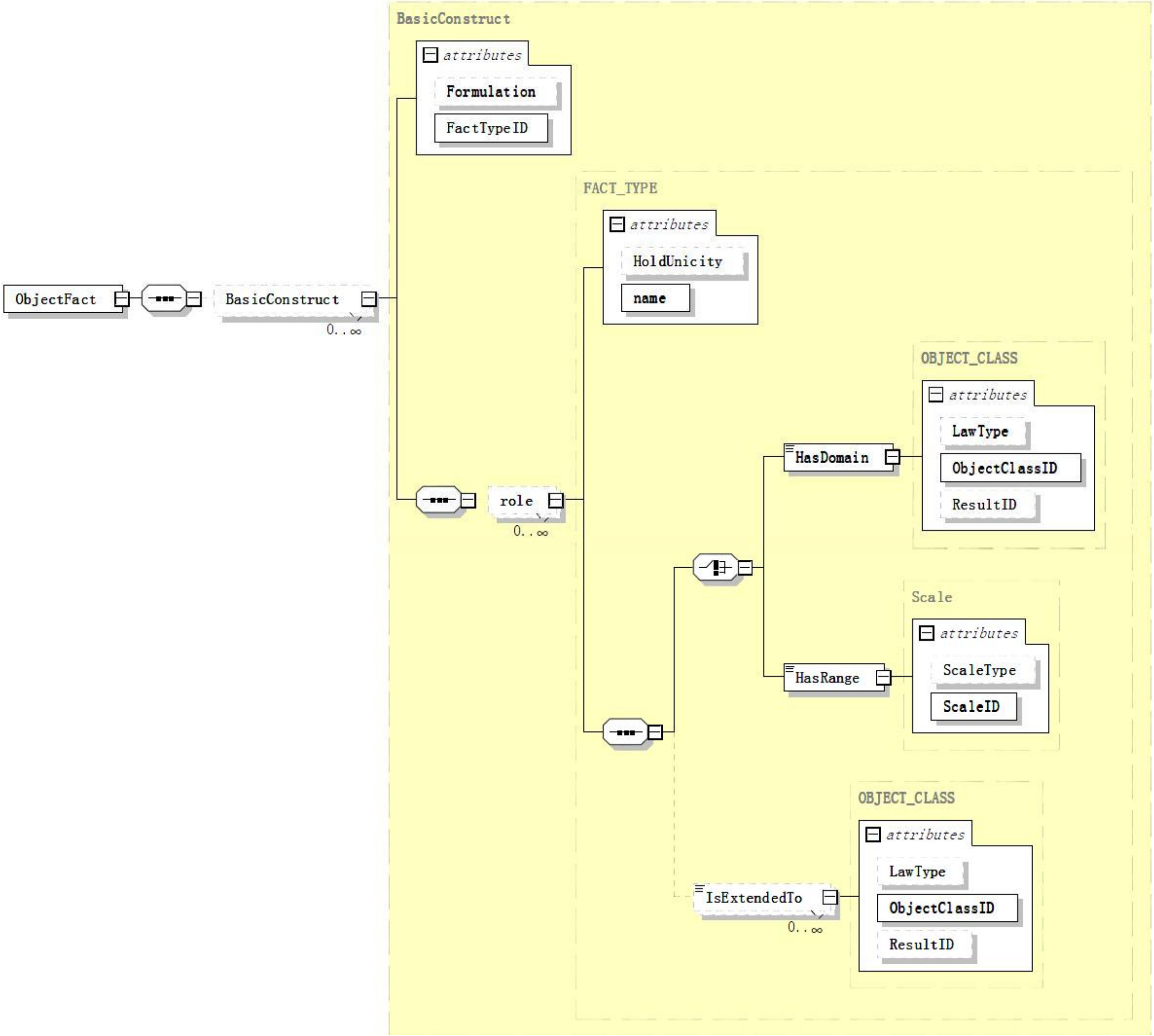


Figure B.4: The graphical representation of the schema for SM

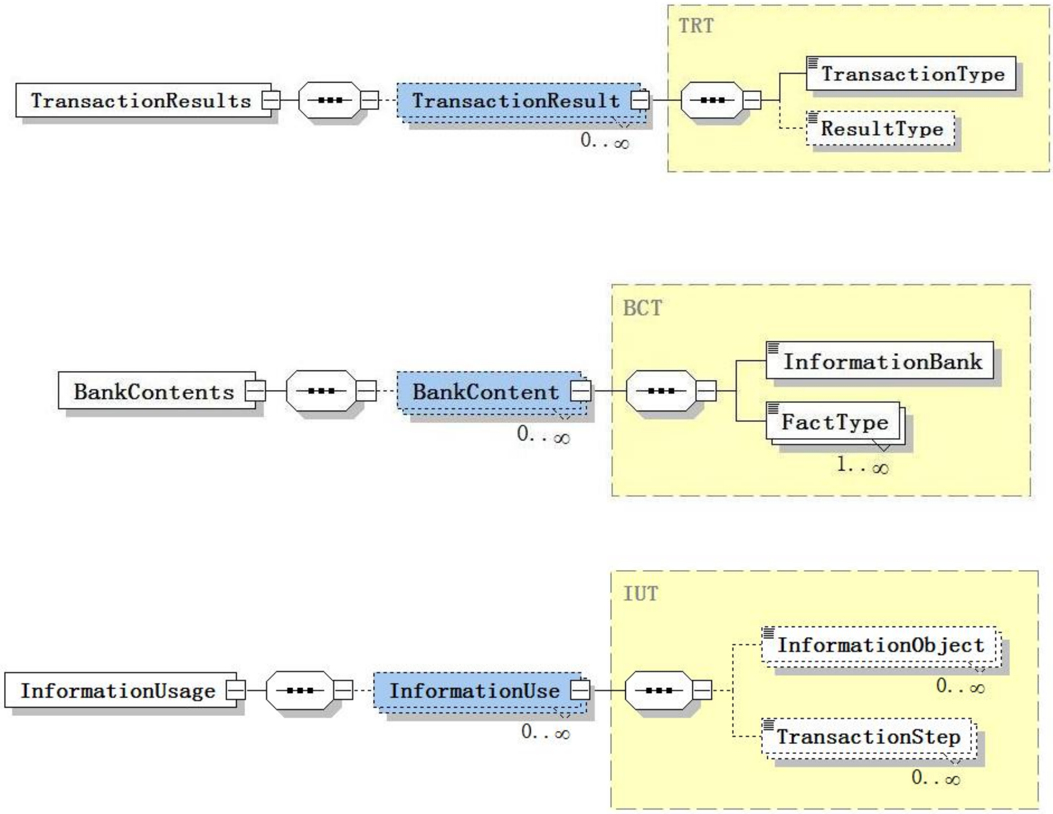
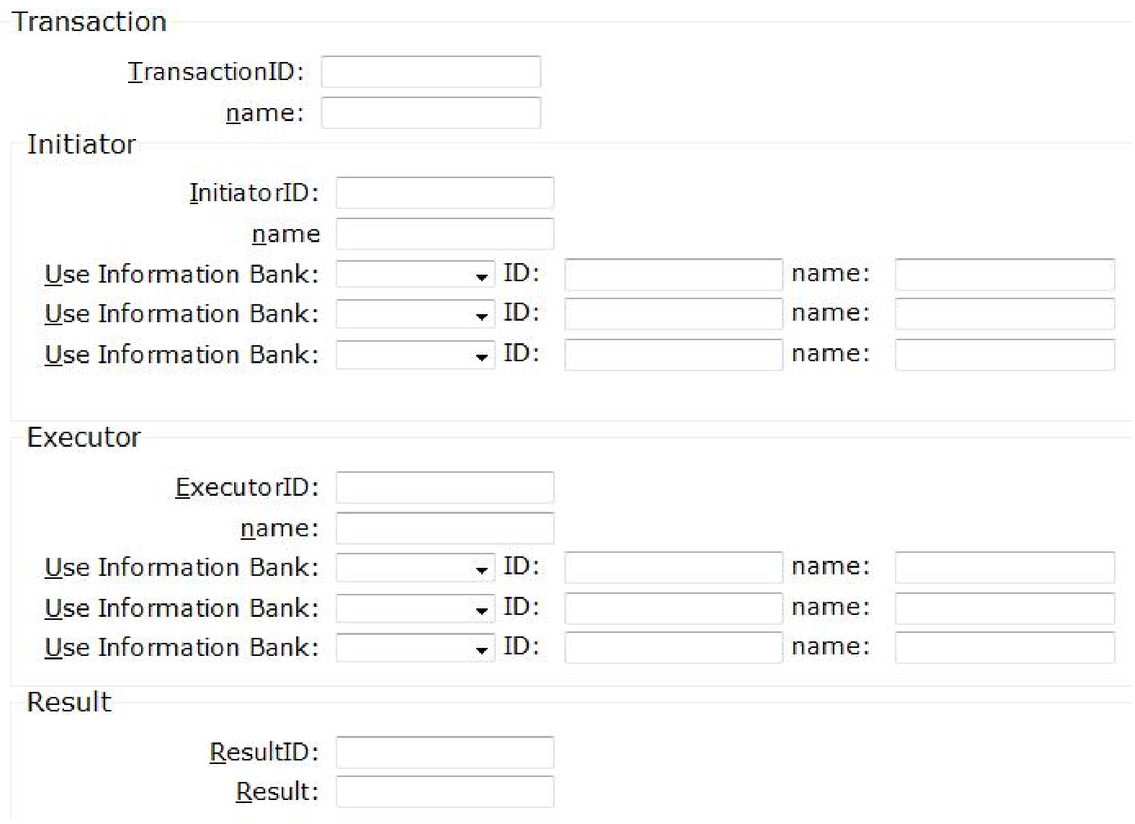


Figure B.5: The graphical representation of the schema for TRT, BCT and IUT (in order)

Appendix C

Application Screenshot



Transaction

TransactionID:

name:

Initiator

InitiatorID:

name

Use Information Bank: ID: name:

Use Information Bank: ID: name:

Use Information Bank: ID: name:

Executor

ExecutorID:

name:

Use Information Bank: ID: name:

Use Information Bank: ID: name:

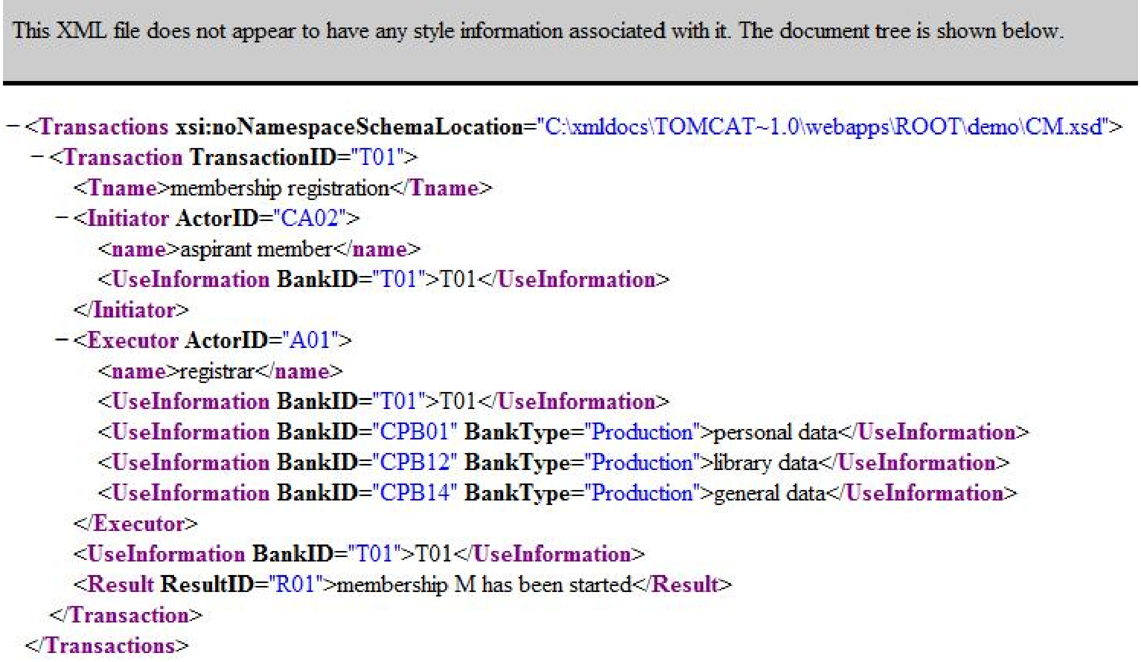
Use Information Bank: ID: name:

Result

ResultID:

Result:

Figure C.1: The information items required for CM



This XML file does not appear to have any style information associated with it. The document tree is shown below.

```
- <Transactions xsi:noNamespaceSchemaLocation="C:\xmldocs\TOMCAT~1.0\webapps\ROOT\demo\CM.xsd">
  - <Transaction TransactionID="T01">
      <Tname>membership registration</Tname>
    - <Initiator ActorID="CA02">
        <name>aspirant member</name>
        <UseInformation BankID="T01">T01</UseInformation>
      </Initiator>
    - <Executor ActorID="A01">
        <name>registrar</name>
        <UseInformation BankID="T01">T01</UseInformation>
        <UseInformation BankID="CPB01" BankType="Production">personal data</UseInformation>
        <UseInformation BankID="CPB12" BankType="Production">library data</UseInformation>
        <UseInformation BankID="CPB14" BankType="Production">general data</UseInformation>
      </Executor>
      <UseInformation BankID="T01">T01</UseInformation>
      <Result ResultID="R01">membership M has been started</Result>
    </Transaction>
  </Transactions>
```

Figure C.2: The exemplary XML document for CM

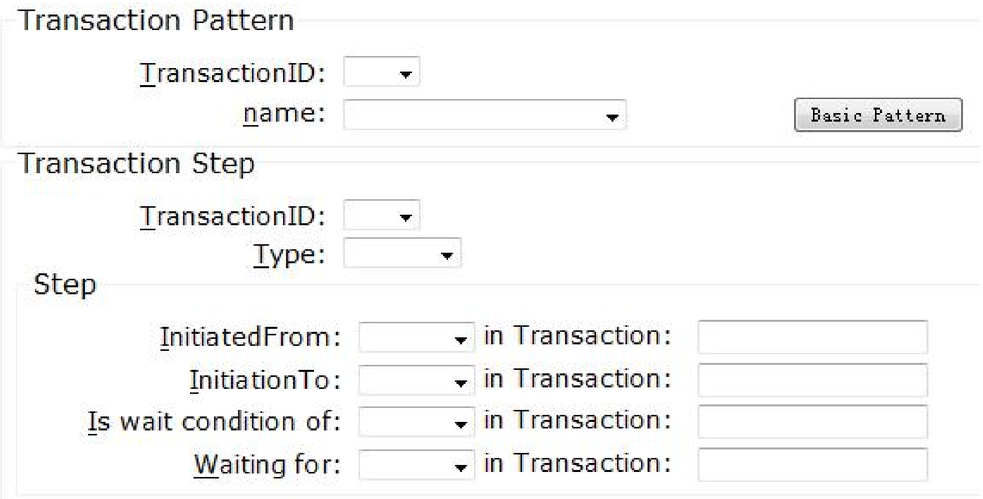
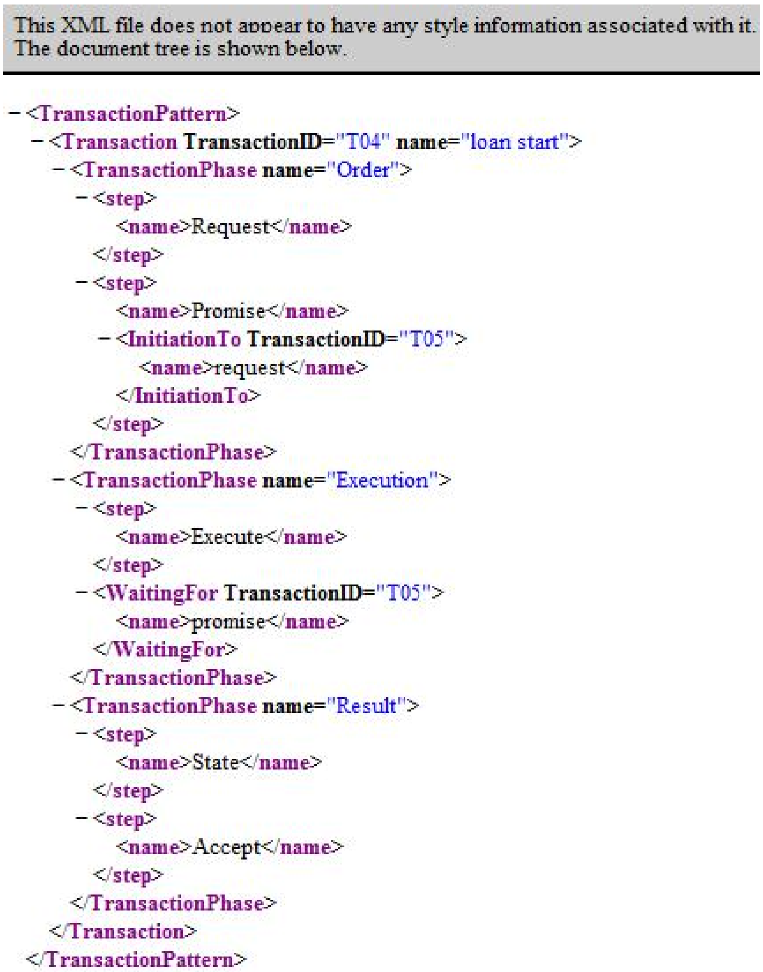


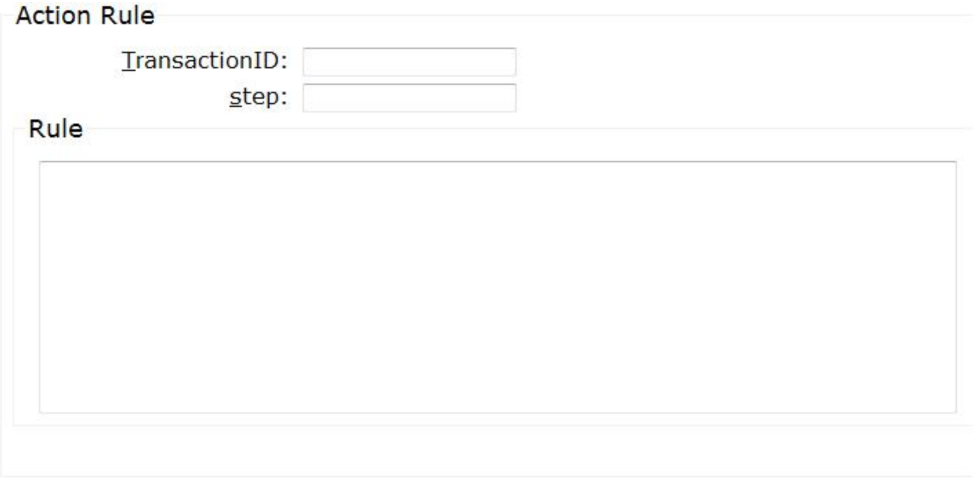
Figure C.3: The information items required for PM



This XML file does not appear to have any style information associated with it. The document tree is shown below.

```
- <TransactionPattern>
  - <Transaction TransactionID="T04" name="loan start">
    - <TransactionPhase name="Order">
      - <step>
          <name>Request</name>
        </step>
      - <step>
          <name>Promise</name>
        - <InitiationTo TransactionID="T05">
            <name>request</name>
          </InitiationTo>
        </step>
      </TransactionPhase>
    - <TransactionPhase name="Execution">
      - <step>
          <name>Execute</name>
        </step>
      - <WaitingFor TransactionID="T05">
          <name>promise</name>
        </WaitingFor>
      </TransactionPhase>
    - <TransactionPhase name="Result">
      - <step>
          <name>State</name>
        </step>
      - <step>
          <name>Accept</name>
        </step>
      </TransactionPhase>
    </Transaction>
  </TransactionPattern>
```

Figure C.4: The exemplary XML document for PM



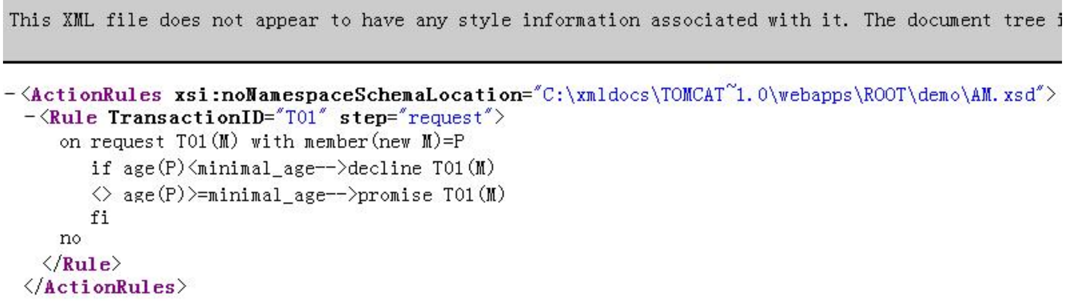
Action Rule

TransactionID:

step:

Rule

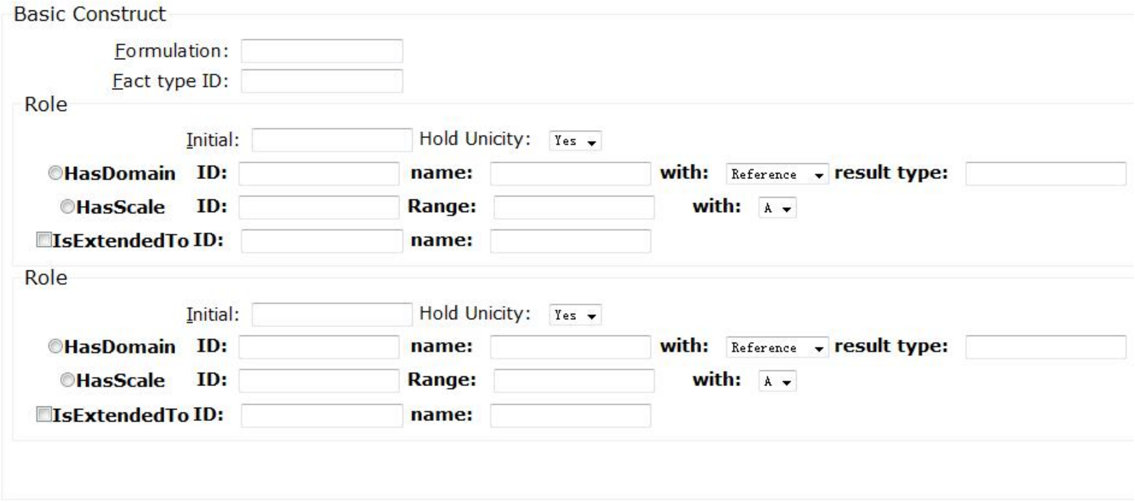
Figure C.5: The information items required for AM



```
This XML file does not appear to have any style information associated with it. The document tree i

- <ActionRules xsi:noNamespaceSchemaLocation="C:\xmldocs\TOMCAT~1.0\webapps\ROOT\demo\AM.xsd">
  - <Rule TransactionID="T01" step="request">
      on request T01(M) with member(new M)=P
         if age(P)<minimal_age-->decline T01(M)
         <> age(P)>=minimal_age-->promise T01(M)
         fi
      no
    </Rule>
  </ActionRules>
```

