UNIVERSITY OF OSLO
Department of Informatics

MODI framework - A model-based approach to data integration

Master thesis

Mohammad Asaf Khan

Khudija Mahmood

30 July 2005

Acknowledgement

This master thesis is submitted in fulfilment of the Master degree in Informatics at the Department of Informatics, University of Oslo, 2005. The work on this thesis was done at SINTEF, Department of Information and Communication Technology (ICT), Cooperative and Trusted Systems.

We would like to thank our supervisor Arne-Jørgen Berre for his guidance and patience. We would also like to give many thanks to Ida Solheim for providing so much help about writing style and motivated us to work hard. Also, we are very grateful for help from Andreas Limyr and Tor Neple. In addition, we would like to thank our contact person Jeanine Lilleng, who has contributed with information concerning the project case NDR. Further, we thank SINTEF for giving information about the ATHENA project and a place to work on this thesis. Lastly we would like to thank our families for support and patience through this period.

Abstract

In this thesis we propose a model-based approach to support data integration between heterogeneous enterprise systems. It reviews literature about interoperability, and presents several aspects of data integration problems. Further, it intends to give the reader an understanding of model-driven development which offers different standards for modeling and model transformation. The work of this thesis presents difficulties encountered in data integration by analysing problem examples. Based on the analysis, data integration problems are defined. We examine technologies related to interoperability, data integration and mapping. In addition, we present existing solution approaches to deal with the problem examples. The main goal is to specify how to develop tools for solving data integration problems by describing and realizing mapping between models. The technique which is specified to realize the mapping is presented in our proposed solution, which we have called the MODI Framework.

Table of contents

List of figures

List of tables

1 Introduction

Nowadays, enterprise information systems have a growing need to respond more effectively to changing market conditions and new emerging technologies. For this reason, enterprises have for the past years increasingly been looking for opportunities to utilize innovative Internet technologies to improve communication and collaboration in providing information and services. The interest in system interoperability is driving the continuous need for integration of new, legacy and evolving systems, particularly in the context of networked businesses and e-Government.

While enterprises are trying to move to this arena, they are often hindered by their large, heterogeneous, distributed and evolving information systems. These systems are typically legacy systems that are highly complicated, time-consuming and expensive. In spite of this hinder, some enterprises have made a significant contribution to productivity and inventory control when collaborating electronically without redesigning their systems. Unfortunately, integration with newer systems is difficult because new software may use completely different technologies. Furthermore, it is complex, time-consuming and costly to implement proprietary converting solutions. The proprietary formats generally have quite different syntaxes, structure and semantics to process the same information, which makes it hard to integrate data. Before enterprise systems can integrate data they need to support mutual understanding of shared information through interoperability.

1.1 Interoperability – a review

Interoperability, in a general sense, refers to “the ability of two or more systems or components to exchange information and to use the information that has been exchanged” [1]. It requires compatibility between the communicating systems, on formats and application domain concepts, to enable correct interpretation of transferred data.

1.1.1 Levels of interoperability

Interoperability at different levels is needed to integrate enterprise systems. Figure 1-1 shows how interoperability between two enterprise systems can be achieved on different levels of abstraction and complexity; namely organisational-, business- and technical level [2, 3]. An important fact is that these levels are interdependent, where each level depends on a lower level being functional.

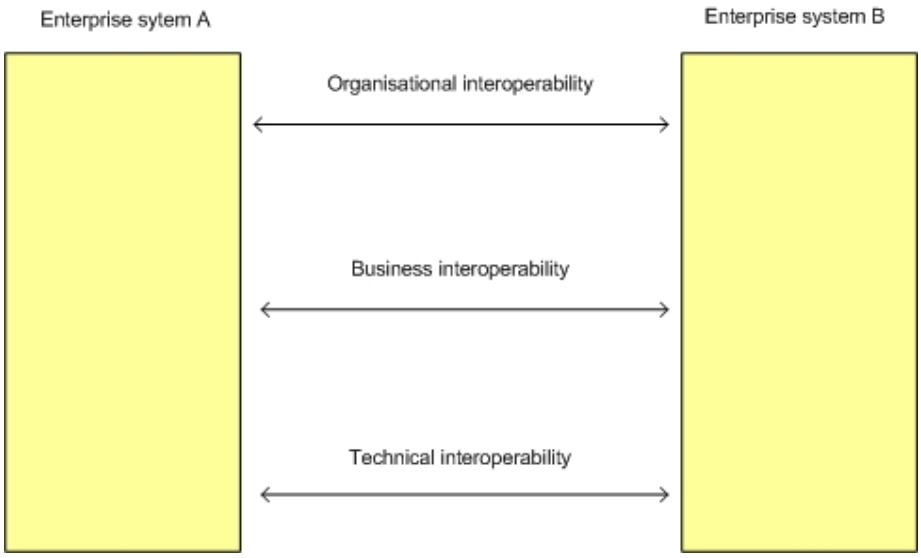


Figure 1-1 Different levels of interoperability

Organisational interoperability

This level of interoperability deals with organisational processes, goals, objectives and how they interoperate through business services. Organisational interoperability is concerned with enabling the collaboration of organisations that wish to exchange information and may have different internal structures and processes.

Business interoperability

This level deals with business services, processes and objects. As illustrated in Figure 1-2, business interoperability is concerned with bringing about collaboration of enterprises' from different aspects. The figure shows interaction between one business service, but the enterprises can have more business services.

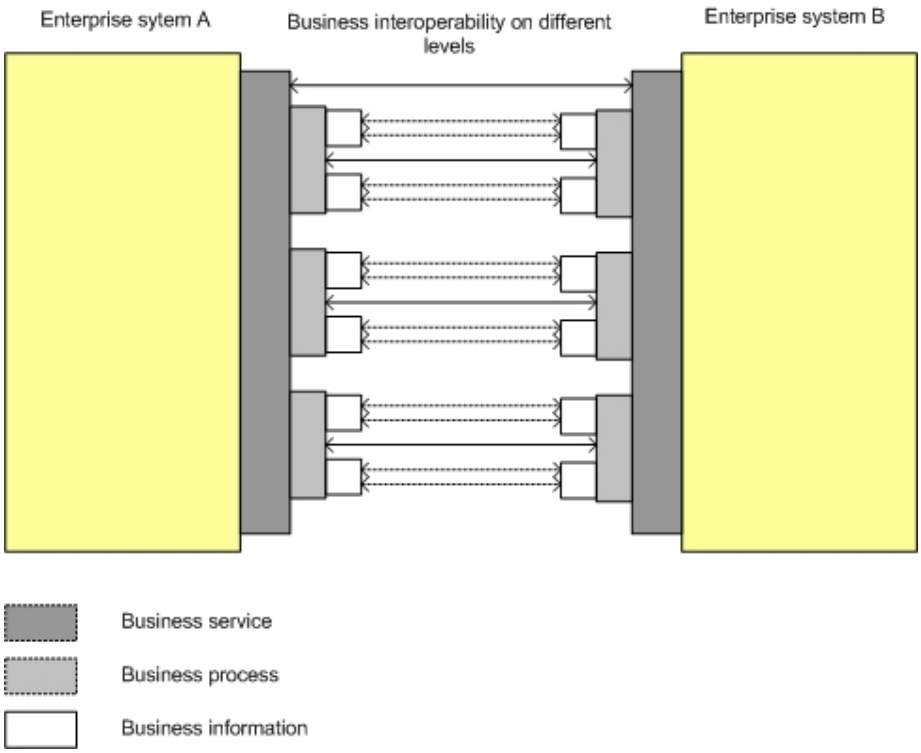


Figure 1-2 Business interoperability

- *Service interoperability* deals with achieving interoperability between different enterprise systems' business services. Services can be seen as an abstraction of functionality encapsulated and provided by an autonomous entity. Typically these services are provided through interfaces and contracts guiding their usage and behaviour.

- *Process interoperability* deals with comparing and integrating business processes. Syntactic, structural and semantic differences need to be taken into account when comparing these. Processes describe sequencing of work in terms of actions, control flows, information flows, interactions, protocols etc. They can be applied to business aspects as well as technical aspects.

- *Information interoperability* deals with comparing and integrating enterprise systems' data. Similar to process comparing syntax, structure and semantics of the data needs to be taken into account. This is because data can be represented in many ways at different enterprise systems. Comparing syntax focuses on the representation of data to be exchanged. However, semantic comparison centres on the meaning of data to be exchanged. The aim is to make the precise meaning of exchanged data understandable by any enterprise system supporting other semantic notations.

Additionally, *Non-Functional Aspects* (NFA) [2] needs to be considered for collaboration between enterprise systems. NFA are driven by need for separation of concerns. These aspects include quality properties such as:

- *Security* describes a solution's ability to protect enterprise resources and control access to them, including authentication, authorization, and data encryption.
- *Scalability* refers to a solution's ability to adjust to an increased number of business tasks.
- *Evolution* refers to the ability of the system to react to changing requirements. E.g. when new functionality is required existing software often needs to be upgraded as a whole. Alternatively, only those components could be exchanged that are affected by required changes. A solid architecture of the system is required.
- *Performance* refers to a solution's ability to rapidly execute a business task and to retrieve and return information in a timely manner.
- *Availability* is a solutions availability to be accessible.
- *Portability* refers to a solution's ability to be used on different hardware platforms, operating systems, and run-time environments with little modifications of the solution.

Technical interoperability
This level deals with linking computer systems and services. Some examples are middleware, open interfaces, interconnection services, data presentation and exchange, accessibility and security services. Technical interoperability makes it possible for computers to exchange signals.

1.1.2 Interoperability problem

Enterprise systems often use different syntax, structure and semantic to represent their data. This becomes a problem when these enterprise systems want to collaborate electronically. The interoperability problem may be considered from various aspects and on increasing levels of complexity. Definition of service, process and information for different enterprise systems can differ from each other in different ways.

Enterprise systems may be poorer at semantic definition, than syntax and structure. Examples are description of what a service does, how well the service works, how the service is carried out, which processes it contains etc. may be missing or insufficient. This can be referred to as service interoperability problems.

Another example is differences in processes that may cause process interoperability problems. First of all, one process may require a set of activities to be carried out in sequence, while another similar process allows them to be carried out in parallel.

Second, one process may require an acknowledgement message, while a similar process does not. Third, one process may send and receive complex messages in one single activity, while a similar process divides the message between several activities.

Further, the information provided by the process, such as arguments may be defined differently. One of the information interoperability problems which have been investigated for the past years is integration of heterogeneous data [4]. Data integration problems occurs when there is disagreement about data, such as the data's meaning, representation and structure among enterprise systems.

Requirements to any interoperability solution would be to deal with these problems. However, the core of our research is information interoperability dealing with data integration. Service and process interoperability, and NFA are left for further work.

1.1.3 Complexity of interoperability

In recent years much technical and scientific work has been committed to solve information interoperability problems, and suggests how interoperability can be addressed in different ways. In the field of interoperability for enterprise applications and software, interesting results have been produced [5]. Another important area is represented by *Enterprise Application Integration* (EAI) [6, 7]. Important results have been achieved in the area of databases, aiming at the integration of heterogeneous data [4]. The complexity of interoperability lies in synchronizing heterogeneous enterprise systems, typically built at different times, by different people, usually by means of different technologies.

Figure 1-3 shows two compatible enterprises systems using the same set of format, and execution platforms.

Figure 1-3 Enterprise systems using same set of format and execution platforms

However, this is not the situation for those enterprises systems that use proprietary formats. A challenge is to make collaboration possible without requiring enterprise systems to modify their software or their data organisation. The next two solutions consider the case where two enterprise systems need to exchange information with different formats.

One solution deals with enterprises utilizing different solutions than others. An example is by providing a piece of software such as an adapter which in principle is

able to transform data produced by one enterprise system in the format required by the other enterprise system. There are many disadvantages with this solution [8]. Firstly, it is technically difficult to build adapters. An adapter is complicated since it requires a complete understanding of the data organisation within two enterprise systems. Unfortunately, there is not often sufficient knowledge about the data organisation such as the semantics of data. For data to be correctly transformed and interpreted, knowledge of semantic data is necessary. Secondly, this solution is suitable in an environment involving only a few cooperating partners. However, in the case where more and more enterprise systems become involved, this approach becomes complex. To maintain as many different solutions to communicate as there are enterprises involved is inefficient, and leads to high costs. Given N systems that need to cooperate, it needs to be developed $N^2 - N$ adapters. This solution is referred to as point-to-point, and illustrated in Figure 1-4. The squares to the left and right shows two enterprise systems, and the square in the centre shows an adapter.

Figure 1-4 Point-to-point solution

Another solution which reduces the development of adapters is the case where enterprise systems adopt the same set of agreements for interoperability solutions, e.g. by using a middleware. This solution is represented by the definition of a common interchange format or standard which is to be imposed to every enterprise system involved. Further, each of the involving enterprises can get benefits of a single solution that needs to be developed only once. A drawback is that it can for many reasons be difficult for large enterprises to standardise on a single middleware platform [9]. The difficulty includes differing requirements in different departments, mergers, interoperability with customer and suppliers, and *Business-to-Business* (B2B) markets. This solution is shown in Figure 1-5. The outer squares show enterprises, and the centre square represents the same set of agreements for interoperability solutions. The inner squares, between the outer and centre squares, refers to conversion from proprietary format to same set of agreement.

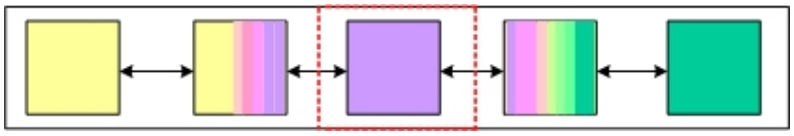


Figure 1-5 Enterprises that adopt same set of agreements

Enterprises need a way to maintain middleware flexibility. We consider an approach where interoperability solutions should be driven by business needs first and software solutions second.

An area that addresses the problem of interoperability in distributed developing environments, is the *model-driven development* (MDD) and in particular the *Object Management Group's* (OMG) [10] *Model-Driven Architecture* (MDA) [11, 12]. OMG is a non-profit organisation established in 1989. Its mission is to help computer users solve integration problems by supplying open, vendor-neutral interoperability specifications. MDD is an architectural business-driven approach for developing software systems based on requirements derived from enterprise and business models. In MDD models are the prime artefacts. Essentially meaning, models are in use from the early capture of user requirements to the production of executable code. Model reuse is essential and also model transformation, which preferably should be automated. MDA can be seen as a specific implementation of MDD with respect to software systems development. According to [13] MDD can contribute with model-driven information integration by addressing MDA. MDA provides an approach that separates what systems must do from how it is implemented.

1.2 Model Driven Architecture (MDA) to facilitate interoperability

MDA is "an approach to using models in software development" [14] and aims to provide a platform-independent approach to domain-specific application development. It promotes the creation of software systems through modeling machine-readable highly abstract models and model transformation. These models are developed independently of the implementation technology and stored in standardized repositories. The strength of storing models in repositories is their repeated accessibility and ability to be transformed automatically by tools into schemas, code skeletons, test harnesses, integration code and deployment scripts for different platforms. Models are no longer merely used as a sketch before starting to code on a software project. Instead, the models are understood by computers enabling them to be consistent with the code at all times during the project. MDA integrates what has been built, with what is being built and what will be built in the future.

The MDA approach promotes to create good designs that cope with multiple-implementation technologies and extended software lifetime. Figure 1-6 shows MDA's three main parts and is taken from [15]. The core of MDA is shown in the centre of this figure which includes widely-used OMG modeling standards: *Unified Modeling Language* (UML) [16], *Meta Object Facility* (MOF) [17] and *Common Warehouse Metamodel* (CWM) [18]. The modeling language UML has in the recent years outgrown its initial purpose as a standard notation for constructing models of object-oriented software. UML allows an application model to be constructed, viewed, developed, and manipulated in a standard way at analysis and design time. Just as blueprints represent the design for an office building, UML models represent the design for an application. In MDA, UML is used for visualizing, storing, and

exchanging software design and models. MOF is a model-driven framework for specifying, constructing, managing interchange and integrating metadata in software systems. It represents metamodels and how to manipulate them. In addition, it has a repository service for storing abstract models used in distributed object-oriented software development. Also, it is a metamodeling language for the rapid specification, construction and management of domain-specific technology-neutral modeling languages. CWM is a specification that describes metadata interchange among data warehousing, business intelligence, knowledge management and portal technologies.