Figure C.6: The exemplary XML document for AM



Basic Construct

Formulation:

Fact type ID:

Role

Initial: Hold Unicity: Yes

○HasDomain ID: name: with: Reference result type:

○HasScale ID: Range: with: A

□IsExtendedTo ID: name:

Role

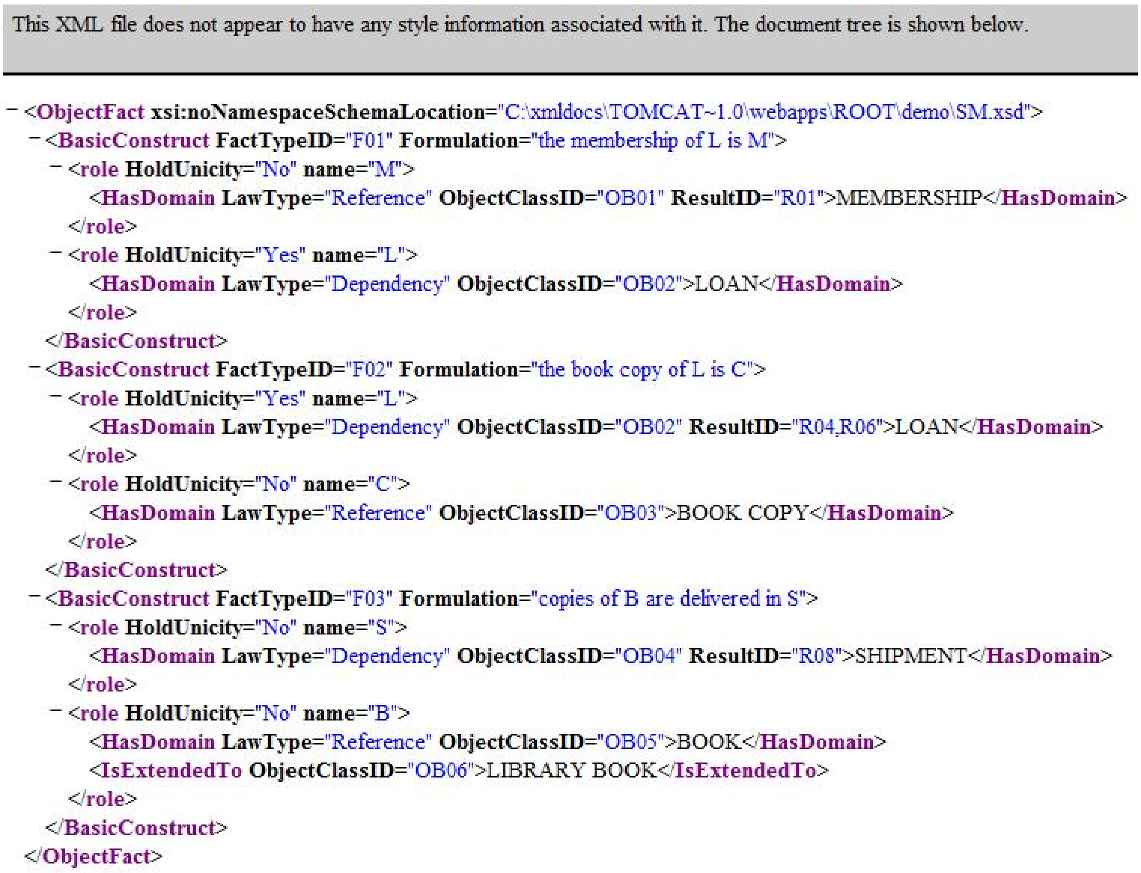
Initial: Hold Unicity: Yes

○HasDomain ID: name: with: Reference result type:

○HasScale ID: Range: with: A

□IsExtendedTo ID: name:

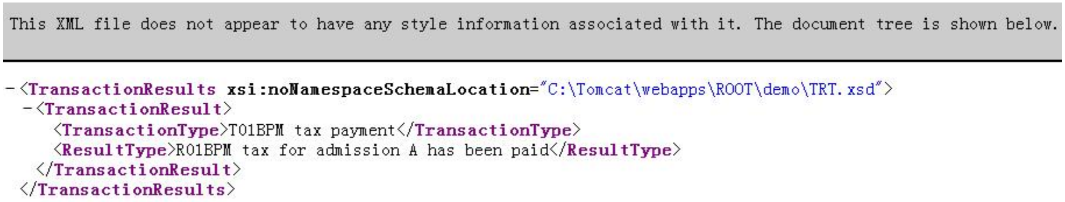
Figure C.7: The information items required for SM



This XML file does not appear to have any style information associated with it. The document tree is shown below.

```
- <ObjectFact xsi:noNamespaceSchemaLocation="C:\xmldocs\TOMCAT~1.0\webapps\ROOT\demo\SM.xsd">
  - <BasicConstruct FactTypeID="F01" Formulation="the membership of L is M">
    - <role HoldUnicity="No" name="M">
        <HasDomain LawType="Reference" ObjectClassID="OB01" ResultID="R01">MEMBERSHIP</HasDomain>
      </role>
    - <role HoldUnicity="Yes" name="L">
        <HasDomain LawType="Dependency" ObjectClassID="OB02">LOAN</HasDomain>
      </role>
    </BasicConstruct>
  - <BasicConstruct FactTypeID="F02" Formulation="the book copy of L is C">
    - <role HoldUnicity="Yes" name="L">
        <HasDomain LawType="Dependency" ObjectClassID="OB02" ResultID="R04,R06">LOAN</HasDomain>
      </role>
    - <role HoldUnicity="No" name="C">
        <HasDomain LawType="Reference" ObjectClassID="OB03">BOOK COPY</HasDomain>
      </role>
    </BasicConstruct>
  - <BasicConstruct FactTypeID="F03" Formulation="copies of B are delivered in S">
    - <role HoldUnicity="No" name="S">
        <HasDomain LawType="Dependency" ObjectClassID="OB04" ResultID="R08">SHIPMENT</HasDomain>
      </role>
    - <role HoldUnicity="No" name="B">
        <HasDomain LawType="Reference" ObjectClassID="OB05">BOOK</HasDomain>
        <IsExtendedTo ObjectClassID="OB06">LIBRARY BOOK</IsExtendedTo>
      </role>
    </BasicConstruct>
  </ObjectFact>
```

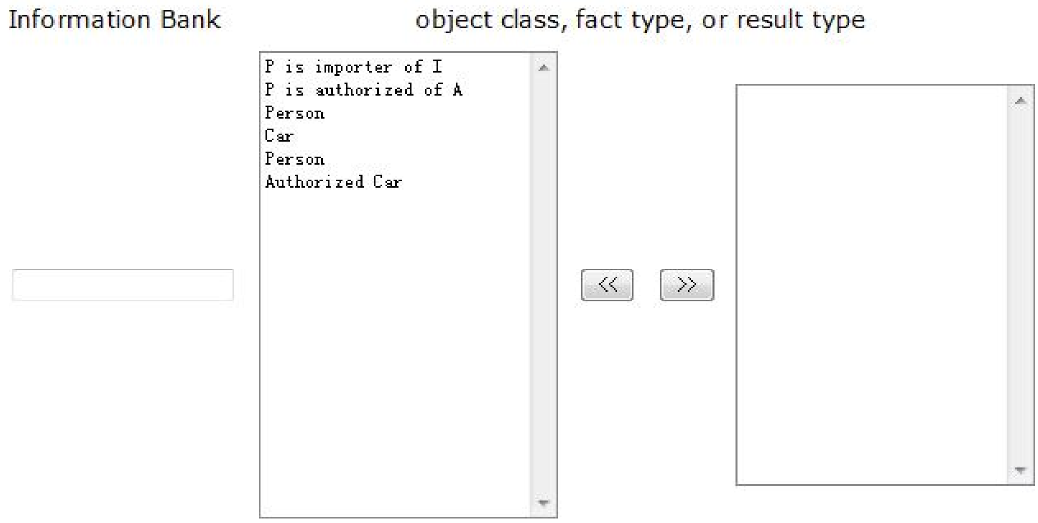
Figure C.8: The exemplary XML document for SM



This XML file does not appear to have any style information associated with it. The document tree is shown below.

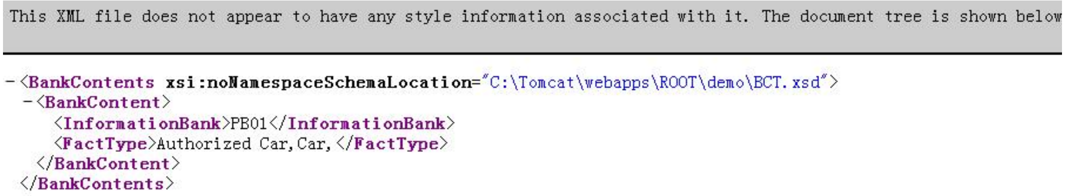
```
- <TransactionResults xsi:noNamespaceSchemaLocation="C:\Tomcat\webapps\ROOT\demo\TRT.xsd">
  - <TransactionResult>
      <TransactionType>T01BPM tax payment</TransactionType>
      <ResultType>R01BPM tax for admission A has been paid</ResultType>
    </TransactionResult>
  </TransactionResults>
```

Figure C.9: The exemplary XML document for TRT



P is importer of I
P is authorized of A
Person
Car
Person
Authorized Car

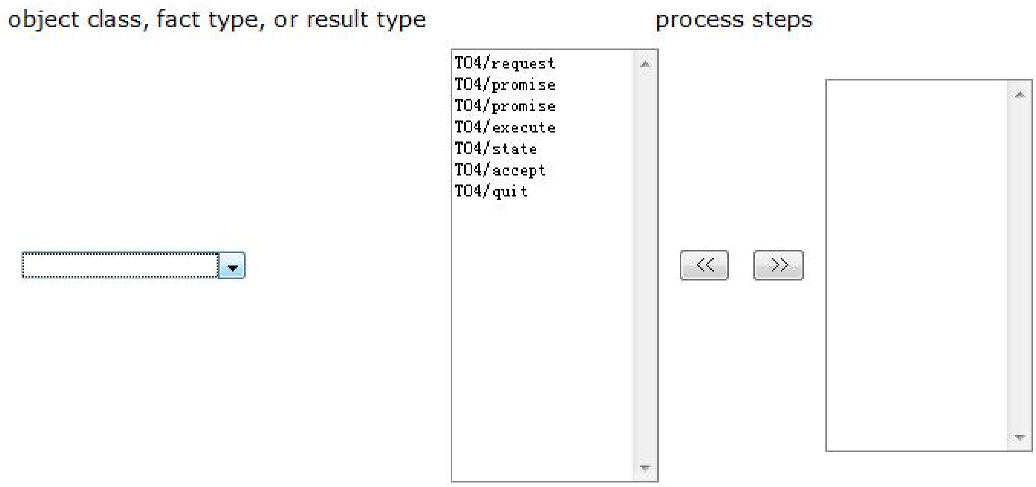
Figure C.10: The Construction of the BCT



This XML file does not appear to have any style information associated with it. The document tree is shown below

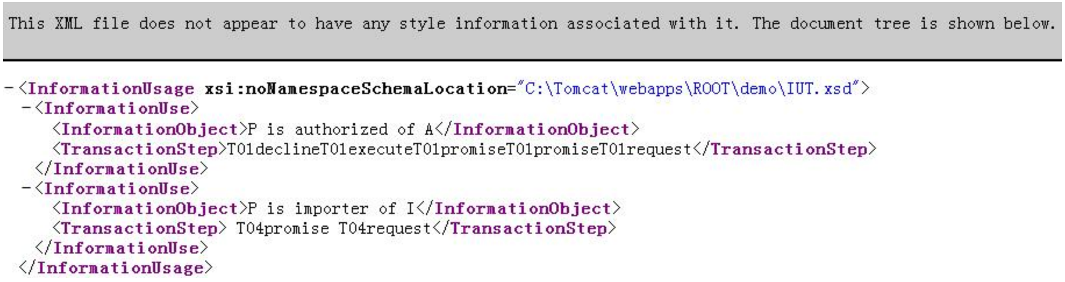
```
- <BankContents xsi:noNamespaceSchemaLocation="C:\Tomcat\webapps\ROOT\demo\BCT.xsd">
  - <BankContent>
      <InformationBank>PB01</InformationBank>
      <FactType>Authorized Car,Car,</FactType>
    </BankContent>
  </BankContents>
```

Figure C.11: The exemplary XML document for BCT



T04/request
T04/promise
T04/promise
T04/execute
T04/state
T04/accept
T04/quit

Figure C.12: The Construction of the IUT



```
This XML file does not appear to have any style information associated with it. The document tree is shown below.

-<InformationUsage xsi:noNamespaceSchemaLocation="C:\Tomcat\webapps\ROOT\demo\IUT.xsd">
  -<InformationUse>
      <InformationObject>P is authorized of A</InformationObject>
      <TransactionStep>T01declineT01executeT01promiseT01promiseT01request</TransactionStep>
    </InformationUse>
  -<InformationUse>
      <InformationObject>P is importer of I</InformationObject>
      <TransactionStep> T04promise T04request</TransactionStep>
    </InformationUse>
  </InformationUsage>
```

Figure C.13: The exemplary XML document for IUT

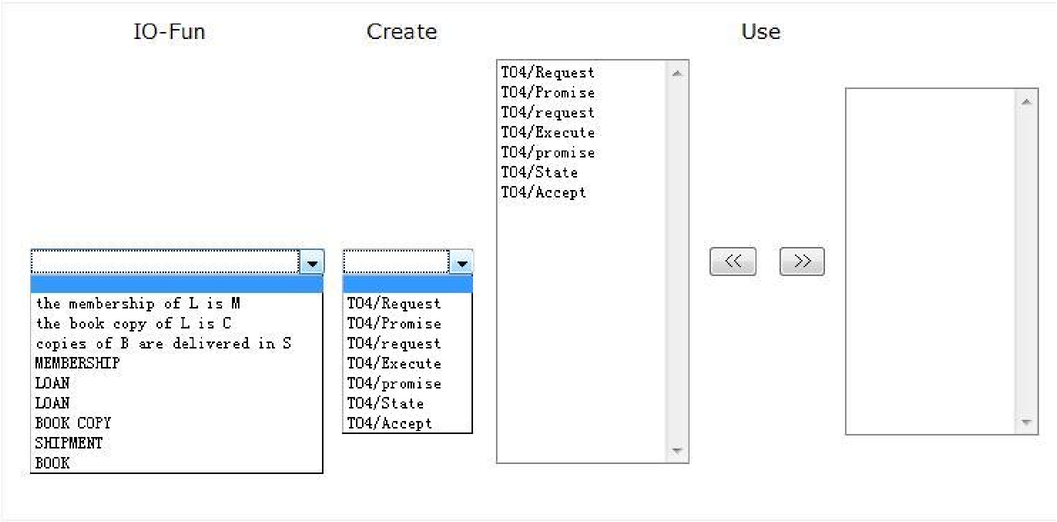


Figure C.14: The Construction of the Create / Use Table

Appendix D

Exemplary XML Documents of the Library Case

D.1 XML documents for CM

Listing D.1: The exemplary XML document of CM

```
<?xml version=”1.0” encoding=”UTF−8” standalone=”no”?>
<Transactions xmlns:xsi=”http://www.w3.org/2001/XMLSchema−instance” xsi:
    noNamespaceSchemaLocation=”C:\Tomcat\webapps\ROOT\demo\CM.xsd”>

<Transaction TransactionID=”T01”>
        <Tname>membership registration </Tname>
        <Initiator ActorID=”CA02”>
            <name>aspirant member</name>
            <UseInformation BankID=”PB01”>PB01</UseInformation>
        </Initiator>
        <Executor ActorID=”A01”>
            <name>registrar </name>
            <UseInformation BankID=”PB01”>PB01</UseInformation>
            <UseInformation BankID=”CPB14” BankType=”Production”>general
                data</UseInformation>
            <UseInformation BankID=”CPB11” BankType=”Production”>personal
                data</UseInformation>
        </Executor>
        <UseInformation BankID=”PB01”>PB01</UseInformation>
        <Result ResultID=”R01”>membership M has been started</Result>
    </Transaction>
<Transaction TransactionID=”T02”>
        <Tname>membership fee payment</Tname>
        <Initiator ActorID=”A01”>
            <name>register </name>
            <UseInformation BankID=”PB02”>PB02</UseInformation>
            <UseInformation BankID=”CPB14” BankType=”Production”>general
                data</UseInformation>
        </Initiator>
        <Initiator ActorID=”A10”>
            <name>annual fee controller </name>
            <UseInformation BankID=”PB02”>PB02</UseInformation>
            <UseInformation BankID=”CPB14” BankType=”Production”>general
                data</UseInformation>
        </Initiator>
        <Executor ActorID=”CA02”>
            <name>aspirant member</name>
            <UseInformation BankID=”PB02”>PB02</UseInformation>
        </Executor>
        <UseInformation BankID=”PB02”>PB02</UseInformation>
        <Result ResultID=”R02”>the fee for membership M in year Y has been
              paid</Result>
    </Transaction>
<Transaction TransactionID=”T03”>
        <Tname>reduced fee approval </Tname>
        <Initiator ActorID=”A01”>
            <name>register </name>
            <UseInformation BankID=”PB03”>PB03</UseInformation>
            <UseInformation BankID=”CPB12” BankType=”Production”>library
                data</UseInformation>
            <UseInformation BankID=”CPB14” BankType=”Production”>general
                data</UseInformation>
        </Initiator>
        <Initiator ActorID=”A10”>
            <name>annual fee controller </name>
            <UseInformation BankID=”PB02”>PB02</UseInformation>
            <UseInformation BankID=”CPB14” BankType=”Production”>general
                data</UseInformation>
        </Initiator>
        <Executor ActorID=”CA01”>
            <name>board </name>
            <UseInformation BankID=”PB03”>PB03</UseInformation>
        </Executor>
        <UseInformation BankID=”PB03”>PB03</UseInformation>
        <Result ResultID=”R03”>the reduced fee for M in year Y is approved
            </Result>
    </Transaction>
<Transaction TransactionID=”T04”>
        <Tname>loan start </Tname>
        <Initiator ActorID=”CA04”>
            <name>member</name>
            <UseInformation BankID=”PB04”>PB04</UseInformation>
        </Initiator>
        <Executor ActorID=”A04”>
            <name>loan creator </name>
            <UseInformation BankID=”PB04”>PB04</UseInformation>
            <UseInformation BankID=”PB01”>PB01</UseInformation>
            <UseInformation BankID=”CPB12” BankType=”Production”>library
                data</UseInformation>
            <UseInformation BankID=”CPB14” BankType=”Production”>general
                data</UseInformation>
        </Executor>
        <UseInformation BankID=”PB04”>PB04</UseInformation>
        <Result ResultID=”R04”>loan L has been started </Result>
    </Transaction>
</Transactions>
```

D.2 XML documents for PM

Listing D.2: The exemplary XML document of PM

```
<?xml version=”1.0” encoding=”UTF-8” standalone=”no”?>
<TransactionPattern xmlns:xsi=”http://www.w3.org/2001/XMLSchema-instance”
    xsi:noNamespaceSchemaLocation=”C:\Tomcat\webapps\ROOT\demo\PM.xsd”>
```

```
<Transaction TransactionID=”T01” name=”membership registration”>
        <TransactionPhase name=”Order”>
            <step>
                <Name>Request </Name>
            </step>
            <step>
                <Name>Promise </Name>
                <InitiationTo TransactionID=”T03”>
                    <Name>Request </Name>
                </InitiationTo>
                <InitiationTo TransactionID=”T02”>
                    <Name>Request </Name>
                </InitiationTo>
            </step>
        </TransactionPhase>
        <TransactionPhase name=”Execution”>
            <step>
                <Name>Execute </Name>
                <WaitingFor TransactionID=”T02”>
                    <Name>Accept </Name>
                </WaitingFor>
            </step>
        </TransactionPhase>
        <TransactionPhase name=”Result”>
            <step>
                <Name>State </Name>
            </step>
            <step>
                <Name>Accept </Name>
            </step>
        </TransactionPhase>
    </Transaction>
<Transaction TransactionID=”T03” name=”reduced fee approval”>
        <TransactionPhase name=”Order”>
            <step>
                <Name>Request </Name>
                <InitiatedFrom TransactionID=”T01”>
                    <Name>Promise </Name>
                </InitiatedFrom>
            </step>
            <step>
                <Name>Promise </Name>
            </step>
        </TransactionPhase>
        <TransactionPhase name=”Execution”>
            <step>
                <Name>Execute </Name>
            </step>
        </TransactionPhase>
        <TransactionPhase name=”Result”>
            <step>
                <Name>State </Name>
            </step>
            <step>
                <Name>Accept </Name>
                <IsWaitConditionOf TransactionID=”T02”>
                    <Name>Request </Name>
                </IsWaitConditionOf>
            </step>
        </TransactionPhase>
    </Transaction>
```

```
<Transaction TransactionID=”T02” name=”membership fee payment”>
        <TransactionPhase name=”Order”>
            <step>
                <Name>Request</Name>
                <InitiatedFrom TransactionID=”T01”>
                    <Name>Promise</Name>
                </InitiatedFrom>
            </step>
            <step>
                <Name>Promise</Name>
            </step>
        </TransactionPhase>
        <TransactionPhase name=”Execution”>
            <step>
                <Name>Execute</Name>
            </step>
        </TransactionPhase>
        <TransactionPhase name=”Result”>
            <step>
                <Name>State</Name>
            </step>
            <step>
                <Name>Accept</Name>
                <IsWaitConditionOf TransactionID=”T01”>
                    <Name>Execute</Name>
                </IsWaitConditionOf>
            </step>
        </TransactionPhase>
    </Transaction>
</TransactionPattern>
```

D.3 XML documents for AM

Listing D.3: The exemplary XML document of AM

```
<?xml version=”1.0” encoding=”UTF–8” standalone=”no”?>
<ActionRules xsi:noNamespaceSchemaLocation=”C:\Tomcat\webapps\ROOT\demo\AM
    .xsd”>
        <Rule TransactionID=”T01” step=”request”>
                on request T01(M) with member (new M)=P
                        if age(P)<minimal_age —>decline T01(M)
                        <> age(P)>=minimal_age —>promise T01(M)
                        fi
                no
</Rule>
</ActionRules>
```

D.4 XML documents of SM

Listing D.4: The exemplary XML document of SM

```
<?xml version=”1.0” encoding=”UTF–8” standalone=”no”?>
<ObjectFact xmlns:xsi=”http://www.w3.org/2001/XMLSchema–instance” xsi:
    noNamespaceSchemaLocation=”C:\Tomcat\webapps\ROOT\demo\SM.xsd”>
<BasicConstruct FactTypeID=”F01” Formulation=”the membership of L is M”>
```

```
        <role HoldUnicity=”No” name=”M”>
                <HasDomain LawType=”Reference” ObjectClassID=”OB01”
                     ResultID=”R01”>MEMBERSHIP</HasDomain>
        </role>
        <role HoldUnicity=”Yes” name=”L”>
                <HasDomain LawType=”Dependency” ObjectClassID=”OB02”>LOAN
                     </HasDomain>
        </role>
</BasicConstruct>
<BasicConstruct FactTypeID=”F02” Formulation=”the book copy of L is C”>
        <role HoldUnicity=”Yes” name=”L”>
                <HasDomain LawType=”Dependency” ObjectClassID=”OB02”
                     ResultID=”R04,R06”>LOAN</HasDomain>
        </role>
        <role HoldUnicity=”No” name=”C”>
                <HasDomain LawType=”Reference” ObjectClassID=”OB03”>BOOK
                     COPY</HasDomain>
        </role>
</BasicConstruct>
<BasicConstruct FactTypeID=”F03” Formulation=”copies of B are delivered in
     S”>
        <role HoldUnicity=”No” name=”S”>
                <HasDomain LawType=”Dependency” ObjectClassID=”OB04”
                     ResultID=”R08”>SHIPMENT</HasDomain>
        </role>
        <role HoldUnicity=”No” name=”B”>
                <HasDomain LawType=”Reference” ObjectClassID=”OB05”>BOOK</
                     HasDomain>
                <IsExtendedTo ObjectClassID=”OB06”>LIBRARY BOOK</
                     IsExtendedTo>
        </role>
</BasicConstruct>

</ObjectFact>
```

D.5 XML documents for Cross-Model Tables

Listing D.5: The exemplary XML document for TRT

```
<?xml version=”1.0” encoding=”UTF−8” standalone=”no”?>
<TransactionResults xmlns:xsi=”http://www.w3.org/2001/XMLSchema−instance”
xsi:noNamespaceSchemaLocation=”C:\Tomcat\webapps\ROOT\demo\TRT.xsd”>
<TransactionResult>
          <TransactionType>T01membership registration</TransactionType>
          <ResultType>R01membership M has been started</ResultType>
</TransactionResult>
<TransactionResult>
          <TransactionType>T02membership fee payment</TransactionType>
          <ResultType>R02the fee for membership M in year Y has been paid
               </ResultType>
</TransactionResult>
<TransactionResult>
          <TransactionType>T03reduced fee approval</TransactionType>
          <ResultType>R03the reduced fee for M in year Y is approved</
               ResultType>
</TransactionResult>
<TransactionResult>
          <TransactionType>T04loan start</TransactionType>
                <ResultType>R04loan L has been started</ResultType>
```

```
</TransactionResult>

</TransactionResults>
```

Listing D.6: The exemplary XML document for BCT

```
<?xml version=”1.0” encoding=”UTF−8” standalone=”no”?>
<BankContents xmlns:xsi=”http://www.w3.org/2001/XMLSchema−instance” xsi:
    noNamespaceSchemaLocation=”C:\Tomcat\webapps\ROOT\demo\BCT.xsd”>
<BankContent>
                <InformationBank>PB01</InformationBank>
    <FactType>MEMBERSHIP,P is the member in M,membership M has been
        started,</FactType>
</BankContent>
<BankContent>
    <InformationBank>PB02</InformationBank>
    <FactType>the fee for membership M in year Y has been paid,</FactType>
</BankContent>
<BankContent>
    <InformationBank>PB03</InformationBank>
    <FactType>the reduced fee for membership M in year Y has been approved
        ,</FactType>
</BankContent>
<BankContent>
    <InformationBank>PB04</InformationBank>
    <FactType>LOAN,loan L has been started,the membership of L is M,the
        book copy of L is C,</FactType>
</BankContent>
</BankContents>
```

Listing D.7: The exemplary XML document for IUT

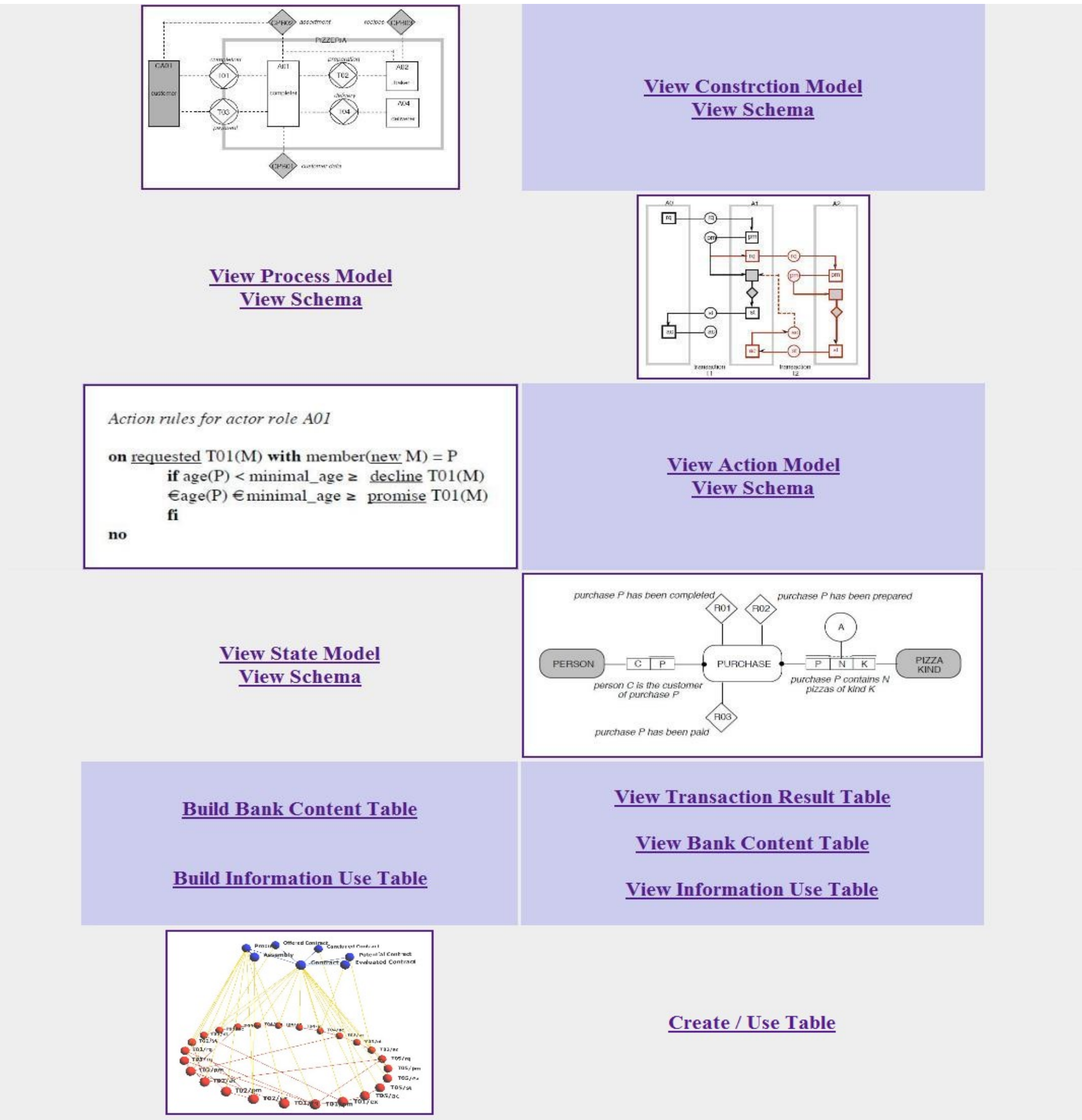
```
<?xml version=”1.0” encoding=”UTF−8” standalone=”no”?>
<InformationUsage xmlns:xsi=”http://www.w3.org/2001/XMLSchema−instance”
    xsi:noNamespaceSchemaLocation=”C:\Tomcat\webapps\ROOT\demo\IUT.xsd”>
<InformationUse>
                <InformationObject>MEMBERSHIP</InformationObject>
                <TransactionStep>T01request,T01promise,T04request</
                    TransactionStep>
</InformationUse>
<InformationUse>
                <InformationObject>P is the member in M</InformationObject
                    >
                <TransactionStep>T01request</TransactionStep>
</InformationUse>
<InformationUse>
                <InformationObject>LOAN</InformationObject>
                <TransactionStep>T04request,T06promise</TransactionStep>
</InformationUse>
<InformationUse>
                <InformationObject>the membership of L is M</
                    InformationObject>
                <TransactionStep>T04request</TransactionStep>
</InformationUse>
</InformationUsage>
```

Appendix E

Manual

Manual

This is the application for supporting the DEMO model transformation, which is proposed in the master thesis “Transformation of DEMO models into exchangeable format”. It has five functions, which are building the Construction Model, building the Process Model, building the Action model, building the State Model, and building the Create / Use Table. This application is built in web pages by using the Hyper Text Markup Language (HTML)[1], and uses Java Server Page (JSP) [2]technology and JavaScript[3] to create and access the dynamic web content. The server for running the JSP files is the TOMCAT 6.0.



View Constrction Model
View Schema

View Process Model
View Schema

```
Action rules for actor role A01

on requested T01(M) with member(new M) = P
        if age(P) < minimal_age ≥  decline T01(M)
        €age(P) €minimal_age ≥  promise T01(M)
        fi
no
```

View Action Model
View Schema

View State Model
View Schema

Build Bank Content Table

Build Information Use Table

View Transaction Result Table

View Bank Content Table

View Information Use Table

Create / Use Table

PIZZERIA
CA01
customer
T01
T02
T03
T04
A01
A02
A04
purchase P has been completed
purchase P has been prepared
R01
R02
R03
A
PERSON
C
P
PURCHASE
N
K
PIZZA KIND
person C is the customer of purchase P
purchase P contains N pizzas of kind K
purchase P has been paid

[1] http://www.w3schools.com/html/DEFAULT.asp

[2] http://java.sun.com/products/jsp/

[3] http://www.w3schools.com/JS/default.asp

Construction Model

1. To start constructing the CM, please click the CM diagram on the left side in the home page, as shown in Figure 1. Go to 2.

   To view the exemplary XML document for CM, please click “View Construction Model” on the right side. Go to 9.

   To view the XML schema of the CM, please click “View Schema” on the right side. Go to 10.

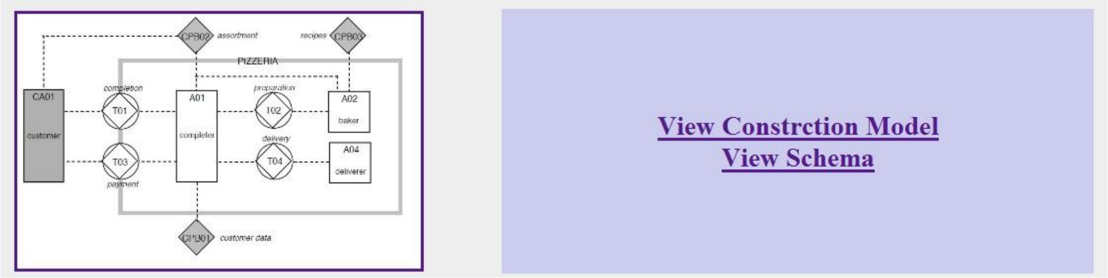
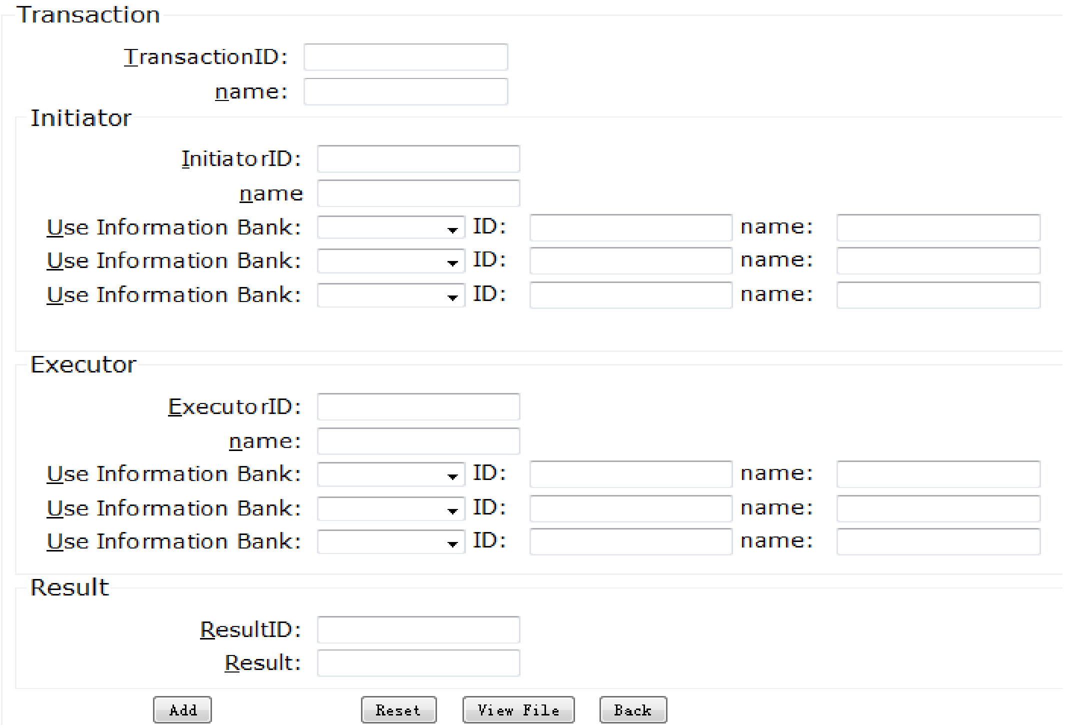


Figure 1 CM Home Page

2. The page for creating the XML document for CM is shown in Figure 2. The CM is constructed by adding transactions one by one. Information that required as input for one transaction is listed in the form below. You can always click button “View File” to check what you create in the XML document for CM.



Transaction

TransactionID:

name:

Initiator

InitiatorID:

name

Use Information Bank: ID: name:

Use Information Bank: ID: name:

Use Information Bank: ID: name:

Executor

ExecutorID:

name:

Use Information Bank: ID: name:

Use Information Bank: ID: name:

Use Information Bank: ID: name:

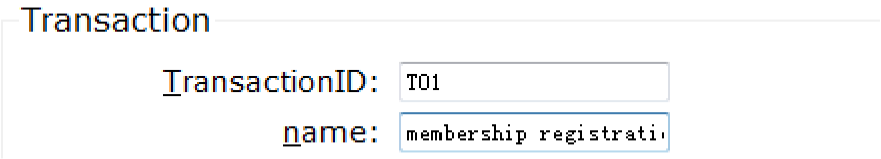
Result

ResultID:

Result:

Figure 2 Complete form for Adding one transaction

3. Please fill in the transaction kind information as shown in Figure 3.



Transaction

TransactionID: T01

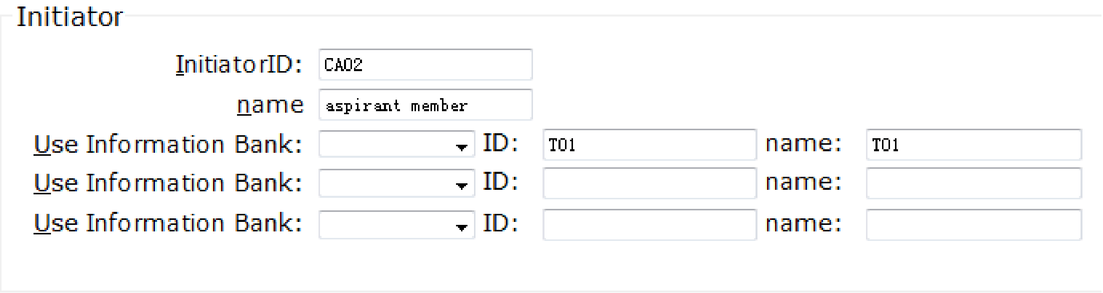
name: membership registrati

Figure 3 Exemplary transaction kind information

4. Please fill in the information related to the initiator, as shown in Figure 4, including the information about the initiator, the information banks used by the initiator.

   You can choose the information bank type in the select list, but it is not necessary to give the type of the information bank when it comes to the information bank that belongs to the transaction.

   Leave the text area blank, if there is no more used information bank.



Initiator

InitiatorID: CA02

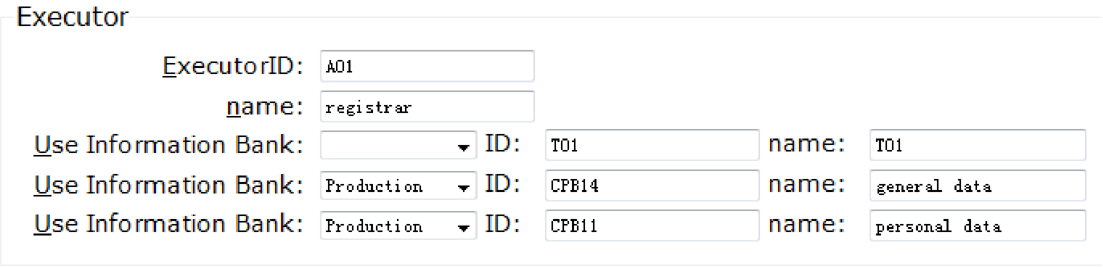
name aspirant member

Use Information Bank:	▾	ID:	T01	name:	T01
Use Information Bank:	▾	ID:		name:	
Use Information Bank:	▾	ID:		name:	

Figure 4 Exemplary initiator information

5. Please fill in the information related to the executor, as shown in Figure 5, including the information about executor, the information banks used by the executor.

   Choose the information bank type from the select list, if it is necessary.



Executor

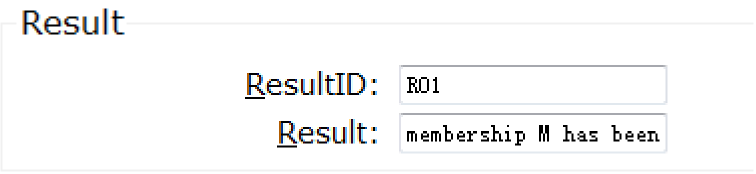
ExecutorID: A01

name: registrar

Use Information Bank:	▾	ID:	T01	name:	T01
Use Information Bank:	Production ▾	ID:	CPB14	name:	general data
Use Information Bank:	Production ▾	ID:	CPB11	name:	personal data

Figure 5 Exemplary executor information

6. Please fill in the transaction result information, as shown in Figure 6.



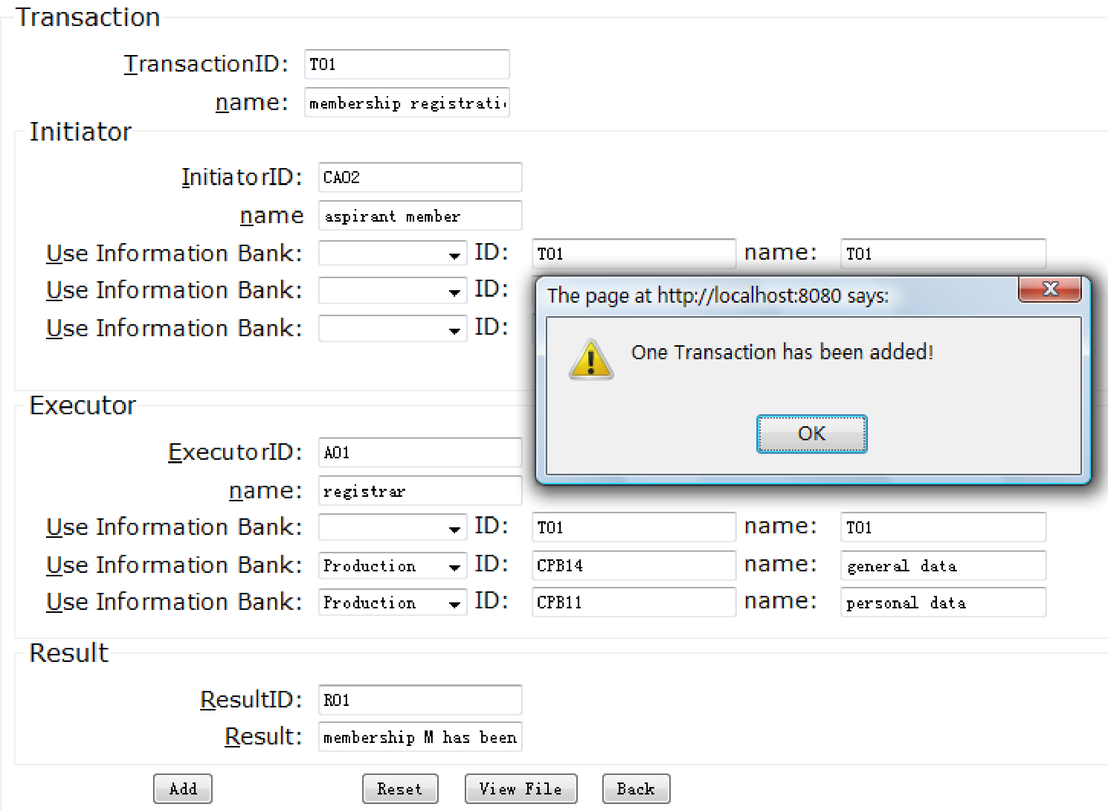
Result

ResultID: R01

Result: membership M has been

Figure 6 Exemplary transaction result information

7. Click button “Add” to add a new transaction with the given information into the CM.xml file. You will see the confirmation message, as shown in Figure 7. The transaction has been saved in the XML document for CM.



Transaction

TransactionID: T01

name: membership registrati

Initiator

InitiatorID: CA02

name aspirant member

Use Information Bank: ID: T01 name: T01

Use Information Bank: ID:

Use Information Bank: ID:

The page at http://localhost:8080 says:

One Transaction has been added!