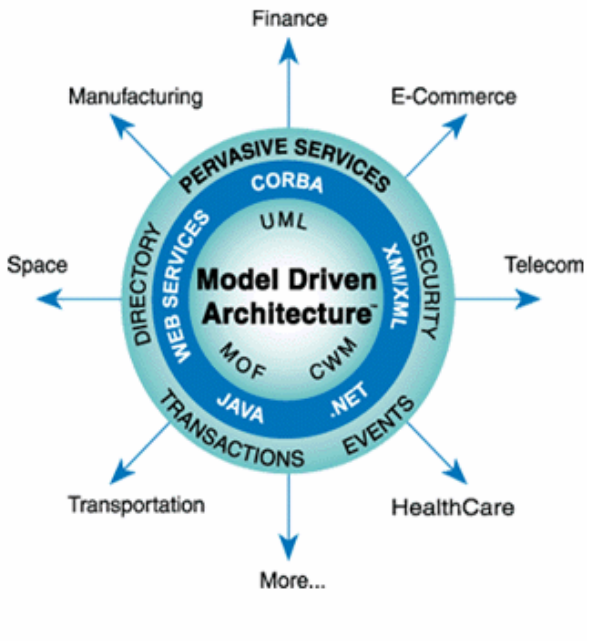


Figure 1-6 OMG's Model Driven Architecture

The next circle includes the proprietary target platforms which are current targets of MDA. These are CORBA, JAVA, .NET, XMI/XML and Web-based platforms. The outermost circle shows the pervasive services that are common for all enterprise systems regardless of what platforms they are based on. These services are directory, transactions, events and security. The arrows indicate that MDA can be used in many market places.

The MDA defines an architecture for models which provides a set of guidelines for structuring specifications which are expressed as models. In the MDA development life cycle, models that can be understood by computers are created: *Platform Independent Model* (PIM) and *Platform Specific Model* (PSM).

1.2.1 Platform Independent Model (PIM)

PIM is a model with a high level of abstraction defined in UML. It specifies services and interfaces independent of software technology platforms. A PIM looks at the enterprise system from the viewpoint of how it can best support the enterprise. It is concerned with modeling business processes and functionality on a platform-independent level. For example a PIM may allocate several logical business objects to one software component. These models are computational in that they may be converted into executable software. The PIM may incorporate decisions regarding distribution of components to meet performance and security requirements. Additionally, an MDA application can be produced on multiple middleware platforms from a single PIM.

1.2.2 Platform Specific Model (PSM)

In the same way as PIMs are constrained by platform-independent UML profile, PSMs are constrained by profiles specific to the technologies they represent, such as UML profile for CORBA. A PSM adds more details to a PIM. The PSM adheres to constraints and conventions imposed by a specific software technology platform, such as CORBA, J2EE or Web Services. The PSM stands relatively close to the actual code, e.g. Java code.

1.2.3 Mapping and transformation

One of the core characteristics in MDA is mapping of models. The mapping process uses a set of rules and techniques to modify one model to obtain another. When transforming from one model to another, mapping is used at several occasions. Figure 1-7 shows the MDA metamodel description which illustrates various mappings and is taken from [19].

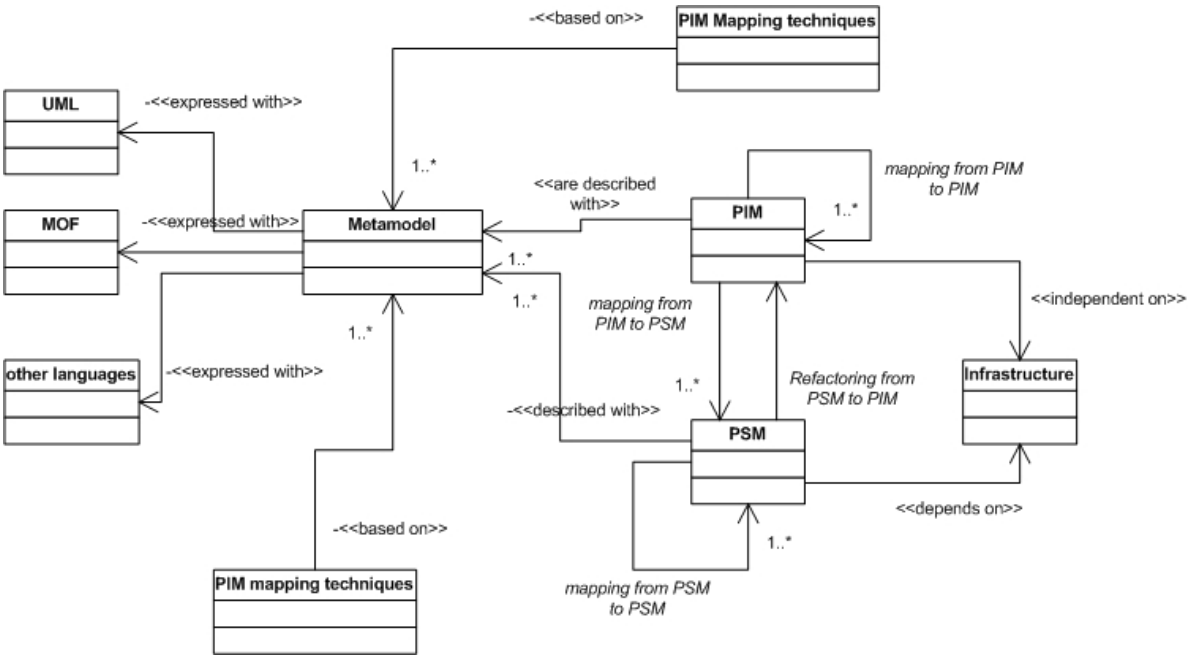


Figure 1-7 MDA metamodel description

Mappings are used for transforming of models from:

PIM to PIM: Transformations between these models at this level are related to model refinement, filtering of the model, omitting platform dependent issues. PIM to PIM mapping is an iterative process independent of platform details. In each iteration the generated output model contains more details about the problem domain than the one in the previous iteration. For example some details are abstracted in the analysis model, but are elaborated in the design model.

PIM to PSM: This transformation is used when the PIM is refined enough with complete details and has to be projected to some specific technology platform. For example, a mapping can be transforming from a logical model to a specific platform like CORBA. PIM to PSM mapping is also an iterative process, but dependent on platform specific details.

PSM to PSM: This transformation deals with model refinement during realization and deployment of components. An example for this transformation is the selection of services and preparation of their configuration.

PSM to PIM: This transformation is concerned with reverse engineering operations. These transformations are needed to abstract models from existing implementations in a specific technology into a PIM.

1.2.4 Integrating legacy systems

In addition to the MDA approach, the OMG define an approach which specifies how to integrate and modernise existing legacy systems according to new business needs. This is a reverse engineering approach known as the *Architecture-Driven Modernization* (ADM) [20]. It allows any legacy system based on a UML model and a supported middleware platform to be included in an enterprise's circle of MDA interoperability. In particular, ADM aims at assessing and synthesizing several MDA related standards for the purpose of mining legacy systems, recovering their architecture, identifying inconsistencies in them. Also, migrating them into new, revitalized system.

1.2.5 MDA – a middleware

It is usual that enterprises typically define computing standards in a specific technology. This is necessary to guarantee interoperability, but requires every enterprise to use the same middleware. Another disadvantage is the case where enterprises advances and the chosen middleware platform are superseded, the standard and all of its users are forced to change to something new. By defining standards in the MDA, enterprises avoid both of these severe disadvantages. Their standard can be implemented equivalently and interoperable on multiple middleware platforms by defining their business services and interfaces as a PIM. Over time, if one or some of these platforms become obsolete, the enterprise can define new implementations on new platforms from the original PIM.

1.2.6 MDA tools

There exist MDA-oriented tools that are available. Certain tools are pure code generation tools and others are more completely developed model-driven tools. UML tools can also be thought of as MDA tools. Examples of these kinds of tools are: OptimalJ, UMT, ATL, MOFScript. For a more detailed description of these and other related tools, see [21].

1.3 Goal of this thesis

The goal for this thesis is to outline a model-based approach to data integration with main emphasis on how to integrate heterogeneous data from one enterprise's format into another enterprise's format with aid of models. Further, to provide syntactic, structure and semantic integration of data.

The enterprises should have common understanding of the data to be exchanged. With common understanding enterprises can more easily do business, and more efficient collaboration with several business partners without being concerned about who is

using which format. Figure 1-8 below shows a goal model with the goals defined above.

Figure 1-8 Goal model

1.4 Methodology of work

We are two students who have worked on this thesis. For this reason we have divided the work in two parts. Two projects have been examined while using cases to define the problem areas. We divided responsibility for each case, but changed the responsibility on the way so both of us could have the same understanding and knowledge about the research areas. The remaining work on this thesis has been done together.

Resources that have been used are mainly books related to the research area and the Internet. In addition, we have used project documentation as input to the cases defined and the existing solution approaches. The resources used for these cases and existing solution approaches have been available on the Internet and given by our supervisor. These resources have not been sufficient since the projects are at the time of writing ongoing. Also, the documentation has been dynamic. However, these resources have been useful for describing the problem area and for input to our proposed solution.

1.5 Structure of this thesis

The structure of this thesis is illustrated in Figure 1-9 and is organized in the following way. There are 9 chapters, and chapter 2 and onwards are built in a manner where a new chapter builds on a previous chapter. In chapter 2, two problem examples are presented to give an understanding of the problem in detail. These are project cases considering enterprise collaboration with metadata and data integration problems. Further, general data integration problems related to information interoperability is defined. With these problems in mind, requirements to solutions for data integration are specified. The goal for chapter 3 is to examine technologies

related to the problem area, and evaluate them according to the requirements. Chapter 4 analyses existing solution approaches to the project cases presented in chapter 2, and evaluate the approaches according to the requirements. In chapter 5, our proposed solution, MODI Framework is presented. It is a solution for data integration established with a model-based approach. In chapter 6 and 7 the MODI Framework is applied to the two project cases. In chapter 8 an evaluation of the MODI Framework is given. Finally, chapter 9 contains a conclusion and suggestions for possible improvements that could be applied to in future work.

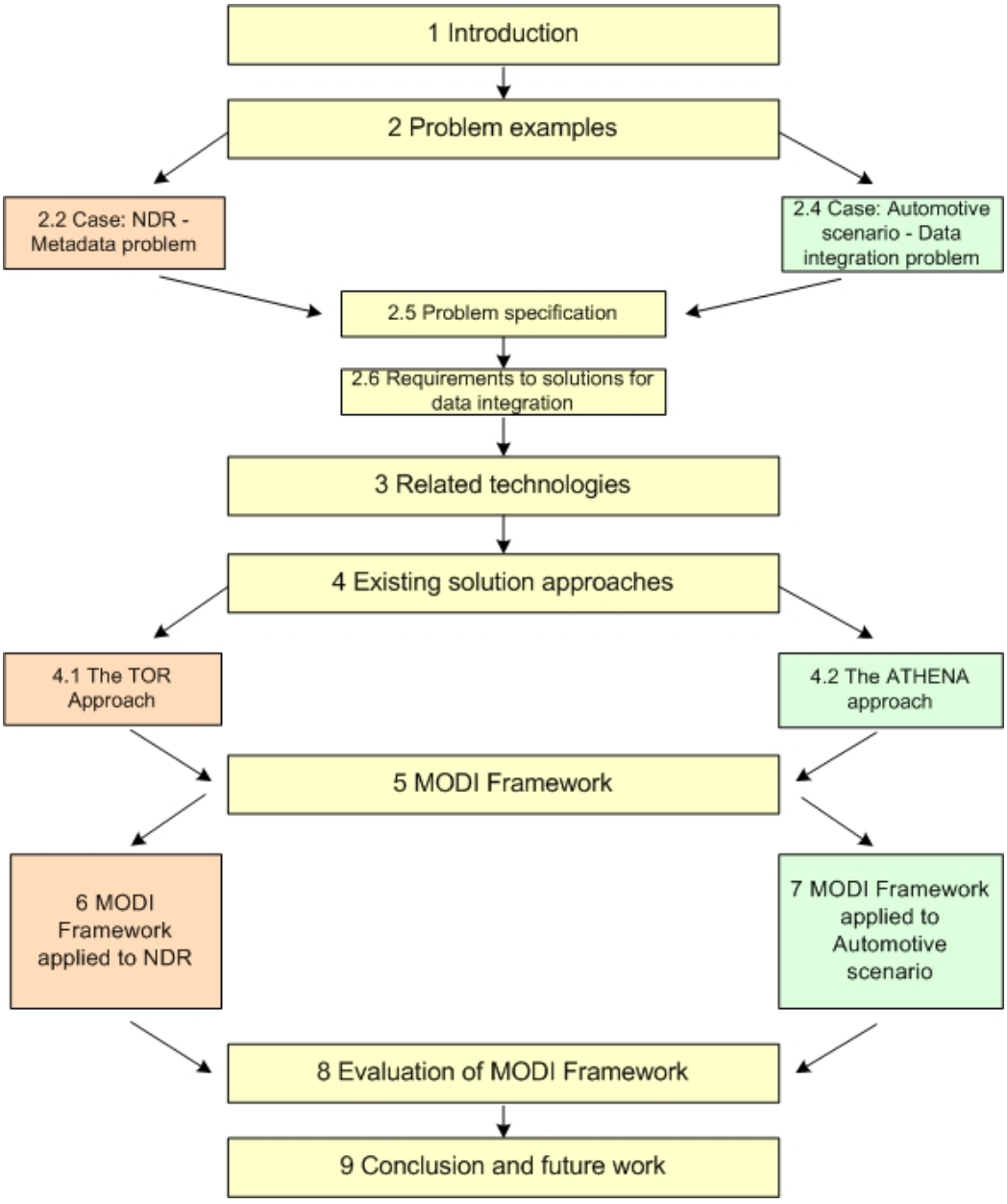


Figure 1-9 Structure of this thesis

2 Problem examples

The problem we address in this chapter is based upon projects we have analyzed. The main discussion topic is interoperability, with focus on metadata and data integration. First, we present the OR project, and then we present a case where metadata problem is discussed. Second, we present the ATHENA project. Based on the latter project, we present a case where data integration problems are discussed. This case describes a scenario which is used to show data integration problems between two heterogeneous enterprises. At the end of this chapter, we present the problem specification, and requirements to solutions for data integration.

2.1 National Data Registry (NDR)

National Data Registry (NDR) is the name we use to refer to the project concerning *The Register of Reporting Obligations of Enterprises* (OR[1]). OR was established by the Brønnøysund Register Centre in 1997 [22]. It is a national infrastructure for handling reporting obligations and one of many governmental registries in the Brønnøysund Register Centre. OR keeps track of all reporting obligations of enterprises in Norway, and develop implementation strategies for data collection related to these obligations. OR's intention is to achieve correct and efficient reporting, e.g. by identifying and preventing multiple reporting of the same information from enterprises and citizens to government departments. Thus, prevent superfluous collection and registration of information from enterprises.

OR was originally created to obtain an overview over all forms that are reported to government departments. Gradually information about fields in the forms were added. Further, it was realized that this could be used to create XML Schema definitions to define content in electronic forms, but then representation format had to be added. OR's main responsibility is to have an overview of reporting obligations, and over all forms (including fields) reported from enterprises to the government departments. In addition, OR offers XML Schema definitions in connection with electronic forms. The challenge with this solution, according to OR, is overlap detection and information exchange between the departments.

2.1.1 OR system design and architecture

Enterprises report obligations to departments, also called central government or just *receivers*, through a reporting service. The reporting service is a web portal, and Altinn [23] is an example of this kind of web portal. Altinn is used to send public

[1] In Norwegian: Oppgaveregisteret

forms through Internet. Altinn uses metadata from OR to generate forms. In addition, it centres on message- and application descriptions for different forms. A *message description* is an XML message describing how data should be represented on a form. An *application description* is generated by XForm [24] and describes how data definitions (metadata) shall be used to build a web-based application. In addition, it describes how data from the web-application shall be represented, modelled and validated.

The OR system contains *data definitions* which is reported by departments. This solution resembles the early data dictionary initiatives that attempted to create a central repository for storing and accessing technical definitions for the attributes and entities used in a company's IT system [25]. The data definitions describes the information requested by departments and are gathered in a database. All the data definitions can be found listed at [26]. This list is tabulator divided: Data identifier (*id_id*), *Name, Group, Type, Category.* Group, Type and Category is used to easier find the correct data definitions for reuse. More about the structure can be found here [27].

The OR system is divided into the following parts; ORdb, ORsys, ORetat and ORnett. Figure 2-1 shows the OR system and how the different parts are related to each other.