Executor

ExecutorID: A01

name: registrar

Use Information Bank: ID: T01 name: T01

Use Information Bank: Production ID: CPB14 name: general data

Use Information Bank: Production ID: CPB11 name: personal data

Result

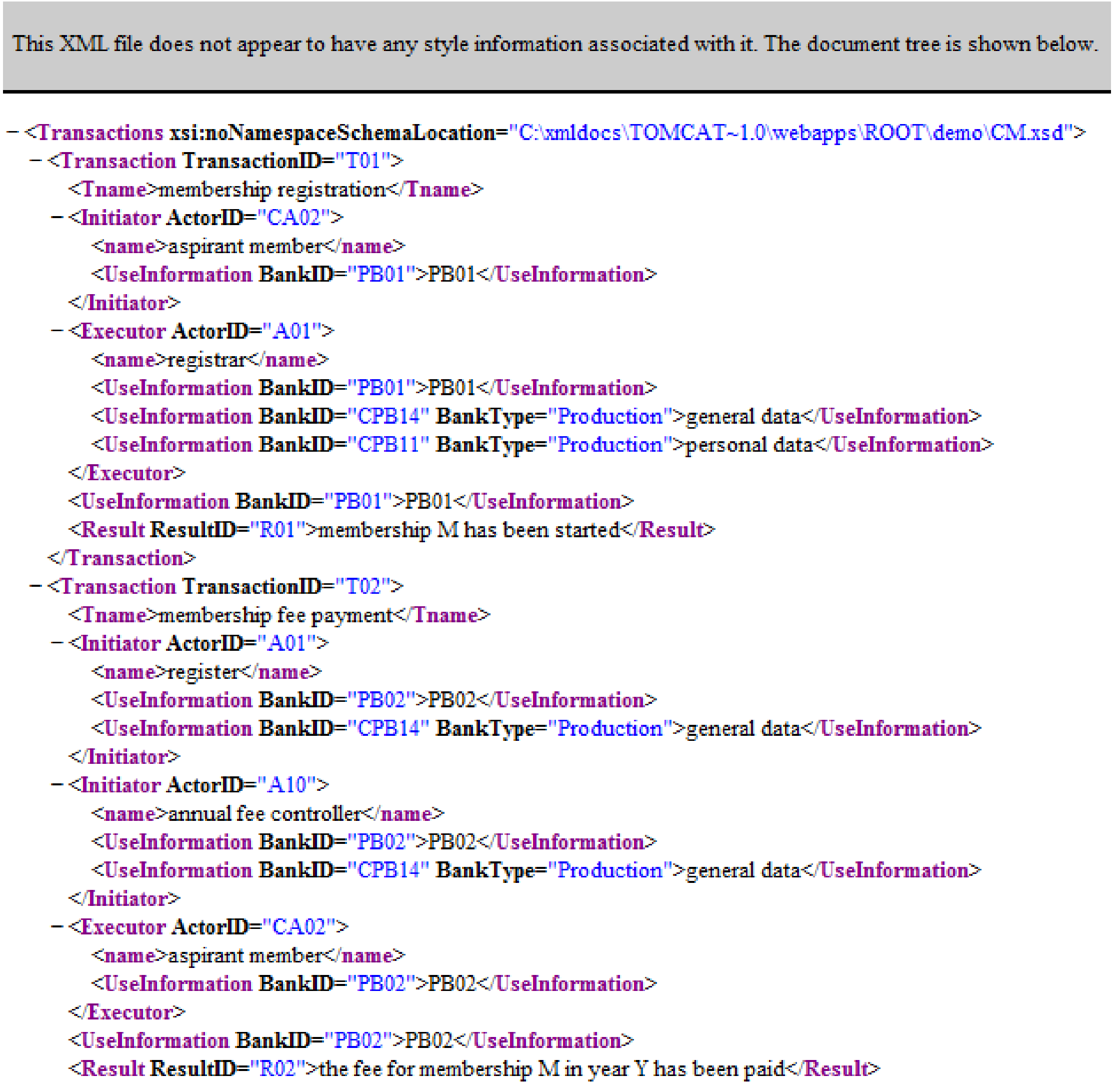
ResultID: R01

Result: membership M has been

Figure 7 Confirmation message

8. Click “ok”, the page returns back to the same form as shown in Figure 2. Repeat step 2-8 if you need to add more transactions, or click button “Back” to return to the home page.

9. After clicking “View Construction Model” on the home page or the button “View File” on the page of constructing the CM, you will see the XML document for CM which you create as shown in Figure 8.

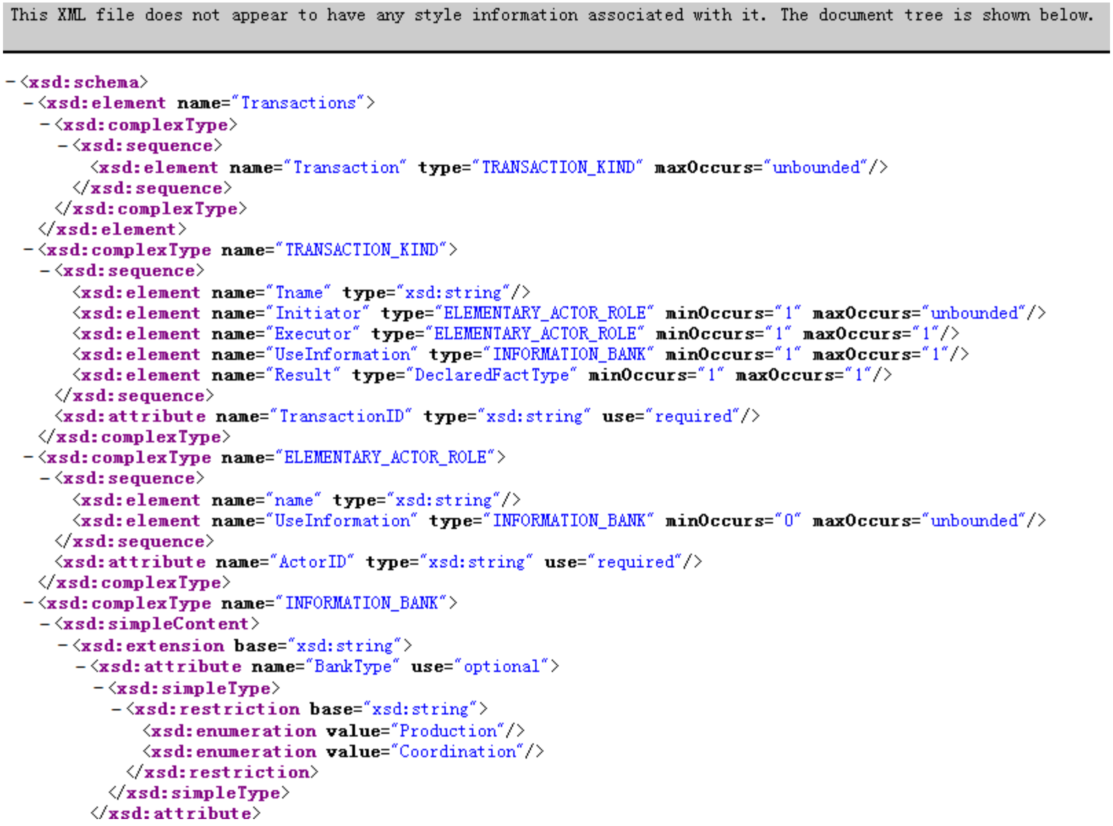


This XML file does not appear to have any style information associated with it. The document tree is shown below.

```
– <Transactions xsi:noNamespaceSchemaLocation="C:\xmldocs\TOMCAT~1.0\webapps\ROOT\demo\CM.xsd">
  – <Transaction TransactionID="T01">
      <Tname>membership registration</Tname>
    – <Initiator ActorID="CA02">
        <name>aspirant member</name>
        <UseInformation BankID="PB01">PB01</UseInformation>
      </Initiator>
    – <Executor ActorID="A01">
        <name>registrar</name>
        <UseInformation BankID="PB01">PB01</UseInformation>
        <UseInformation BankID="CPB14" BankType="Production">general data</UseInformation>
        <UseInformation BankID="CPB11" BankType="Production">personal data</UseInformation>
      </Executor>
      <UseInformation BankID="PB01">PB01</UseInformation>
      <Result ResultID="R01">membership M has been started</Result>
    </Transaction>
  – <Transaction TransactionID="T02">
      <Tname>membership fee payment</Tname>
    – <Initiator ActorID="A01">
        <name>register</name>
        <UseInformation BankID="PB02">PB02</UseInformation>
        <UseInformation BankID="CPB14" BankType="Production">general data</UseInformation>
      </Initiator>
    – <Initiator ActorID="A10">
        <name>annual fee controller</name>
        <UseInformation BankID="PB02">PB02</UseInformation>
        <UseInformation BankID="CPB14" BankType="Production">general data</UseInformation>
      </Initiator>
    – <Executor ActorID="CA02">
        <name>aspirant member</name>
        <UseInformation BankID="PB02">PB02</UseInformation>
      </Executor>
      <UseInformation BankID="PB02">PB02</UseInformation>
      <Result ResultID="R02">the fee for membership M in year Y has been paid</Result>
```

Figure 8 Exemplary XML document for CM

10. After clicking “View Schema”, you will see the XML schema of the CM as shown in Figure 9.



```
This XML file does not appear to have any style information associated with it. The document tree is shown below.

-<xsd:schema>
  -<xsd:element name="Transactions">
    -<xsd:complexType>
      -<xsd:sequence>
          <xsd:element name="Transaction" type="TRANSACTION_KIND" maxOccurs="unbounded"/>
        </xsd:sequence>
      </xsd:complexType>
    </xsd:element>
  -<xsd:complexType name="TRANSACTION_KIND">
    -<xsd:sequence>
        <xsd:element name="Tname" type="xsd:string"/>
        <xsd:element name="Initiator" type="ELEMENTARY_ACTOR_ROLE" minOccurs="1" maxOccurs="unbounded"/>
        <xsd:element name="Executor" type="ELEMENTARY_ACTOR_ROLE" minOccurs="1" maxOccurs="1"/>
        <xsd:element name="UseInformation" type="INFORMATION_BANK" minOccurs="1" maxOccurs="1"/>
        <xsd:element name="Result" type="DeclaredFactType" minOccurs="1" maxOccurs="1"/>
      </xsd:sequence>
      <xsd:attribute name="TransactionID" type="xsd:string" use="required"/>
    </xsd:complexType>
  -<xsd:complexType name="ELEMENTARY_ACTOR_ROLE">
    -<xsd:sequence>
        <xsd:element name="name" type="xsd:string"/>
        <xsd:element name="UseInformation" type="INFORMATION_BANK" minOccurs="0" maxOccurs="unbounded"/>
      </xsd:sequence>
      <xsd:attribute name="ActorID" type="xsd:string" use="required"/>
    </xsd:complexType>
  -<xsd:complexType name="INFORMATION_BANK">
    -<xsd:simpleContent>
      -<xsd:extension base="xsd:string">
        -<xsd:attribute name="BankType" use="optional">
          -<xsd:simpleType>
            -<xsd:restriction base="xsd:string">
                <xsd:enumeration value="Production"/>
                <xsd:enumeration value="Coordination"/>
              </xsd:restriction>
            </xsd:simpleType>
          </xsd:attribute>
```

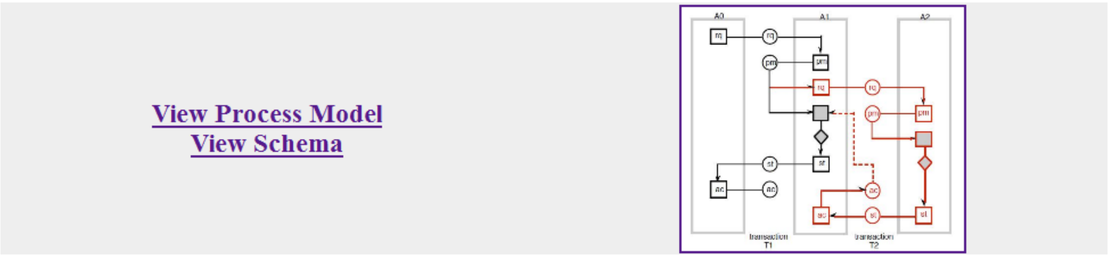
Figure 9 XML schema of the CM

Process Model

1. To start constructing the PM, please click the PM diagram on the right side in the home page, as shown in Figure 10. Go to 2.

   To view the exemplary XML document for PM, please click “View Process Model” on the left side. Go to 10.

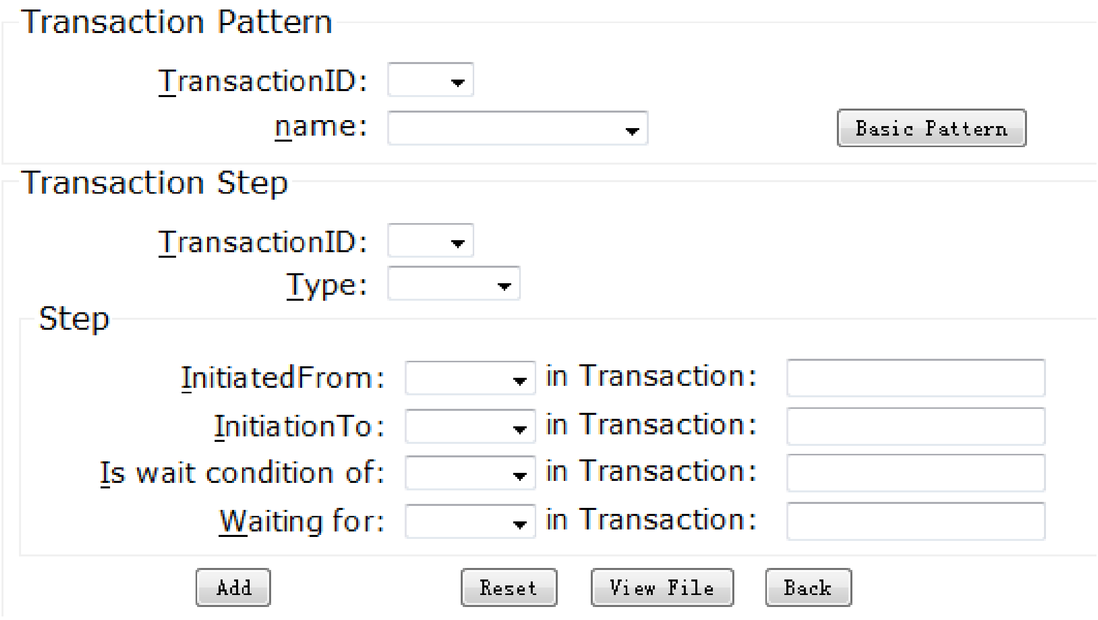
   To view the XML schema of the PM, please click “View Schema” on the left side. Go to 11.



View Process Model
View Schema

Figure 10 PM Home Page

2. In Figure 11, the page for creating the XML document for PM is shown. The PM is constructed by adding the transaction steps for each transaction. All the information regarding to a process step can be filled in the form below, in Figure 11. You can always click button “View File” to check what you create in the XML document for PM.



Transaction Pattern

TransactionID:

name:

Transaction Step

TransactionID:

Type:

Step

InitiatedFrom: in Transaction:

InitiationTo: in Transaction:

Is wait condition of: in Transaction:

Waiting for: in Transaction:

Figure 11 Complete form for adding information about a process step

3. In the form of Transaction Pattern, as shown in Figure 12, it lists all the transaction id and names from the XML document for CM. Choose the transaction kind information from the select lists, to which you want to add transaction steps. Click button “Basic Pattern” to add the basic transaction steps to the transaction.

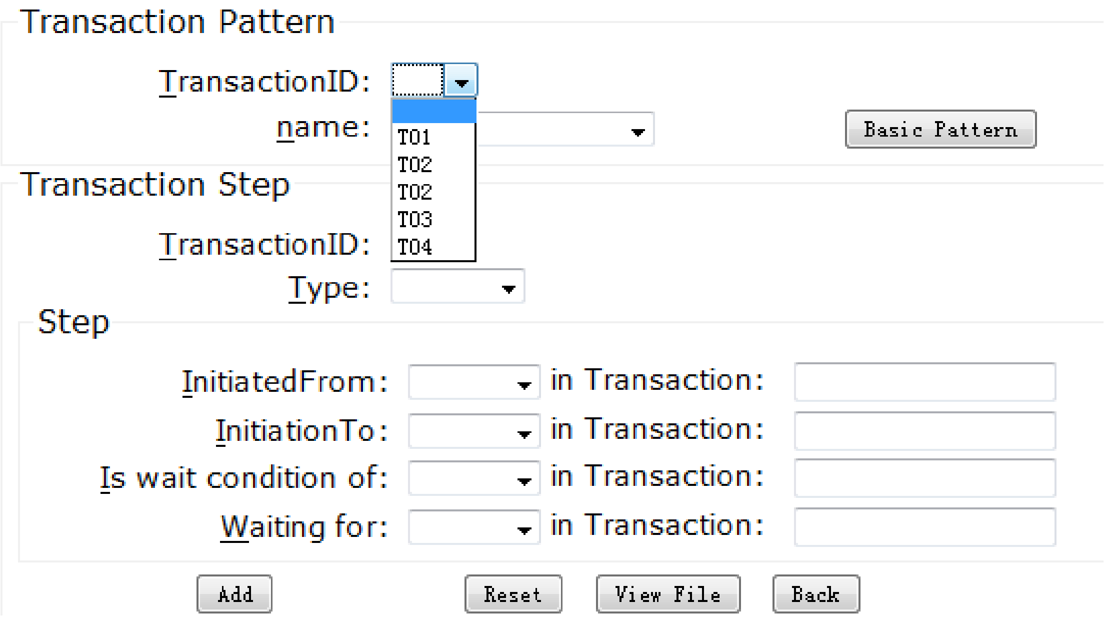
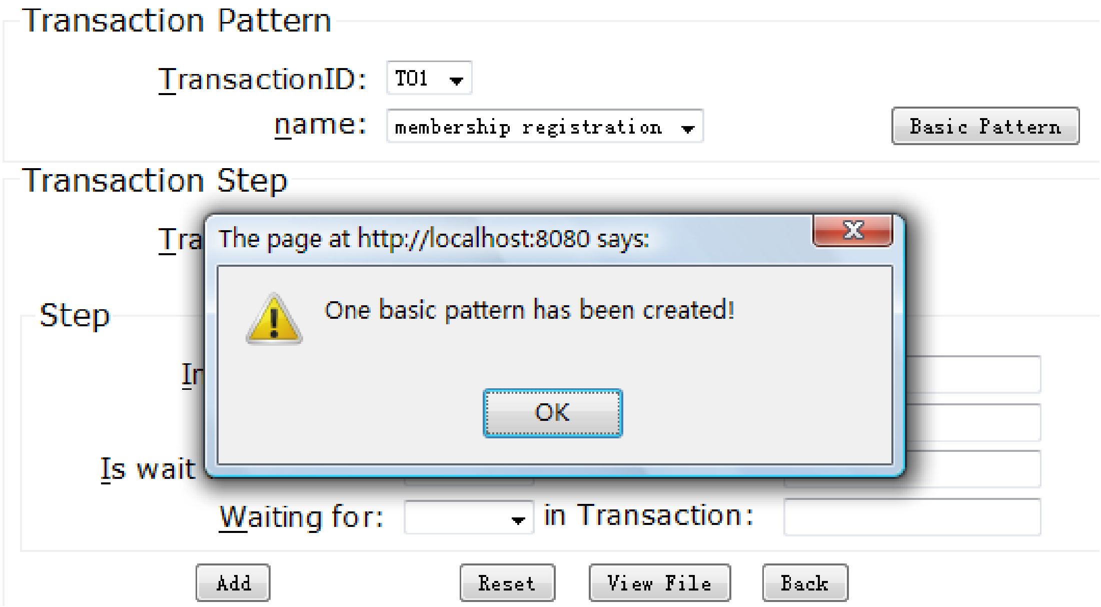


Figure 12 Exemplary transaction kind information

4. After clicking button “Basic Pattern”, you will see the confirmation message as shown in Figure 13. One basic pattern of transaction steps are added to the certain transaction in the XML document for PM.



TransactionID: T01

name: membership registration

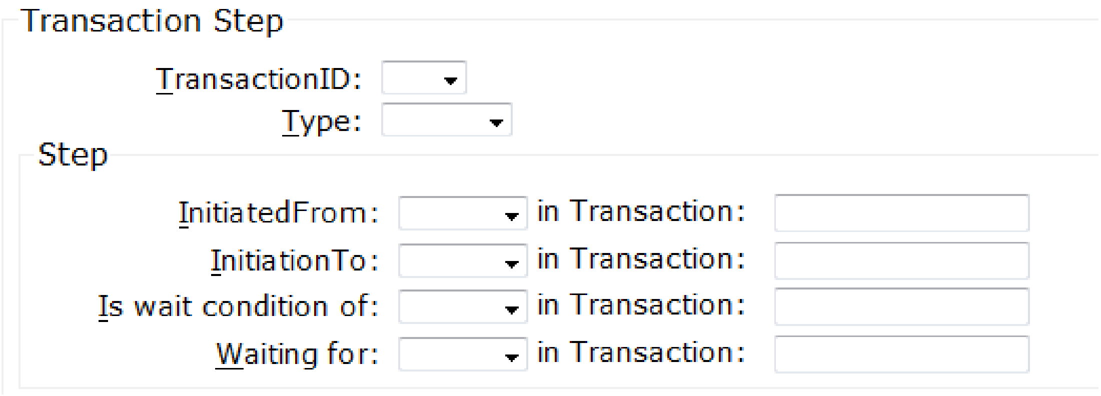
The page at http://localhost:8080 says:

One basic pattern has been created!

Figure 13 Confirmation message

5. Click “OK” to return to the complete form as shown in Figure 11. Repeat step 2-5, if you need to add transaction steps for more transactions, or click the button “Back” to return to the home page. If there is any additional condition within the transaction steps other than the basic pattern, go to 6.

6. If there are more interrelationships among the transaction steps other than a basic pattern in a transaction. Please fill the relevant information in the form as shown in Figure 14.



Transaction Step

TransactionID:

Type:

Step

InitiatedFrom: in Transaction:

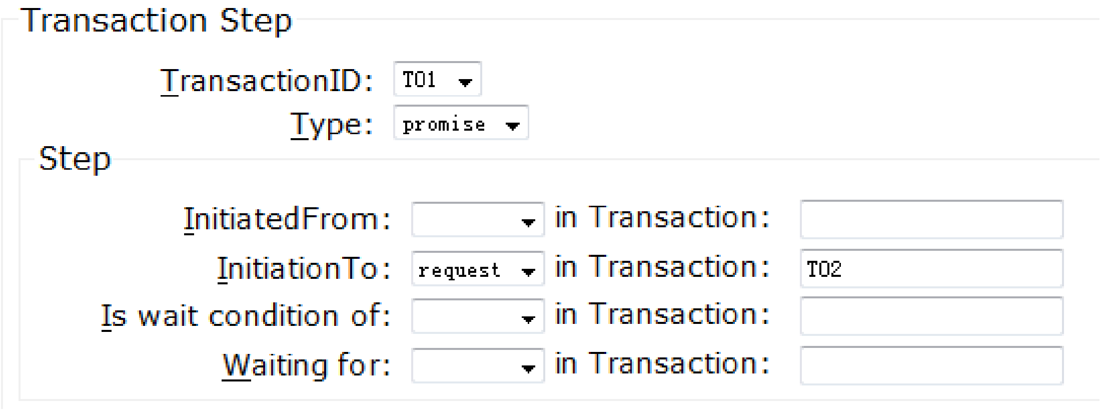
InitiationTo: in Transaction:

Is wait condition of: in Transaction:

Waiting for: in Transaction:

Figure 14 Form for additional information in a transaction step

7. Fill the additional information in the form as shown in Figure 15. Specify to which transaction step, the additional information will be added. The select list “TransactionID” shows all the existing transactions in the XML document for PM. Choose the transaction ID, and choose the transaction step. Fill in the additional information, the related step type and the transaction kind, in the relevant area.



Transaction Step

TransactionID: T01

Type: promise

Step

InitiatedFrom: in Transaction:

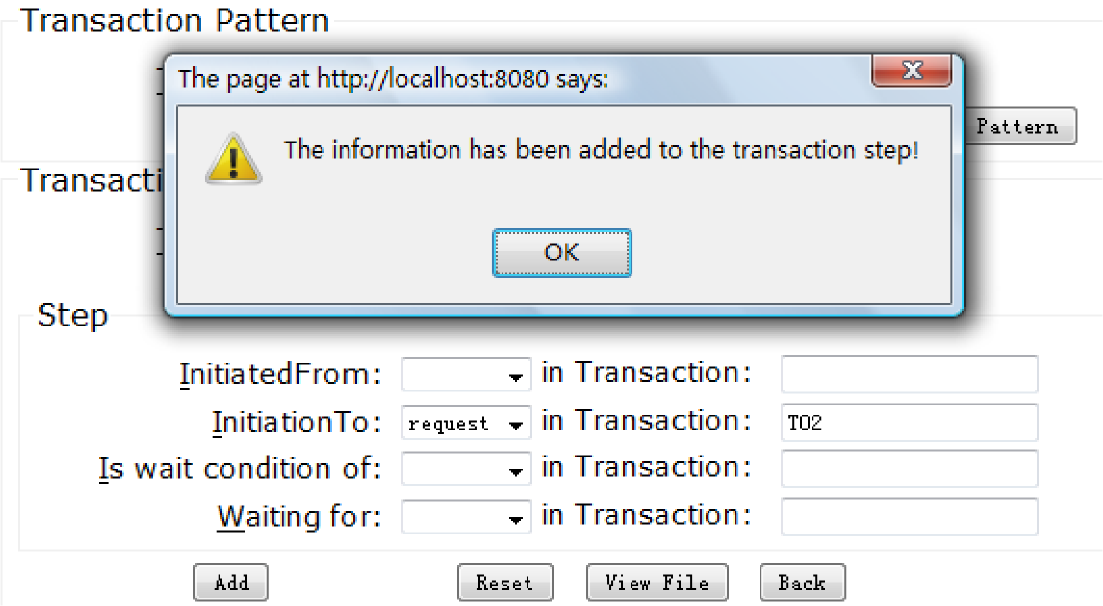
InitiationTo: request in Transaction: T02

Is wait condition of: in Transaction:

Waiting for: in Transaction:

Figure 15 Exemplary additional information to a transaction step

8. Click the button “Add”, you will see the confirmation message as shown in Figure 16. The additional information has been added to the certain transaction step in the XML document for PM.



The page at http://localhost:8080 says:

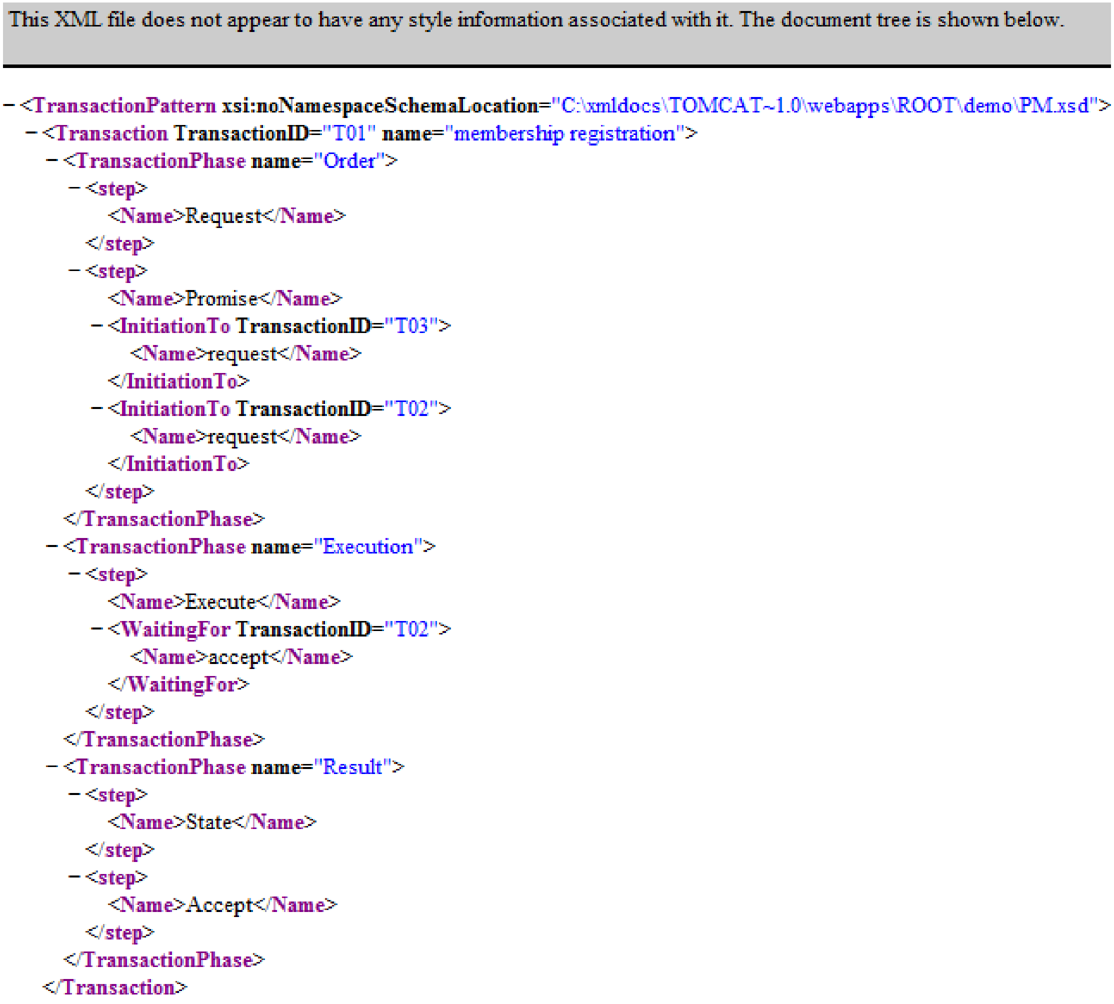
The information has been added to the transaction step!

InitiatedFrom:		in Transaction:	
InitiationTo:	request	in Transaction:	T02
Is wait condition of:		in Transaction:	
Waiting for:		in Transaction:	

Figure 16 Confirmation message

9. Click “OK” to return to the complete form as shown in Figure 11. Repeat step 6-9 if you need to add more information to the transaction steps, or click the button “Back” to return to the home page.

10. After clicking “View Process Model”, you will see the exemplary XML document for PM as shown in Figure 17.

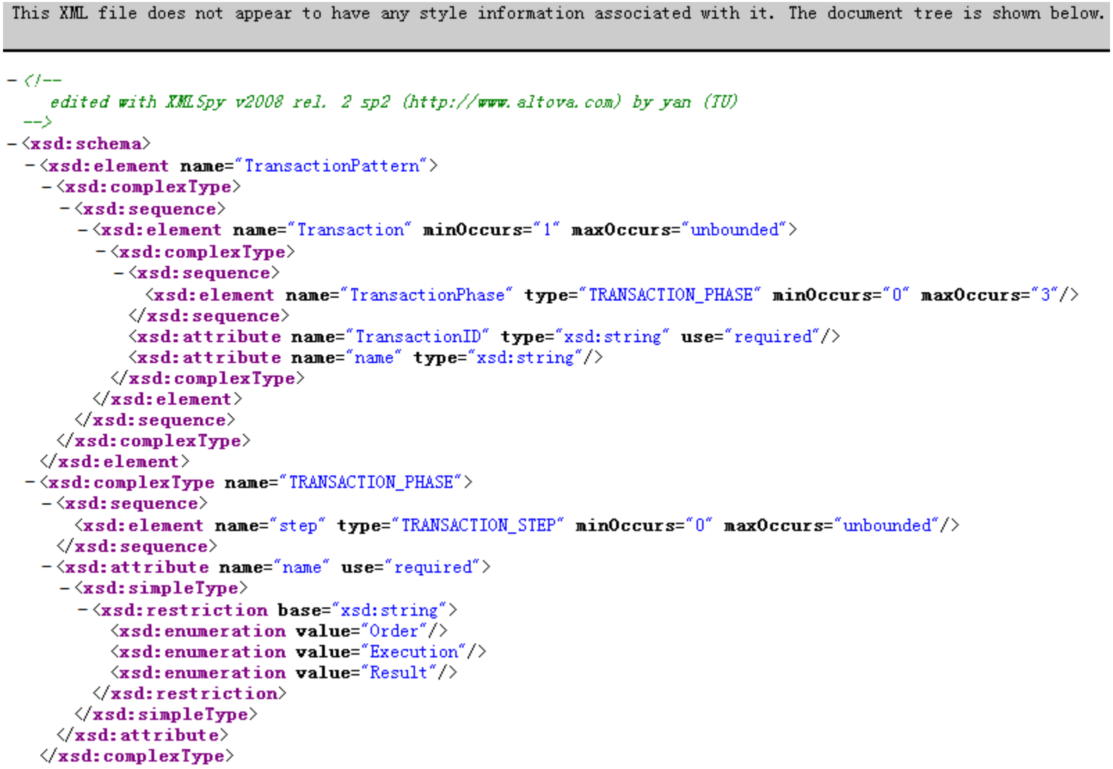


This XML file does not appear to have any style information associated with it. The document tree is shown below.

```
– <TransactionPattern xsi:noNamespaceSchemaLocation="C:\xmldocs\TOMCAT~1.0\webapps\ROOT\demo\PM.xsd">
  – <Transaction TransactionID="T01" name="membership registration">
    – <TransactionPhase name="Order">
      – <step>
          <Name>Request</Name>
        </step>
      – <step>
          <Name>Promise</Name>
        – <InitiationTo TransactionID="T03">
            <Name>request</Name>
          </InitiationTo>
        – <InitiationTo TransactionID="T02">
            <Name>request</Name>
          </InitiationTo>
        </step>
      </TransactionPhase>
    – <TransactionPhase name="Execution">
      – <step>
          <Name>Execute</Name>
        – <WaitingFor TransactionID="T02">
            <Name>accept</Name>
          </WaitingFor>
        </step>
      </TransactionPhase>
    – <TransactionPhase name="Result">
      – <step>
          <Name>State</Name>
        </step>
      – <step>
          <Name>Accept</Name>
        </step>
      </TransactionPhase>
    </Transaction>
```

Figure 17 Exemplary XML document for PM

11. After clicking “View schema”, you will see the XML schema of the PM as shown in Figure 18.



This XML file does not appear to have any style information associated with it. The document tree is shown below.

```
- <!--
    edited with XMLSpy v2008 rel. 2 sp2 (http://www.altova.com) by yan (TU)
  -->
- <xsd:schema>
  - <xsd:element name="TransactionPattern">
    - <xsd:complexType>
      - <xsd:sequence>
        - <xsd:element name="Transaction" minOccurs="1" maxOccurs="unbounded">
          - <xsd:complexType>
            - <xsd:sequence>
                <xsd:element name="TransactionPhase" type="TRANSACTION_PHASE" minOccurs="0" maxOccurs="3"/>
              </xsd:sequence>
              <xsd:attribute name="TransactionID" type="xsd:string" use="required"/>
              <xsd:attribute name="name" type="xsd:string"/>
            </xsd:complexType>
          </xsd:element>
        </xsd:sequence>
      </xsd:complexType>
    </xsd:element>
  - <xsd:complexType name="TRANSACTION_PHASE">
    - <xsd:sequence>
        <xsd:element name="step" type="TRANSACTION_STEP" minOccurs="0" maxOccurs="unbounded"/>
      </xsd:sequence>
    - <xsd:attribute name="name" use="required">
      - <xsd:simpleType>
        - <xsd:restriction base="xsd:string">
            <xsd:enumeration value="Order"/>
            <xsd:enumeration value="Execution"/>
            <xsd:enumeration value="Result"/>
          </xsd:restriction>
        </xsd:simpleType>
      </xsd:attribute>
    </xsd:complexType>
```

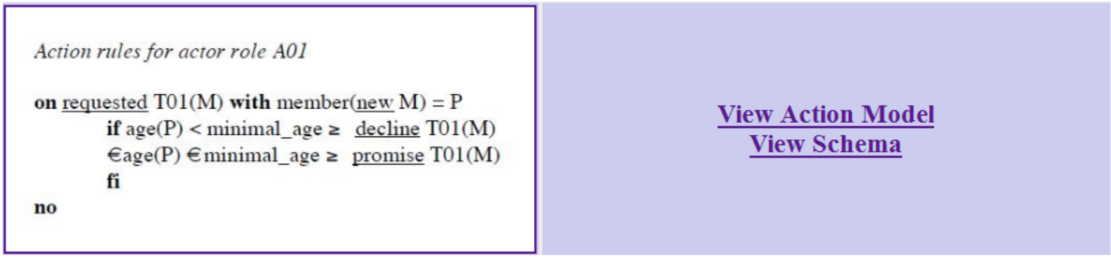
Figure 18 XML schema of the PM

Action Model

1. To start constructing the AM, please click the action rule on the left side in the home page, as shown in Figure 19. Go to 2.

   To view the exemplary XML document for AM, please click “View Action Model” on the right side. Go to 6.

   To view the XML schema of the AM, please click “View Schema” on the right side. Go to 7.



Action rules for actor role A01

```
on requested T01(M) with member(new M) = P
        if age(P) < minimal_age ≥  decline T01(M)
        €age(P) €minimal_age ≥  promise T01(M)
        fi
no
```

Figure 19 AM home page

2. Now it is the page of adding action rules. Figure 20 shows the page with the complete form for adding a new action rule. Information needed to be filled in this form includes the action rule and a specific transaction step, to which the action rule is assigned. You can always click button “View File” to check what you create in the XML document for AM.

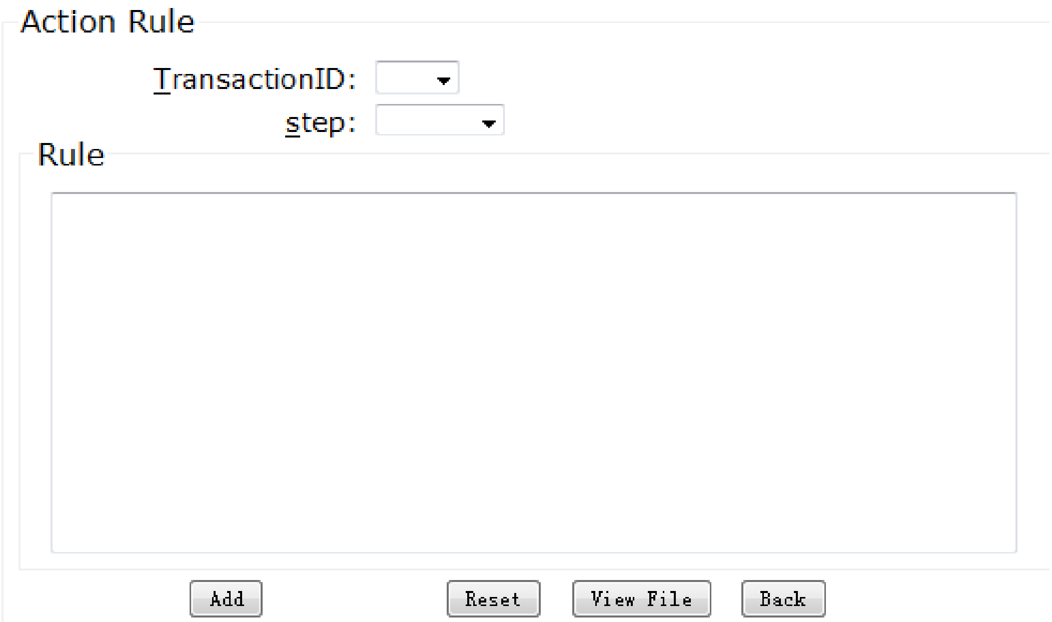
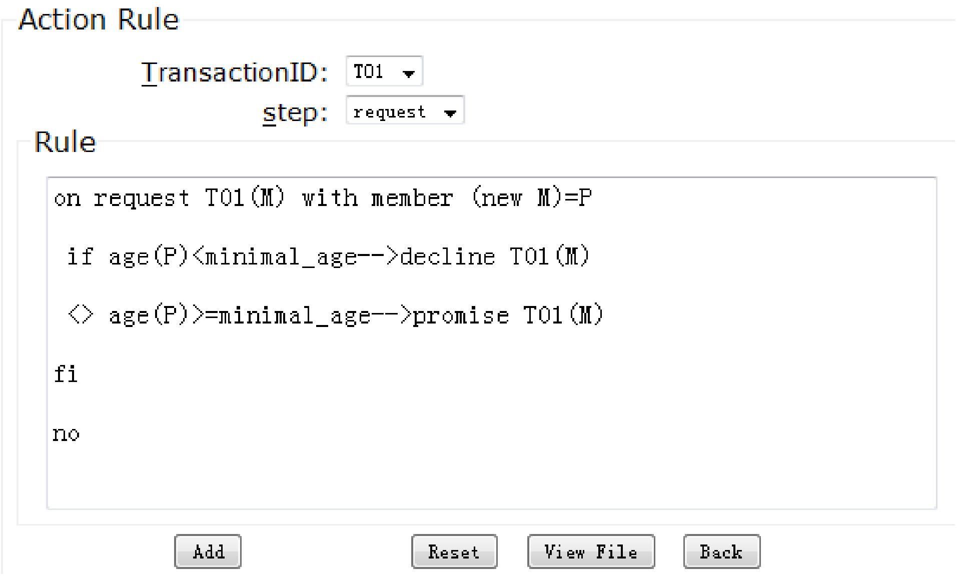


Figure 20 Complete form for adding action rules

3. Click the button “Add” when all the necessary information is given and ready to be saved, as shown in Figure 21.



Action Rule

TransactionID: T01

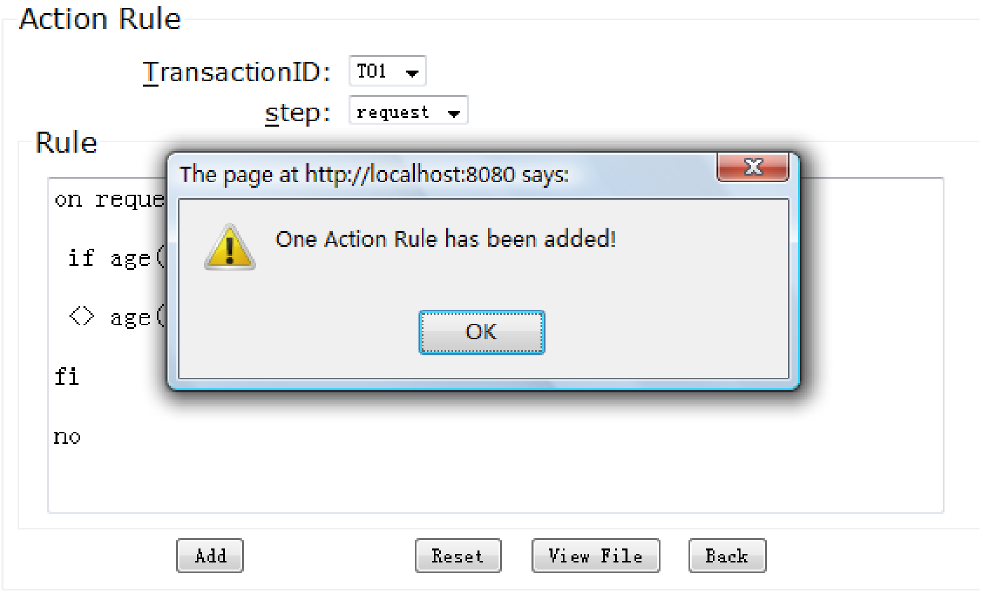
step: request

Rule

```
on request T01(M) with member (new M)=P
 if age(P)<minimal_age-->decline T01(M)
 <> age(P)>=minimal_age-->promise T01(M)
fi
no
```

Figure 21 Exemplary action rule information

4. After clicking the button “Add”, you will see the confirmation message as shown in Figure 22. The action rule has been saved in the XML document for AM.



Action Rule

TransactionID: T01

step: request

Rule

```
on reque
 if age(
 <> age(
fi
no
```

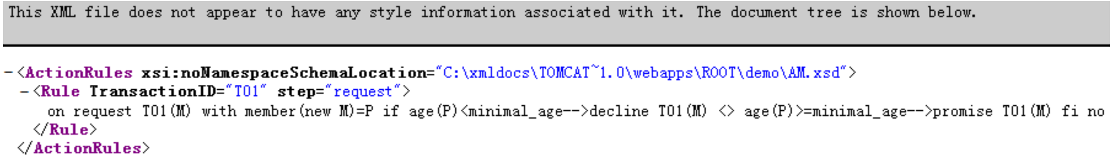
The page at http://localhost:8080 says:

One Action Rule has been added!

Figure 22 Confirmation message

5. Click “OK” to return to the form as shown in Figure 20. Repeat step 2-5 if you need to add more action rules, or click the button “Back” to return to the home page.

6. After clicking “View Action Model”, you will see the exemplary XML document for AM as shown in Figure 23.

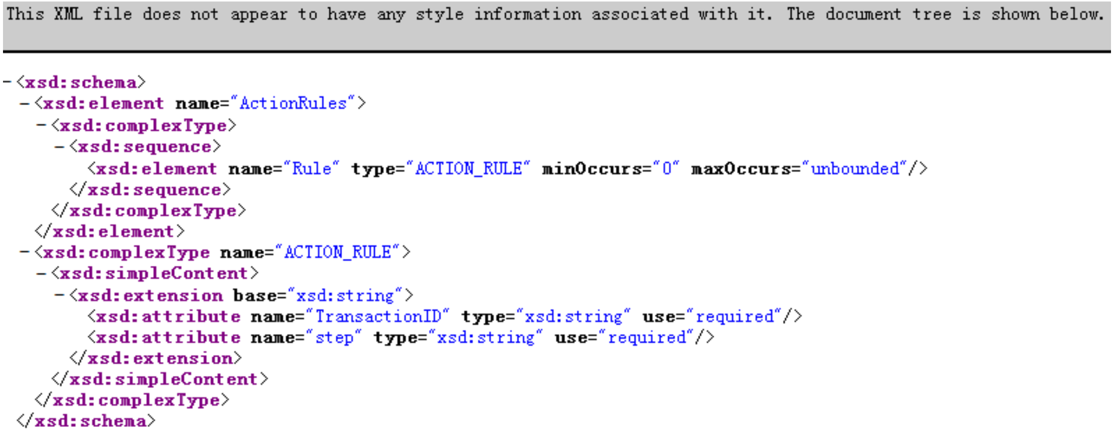


```
This XML file does not appear to have any style information associated with it. The document tree is shown below.

-<ActionRules xsi:noNamespaceSchemaLocation="C:\xmldocs\TOMCAT~1.0\webapps\ROOT\demo\AM.xsd">
  -<Rule TransactionID="T01" step="request">
     on request T01(M) with member(new M)=P if age(P)<minimal_age-->decline T01(M) <> age(P)>=minimal_age-->promise T01(M) fi no
   </Rule>
 </ActionRules>
```

Figure 23 Exemplary XML document for AM

7. After clicking "View Schema", you will see the XML schema of the AM as show in Figure 24.



```
This XML file does not appear to have any style information associated with it. The document tree is shown below.

-<xsd:schema>
  -<xsd:element name="ActionRules">
    -<xsd:complexType>
      -<xsd:sequence>
          <xsd:element name="Rule" type="ACTION_RULE" minOccurs="0" maxOccurs="unbounded"/>
        </xsd:sequence>
      </xsd:complexType>
    </xsd:element>
  -<xsd:complexType name="ACTION_RULE">
    -<xsd:simpleContent>
      -<xsd:extension base="xsd:string">
          <xsd:attribute name="TransactionID" type="xsd:string" use="required"/>
          <xsd:attribute name="step" type="xsd:string" use="required"/>
        </xsd:extension>
      </xsd:simpleContent>
    </xsd:complexType>
  </xsd:schema>
```

Figure 24 XML schema of the AM

State Model

1. To start constructing the SM, please click the SM diagram on the right side in the home page, as shown in Figure 25. Go to 2.