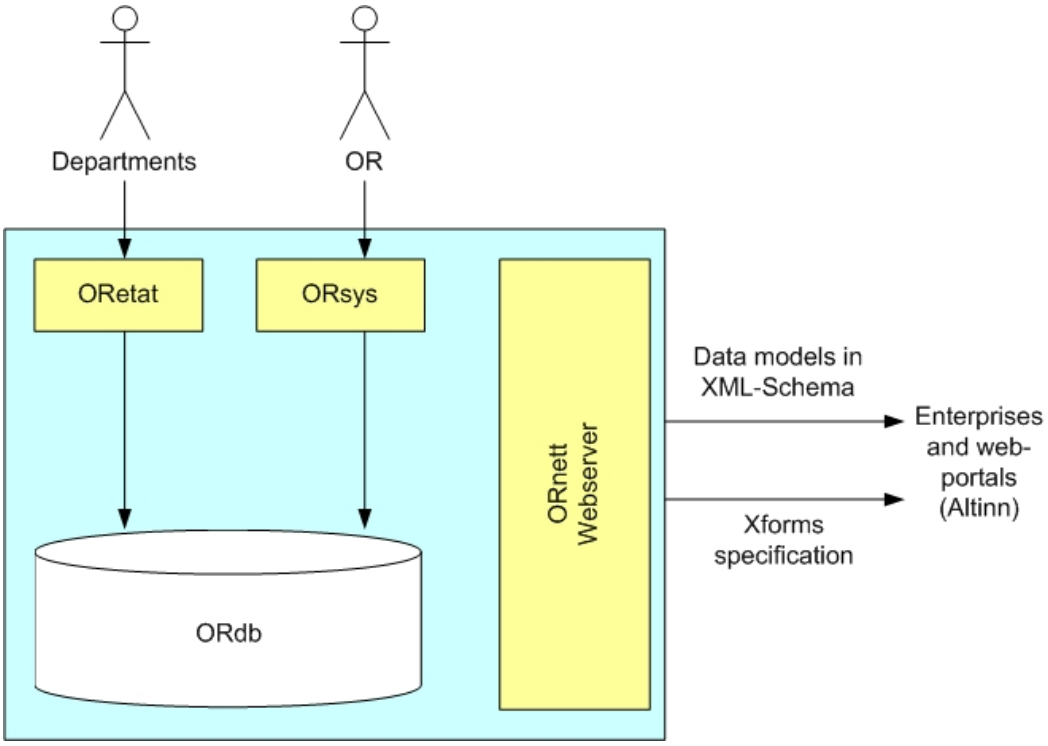


Figure 2-1 Overview of the OR system

- *ORdb* is the database of data definitions pertaining to reporting obligations. Every field in a form is identified in conformity with the minimum requirement formulated in *Metadata registries*, ISO-11179 [28].

- *ORsys* is the case handler system of OR. It is an internal system which is used by OR and the case handlers at OR. ORsys is used to maintain a list over registered metadata. In ORsys the metadata's format and semantics is not separated.

- *ORetat* is a data modeling tool based on the metadata in ORsys. The metadata are listed as attributes and further used in a specific message data model. It is available externally to departments through Altinn. ORetat transforms the data model to an XML Schema model and X-Forms

- *ORnett* is the open web-sites of OR and giving access to published messages and guidance from ORetat.

Metadata involved for the different reporting obligations are registered in OR. Additionally, the departments and enterprises involved for a reporting obligation are registered. OR, only coordinates data models and other general information about data definitions.

2.2 Case: NDR – metadata problem

Due to the departments' different assignments, they handle their information in different ways, such as use of different representation format. The reason for OR to do overlap search is to check if other departments already are using requested data definitions. Assume that two or more departments demand the same information from the same type of enterprise. Then, the department which needs the information already in use has to request it from the department that has the needed information in order to issue one information request. OR has to know which departments need to collaborate, and has to inform the departments involved about this overlap case.

2.2.1 OR and overlap detection

The departments are responsible for collecting information from enterprises, and for processing this information in their respective systems. The departments' systems are heterogeneous. Departments and OR collaborate as follows: A case handler from a department informs OR about data (attributes) to be reported. Then OR ensures that data definitions are consistent with the rules specified for the register, and identifies overlap with previously defined reporting obligations. Overlapping information is identified by comparing forms from different departments. Additionally, attributes not already registered in OR, are entered in the register. Further, the department places the attributes in one or several message data models. This approach is similar to the

ebXML Core Component and *Business Information Entity* (BIE) modeling methodology [29].

The information submitted by enterprises and citizens is often submitted several times, but to different departments. Every department has to verify the collected information. In the following tables we show an example of three forms we use to illustrate this. The forms presented are parts of a bigger form. Further, these forms contain some similar information and have some overlaps. In this example each of the forms are represented by different departments collecting same kind of information from same type of enterprise. The departments shown in the examples are: *Directorate of tax* (Form 1) shown in Table 2-1, *Food supervision* (Form 2) shown in Table 2-2 and *Food supervision 2* (Form 3) shown in Table 2-3.

Table 2-1 Form 1

Form 1				
Directorate of tax				
Name:		Personal identification number		
Firstname	*Lastname*	*Personal identification number*		*Land No Title number*
	Overview of animal Per 31.12.04	per 31.12.03		
Horse	*Horse*	*Horse*		
Cattle	*Cattle*	*Cattle*		
Pigs	*Pigs*	*Pigs*		
Sheep	*Sheep*	*Sheep*		
Goat	*Goat*	*Goat*		
Chicken	*Chicken*	*Chicken*		
Other feather animal	*other feather animal*	*Other feather animal*		
Fur-bearing animal	*Fur-bearing animal*	*fur-bearing animal*		
Reindeer	*Reindeer*	*Reindeer*		
Renting:		Completely/		
Type home unity	*Type home unity*			
Name of renter	*firstname*	*Lastname*		
Renting period	Start:	*Startdate*	end:	*enddate*
Rented completely / partially cost free?		*Rented completely /partially cost free*		
Rented as a part of working conditions		*Rented as a part of working conditions*		
Rent value free of charge		*Rent value free of charge*		
Paid rent		*Paid rent*		
Expenses on rented homes				
Capitalized costs		*Capitalized cost*		
Running expenses		*Running expenses*		
Maintenance expenses		*Maintenance expenses*		

Form 1, Form 2 and Form 3 are taken out of a larger form to make the understanding of overlap easier. Form 2 is a real subset of Form 1. The department using Form 2 is demanding exactly the same information as the department using Form 1. If the Food supervision knows that Directorate of tax already collects the same information, they could request it from them. Consequently, it helps eliminate a lot of work with issuing forms and collecting and verifying the same information.

Table 2-2 Form 2

Form 2			
Food supervision			
First name:	Lastname	Personal identification number	
Firstname	*Lastname*	*Personal identification number*	*Land No Title number*
	Overview of animal per 31.12.04	per 31.12.03	
Horse	*Horse*	*Horse*	
Cattle	*Cattle*	*Cattle*	
Pigs	*Pigs*	*Pigs*	
Sheep	*Sheep*	*Sheep*	
Goat	*Goat*	*Goat*	
Chicken	*Chicken*	*Chicken*	
Other feather animal	*other feather animal*	*other feather animal*	
Fur-bearing animal	*fur-bearing animal*	*fur-bearing animal*	
Reindeer	*Reindeer*	*Reindeer*	

Form 3 and Form 1 collect much identical information. At the same time each form collects different information. For instance, one of the differences contained in Form 1 allows entry of renting and in addition registration of animals at two occasions. However, Form 3 contains a field not included in Form 1 which is *Dead animals last year*. This problem can be solved in two ways. A suggested solution is to make one collective form with all the required information or by keeping two forms, where one form contains the common information and the other with the remaining information. Another problem with Form 1 and Form 3 is that they collect the same information, but at different points in time. This is not necessarily an overlap, but there is potential for simplification by collecting the information at the same time. This is also considered as an important type of overlap.

Table 2-3 Form 3

Form 3			
Food supervision 2			
Name:		Personal identification number	
Firstname	*Lastname*	*Personal identification number*	*Land No Title number*
	Overview of animal Per 31.07.04		
Horse	*Horse*		
Cattle	*Cattle*		
Pigs	*Pigs*		
Sheep	*Sheep*		
Goat	*Goat*		
Chicken	*Chicken*		
Other feather animal	*other feather animal*		
Fur-bearing animal	*Fur-bearing animal*		
Reindeer	*Reindeer*		
Dead animals last year	*Dead animals last year*		

In the case where there is not need to exchange information between the departments, different information handling is not a problem. However, in the opposite case it is not possible to exchange information electronically between different databases of departments directly. Our next discussion topic is the way departments collaborate to issue one information request and which problems arise.

2.2.2 Metadata problem leading to data integration problems

Presently, OR has already created a number of data definitions only for the *name* of an enterprise because of different proprietary solutions at the various departments. To view an example see here [30]. The terminology's intended meaning at the different departments is not clear between them. Further, if a department wants another format on a data definition which already exists, it will result in that OR has to create a new data definition. In this case the semantic meaning is the same, but the format is changed. This can be referred to as *semantic heterogeneity*, which in this case leads to different identification and treatment of forms that in principal are the same. The problem is to identify data definitions that refer to the same concept, since the departments define data according to their systems. This fact makes it harder to reuse data, since they are too specific and they do not separate syntax from semantics. Consequently, causes multiple reporting of the same information. Additionally, it is

hard to find data definitions for reuse because they exist in many versions. The lack of reuse also reduces the possibility for overlap detection, since it is not known to which extent different data definitions are semantically equivalent. Further, few identical data definitions are identified when OR does overlap detection, since the metadata about format is included in the XML Schema description .

The Brønnøysund Register Centre has, at the time of writing, an ongoing project to solve the problems mentioned above. This project is further described as an existing solution approach in chapter 4. As the departments have information represented differently, it is hard to exchange the information between them. A department model their information differently according to their needs and demands. After studying the NDR case, we conclude that several data integration problems may arise in the case departments need to collaborate. The problems defined above needs to be solved before departments can start integrating data.

2.3 The ATHENA Project

Advanced Technologies for interoperability of Heterogeneous Enterprise Networks and their Applications (ATHENA) is an Integrated Project funded by the European Commission, initiated in 2004, and scheduled to last 36 months [31]. The ATHENA consortium consists of 19 partners which are, Aidima and ESI (Spain), Computas and Sintef (Norway), Cr-Fiat, Leks, TXT and Formula (Italy), Dfki, FHG IPK Simens and SAP AG (Denmark), Eads-ccr, Graisoft and University Bordeaux I (France), IBM and IC-Focus (Great Britain), Intracom (Germany), Uninova (Portugal).

The ATHENA project is concerned with enterprises that are transforming themselves into networked organisations. ATHENA's main objective is to remove interoperability barriers. In addition, they will enable interoperability by providing a comprehensive Interoperability Framework.

Furthermore, ATHENA has defined four business scenarios that capture industry specific requirements: Collaborative Product Design (automotive sector), Supply Chain Management (aerospace sector), e-Procurement (furniture sector) and Product Portofolio Management (telecommunication sector) [32].

We have analyzed the automotive sector, Fiat Auto case. This case focuses on the *Product Development Process* (PDP) portion which prescribes suppliers involvement in the objectives definition and on product planning, called *Collaborative Product Development* (CPD). The main emphasis is on collaboration between FIAT and suppliers, and integration aspects between the two actors.

With the automotive sector in mind, we and some other students from the University of Oslo have described a case. The case deals with data integration problems that may

arise between two enterprise systems using different formats. We have called the case *Automotive scenario*.

2.4 Case: Automotive scenario – data integration problem

In the Automotive scenario, a car manufacturer spends considerable resources to handle logistics, warehouse and contracts. It is therefore desirable to reduce the administrative overhead and warehouse costs. In addition, it is vital for enterprises in the car manufacturing supply chain to be able to share information. The Automotive scenario presents a case where enterprise systems using different formats face interoperability problems. In this scenario, information interoperability problems about data integration are considered.

The scenario's main focus is order and invoice, by which customers can order products from suppliers. To simplify the situation, we include only one customer and one supplier. Both customer and supplier provide interfaces to their systems, enabling to order products and send invoice by using a so-called *Application Program Interface* (API). Two UML use case diagrams illustrate views of the customer's and the supplier's system functionality. Figure 2-2 shows a use case diagram depicting a customer ordering products.

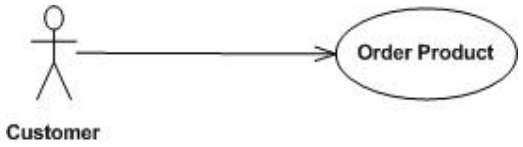


Figure 2-2 Order Product use case

Figure 2-3 shows a use case diagram depicting a supplier sending an invoice.

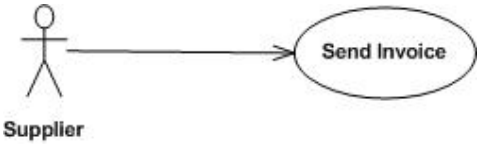


Figure 2-3 Send invoice use case

The UML sequence diagram in Figure 2-4 illustrates how customer and supplier interact with messages. The messages contain arguments referring to the information to be exchanged. We only consider a happy scenario in which the customer first orders the product, and then receives an invoice from the supplier with no negative occurrence, such as products being out of stock etc. Further, we assume that the customer to be a car company like Fiat and supplier to be a broker that purchases small car parts like lights, bumpers, etc. like Bosch.

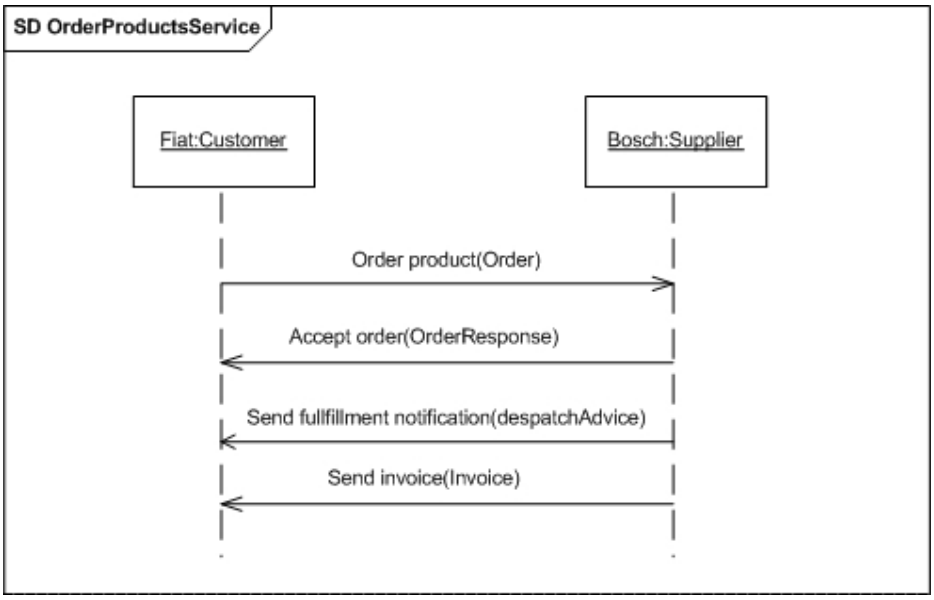


Figure 2-4 Interaction between customer and supplier

From the figure above we can describe the following messages:

1. Customer *orders* products from supplier
2. Supplier responds with *accept*
3. Supplier *sends fulfillment notification* to the customer
4. Supplier *sends invoice* to the customer

In the next section we discuss various data integration problems that may arise depending on several conditions.

2.4.1 Problem description

In the case where the customer's and the supplier's systems are heterogeneous, different data integration problems need to be considered in order to achieve information interoperability. The way the customer and the supplier define data may differ syntactically and semantically. If these differences are not identified and dealt with, collaboration between the customer and the supplier is inconsistent. Figure 2-5 shows an inconsistent state between the customer and the supplier, since they are using different technologies. The bold line in the middle of the figure indicates incompatible message exchange.

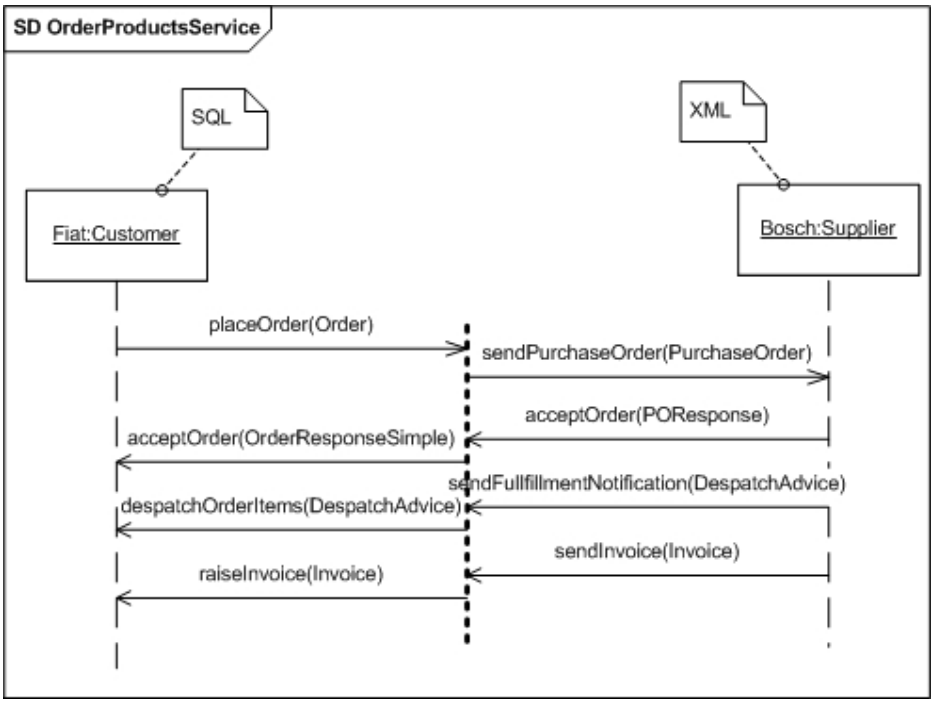


Figure 2-5 Inconsistent state between customer and supplier

From the figure we see that the customer uses *Structured Query Language* (SQL) [33] when specifying order and invoice. SQL is a query language based on the relational model of database systems. Further, it includes statements for modifying the database, and for declaring a database schema. It serves as both a *data manipulation language* (DML) and a *data definition language* (DDL). The DDL code for the customer is shown in Table 2-4.

Table 2-4 SQL code for customer

SQL code for customer expressed in DDL	
CREATE TABLE Order(orderID INTEGER(10), issueDate DATE, comment VARCHAR(50), expireDate DATE, ID INTEGER(5),) CREATE TABLE Orderline (orderID INTEGER(10), comment VARCHAR(50),) CREATE TABLE LineItem(ItemID INTEGER(10), quantity INTEGER(10), taxAmountTotal INTEGER(15), lineStatusCode VARCHAR(10), comment VARCHAR(50), orderID INTEGER(10),) CREATE TABLE Buyer(ID INTEGER(5), firstName VARCHAR(15), lastName VARCHAR(15), address VARCHAR(15), city VARCHAR(10), country VARCHAR(15), telephonenr VARCHAR(8),)	CREATE TABLE Invoice(invoiceID INTEGER(10), issueDate DATE, comment VARCHAR(50), lineItemCount INTEGER(10),) CREATE TABLE InvoiceLine(invoiceID INTEGER(10), lineStatusCode VARCHAR(10), comment VARCHAR(50), itemID INTEGER(10),) CREATE TABLE Item(itemID INTEGER(10), name VARCHAR(25), description VARCHAR(30), packquantity INTEGER(10),)

Figure 2-6 presents the customer's data in a *Entity Relationship* (ER) model.

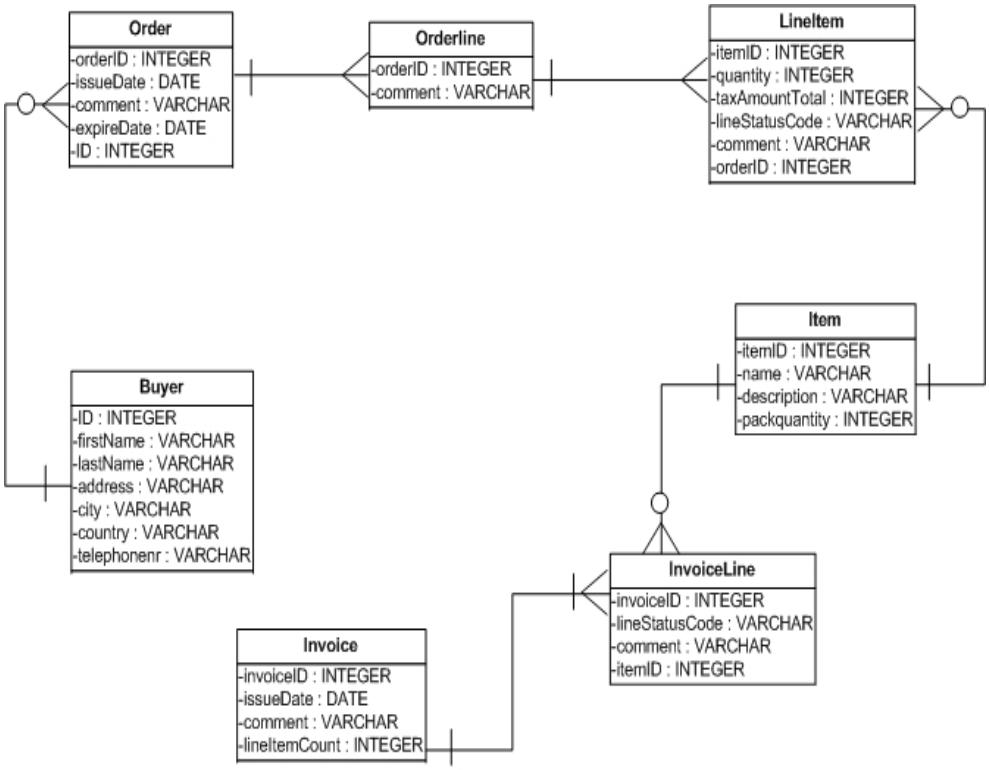


Figure 2-6 SQL relational model – ER model

A segment of the supplier's XML Schema is shown in Table 2-5. The XML Schema represents the data in elements. The elements can either be of a *complex type* or *simple type*. The complex type contains other elements, such as simple types. However, simple types do not contain other elements. The relational model represents the data in tables and attributes. Only information relevant to our scenario is taken into account in the models. The models define the information used by the customer and the supplier differently. The differences are listed below:

- The models are structured differently; XML uses inheritance in contrast to the SQL relational model. In addition, they use totally different syntax, e.g. a complex type in XML corresponds to a table in SQL.
- They are using different names on most of their data.
- Some data in XML are not managed in SQL, and the other way aorund.
- They use different datatypes in some cases.

Table 2-5 XML Schema for supplier

XML Schema for supplier

```
<?xml version="1.0"?>
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema"
targetNamespace="http://www.automotivescenario.com"
xmlns="http://www.automotivescenario.com"
elementFormDefault="qualified">

<xs:element name="Customer"  minOccurs=”1” maxOccurs="1”>
  <xs:complexType>
    <xs:sequence>
         <xs:element name="ID" type="xs:integer"/>
         <xs:element name="firstName" type="xs:string"/>
         <xs:element name="middleName" type="xs:string"/>
         <xs:element name="lastName" type="xs:string"/>
         <xs:element name="city" type="xs:string"/>
         <xs:element name="country" type="xs:string"/>
         <xs:element name="phone" type="xs:string"/>
    </xs:sequence>
  </xs:complexType>
</xs:element>

<xs:element name="PrisedDocument">
  <xs:complexType>
    <xs:sequence>
         <xs:element name="ID" type="xs:integer"/>
         <xs:element name="orderDate" type="xs:date"/>
         <xs:element name="note" type="xs:string"/>
         <xs:element name="lineItemCount" type="xs:integer"/>
         <xs:element name="prisingCurrencyCode" type="xs:string"/>
    </xs:sequence>
</xs:complexType>
</xs:element>

<xs:element name="PurchaseOrder"  type=”PrisedDocument”>
  <xs:complexType>
    <xs:sequence>
         <xs:element name="earliestDate" type="xs:date"/>
         <xs:element name="expiryDate" type="xs:date"/>
         <xs:element name="totalPackageQuantity" type="xs:integer"/>
    </xs:sequence>
  </xs:complexType>
</xs:element>
```

Table 2-6 shows a detailed mapping between SQL table Order and XML element PurchaseOrder, and SQL table Buyer and XML element Customer. From the table we can see that the two models use different syntax to denote the same information. Below we elaborate for the differences arising when comparing the two different formats. These differences lead to data integration problems.

Table 2-6 Detailed mapping between SQL and XML

SQL	XML
OrderProductsService	
Order	**PurchaseOrder**
ordered	ID
issueDate	orderDate
comment	note
Not handled	lineItemCount
Not handled	prisingCurrencyCode
Not handled	earliestDate
expiryDate	expiryDate
Not handled	totalPackageQuantity
ID (foreign key,primary key in table Buyer)	***Not handled***
Buyer	**Customer**
ID	ID
firstName	firstName
Not handled	middleName
lastName	lastName
address	***Not handled***
city	city
country	country
telephonenr	phone

Mapping between Order and PurchaseOrder

The SQL table Order uses the terms orderID, issueDate and comment. While, the XML element PurchaseOrder uses the terms ID, orderDate and note to define the same *semantic* concepts.

Another concern is that XML uses inheritance, and SQL does not. Some of the terms are represented at *different aggregation levels* in XML. In this case the attributes ID, orderDate, note and lineItemCount are part of the super-class PricedDocument in XML.

Further, some terms are *represented differently*. issueDate and expiryDate for SQL table Order are represented as *dd.mm.yy*, but orderDate and expiryDate for XML element PurchaseOrder are represented as *yy.mm.dd*.

Additionally, some of the terms in XML element PurchaseOrder are *not handled in* SQL table Order. These terms are: lineItemCount, earliestDate, totalPackageQuantity and prisingCurrencyCode. Since SQL uses foreign key as relation, the foreign key ID from table Buyer is contained in table Order. For this reason, the latter mentioned term is not handled in XML element PurchaseOrder.

Mapping between Buyer and Customer

The SQL table Buyer and the XML element Customer uses the term ID to denote completely *different concepts*. In table Buyer the term ID is a system generated number which identifies the buyer. However, in element Customer the term ID is a national identity number. These identical terms are semantically unrelated concepts.

Some *properties are modelled differently* in table Buyer and element Customer. Table Buyer uses firstName and lastName to denote name, while element Customer uses middleName in addition to firstName and lastName. Besides, the term address in table Buyer is not handled in element Customer.

The terms telephonenr in table Buyer, and phone in element Customer are synonymous. In addition the terms are represented by different datatypes. The term telephonenr uses the datatype VARCHAR and the term phone uses the datatype INTEGER.

Finally, another conflict that is not considered in the mapping above is the case where same term uses *different default values*. For instance a term *tax* has the default value 20% one place and 25% at the other.

2.5 Problem specification

After looking at the two cases, we conclude that both have data integration problems. The NDR case takes into account metadata problems, which leads to the difficulty of integration of heterogeneous data between departments. The Automotive scenario case is concerned about data integration problems that may arise between enterprise systems using different formats. The problem of different format use can be divided into two parts. The first part is concerned about dealing with different possibilities of representing information. The second part has to do with interpreting the information.

Data integration between enterprise systems is a very general interoperability problem. It can be explained in some interrelated sub-problems, such as data value- , schema- , data model- , syntactic- and semantic conflicts:

- *Data value conflicts* are related to the representation or interpretation of the data values. For example these conflicts are discrepancies of type, unit, precision, allowed values or spelling.

- *Schema conflicts* are concerned about different alternatives provided by one datamodel to develop schemas for the same situation. For instance what is modelled as an attribute in one relational schema might be modelled as a relation in another relational schema for the same application domain. This can also be referred to as a data precision conflict.

- *Data model conflicts* arise when databases uses different data models. E.g. one database is designed according to the relational model and another one object-oriented.

- *Syntactic conflicts* refer to data representation discrepancies. In other words, when data involved in an integration process is designed with different approaches.

- *Semantic conflicts* refer to difference of opinion about the meaning or interpretation of the same data. Consequently, it refers to discrepancies associated with representation with real world objects and phenomena. The case where syntactic conflicts can have a semantic counterpart is not considered.

From the discussion above general data integration problems are defined in Table 2-7. These problems are based on the conflicts described in [4]. Mostly we consider the representation and interpretation concerns, accordingly syntactic-, structural and semantic conflicts.

Table 2-7 General data integration problems

Problem	Description
Synonyms	Semantic equivalence: Different terms referring to same semantic concept, ergo same information, different attribute names
Homonyms	Semantic incompatibility: Same term used to denote different concepts, ergo different information, same attribute names
Data representation conflicts	Semantic equivalence: Type mismatch, e.g. different units of measurements used (cm vs. km)
Differences in properties	Two systems model properties differently. (first name, middle name, last name, vs. name)
Data precision	Semantic relationship: Some elements are represented as relation/attribute, also called different aggregation level.
Default value	Semantic relevance: Different default values for e.g. tax
Attribute integrity constraint	Semantic resemblance: Different data types for same attribute (string vs. integer)
Data lacking	Information elements are missing or not accounted for in one specification and provided in the other.

2.6 Requirements to solutions for data integration

The data integration problems can be overcome. Several requirements are vital to enable enterprises systems with different representation, structure and meaning of data to integrate their data. The solution should be a general description of a framework for enabling enterprise systems using proprietary formats to integrate data.

Metadata enrichment is an important requirement to support semantic matching among data items from different enterprise systems. This helps obtain the correct meaning of data.

Metadata is vital for identifying differences in data such as synonyms, homonyms, data representation conflicts etc.

Another important requirement is requiring enterprise systems to support use of platform independent data model as exchange format. The data model should allow for identification of differences in their information systems data at a platform independent level.

Further, tool support to manage data mapping and integration is another vital requirement. A tool will provide interactive ways to manipulate data and perform mapping. Enterprises should be able to combine their information systems with modern mapping techniques in a consistent manner. This requirement will allow enterprises to perform the actual mapping. From the considerations above, reasonable requirements we find to solutions for data integration are defined in Table 2-8.

Table 2-8 Requirement to solutions for data integration

Requirements	Description
Metadata enrichment	The solution shall specify how to integrate data with different syntax, structure and semantic, by obtaining correct meaning of data.
Synonyms	The solution shall specify how to integrate data that are synonyms
Homonyms	The solution shall specify how to integrate or manage data that are homonyms
Data representation conflicts	The solution shall specify how to integrate data with different representations.
Differences in properties	The solution shall specify how to integrate data with differences in properties.
Data precision conflicts	The solution shall specify how to integrate data concerning precision conflicts
Default value conflicts	The solution shall specify how to integrate data with different default values
Attribute integrity constraint conflicts	The solution shall specify how to integrate data concerning attribute integrity constraint conflicts
Data lacking	The solution shall specify how to deal with data lacking
Platform independent data model	The solution must have support for data integration through a data model which shall not be based on any specific implementation platform. The data model for data integration must be versatile and self-explanatory.
Tool support	The solution must have support for tools to manage data mapping and integration.

2.7 Summary

In this chapter we have presented two problem examples and analysed them. With these problem examples in mind we have identified and given an insight to some general data integration problems. At the end of this chapter requirements to solutions for data integration are defined. In the continuation, we look into related technologies dealing with interoperability and data integration.

3 Related technologies

Different consortiums, such as IBM, Microsoft, OASIS have initiated technologies to facilitate interoperability and support data integration. The technologies that are examined in this chapter are XML, XSLT, ebXML, MOF, BizTalk and Altova MapForce 2005. At the end of this chapter these technologies are evaluated according to the requirements described in the previous chapter.

3.1 Extensible Markup Language (XML) as a syntactic standard

A standard format for data representation and exchange in the Internet which recently has emerged is XML [34]. XML allows parties to exchange and integrate structured data, similar to information stored in databases, over the Internet. It is an open and freely available document from *World Wide Web Consortium* (W3C) [35]. Besides, it has the support of the leading technology companies who are mowing toward adopting it. They are either using it as an internal data representation model for their software or for data exportation and importation among different applications and platforms. Another good thing, XML supports Unicode that enables the display and exchange of most of the world's written languages.

Like HTML, XML employs tags to structure data, but in XML tags are defined for each application and can be used to identify the meaning of data. Web servers and enterprises' systems encoding their data in XML can quickly make their information available in a simple and generally accepted format. Information content is separated from information rendering, making it easy to provide multiple views of the same data.

Enterprises systems can represent their data more generally and flexible by using XML than other standards, including the relational data model. Furthermore, the data in XML format can make the Internet a huge data source for all sorts of information. Using XML as a data representation standard can bring many benefits for data integration. Since XML is a semi-structured data model it can increase flexibility, and aid in data representation. The activities involved in the data integration process can be simplified and distributed, if semantics are associated to XML data and their markup. In the case where XML is without agreement upon semantics associated to data and tags, it does nothing to support integration except for providing common syntax.

We may verify that use of XML as a standard enables heterogeneous enterprise systems to integrate data by comparing syntax and structure, but not semantics. XML does not by itself support semantic description. To achieve information interoperability with XML it is necessary to establish multiple agreements on application domain terminologies, taxonomies and representations. Although there are advantages with XML such as that it is low-risk, stable and reliable way of transporting data.

Furthermore, there is existence of different XML versions for definition of message exchange. Enterprises using different XML formats can collaborate by using a transformation technology for XML documents.