   To view the exemplary XML document for SM, please click “View State Model” on the left side. Go to 8.

   To view the XML schema of the SM, please click “View Schema” on the left side. Go to 9.

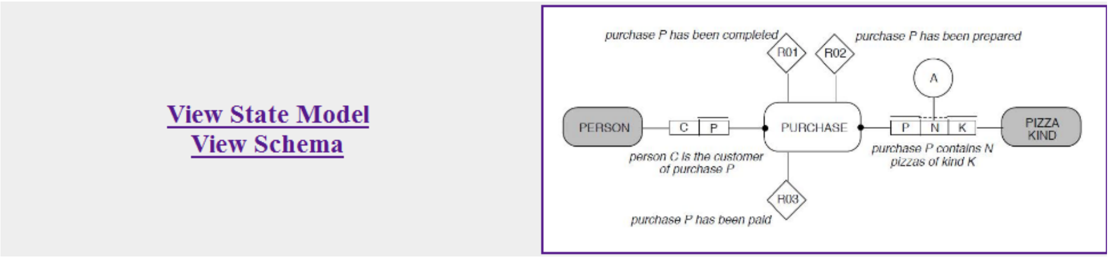

View State Model
View Schema
purchase P has been completed
R01
R02
purchase P has been prepared
A
PERSON
C
P
PURCHASE
P
N
K
PIZZA KIND
person C is the customer of purchase P
purchase P contains N pizzas of kind K
R03
purchase P has been paid

Figure 25 SM home page

2. Now it is the page of adding one basic construct. The complete form including all the required information for adding one basic construct is shown in Figure 26. The required information includes the general information about the basic construct, the information about the two roles within the basic construct. You can always click button “View File” to check what you create in the XML document for SM.

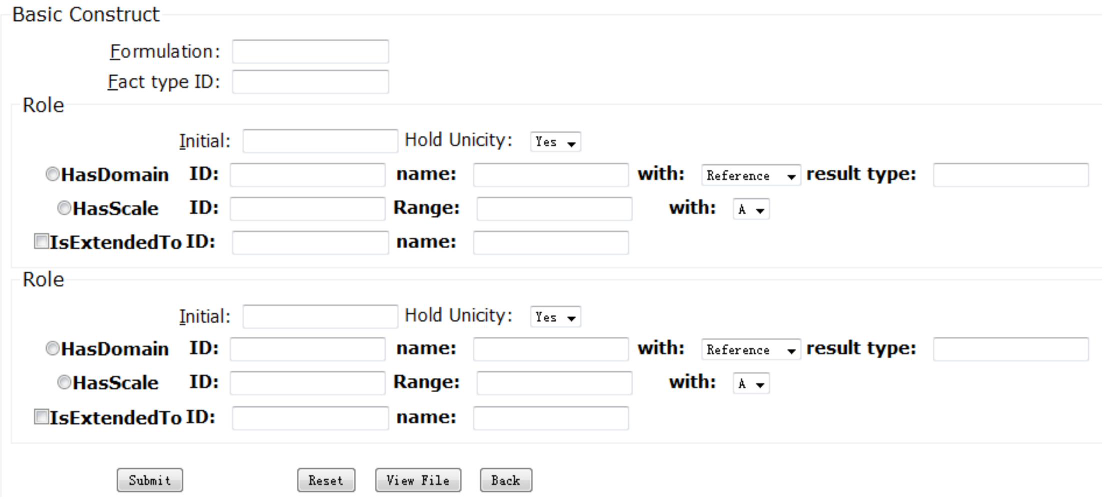
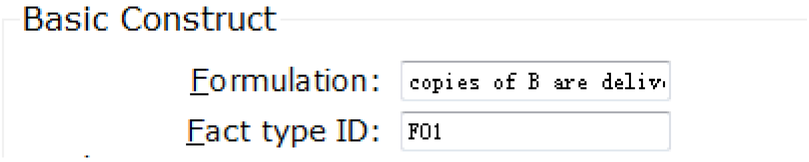


Figure 26 Complete form of adding one basic construct

3. First please give the general information about the basic construct, which includes the formulation of the basic construct and a fact type ID, as shown in Figure 27.



Basic Construct

Formulation: copies of B are deliv

Fact type ID: F01

Figure 27 Exemplary general information about the basic construct

4. For each role within the basic construct, give a letter as its initial.

   Select “Yes” in the select list “Hold Unicity” if the role holds a unicity law, and select “No” if not.

   If the role has an object class as its domain, select “HasDomain”, otherwise select “HasScale”.

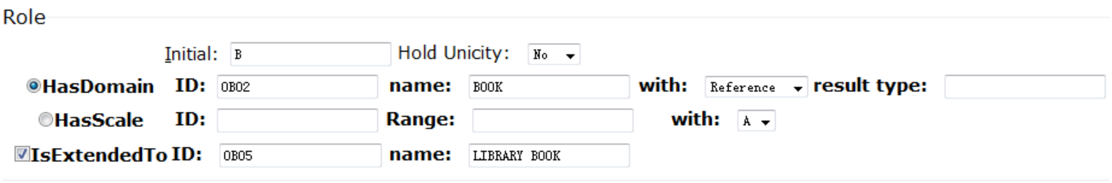
   If “HasDomain” is selected, give the ID and the name of the object class. Depends on what kind of law hold between the role and its domain, choose dependency law or reference law from the select list.

   If there is any result type declared on the object class, give the result type ID.

   If “HasScale” is selected, give the ID and the name of the range. Choose the scale type from the select list.

   If the role is extended to another object class, tick the checkbox “IsExtendedTo”, give the ID and the name of the new object class.

   An exemplary role information is shown in Figure 28.



Role

Initial: B Hold Unicity: No

● HasDomain ID: OB02 name: BOOK with: Reference result type:

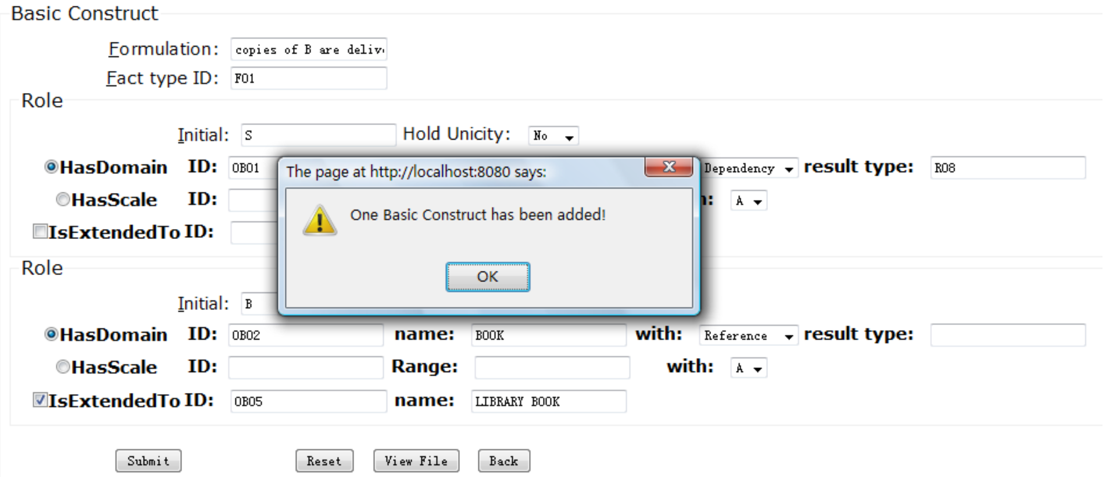
○ HasScale ID: Range: with: A

☑ IsExtendedTo ID: OB05 name: LIBRARY BOOK

Figure 28 Exemplary role information

5. Fill in the role information for another role following the same way as explained in 4.

6. After filling all the necessary information, click button “Add” to save the information in the XML document for SM. You will see the confirmation message as shown in Figure 29.



Basic Construct

Formulation: copies of B are deliv

Fact type ID: F01

Role

Initial: S Hold Unicity: No

● HasDomain ID: 0B01 Dependency result type: R08

○ HasScale ID: A

□ IsExtendedTo ID:

The page at http://localhost:8080 says:

One Basic Construct has been added!

Role

Initial: B

● HasDomain ID: 0B02 name: BOOK with: Reference result type:

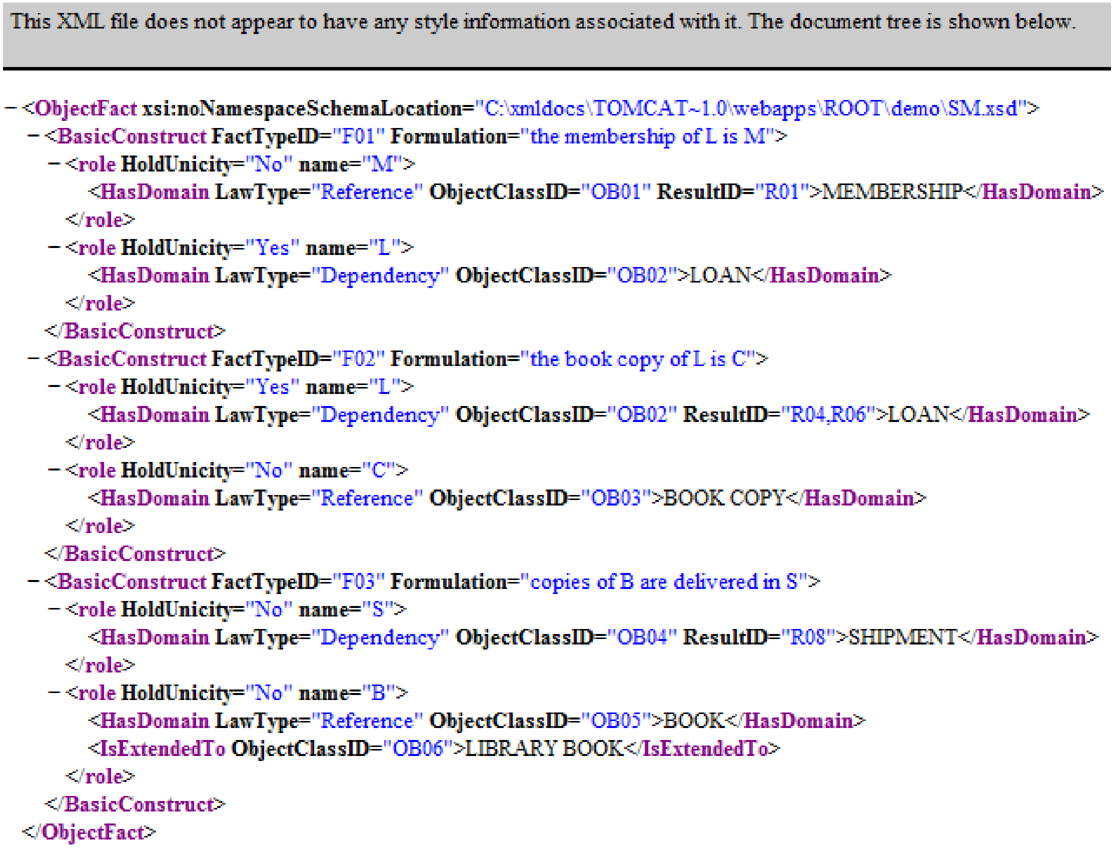
○ HasScale ID: Range: with: A

☑ IsExtendedTo ID: 0B05 name: LIBRARY BOOK

Figure 29 Confirmation message

7. Click “OK” to return to the complete form as shown in Figure 26. Repeat step 2-7 if you need to add more basic constructs, or click button “Back” to return to the home page.

8. After clicking “View State Model”, you will see the exemplary XML document for SM as shown in Figure 30.

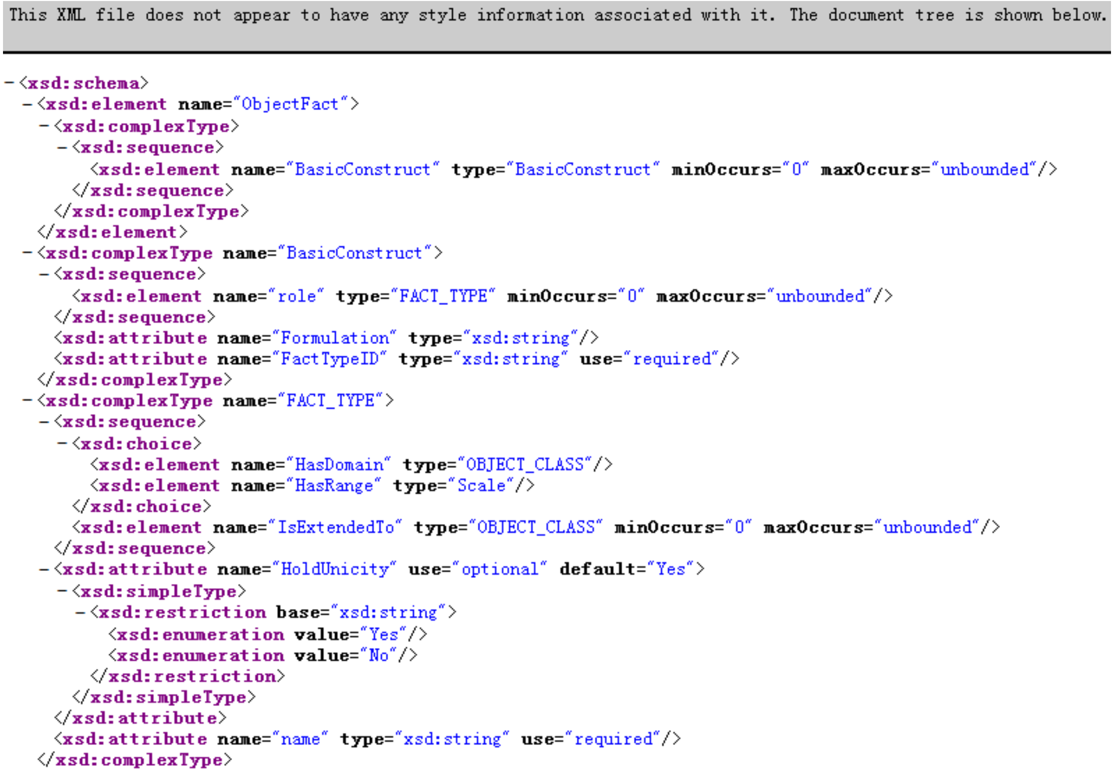


This XML file does not appear to have any style information associated with it. The document tree is shown below.

```
– <ObjectFact xsi:noNamespaceSchemaLocation="C:\xmldocs\TOMCAT~1.0\webapps\ROOT\demo\SM.xsd">
  – <BasicConstruct FactTypeID="F01" Formulation="the membership of L is M">
    – <role HoldUnicity="No" name="M">
        <HasDomain LawType="Reference" ObjectClassID="OB01" ResultID="R01">MEMBERSHIP</HasDomain>
      </role>
    – <role HoldUnicity="Yes" name="L">
        <HasDomain LawType="Dependency" ObjectClassID="OB02">LOAN</HasDomain>
      </role>
    </BasicConstruct>
  – <BasicConstruct FactTypeID="F02" Formulation="the book copy of L is C">
    – <role HoldUnicity="Yes" name="L">
        <HasDomain LawType="Dependency" ObjectClassID="OB02" ResultID="R04,R06">LOAN</HasDomain>
      </role>
    – <role HoldUnicity="No" name="C">
        <HasDomain LawType="Reference" ObjectClassID="OB03">BOOK COPY</HasDomain>
      </role>
    </BasicConstruct>
  – <BasicConstruct FactTypeID="F03" Formulation="copies of B are delivered in S">
    – <role HoldUnicity="No" name="S">
        <HasDomain LawType="Dependency" ObjectClassID="OB04" ResultID="R08">SHIPMENT</HasDomain>
      </role>
    – <role HoldUnicity="No" name="B">
        <HasDomain LawType="Reference" ObjectClassID="OB05">BOOK</HasDomain>
        <IsExtendedTo ObjectClassID="OB06">LIBRARY BOOK</IsExtendedTo>
      </role>
    </BasicConstruct>
  </ObjectFact>
```

Figure 30 Exemplary XML document for SM

9. After clicking “View Schema”, you will see the XML schema of the SM as shown in Figure 31.



```
This XML file does not appear to have any style information associated with it. The document tree is shown below.

- <xsd:schema>
  - <xsd:element name="ObjectFact">
    - <xsd:complexType>
      - <xsd:sequence>
          <xsd:element name="BasicConstruct" type="BasicConstruct" minOccurs="0" maxOccurs="unbounded"/>
        </xsd:sequence>
      </xsd:complexType>
    </xsd:element>
  - <xsd:complexType name="BasicConstruct">
    - <xsd:sequence>
        <xsd:element name="role" type="FACT_TYPE" minOccurs="0" maxOccurs="unbounded"/>
      </xsd:sequence>
      <xsd:attribute name="Formulation" type="xsd:string"/>
      <xsd:attribute name="FactTypeID" type="xsd:string" use="required"/>
    </xsd:complexType>
  - <xsd:complexType name="FACT_TYPE">
    - <xsd:sequence>
      - <xsd:choice>
          <xsd:element name="HasDomain" type="OBJECT_CLASS"/>
          <xsd:element name="HasRange" type="Scale"/>
        </xsd:choice>
        <xsd:element name="IsExtendedTo" type="OBJECT_CLASS" minOccurs="0" maxOccurs="unbounded"/>
      </xsd:sequence>
    - <xsd:attribute name="HoldUnicity" use="optional" default="Yes">
      - <xsd:simpleType>
        - <xsd:restriction base="xsd:string">
            <xsd:enumeration value="Yes"/>
            <xsd:enumeration value="No"/>
          </xsd:restriction>
        </xsd:simpleType>
      </xsd:attribute>
      <xsd:attribute name="name" type="xsd:string" use="required"/>
    </xsd:complexType>
```

Figure 31 XML schema of the SM

Cross-Model Table

1. To start constructing the BCT, please click “Build Bank Contents Table” on the left side in the home page, as shown in Figure 32. Go to 2.

   To view the produced XML document for BCT, please click “View Bank Contents Table” on the right side in the home page. Go to 8.

   To start constructing the IUT, please click “Build Information Use Table” on the left side in the home page. Go to 9.

   To view the produced XML document for IUT, please click “View Information Use Table” on the right side in the home page. Go to 15.

   The XML document for TRT is automatically produced when CM is produced. To view the produced XML document for TRT, please click “View Transaction Result Table” on the right side in the home page. Go to 16.

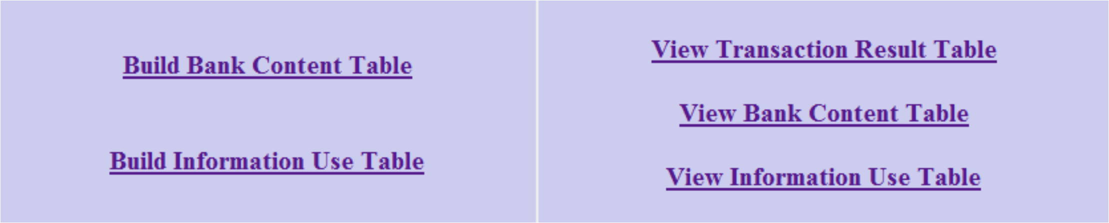
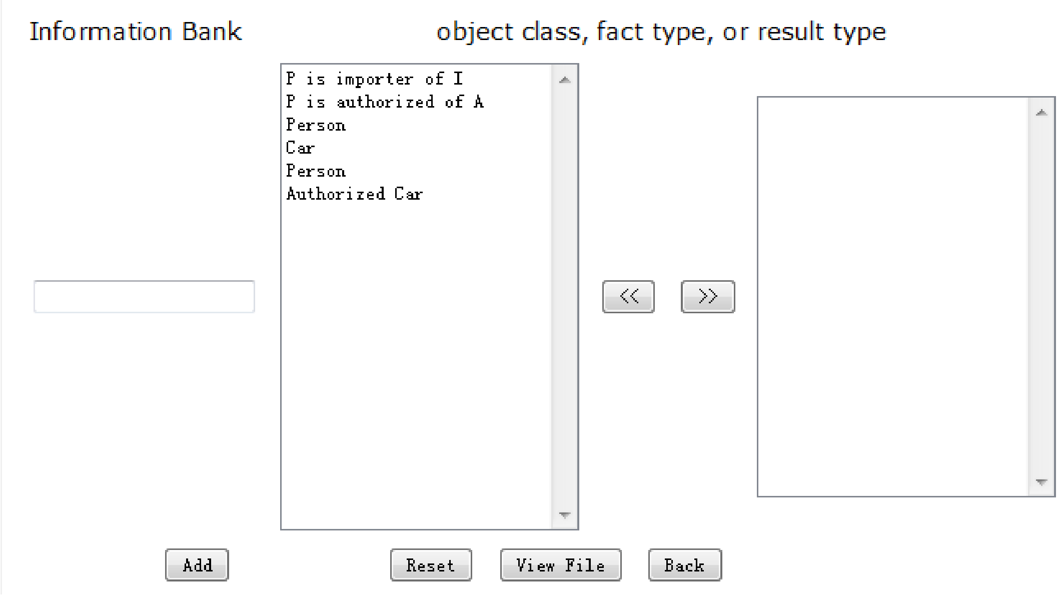

Build Bank Content Table
Build Information Use Table
View Transaction Result Table
View Bank Content Table
View Information Use Table

Figure 32 Cross-Model Table Home Page

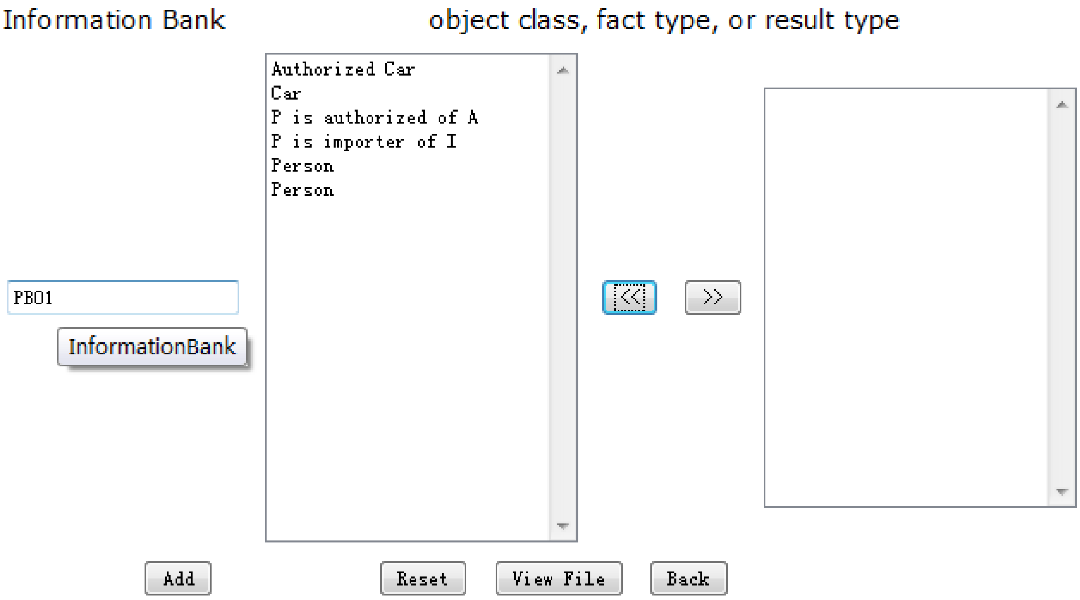
2. In Figure 33, the page for creating the BCT is shown. This form adds one row to the BCT every time. You can always click button “View File” to check what you create in the XML document for BCT.