3.2 XSL Transformations *(XSLT)*

W3C has developed an XML-based style sheet language, called *EXtensible Stylesheet Language* (XSL) [36]. A part of XSL is *XSL Transformations* (XSLT) [37]. XSLT is a language for transforming an XML document into another XML document. To discover information in an XML-document, XSLT uses XPath [38] to navigate in a XML document through elements and attributes.

In the transformation process XSLT defines parts of the source document that should match one or more predefined templates. When a matching is found, XSLT transforms the matching part of the source document into the result document. An example of and XML file with the belonging XSL file is shown in Appendix B.

3.3 Electronic Business XML (ebXML)

Electronic Business XML (ebXML) [39] is an initiative started by *Organization for the Advancement of Structured Information* (OASIS) and *Unified Nations Centre for Trade Facilitation and Electronic Business* (UN/CEFACT) in 1999. OASIS is an international nonprofit consortium that promotes open collaborative development of interoperability specifications [40]. UN/CEFACT is a body of the United Nations whose mandate is to support the worldwide development in the area of trade facilitation and electronic business [41]. ebXML is a technology where XML is one of its technical foundations and enterprises are enabled to conduct business over the Internet by adopting its specifications. It is a set of specifications which enables enterprises to conduct business over the Internet, independent of their size and geographical location. It intends to support description, discovery, composition and execution of business processes and services across the Internet. The goal of ebXML is to provide an infrastructure for a single global electronic market.

3.3.1 ebXML specifications

ebXML includes specifications for public repositories of industry business processes, messages and common data objects that companies need to get started exchanging data, additional to register their capabilities to engage in electronic business. Then, enterprises can use these registries to access the stored data objects and find new suppliers or customers with the ability to provide electronic messages or services. The specifications enable dynamic B2B collaborations which cover the following bases [42]:

Core Components: provides reusable data structures such as party, address, phone, date, currency. They form a single, consistent lexicon used to define business process and information models that facilitates interoperability between heterogeneous enterprise systems. *Universal Business Language* (UBL) [43] is an example on a Core Component which defines a reusable, generic XML interchange format. Besides it defines hierarchical relationships between processes. These relationships are typically used as instruments in both composition and decomposition of processes. UBL is discussed further in section 3.3.3.

Registry/Repository: is an information system which stores XML artefacts (XML schemas, data elements etc.) and supportive documents which are not XML artefacts and metadata of the artefacts. The part of the information system which manages the metadata for the registered objects, is called *Registry*. The storage unit which keeps Registry objects is called *Repository* [44].

Collaborative Protocol Profile **(CPP)**: is a concrete specification of a company's offerings, which are the business scenarios one support, the service interfaces one implements, document format exchanged, technical requirements and options for protocols, security and reliability. The profile is composed of business process models, information models, and context rules. The information model defines the documents, and the industry-specific context in which the transactions take place.

Collaboration Protocol Agreements **(CPAs):** Examination of an enterprise's CCP is done after finding a registry and search for partners in order to ascertain compatibility of the business process and technical specifications. "Rules of engagement" are stipulated and a CPA is produced.

Message Service: is a secure XML messaging service which is required to enforce the rules of engagement in the CPA. The message service is defined transport independent.

3.3.2 ebXML Registry/Repository Example

Figure 3-1 is based on the ebXML Technical Architecture Specification and illustrates a scenario for using the ebXML Registry/Repository. The figure is taken from [45]. The scenario consists of six steps that are further explained:

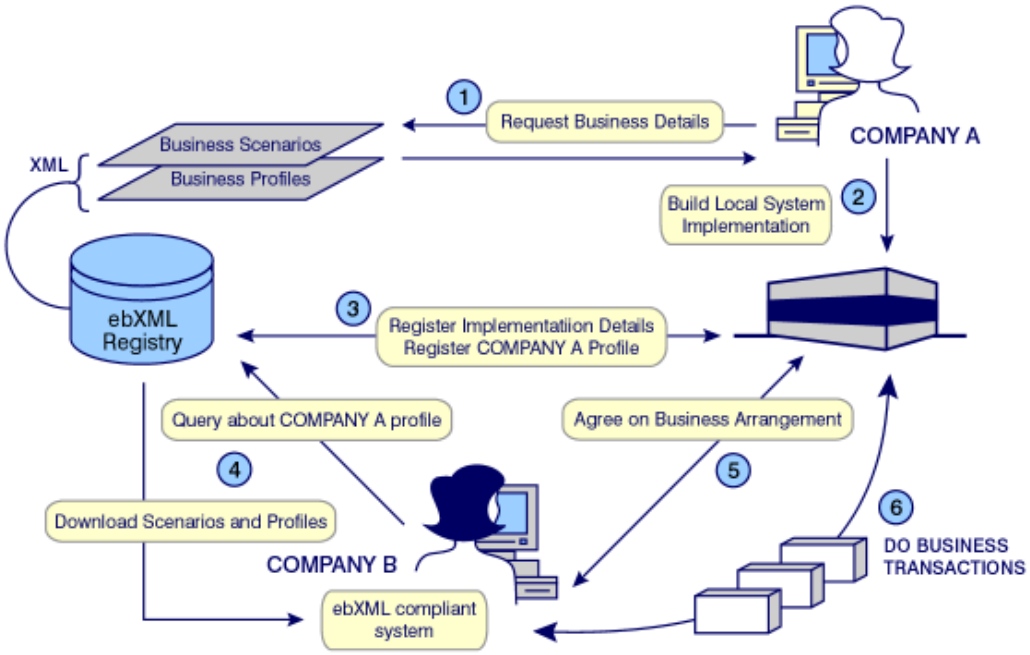


Figure 3-1 A scenario for using ebXML Registry/Repository

1. Request Business Details

COMPANY A wants to see the content in an ebXML Registry. The company searches in the Registry/Repository bases to find out which collaboration models are available and suited for its business. The collaboration of interest and a core library, a set of standard "parts" which can be applied on large ebXML elements, will be downloaded. The core library and maybe other registered business processes will allow COMPANY A to determine the requirements for their own implementation of ebXML and if ebXML is suitable for their business needs.

2. Build Local System Implementation

An adjustment to own system is done. COMPANY A has the opportunity to build or buy an ebXML implementation that fits to their expected ebXML transactions, based on a view of information which is available from an ebXML Registry. Afterwards the COMPANY A's local systems are adapted to the collaboration model.

3. Register Implementation Details- Register COMPANY A Details

The next step is to publish own use. COMPANY A creates and registers a CPP with a Registry. The company can contribute with new business processes to the Registry, or refer to available ones. CPP will consist of information which is necessary so a potential partner can determine business roles which COMPANY A is interested in.

4. **Query about COMPANY A profile**
Finding collaboration partners is the next step. After COMPANY A is registered, COMPANY B accesses the Registry/Repository bases to look for potential collaboration partners, in this case COMPANY A. COMPANY B can now see Company A's CPP and determine if it is compatible with its CPP and requirements.

5. **Agree on Business Agreement**
COMPANY A and B enter into a collaboration agreement to do e-commerce according to some collaboration model. Their systems are configured to mutually be able to do business transactions according to this collaboration model.

6. **Do Business Transactions**
Then the companies can start actual transactions. These transactions involve business messages that further confirm to ebXML standards and recommendations.

ebXML does not just support messages and services among businesses (B2B), but also between businesses and consumers (B2C). However, only the services and architecture are defined by the specifications on the business end, not customer screens or interactions.

3.3.3 ebXML Core Component - Unified Business Language (UBL)

In the case where an enterprise already has agreed on an XML vocabulary, it might need to change its message structure to meet the requirements of ebXML. It offers a common message structure and syntax for exchanging business data over data networks like the Internet using XML. This offer replaces the prospect for interacting with multiple vocabularies. UBL was initiated by OASIS in 1999.

UBL defines a standard of electronic XML business syntax documents such as purchase orders and invoices. It replaces the widespread use of proprietary XML versions for definition of message exchange in electronic commerce. The existence of different formats to accomplish the same purpose in different business domains consists of a number of disadvantages. First, developing and maintaining multiple versions of common business documents leads to duplication. Second, creating and maintaining multiple adapters to enable trading relationships across domain boundaries is ineffective. Third, the existence of multiple XML formats makes it much harder to integrate XML messages. Fourth, the need to support a random number of XML formats makes tools more expensive and trained workers harder to find.

UBL's intention is to solve these disadvantages by providing a library of XML Schemas for reusable data components such as "Address," "Item," and "Payment", common data elements of everyday business documents. Further provide for a small set of XML Schemas for common business documents such as "Order", "Despatch Advice" and "Invoice". These business documents are constructed from the UBL library components and can be used e.g. in

a generic order-to-invoice trading context. Additionally, UBL supports for customization in specific trading relationships.

As mentioned, UBL is designed to operate within a standard business framework such as ISO 15000 (ebXML). The purpose is to provide a complete, standards-based infrastructure that can extend the benefits of existing EDI systems to businesses of all sizes.

UBL schemas are modular, reusable, and extensible in XML-aware ways. Designed as an implementation of ebXML Core Components Technical Specification 2.01, the UBL Library is based on a conceptual model of information components known as *Business Information Entities* (BIE). These components are assembled into specific document models such as Order and Invoice. These document assembly models are then transformed in accordance with UBL Naming and Design Rules into W3C XSD schema syntax. This approach facilitates the creation of UBL-based document types beyond those specified in this 1.0 release. This document describes the basic order-to-invoice business process that the UBL document types are designed to support.

An alternative to ebXML is EDIFACT [41]. Similar with ebXML, EDIFACT is a message platform for message-oriented computing. However, ebXML bases its approach on replacing earlier EDIFACT/EDI standards with similar XML messages and documents. ebXML is complimentary with existing standards, not competitive, such as UN/EDIFACT, etc. Thus preserving much of the existing investment in these applications characteristics with EDIFACT is that it focuses on technical aspects and has little support to semantics. However, UBL is much stronger, easier and better to use than EDIFACT.

In an environment where heterogeneous enterprise systems use the shared repository ebXML they adopt the same set of agreements for interoperability solutions. Further, they conduct business by integrating data. Different from XML, ebXML specifies the use of the core component UBL as format. Enterprises can achieve information interoperability through multiple agreements on application domain terminologies, taxonomies and representations. The disadvantage is that heterogeneous enterprise systems have to adopt the new format to conduct business.

3.4 Meta Object Facility (MOF)

To support the investigation of semantic matching among data items from different enterprise systems, involves metadata enrichment. The MOF specification is one of OMG’s standards for modeling distributed software architectures. It is an extensible model-driven integration framework for defining, manipulating and integrating metadata and data in a platform independent manner. Nowadays, many of the available metadata standards are being aligned with XML.

3.4.1 MOF defining metadata and data

The MOF standard is platform-independent, and it is a meta-language which allows us to define an abstract language. Additionally, it allows a way to specify, build and handle technology neutral metamodels. A metamodel is essentially an abstract language for some metadata. Examples are UML, CWM and MOF itself. The reason for these modeling languages to have a formal definition is so tools can automatically transform the models expressed in these languages. UML and CWM are in use for integrating tools, applications and data. As these languages are defined, tools are able to read and write the standardized languages. An advantage of having a shared metamodel is that generic tools cooperating on any MOF-compliant model can be built.

Figure 3-2 shows a simplified version of the MOF model [11]. To see the whole MOF model, see Appendix B. From the figure we see that the classes constitute the basis for the definition of any modeling language. The elements are used to define metamodels. MOF uses a subset of UML to describe modeling concepts. However, MOF's semantics are more precise and more limited than general UML.

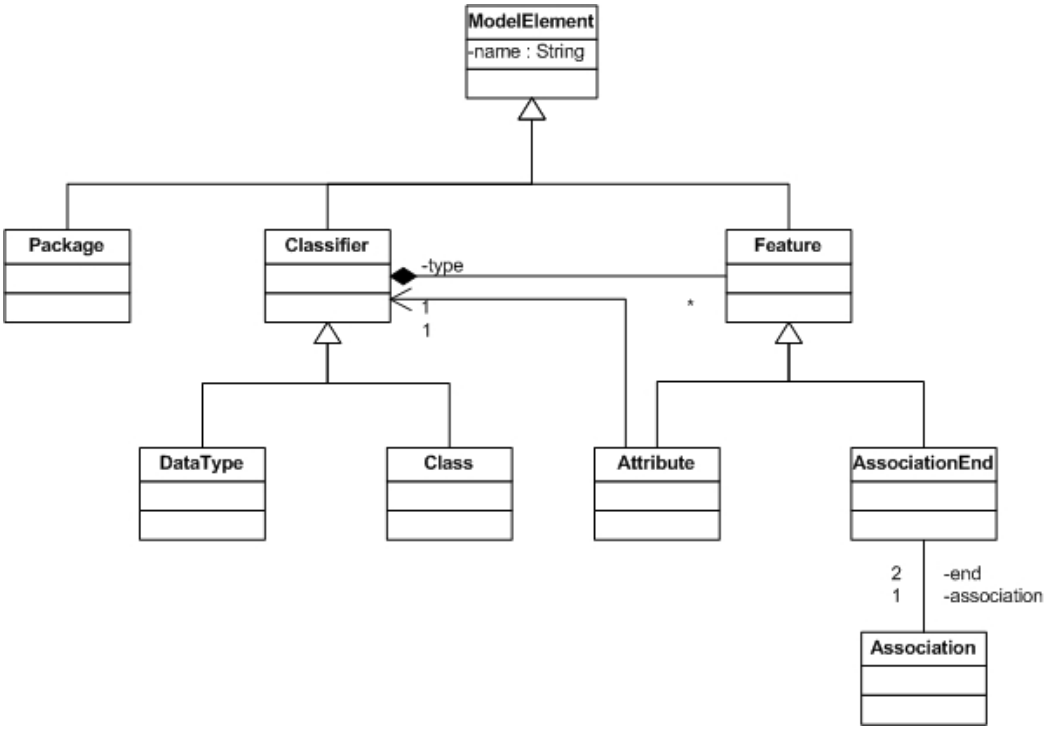


Figure 3-2 Simplified MOF model

3.4.2 Metadata architecture

Since the meta-language MOF is a language itself, it is natural that it can be defined by a metamodel expressed in another meta-language. This can go on and on until we reach an infinite number of layers of relationship. OMG use four layers for modeling architecture and

MOF represent the top-level language in the metadata hierarchy shown in Figure 3-3. The central idea is to define metadata architecture, in a way that metamodels and models based on it can be linked together using a simple language. As MOF correspond to the topmost layer, it can be used to define the semantics and structure of generic meta-models or domain specific ones. More traditional models, like the relational one, can also be represented using the MOF. The advantage with the metadata architecture is semantic constructs refinement through its application on consecutive layers. The elements in a given layer describe elements in the next layer down.

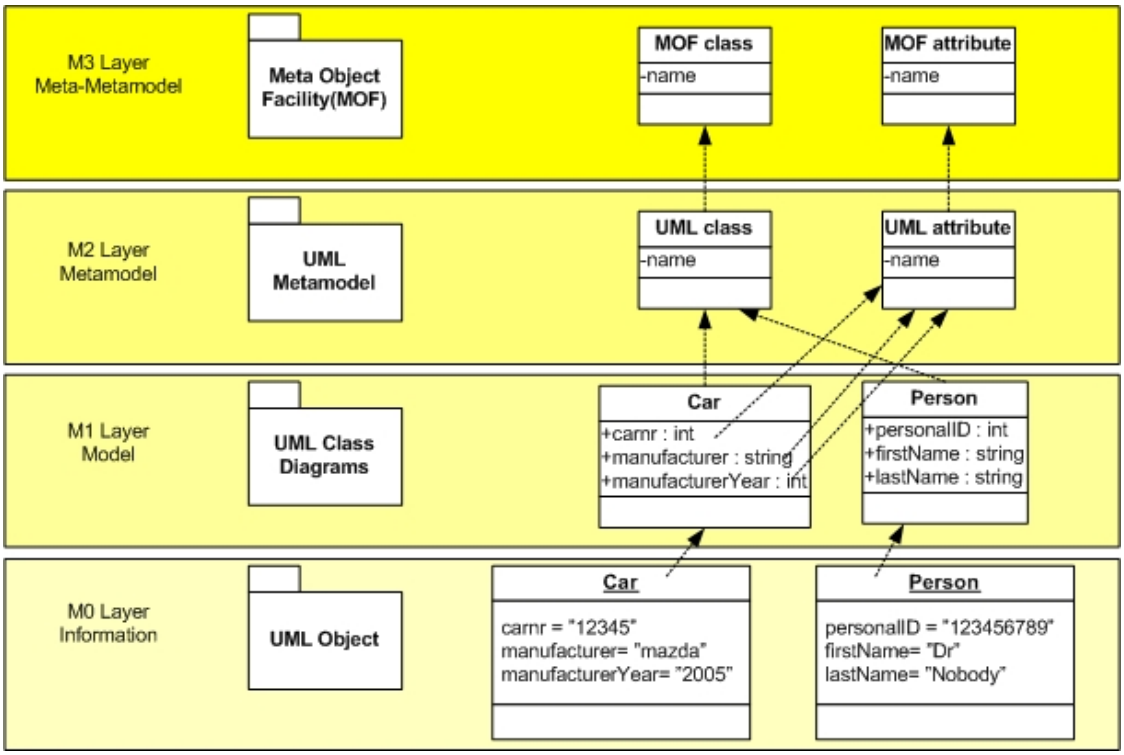


Figure 3-3 Metadata architecture

M3 Layer: Meta-Metamodel

The meta-metamodel is the infrastructure for a metamodeling architecture which defines the language for specifying metamodels. The MOF meta-metamodel is used to define the metamodel UML, e.g. the metaclass MOF class defines the UML class. MOF links the gap between different metamodels by providing a common basis for metamodels. In the case where two dissimilar metamodels are both MOF-conformant, then models based on them can reside in the same repository.

M2 Layer: Metamodel

The metamodel is an instance of the meta-metamodel which defines the language for specifying a model. The UML metamodel is used to define UML models, e.g. class *Car* is

defined with its properties *carnr*, *manufacturer* and *manufacturerYear*. The Car class is and instance of UML class and its properties are instances of UML attribute.

M1 Layer: Model
The model is an instance of the metamodel which defines the language to describe an information domain. The UML model is used to define aspects of a computer system. E.g. The Car class specify how the information or instances of this class would look like.

M0 Layer: Information
The actual information exists on this level and is an instance of model which defines a specific information domain. E.g. the car "Car" has the carnr "12345", manufacturer "mazda" and manufacturerYear "2005". There are usually many car instances, with their own data.

MOF defines a model as an instance of a metamodel which can describe properties of a specific platform. Figure 3-4 shows the relationship between a model, metamodel and platform.

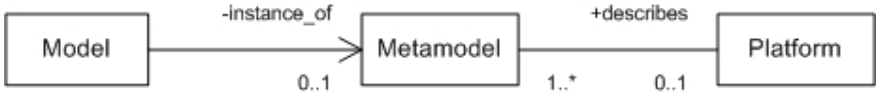


Figure 3-4 Relationship between a model, metamodel and a platform

An issue to note is that the layers of modeling give an understanding of the relationship between the various OMG standards used within the MDA framework. In addition, to defining modeling languages, the MOF is used to enable building of tools for defining modeling languages. Some of the additional functionality provided by MOF is a MOF repository and model interchange.

3.4.3 MOF repository Interface – Java Metadata interface (JMI)

MOF also defines a framework for developing repositories containing metadata, such as models described by metamodels. This framework provides for transformation of MOF metamodels into metadata API's by using standard technology mappings. This gives compatible and interoperable metadata repository APIs for various implementations technologies. The interface for a MOF repository makes it possible to get information about M1 models from a MOF-based repository. This interface is usable in many environments, particular for Java called *Java Metadata Interface* (JMI). The development of JMI is being led by Unisys and includes the participation of Sun Microsystems, Hyperion, IBM, Oracle, and a number of other industry leaders [46].

JMI provides a platform-independent infrastructure for modeling, representing and querying the meaning of a data source's metadata, application, tool, and data integration can be improved. In addition, it provides a metadata framework that has the ability to capture the semantics of data and a common semantic model for describing metadata, a common Java-

based programming model for handling metadata and a common interchange format for exchanging metadata.

JMI is a mapping of the MOF model to the Java Programming Language. JMI basically take a MOF metamodel and generates Java interfaces that can be used to access model instances at run-time. This means that a Java implementation of any MOF-based metadata service can expose both the generic and metamodel-specific interfaces derived from the MOF's interface mapping rules. Java clients have completely portable access to metadata services via JMI. JMI provides an easy mapping from a MOF-compliant data abstraction, usually defined in UML, to the Java programming language. All the Java interfaces for metadata access are automatically generated from the information in the metamodels. In the case where metamodels are in change the interfaces will automatically be changed to reflect it.

The implementation of the JMI specification facilitates the integration of applications, tools and services. JMI helps to reduce the complexity of interoperability and data integration A JMI implementation allows for the generation of pure Java interfaces for programmatic and XMI-based access to repository-based MOF metamodels and their instances. In addition, MOF is used to define file-based interchange format for M1 models.

3.4.4 MOF and interchange – XML Metadata Interchange (XMI)

MOF defines a standard way to generate an interchange format for models in the language defined using a metamodel described in the MOF. This interchange format is an OMG standard called *XML Metadata Interchange* (XMI) [47]. In addition, XMI can be used to generate standard interchange formats for metamodels as well, since MOF is defined using itself. XMI is a standard for representing models and exchanging models in a standardized format using XML technology. XMI provides mechanisms to define, to manipulate and to interchange XML data and objects. XMI is mostly used as an interchange format for UML models. It specifies how to create XML Schemas for UML models. This articulates the value needed to overcome some interoperability problems.

Figure 3-5 shows how a UML model is generated to an XML Schema by XMI. Further it shows the relationship between XML document and XML Schema. The XML document is validated according to the XML Schema.

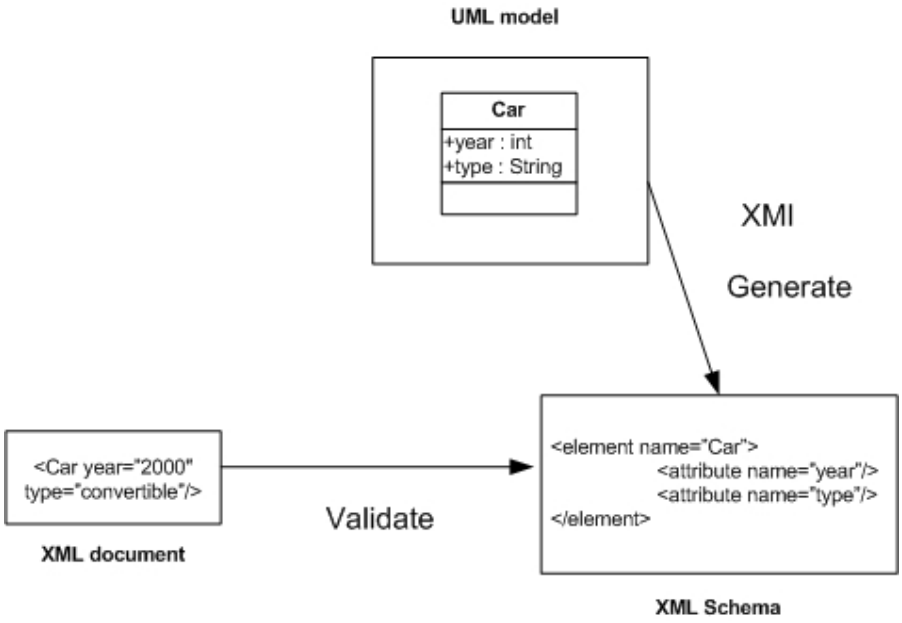


Figure 3-5 Relationship between UML model, XMI, XML Schema and XML

XMI uses XML technologies and enables to use modeling with XML. Further, there are available software that supports XMI to create schemas from models and XMI to provide a higher level of abstraction than XML elements and attributes. All tools that support XMI can read models made in different tools. Besides, XMI helps to produce XML documents that can easily be exchanged and enables to create simple documents and make more advanced ones as an application evolves. In addition XMI enables to tailor the XML representation of objects and document choices in the models and one to work with data and metadata. XMI can be used for any metadata whose metamodels can be expressed in MOF. However, there are some drawbacks with XMI. Visual diagrams can not be interchanged through XMI, only the information from the diagrams is saved. For this reason, XMI-based models are appropriate for interchange between, e.g. a drawing tool and a code generator or model checking tool, but not to exchange diagrams between picture-based tools.

A standard specification of a language suitable for querying and transforming models which are represented according to a MOF metamodel is at the time of writing, currently in the standardization process [48].

3.4.5 Query, View, Transformation (QVT)

The standard specification, called *Query, Views,* and *Transformation* (QVT) [11] is supposed to be standard language to write transformation definitions. QVT is a part of MOF since it addresses the way transformations are achieved between models whose languages are defined using MOF. It is suppose to be a language for creating views on a model, querying on a model and writing transformation definitions. QVT is concerned about navigation across related models and propose a unified approach to model transformation. In Figure 3-6 an overview of the transformation part of QVT is shown.

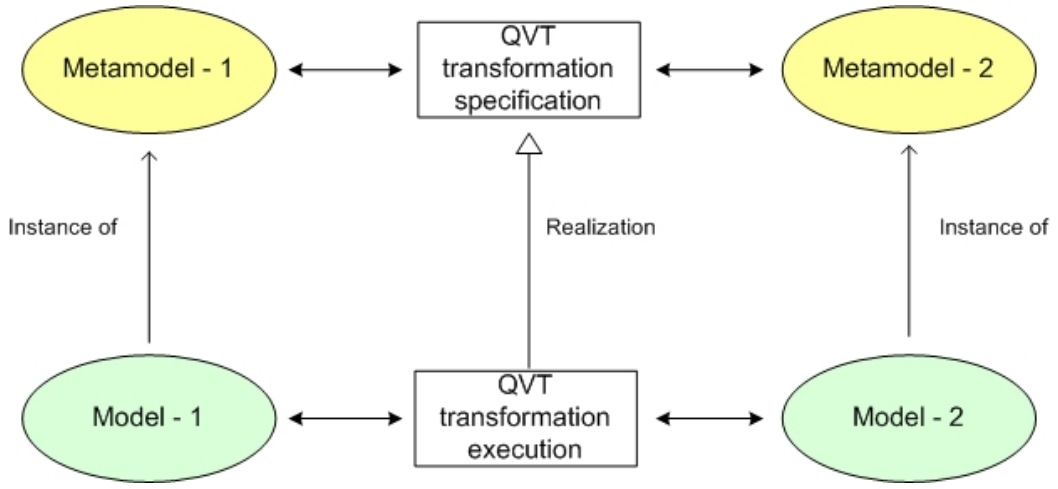


Figure 3-6 Overview of transformations

The fundament is to enable the possibility of defining a transformation between two metamodels, and in addition use QVT as input to a transformation engine that will perform a transformation from a model according to one metamodel to another model adhering to another metamodel. Both metamodels need to adhere to the same meta-metamodel, usually MOF.

Atlas Transformation Language (ATL)

An example of a QVT-based transformation language is the *Atlas Transformation Language* (ATL) Engine [49]. The engine is, at the time of writing, currently available as open source under Eclipse *Generative Model Transformer* (GMT) [50]. ATL is developed as a set of Eclipse plug-ins, and works as a development *Integrated Development* Environment (IDE) for transformations, including execution and debugging. ATL is a combination of declarative and imperative constructions; a hybrid language. It is intended to express model transformations as required by the MDA approach to answer the QVT *Request for Proposal* (RFP) [48]. RFP is one of a series of RFP's associated to developing the next revision of the OMG MOF. An abstract syntax such as MOF meta-model, a textual concrete syntax and additional graphical notation describes ATL. This permits modellers to denote partial views of transformation models. ATL expresses a transformation model as a set of transformation rules. For example, it is possible to write transformations from PIM to PSM. In addition, ATL provides for a repository of models. It supports creation of a library including transformations ranging from uncomplicated examples to reusable components.

UML Model Transformation Tool (UMT) – QVT

An open source tool implemented in Java to support MDD and MDA is UMT-QVT [51]. It is a tool for model transformation, and code generation of UML/XMI models. The tool enables new generators to be plugged in, where the generators are either implemented in Java or

XSLT. It is important to specify that this tool is not an implementation of the approaching OMG QVT standard.

In addition to standard formats and standards for integrating metadata, there are technologies or mapping tools available aiming to integrate heterogeneous data. Mapping is a complicated activity that has been an object of study in various research domains, including distributed databases, digital libraries, and schema integration. The mapping process is about mapping the content of one component to another component. It allows translation between specifications. During mapping, a source specification is compared to a target specification resulting in a collection of partial mappings between elements of the both specifications. BizTalk [52] and Altova MapForce 2005 [53] are examples on these kind of technologies. These technologies support integration of data over the Internet using among other factors XML.

3.5 BizTalk

As earlier mentioned, XML shows some limitations when it comes to integrating different enterprise systems. XML is mostly useful when agreeing on how data should be described. In some cases the document needs to be transferred and received in a commonly recognized format available on all platforms.

BizTalk is an initiative by Microsoft [54] that aims to create a database for XML-based document formats. It is a server which integrates separate enterprise systems together to create a larger system in a B2B environment. BizTalk carries text structured in XML between enterprise systems. The XML document is exchanged by two BizTalk servers over a network. Instead of writing integration code into the system itself, the BizTalk server operates as a messaging hub and allows keeping the integration code on its central server. Figure 3-7 shows the overall BizTalk architecture, taken from [55]. It illustrates XML's role for exchanging data between enterprises systems. The enterprises' systems manage documents in their own formats on their own platforms. Despite architectural differences data can be exchanged with XML.

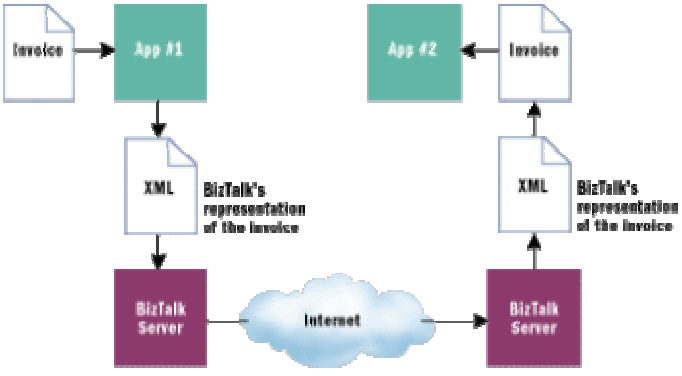


Figure 3-7 BizTalk architecture

In addition, the BizTalk server supplies a tool for translating between message specifications. The tool is called BizTalk Mapper [56].

3.5.1 BizTalk Mapper – a data integration tool

The BizTalk Mapper is a translation design tool which allows for displaying two specifications or XML documents, the source and the destination. Further it gives a programmer permission to specify how records and fields can be mapped to each other. This facilitates data integration between two message specifications with different syntax.

Figure 3-8 shows the BizTalk Mapper tool and it is taken from [57]. The tool displays a mapping diagram on the top. The diagram displays three panes: on the left and right panes tree representation of the source and destination message specification are displayed and a mapping grid in the middle. The lines displayed in the mapping grid between the source and destination message specification are linking the source fields to destination fields. A link is established by dragging a field or record in one specification to the appropriate field or record to the other. This way simple relationship between fields is mapped. The content of the source field or record is copied into the incoming message into the mapped field or record in the destination message specification. The mapper does not only allow for one-to-one relationships, but also one-to-many relationship or reverse. Besides it facilitates data processing during mapping.