P is importer of I
P is authorized of A
Person
Car
Person
Authorized Car

Figure 33 Complete form for making BCT

3. Give a name to the information bank in the column “Information Bank”, as shown in Figure 34.



PB01

Authorized Car
Car
P is authorized of A
P is importer of I
Person
Person

Figure 34 Name the information bank

4. The column “object class, fact type, or result type” has two select lists. The left one lists all the information objects which are imported from the XML document for SM. Select the information objects which belong to the named information bank, as shown in Figure 35.

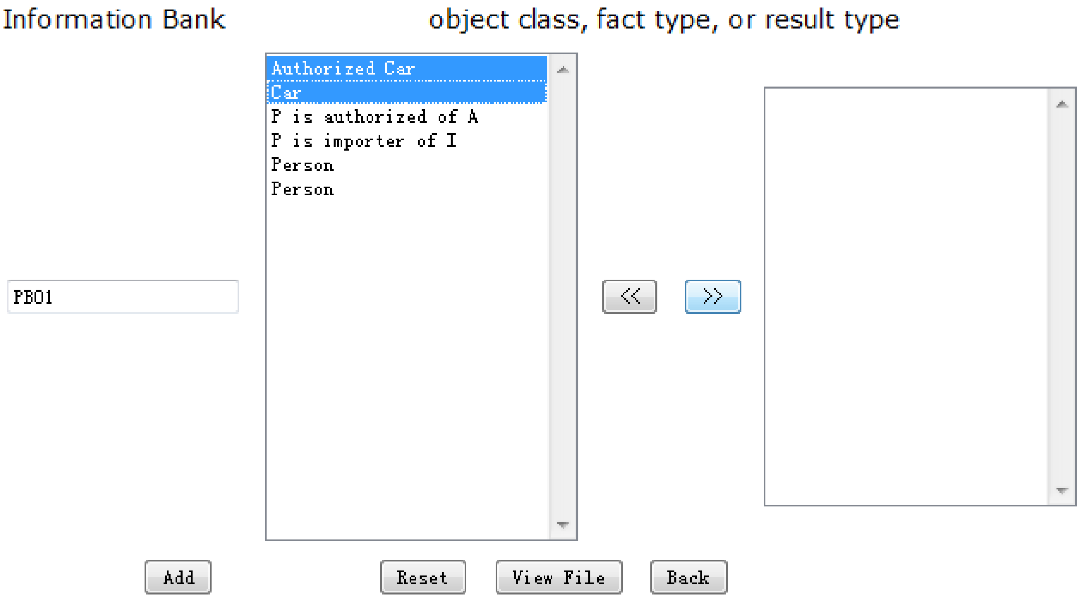


Figure 35 Select the information objects from the left select list

5. Click button “>>” to move the selected information objects in the left select list into the right select list, as shown in Figure 36. Click button “<<” to move the information objects from the right select list to the left select list, if you need to change the chosen information objects.

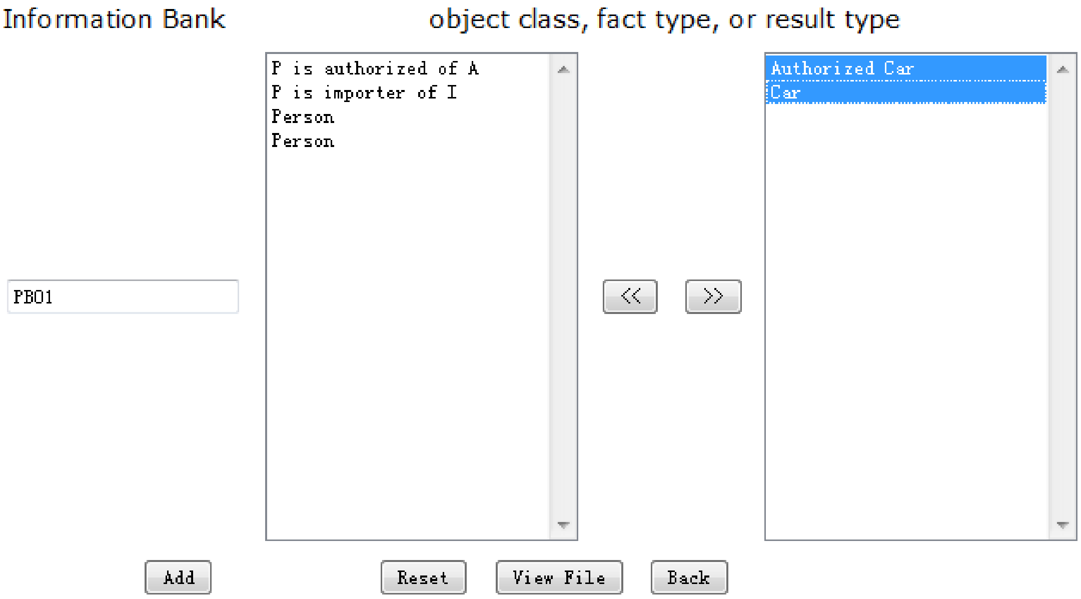
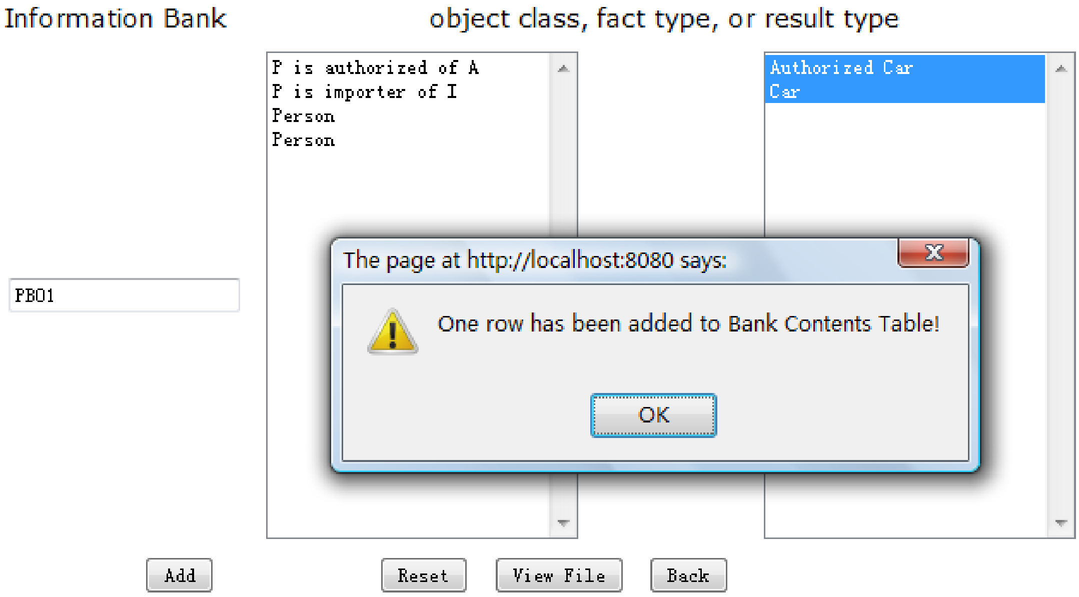


Figure 36 Move the chosen information objects to the right select list

6. First select the information objects in the right select list in the column "object class, fact type, or result type". Then click button "Add" to add the given information into a new row in the BCT. You will see the confirmation message, as shown in Figure 37.



PB01

P is authorized of A
P is importer of I
Person
Person

Authorized Car
Car

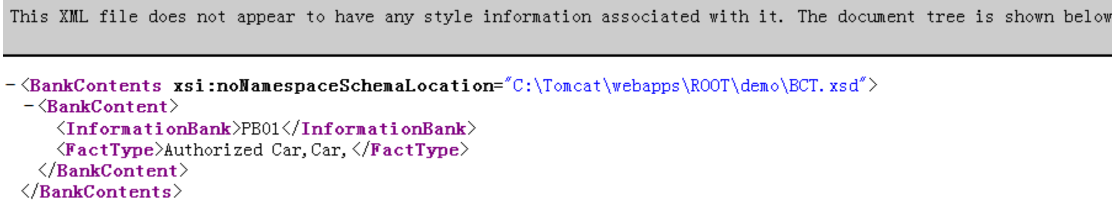
The page at http://localhost:8080 says:

One row has been added to Bank Contents Table!

Figure 37 Confirmation message

7. Click "OK" to return to the form as shown in Figure 33. Repeat step 2-7 if you need to add more rows in the BCT, or click button "Back" to return to the home page.

8. After clicking "View Bank Contents Table", you will see the exemplary XML document for BCT as shown in Figure 38.



This XML file does not appear to have any style information associated with it. The document tree is shown below

```
- <BankContents xsi:noNamespaceSchemaLocation="C:\Tomcat\webapps\ROOT\demo\BCT.xsd">
  - <BankContent>
      <InformationBank>PB01</InformationBank>
      <FactType>Authorized Car,Car,</FactType>
    </BankContent>
  </BankContents>
```

Figure 38 Exemplary XML document for BCT

9. Figure 39 shows the form for making IUT. This form adds one row to the IUT every time. You can always click button “View File” to check what you create in the XML document for IUT.

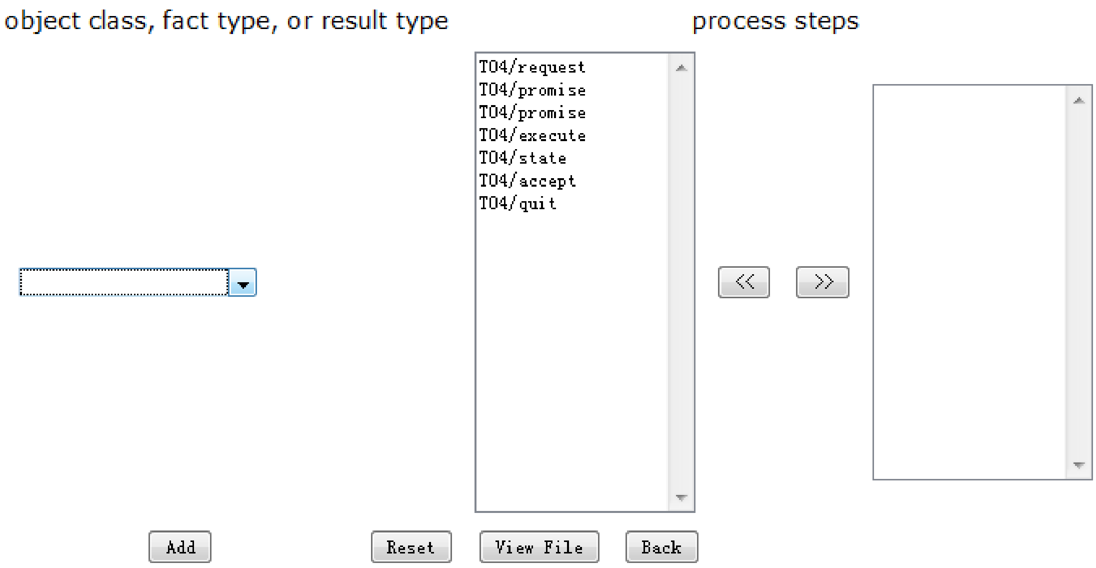


Figure 39 Complete form for making IUT

10. The column “object class, fact type, or result type” lists all the information objects which are imported from the XML document for SM. Select the information object from the select list, as shown in Figure 40.

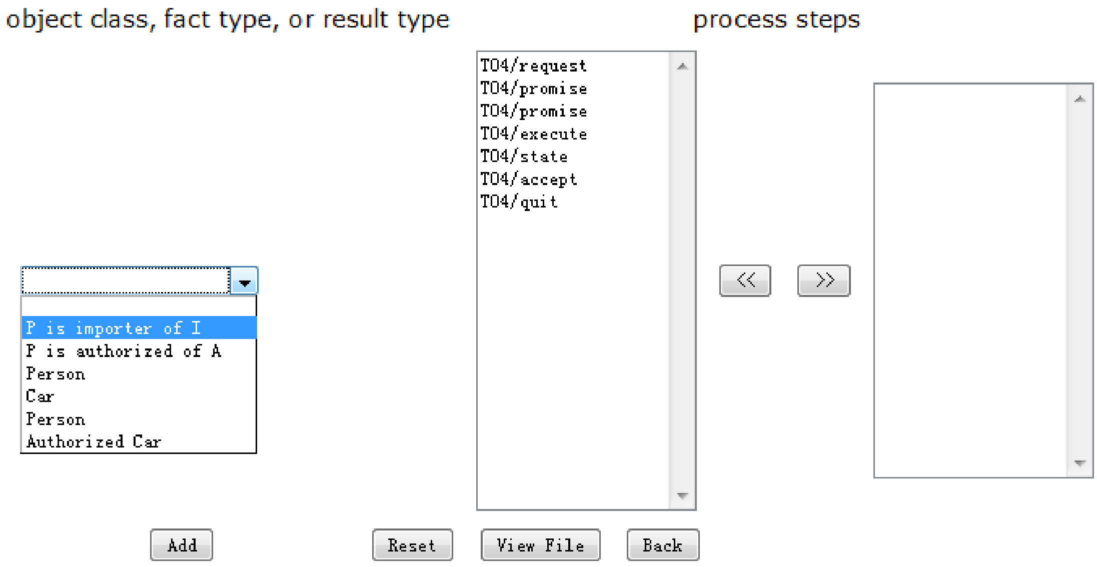


Figure 40 Select information object from select list

11. The column “process steps” has two select lists. The left one lists all the transaction steps which are imported from the XML document for PM. Select the transaction steps in which the selected information object is used, as shown in Figure 41.

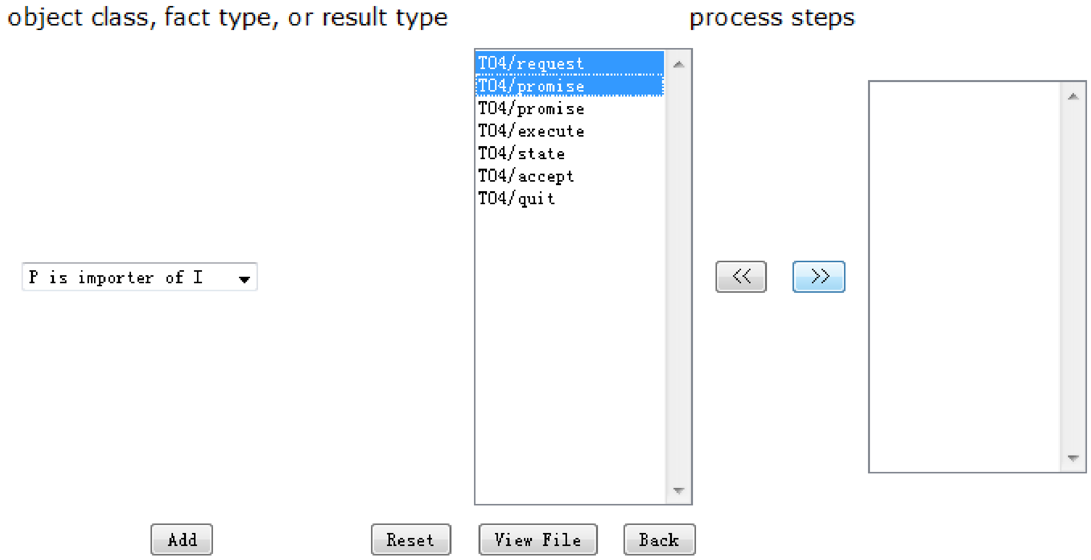


Figure 41 Select the transaction steps from the left select list "process steps"

12. Click button “>>” to move the selected transaction steps in the left select list into the right select list, as shown in Figure 42. Click button “<<” to move the transaction steps from the right select list to the left select list, if you need to change the chosen transaction steps.

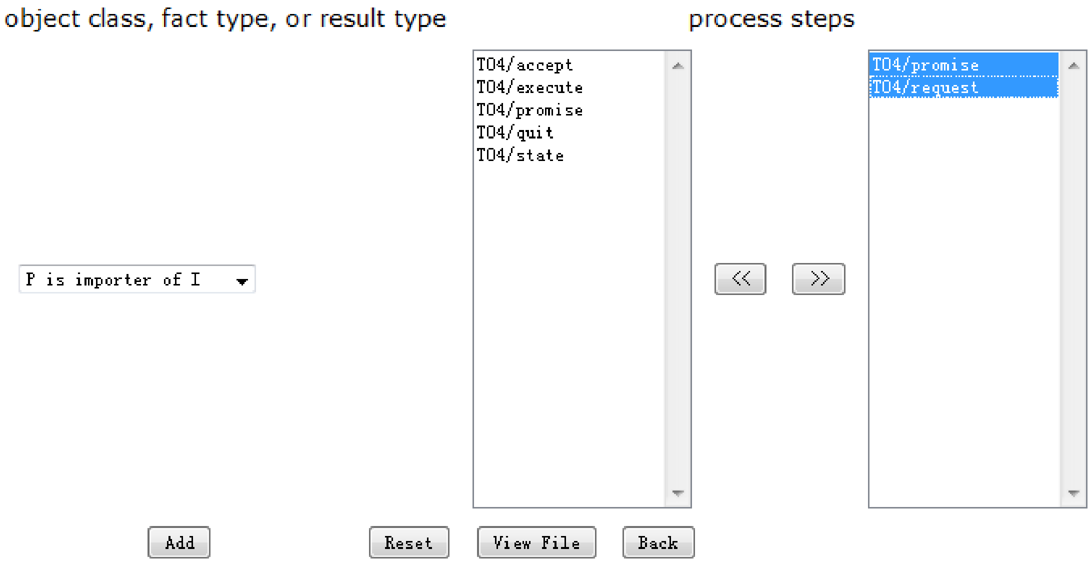
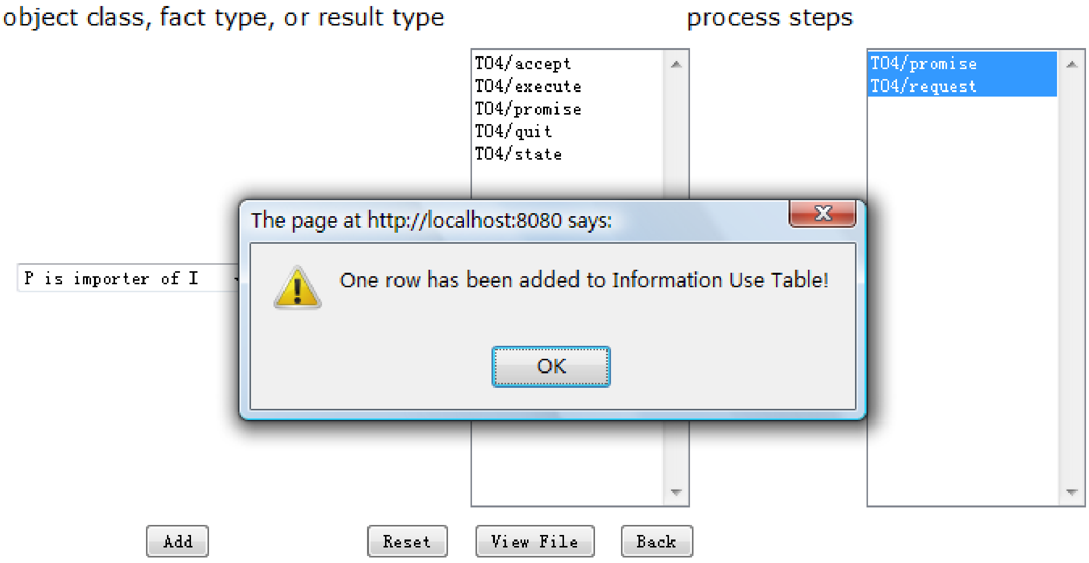


Figure 42 Move the chosen transaction steps to the right select list "process steps"

13. First select the transaction steps in the right select list in the column “process steps”. Then click button “Add” to add the given information into a new row in the IUT. You will see the confirmation message, as shown in Figure 43.



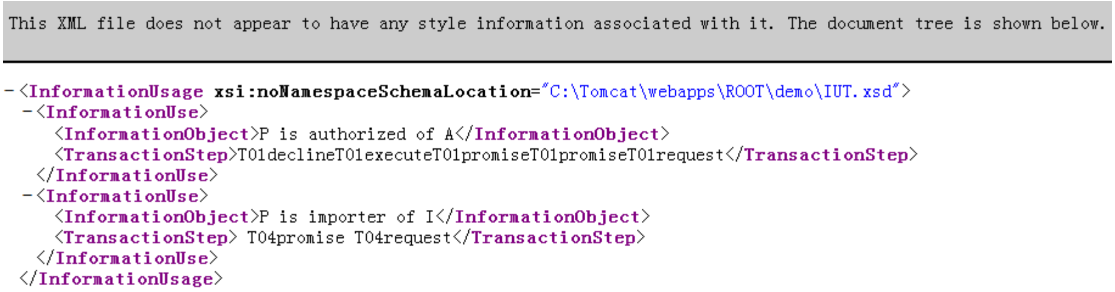
The page at http://localhost:8080 says:

One row has been added to Information Use Table!

Figure 43 Confirmation message

14. Click “OK” to return to the form as shown in Figure 39. Repeat step 9-14 if you need to add more rows in the IUT, or click button “Back” to return to the home page.