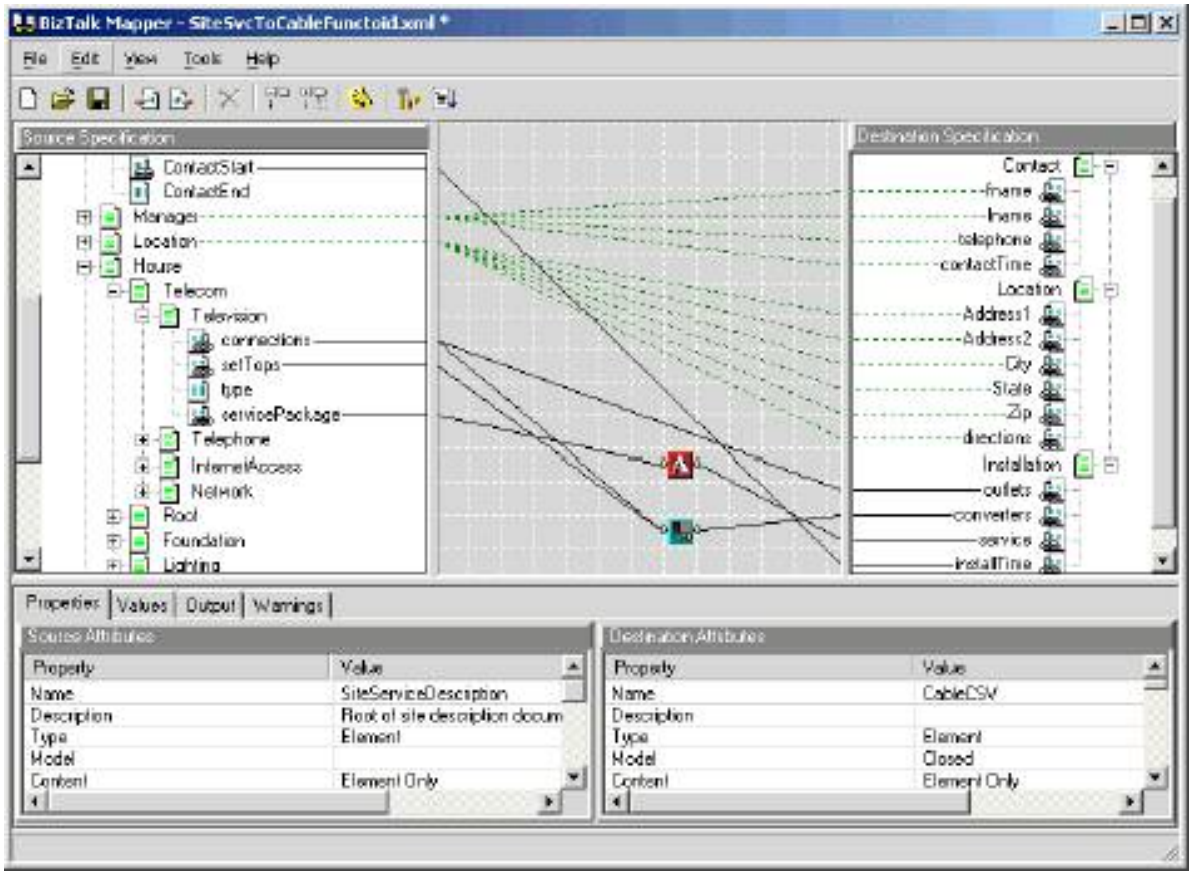


Figure 3-8 BizTalk Mapper interface

BizTalk mapping operations

BizTalk Mapper offer solutions for several mapping scenarios. According to [57], the complexity of a mapping scenario principally depends on an enterprises preferences and business need. The mapping scenarios basically comprises of two mapping categories: *Basic mapping* and *complex mapping*.

- **Basic mapping**: Input and output items have a one to one relationship where they are mapped to each other. Even though several types of transformations and translations are possible with basic mapping, for instance use of multiple functoids and cascading functoids to manipulate the value being copied. In addition, these mapping operations comprise mapping fields from two different parent records to fields under a single parent record in the destination specification (or schema). This enables mapping between data at different aggregation levels, data referring to the same with different terms.

- **Complex mapping:** this mapping is taken into account in the case where records or fields occur several times for a single instance of the record or field element.

Some of the processing that arise during mapping can be complicated. For example, several source fields may need to be combined to form the contents of the destination field. Another issue might be when some data processing may need to be performed on the contents of the source specification field to produce the required contents of the destination field. This solves among other factors the problem with differences in properties described in section 2.5, Table 2-7.

The BizTalk Mapper provides two ways to introduce intermediate data that is processed during the mapping process: *functoids* and *scripts*. Functoids are functional objects that perform simple predefined operations, such as string manipulation or mathematical operations. They range from simple calculations to elaborate script functionality. Short user-written scripts are executed by the script functoid. This allows for more complex data processing. In Figure 3-8, the box containing A in the middle pane is the way functoids and scriptlets are represented in the mapping grid. The BizTalk Mapper uses a *map* to graphically represent the structural information relationship between the specification data elements. The map provides a set of instructions that defines the relationship. When the mapping is completed, a programmer compiles the map using the Mapper. The BizTalk server uses the data provided in the map to generate XSLT. This is shown in Figure 3-9, also taken from [57].

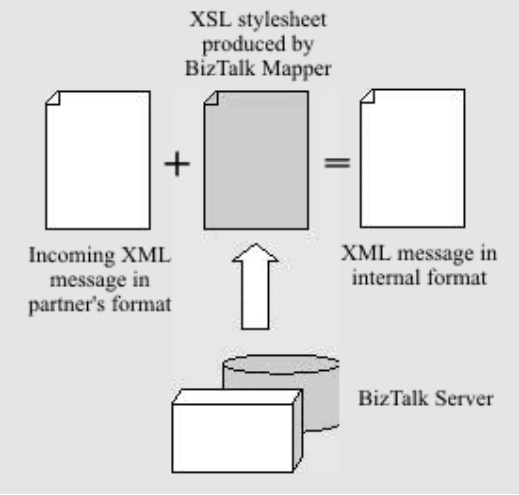


Figure 3-9 Transformation process for mapping

A facility which the BizTalk Mapper does not support is manipulation of generic XML files or non-XML files. In the case there is a need to translate between two generic XML files, it is necessary to import them and save them as specifications into BizTalk Mapper.

The BizTalk Mapper provides for interoperability by allowing for data integration between XML files. It is not adequate enough according to the requirements defined because it is single-vendor, and single-platform. Also, it does not have support for mapping between other formats than XML. Another tool for data integration is the *Altova MapForce 2005* [58]. Different from the BizTalk Mapper it supports mapping between other formats besides XML.

3.6 Altova MapForce 2005

Altova MapForce 2005 (MapForce) is a mapping tool for exchanging data between XML, database, flatfile and *Electronic Data Interchange* (EDI) platforms. It can generate custom mapping code in different programming languages such as XSLT 1.0 and 2.0, XQuery, Java, C#, C++. It supports various mappings such as Schema-to-Schema mapping, Database-to-Schema/XML mapping and vice versa, Database-to-Database mapping, Flat file mapping (CSV and text files). Figure 3-10, taken from [58], illustrates the MapForce architecture.

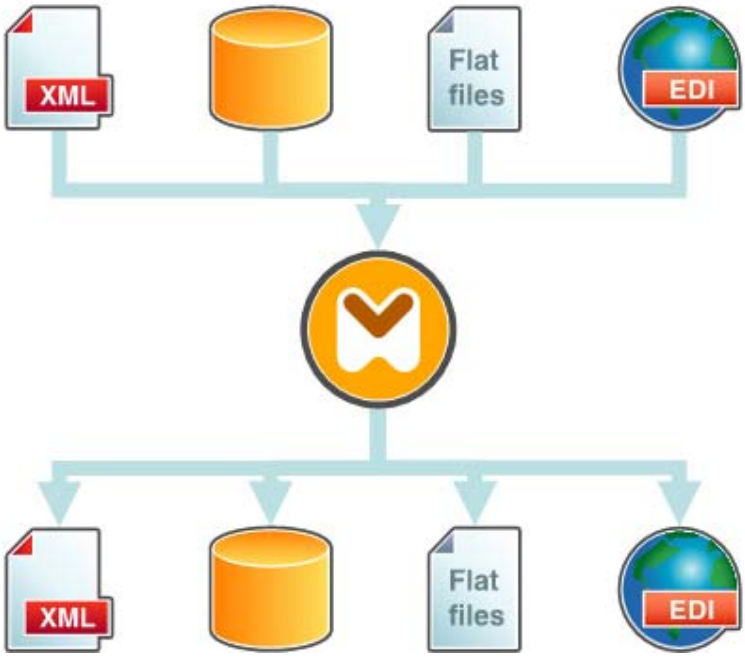


Figure 3-10 MapForce mapping tool architecture

An example on mapping between two different formats: For instance, an XML can be mapped to a different target XML document or database. The mapping is accomplished by an automatically generated XSLT 1.0 or 2.0 Stylesheet, the built in MapForce engine, or generated program code. A *source schema* is mapped to a *target schema* by connecting their elements or attributes, when creating an XSLT transformation. In fact one end up mapping two XML documents, since an XML document instance is associated to and defined by a schema file. Figure 3-11, also taken from [58], illustrates MapForce with Code-Generation.

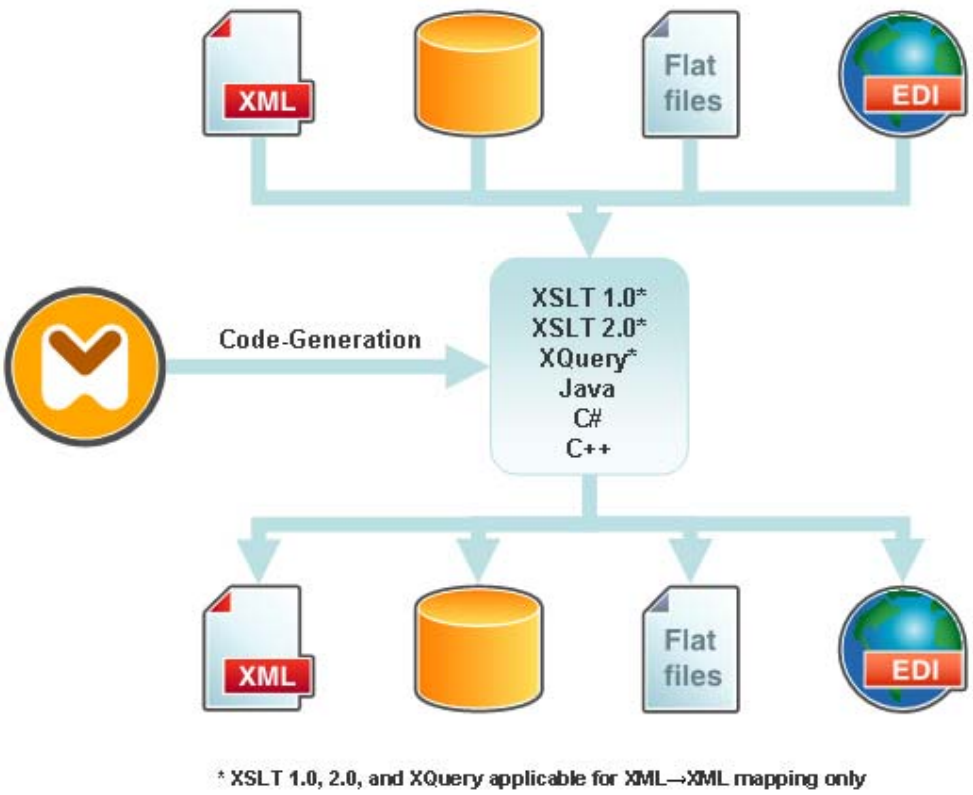


Figure 3-11 MapForce with Code-Generation

3.6.1 MapForce mapping tool

Figure 3-12 is taken from [58] and shows the MapForce mapping tool which consists of four main areas: the *Library pane* at left, the *Mapping Project* tab at right, *Overview* and *Validation* panes below. Language specific and user defined libraries are displayed, and individual library functions are displayed in the Library pane. These functions can directly be dragged into the Mapping Project where the actual mapping process is achieved. The graphical elements that are used to create the mapping between two schemas are displayed in the Mapping Project tab. A preview of the mapping depending on the specific language selected is displayed in the XSLT tab and a preview of the mapping or the mapped data in a text view is displayed in the Output tab. The mapping area is displayed as a red rectangle in the Overview tab. Any validation and warnings or error messages that might occur during the mapping process are displayed in the Validation pane.

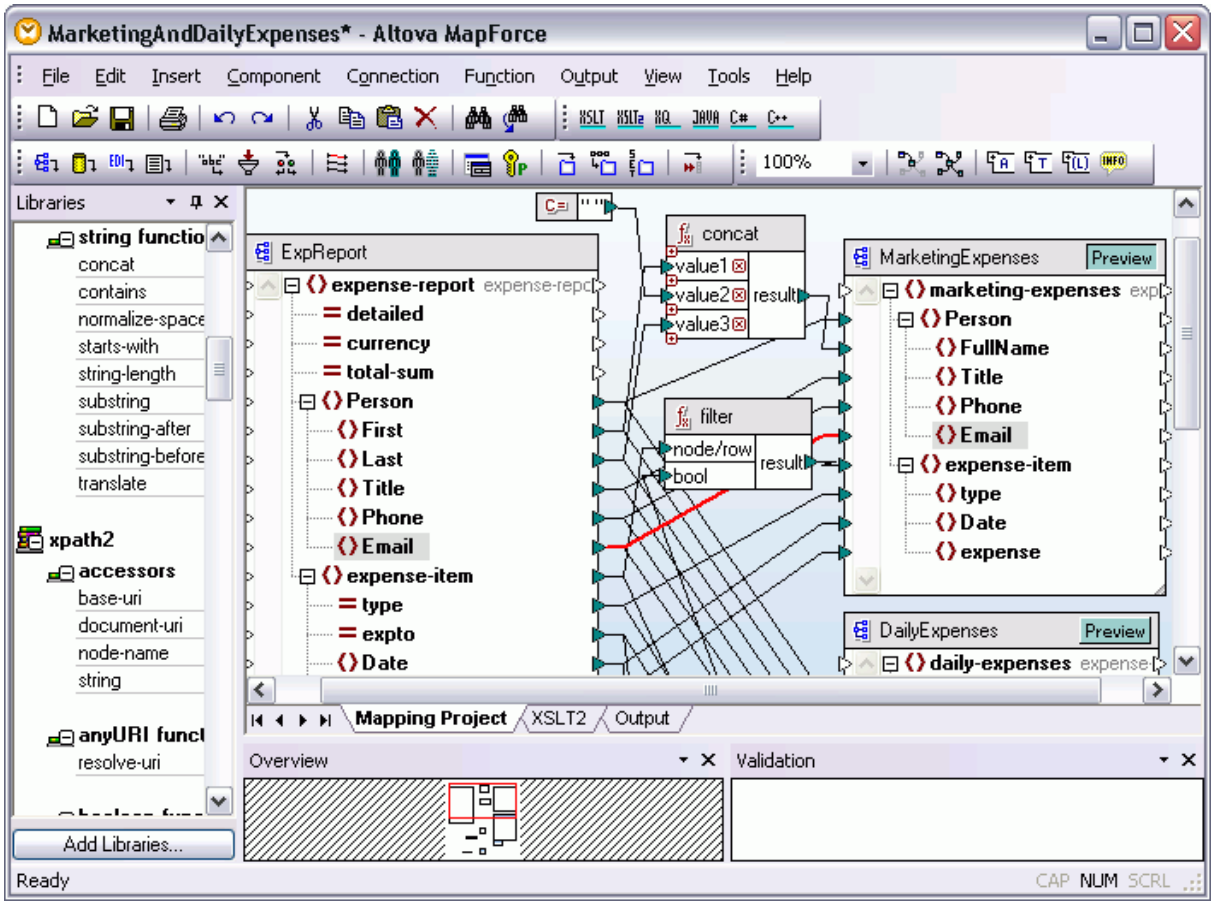


Figure 3-12 MapForce mapping tool

MapForce data processing functions

MapForce supports for defining custom data mapping functions between source and target files. It provides an extensible library of *data processing functions* for manipulating data according to the needs of users' data integration application. Additionally, it has a unique *visual function builder* for defining custom functions that are able to combine multiple operations. It is possible to transmit the output of one function into the input of another function, chaining them together before finishing the data transformation. These kinds of functions can be saved and reused via the visual function builder. To use a data processing function it is simply to drag en drop the wanted function from the function library. A user connects the desired elements from the source file to the data processing function as inputs. Further, the output of the processing function is connected to the target file.

By providing a unique approach to enterprise data integration, MapForce solves some of the general data integration problems defined in section 2.5, Table 2-7. It handles data conversion scenarios with aid of a comprehensive data mapping function library. The library allows for defining mapping rules based on conditions, including string operations, mathematical computations, any user defined operations etc.

Altova MapForce 2005 addresses the synonym problem; the condition is that one has to know which one of the terms are synonyms. Altova MapForce 2005 has a function called *Auto*-mapping. This function allows one to automatically connect child elements with identical names in both schemas. (Child elements are referred to elements inside a complex type or inside a class) This function hinders the ability to detect if these elements are homonyms. Differences in properties are handled with concatenation and constant functions. In the case where data is missing, Altova MapForce 2005 uses a filtering technique. This technique is about filtering out elements which are not available in the target schema, and only passing those elements that are matching. Another service by MapForce is allowing one to produce default values to use for a particular element in the case an output of a mapping is null or absent.

3.7 Evaluation of related technologies

Since MOF- metamodels provide for defining, manipulating and integrating metadata and data in a platform independent manner, it can support metadata enrichment. BizTalk and MapForce are data integration tools which aim at integrating data between two enterprises. However, they emphasise on integrating syntax and structure of data. The data integration tools are poor at availability of data semantic. Both of the tools use a tree representation to represent the specifications to be mapped. This makes it difficult to see the relationship between elements/classes in the specifications.