15. After clicking “View Information Use Table”, you will see the exemplary XML document for IUT as shown in Figure 44.

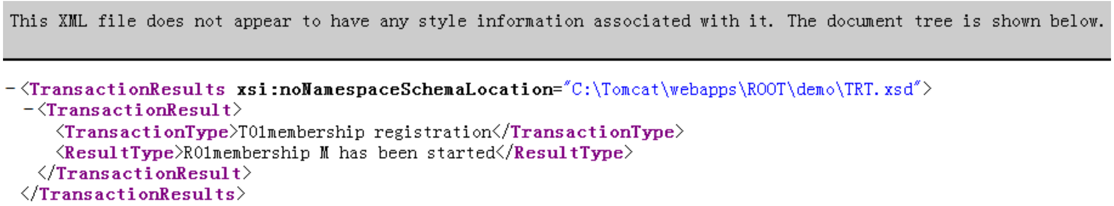


```
This XML file does not appear to have any style information associated with it. The document tree is shown below.

- <InformationUsage xsi:noNamespaceSchemaLocation="C:\Tomcat\webapps\ROOT\demo\IUT.xsd">
  - <InformationUse>
      <InformationObject>P is authorized of A</InformationObject>
      <TransactionStep>T01declineT01executeT01promiseT01promiseT01request</TransactionStep>
    </InformationUse>
  - <InformationUse>
      <InformationObject>P is importer of I</InformationObject>
      <TransactionStep> T04promise T04request</TransactionStep>
    </InformationUse>
  </InformationUsage>
```

Figure 44 Exemplary XML document for IUT

16. After clicking “View Transaction Result Table”, you will see the exemplary XML documents for TRT as shown in Figure 45.



```
This XML file does not appear to have any style information associated with it. The document tree is shown below.

- <TransactionResults xsi:noNamespaceSchemaLocation="C:\Tomcat\webapps\ROOT\demo\TRT.xsd">
  - <TransactionResult>
      <TransactionType>T01membership registration</TransactionType>
      <ResultType>R01membership M has been started</ResultType>
    </TransactionResult>
  </TransactionResults>
```

Figure 45 Exemplary XML document for TRT

Create / Use Table

1. To start constructing the Create / Use Table, please click the diagram on the left side in the home page, as shown in Figure 46. Go to 2.

   To view the created Create / Use Table, click “Create / Use Table”. Go to 9.

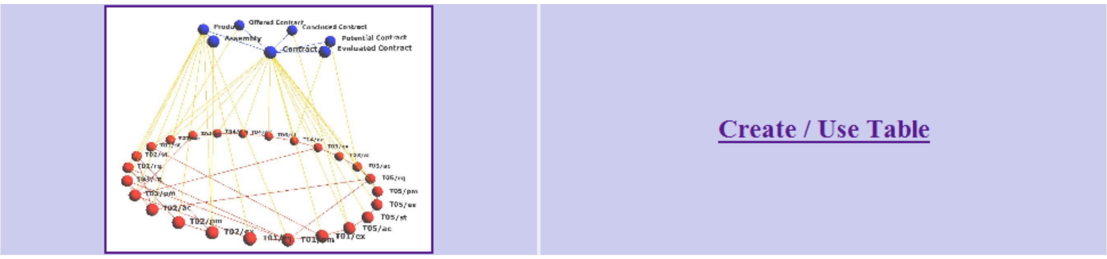


Figure 46 Create / Use Table home page

2. The form for creating the Create / Use Table is shown in Figure 47. This form adds one row to the Create / Use Table every time.

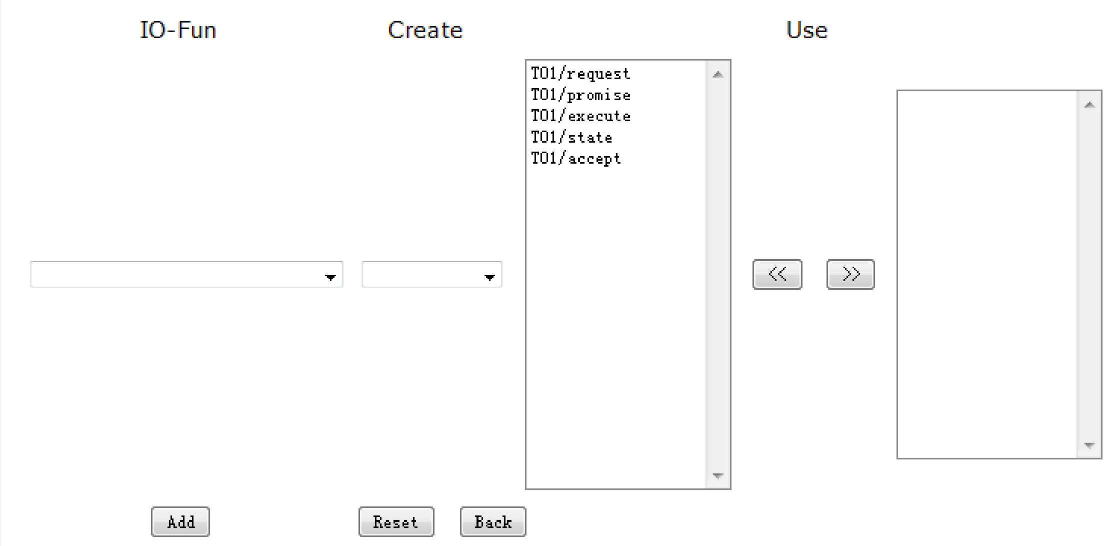


Figure 47 Complete form of making create / use table

3. The column “IO-Fun” lists all the information objects which are imported from the XML document for SM. Select the information object from the select list, as shown in Figure 48.

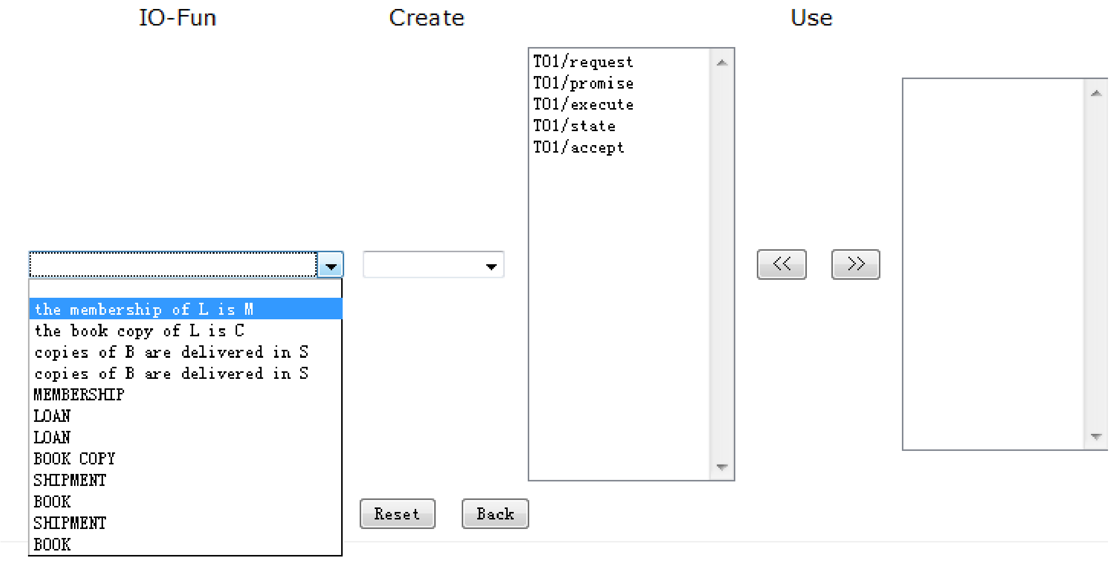


Figure 48 Select information object from the select list “IO-Fun”

4. The column “Create” lists all the transaction steps which are imported from the XML document for PM. Select the transaction step in which the chosen information object in column “IO-Fun” is created from the select list “Create”, as shown in Figure 49.

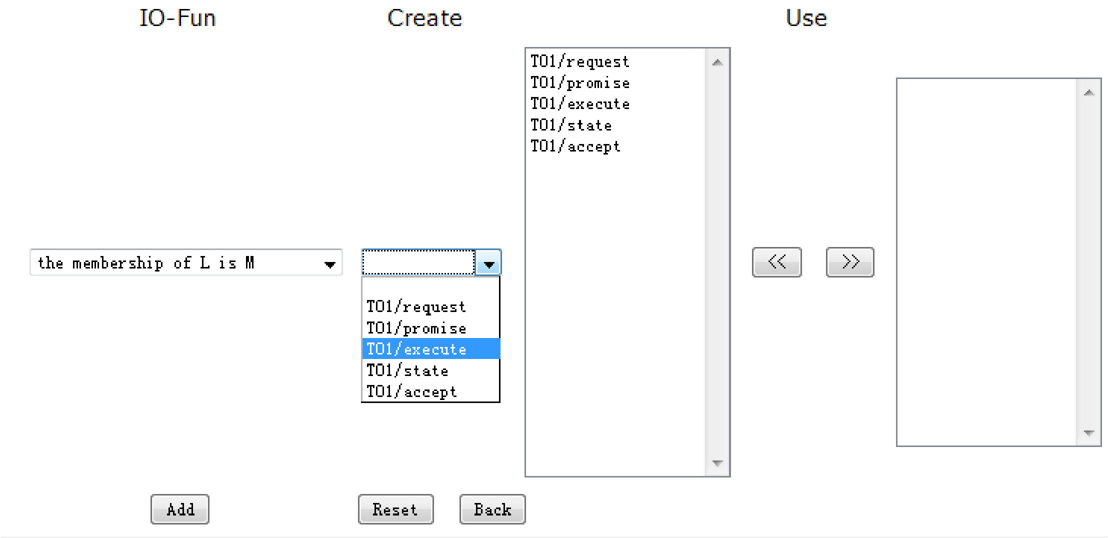
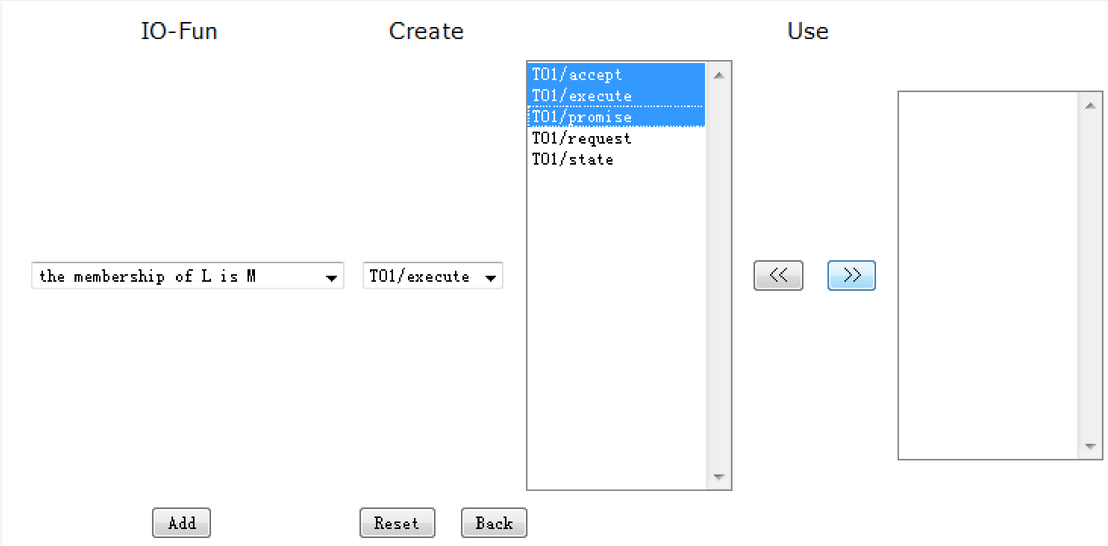


Figure 49 Select the transaction step from the select list “Create”

5. The column “Use” has two select lists. The left one lists all the transaction steps which are imported from the XML document for PM. Select the transaction steps in which the selected information object is used, as shown in Figure 50.


IO-Fun: the membership of L is M

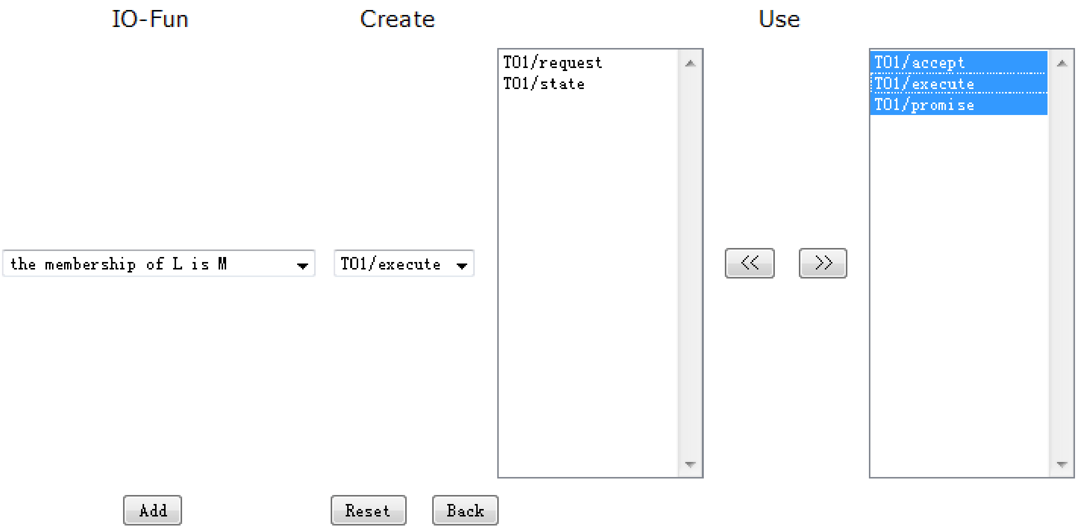
Create: T01/execute

Use (left list): T01/accept, T01/execute, T01/promise, T01/request, T01/state

Use (right list): empty

Figure 50 Select the transaction steps from the left select list “Use”

6. Click button “>>” to move the selected transaction steps in the left select list into the right select list, as shown in Figure 51. Click button “<<” to move the transaction steps from the right select list to the left select list, if you need to change the chosen transaction step.


IO-Fun: the membership of L is M

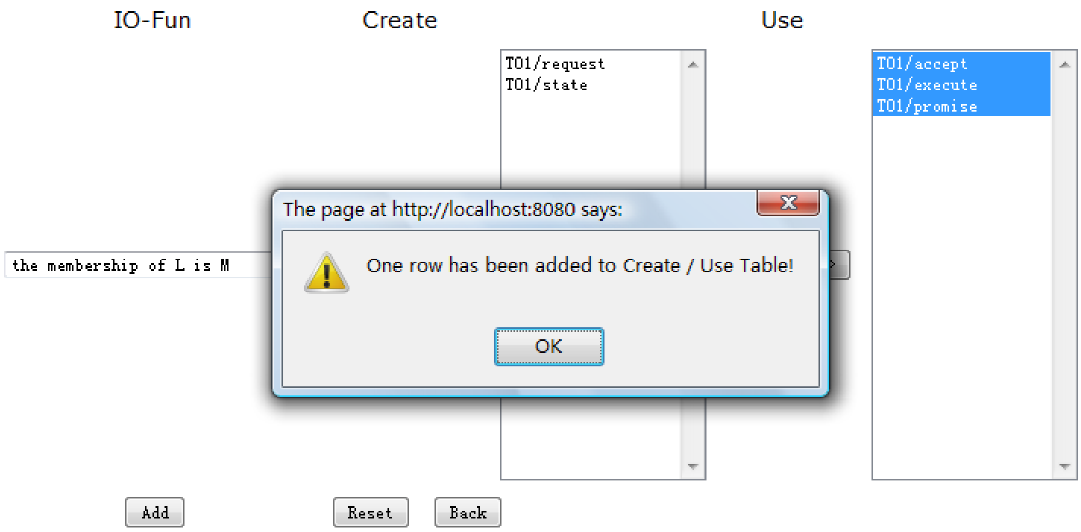
Create: T01/execute

Use (left list): T01/request, T01/state

Use (right list): T01/accept, T01/execute, T01/promise

Figure 51 Move the chosen transaction steps to the right select list “Use”

7. First select the transaction steps in the right select list “Use”. Then click button “Add” to add the given information into a new row in the Create / Use Table. You will see the confirmation message, as shown in Figure 52.



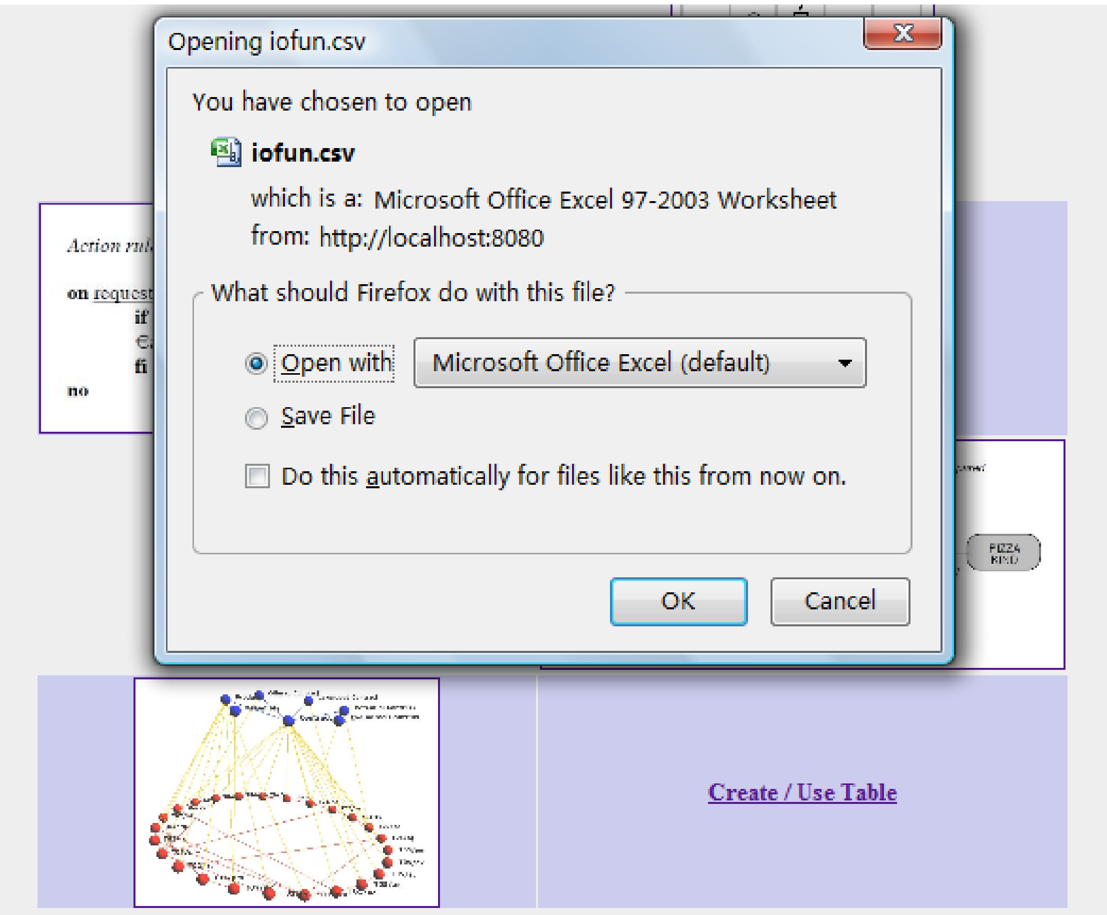
The page at http://localhost:8080 says:

One row has been added to Create / Use Table!

Figure 52 Confirmation message

8. Click “OK” to return to the form as shown in Figure 47. Repeat step 2-8 if you need to add more rows in the Create / Use Table, or click button “Back” to return to the home page.

9. After clicking “Create / Use Table”, you will see the produced create / use table. Click “ok” to open the csv file for view.



Opening iofun.csv

You have chosen to open

iofun.csv

which is a: Microsoft Office Excel 97-2003 Worksheet
from: http://localhost:8080

What should Firefox do with this file?

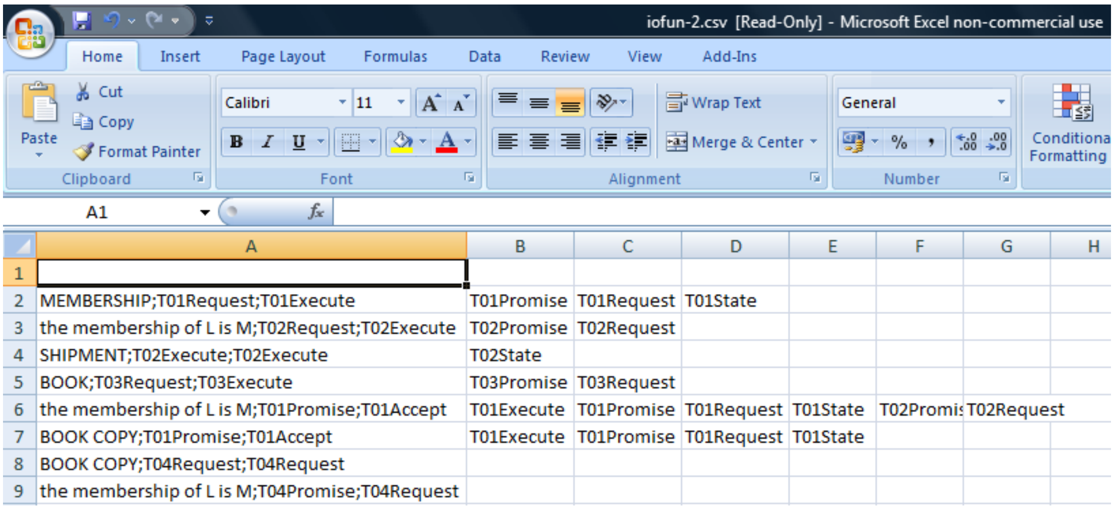
● Open with Microsoft Office Excel (default)

○ Save File

□ Do this automatically for files like this from now on.

Figure 53 View the Create / Use Table

10. An exemplary create / use table is shown in Figure 54.



iofun-2.csv [Read-Only] - Microsoft Excel non-commercial use

	A	B	C	D	E	F	G	H
1								
2	MEMBERSHIP;T01Request;T01Execute	T01Promise	T01Request	T01State				
3	the membership of L is M;T02Request;T02Execute	T02Promise	T02Request					
4	SHIPMENT;T02Execute;T02Execute	T02State						
5	BOOK;T03Request;T03Execute	T03Promise	T03Request					
6	the membership of L is M;T01Promise;T01Accept	T01Execute	T01Promise	T01Request	T01State	T02Promi	T02Request	
7	BOOK COPY;T01Promise;T01Accept	T01Execute	T01Promise	T01Request	T01State			
8	BOOK COPY;T04Request;T04Request							
9	the membership of L is M;T04Promise;T04Request							

Figure 54 An Exemplary create / use table