Both BizTalk and MapForce enables mapping between synonyms, but the condition is that it has to be known which one of the elements are referring to the same semantic concept. MapForce provides for automatic linking, which is a hinder for recognizing homonyms. Also, BizTalk does not provide for recognizing homonyms, e.g. through annotation. This needs to be known in advance. BizTalk and MapForce provides for solving data integration problems concerning differences in properties. In addition, BizTalk and MapForce allows for mapping at different aggregation levels. The data lacking problem can be solved. However, in the case where an enterprise needs to receive information which it does not have support for it can get difficult. However, MapForce provides a mechanism for users to filter out information from certain data. For another case it provides the use of default values for a particular element in the case an output of mapping is null or absent.

XML can be used as a standard for exchanging information, but where XML is without agreement upon semantics associated to data and tags, it does nothing to support integration except for providing common syntax. XSLT can be used to transform between different versions of XML documents. ebXML provides for a repository where enterprises can share business and use UBL as a standard business syntax. By using MOF as a shared metamodel can contribute in using a platform independent data model. Data integration can be supported through this data model, since ATL (QVT) can be used to write transformation definition between models. To solve the homonym problem, semantic annotation is required. Without knowing that data which are to be integrated are homonyms, the mapping can result in an inconsistent mapping. BizTalk and MapForce are data integration tools. UMT-QVT is a tool

for model transformation and code generation of UML/XMI models. In Table 3-1 the related technologies are evaluated according to the requirements defined in the previous chapter.

Table 3-1 Evaluation of related technologies

Requirements	Evaluation
Metadata enrichment	MOF can support metadata enrichment.
Synonyms	ATL, BizTalk and MapForce enables mapping between synonyms.
Homonyms	BizTalk and MapForce does not provide for recognizing homonyms.
Data representation conflicts	ATL can be used to write transformation definitions to specify this conflict.
Differences in properties	ATL can be used to write transformation definitions to specify this conflict. BizTalk and MapForce provides for solving problems concerning differences in properties.
Data precision conflicts	ATL can be used to write transformation definitions to specify this conflict. BizTalk and MapForce allows for mapping at different aggregation levels.
Default value conflicts	ATL can be used to write transformation definitions to specify this conflict.
Attribute integrity constraint conflicts	ATL can be used to write transformation definitions to specify this conflict.
Data lacking	MapForce has some solutions to solve this problem. ATL can be used to write transformation definitions to specify this conflict to some extent.
Platform independent data model	XML can be used as a standard which have support for data integrating by using XSLT. ebXML proposes use of UBL. MOF as a shared metamodel can contribute in using a data model and use the QVT as transformation languages between the models.
Tool support	UMT-QVT, MapForce and BizTalk Mapper.

3.8 Summary

In this chapter related technologies facilitating interoperability, and supporting data integration have been examined and evaluated. In the next chapter we present existing solution approaches to the problem examples introduced in chapter 2.

4 Existing solution approaches

In this chapter existing solution approaches are presented and analysed. These solution approaches are found within existing projects that propose solutions to problems defined in chapter 2, section 2.2.2 and 2.4.1. For this reason, our analysis is based on deliverable project documentation, and in addition on available public documentation about the solution approaches. The TOR[2] project is an existing solution approach to OR. The ATHENA project is analysed as a proposed solution to Automotive scenario case. At the end of this chapter the existing solution approaches are evaluated.

4.1 The TOR approach

The TOR project is a continuation of the OR project, and has in likeness with the OR project been initiated by the Brønnøysund Register Centre [59]. The TOR project intends to develop further on OR based on the problems mentioned in chapter 2. Additionally, the TOR project has the same aim as OR to minimize reporting workload for enterprises and citizens. Further, their aim is to increase the possibilities for reuse of data definitions by identifying identical information request from departments. By finding identical information, an enterprise can send the information to one department, and other departments can get the information needed from the department who collects it. This will release time both for enterprises and departments, and save expenses for the society. As presented in chapter 2, OR has the same intention to avoid multiple submission of the same data from enterprises, but their solution does not detect overlap. Different departments want to define data definitions according to their systems. For this reason, it is impossible for OR to detect identical information. Based on this, OR do not know which departments that need to collaborate to collect the same information.

The TOR approach proposes a solution to the problem concerning semantic not being separated from the format when defining data definitions. It suggests how to separate format and semantics of the information by using models. The two main goals of the TOR project is to simplify the creation of electronic forms for reporting, and provide for that it is again possible to do overlap detection for OR.

[2] The project is named after a Norse god

4.1.1 Using models to organize information and define data definition

Models are a central aspect for the TOR approach. The models are used to describe and define information. All of the information users operate on is in a model. The TOR project uses UML as a modeling language to describe their models. As UML is a huge language, TOR only uses a subset of UML. The subset of UML is shown in Figure 4-1 is taken from [60]. Description of this subset is given in Appendix C.

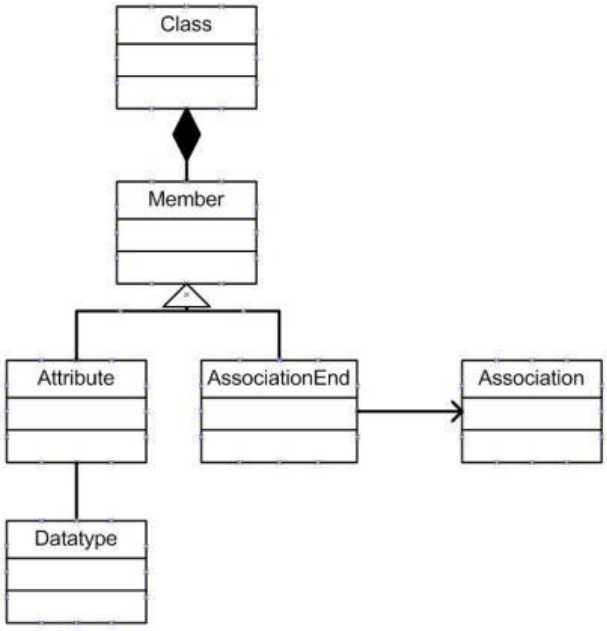


Figure 4-1 Subset of UML

The TOR project operates on the following models; information-, document-, blank form-, message and submitter model. Table 4-1 gives an overview and description of these models.

Table 4-1 TOR models

Modell	Description
Information model	Metadata are structured into an information model to facilitate retrieval by semantic navigation. This model is supposed to describe the information which is going to be reported and relationship between different elements. In other words, this model presents the meta information about the reported information.
Document model	The document model is a segment of the information model. Document models are used to limit the big information model so users making a blank form or a message only need to deal with a smaller part of the information model.
Form model	A form model is a model of a blank form under design. This model deals with formatting: where the different fields shall be, and how the form shall look like. This model consists of components that are put together to describe a blank form. Some of these components are fields that are going to be used for reporting information. These shall contain links to attributes in the information model. A form model can be connected to a document model.
Message model	A message model is a model of a message under design and is an enriched document model. A message model contains attributes. It is the connection between document model and form model.
Submitter model	The submitter model is used for internal reasoning in the system, and can currently not be serialized [61]. It gives information about who shall submit information, and at what time.

The TOR approach enables departments to reuse the information model or the pre-designed form elements. Further, Figure 4-2 shows how the models in TOR are related and their relationships to each other. The figure is taken from [61].

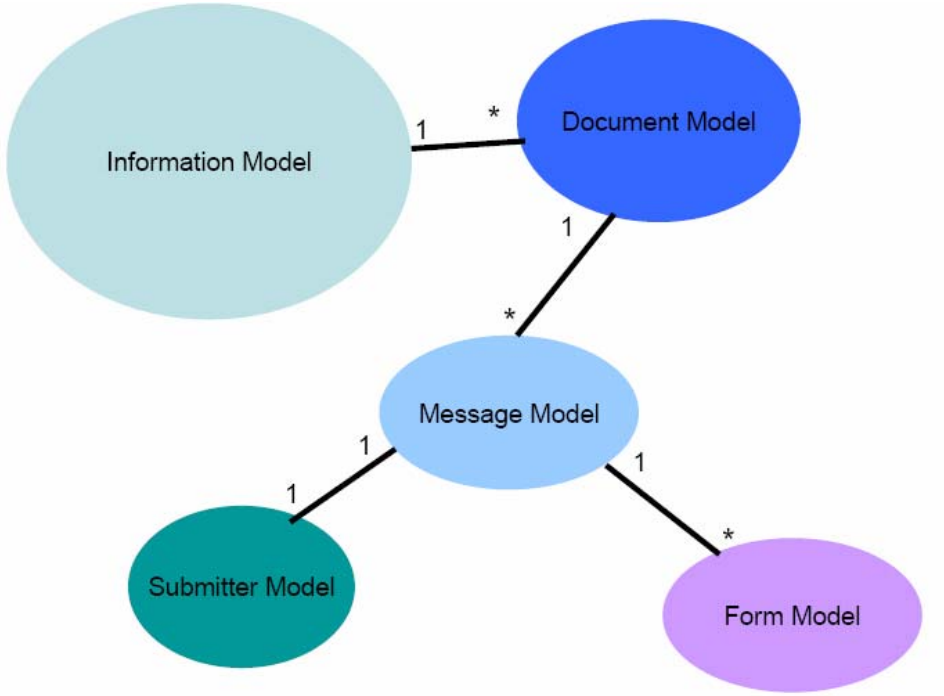


Figure 4-2 Models in TOR and their relationship

These models is supposed to facilitate the separation of syntax and semantic of the data. Further, it will be possible to detect overlapping information since they can define data according to semantics and the departments can choose to use the format that their respective system uses. The format is not taken into account when detecting overlapping information.

Furthermore, the TOR project proposes creation of a system, called TOR system. Next it is given a brief description on how the TOR system is build and what it does.

4.1.2 TOR system

The TOR system is based on the OR system, recall section 2.1.1. The TOR system's main funtion is the ability to detect overlapping information from same kind of enterprise. The system's goal is to achieve semantic interoperability. Semantic interoperability allows for understanding received data and creation of information. Further, technical interoperability is recommended as a necessity to achieve semantic interoperability. Technical interoperability allows for data transfer. Figure 4-3 is taken from [60] and illustrates semantic- and technical interoperability between two legacy systems.

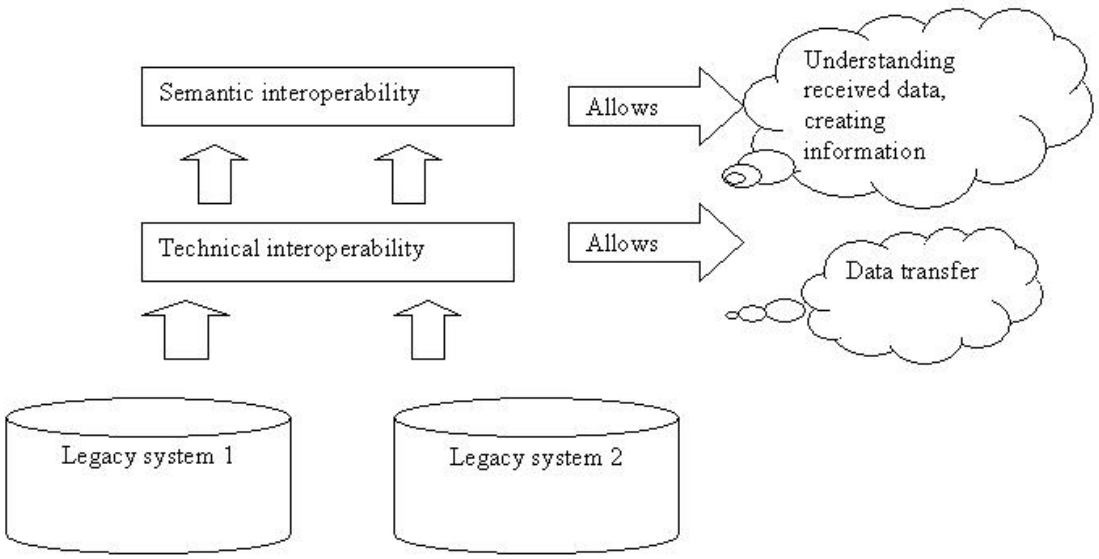


Figure 4-3 Semantic and technical interoperability between two legacy systems

TOR does not use the same architecture of the present OR system, but TOR intends to reuse the parts which are good designed, and add new functionality where necessary.

The TOR system is divided in the following parts; *ORsys, ORdb, TORnett, TORmodell, TORdesign and TORdb*. The description of ORsys and ORdb in this section is a refinement of their previous descriptions in section 2.1.1. Some parts of the present *ORsys* are kept, such as authorities, reporting obligations and statistics of load. These parts are covered by *ORdb*. The remaining parts are moved to the TOR system. *TORnett* authenticate and authorize users. It is

a portal where all users of the TOR system can access the system. Anonymous users, such as users without username and password, only have access to minimum sites. However, users with authorization will have the authority to login and access a major part of the functionality of the system. While OR uses one tool, ORetat, for information modeling and forms, TOR proposes to implement two separate tools; namely TORmodell and TORdesign. TORmodell and TORdesign have new and more demands to database storage. For this reason, there is proposed to separate new and old applications to prevent influence of the old systems' mode of operation by having a separate database for the TORmodell and TORdesign. This database is called TORdb. Figure 4-4 illustrates the proposed TOR system. The arrows indicate the communication between the different parts. The figure is taken from a deliverable of TOR project.

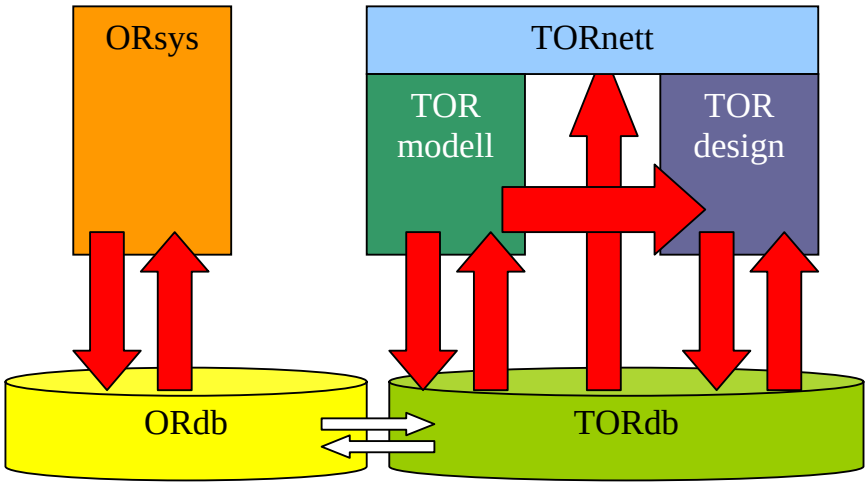


Figure 4-4 TOR system

TORmodell shall support processes of information modeling and TORdesign shall be used for assembly of forms. The connection between the two tools is through a binding in TORdesign from field elements to metadata elements in the information model. The TORmodell and TORdesign replaces ORetat's role as a modeling tool.

TORmodell

The semantic of the data definitions in the ORsys is moved to TORmodell. This tools main function is to give users access to the information model of the TOR system. Further, the TORmodell is supposed to be a representative of the different domain models contained in the information model. The UML like notation for TORmodell is suggested to be used both while viewing and editing the model.

The users of TORmodell will be able to see sections or views relevant to them. Views can also be transformed into a document model, and exported to an XML Schema. TORdesign is

supposed to use this as a basis to form design. The XML Schema documents can be of two kinds. The user can either export the whole model to an XML Schema document, or choose to export only a part of the model.

Another proposed functionality of TORmodell is version control of models, since models will most certainly change over time. Further, all classes have a version. The case where a class is being changed, the previous version remains unchanged. All past versions of the classes will remain in the system, but they will not be visible in the modeling tool. The advantage is that this allows for classes to be dynamic.

TORdesign

TORdesign is a tool to design blank forms, messages specifications and templates. The syntax of the data definitions (representation format) is moved to TORdesign. The tool is accessible from TORnett, and has a generated document specification from TORmodell as an input. The output from TORdesign is specifications for blank forms, and messages that are available for administration in TORnett. TORdesign is designed to work with information or document models that are generated by TORmodell. In addition, TORdesign creates message models.

TORdesign is supposed to take over the part where ORetat adds text. Those restrictions that are blank form dependent, and not a part of TORmodell is a part of TORdesign. Output from TORdesign, are XForm documents.

Another focus of the TOR system is to use open standards like UML, XML, XML Schema and XForm. Use of open standards will make it easy for departments and other users to implement their functionality or only some parts of the system. It will also be easier to change the tools used in some part of the system.

4.1.3 TOR and data integration

The project, at the time we analyze it, does not have support for integration between departments' systems and TOR. However, it is support for integration of information between departments. The message model created in TORdesign are supposed to be used as support integration. The information will be available at the departments through web services where the messages is represented on TOR message model format. The TOR approach have proposed some solutions on how to avoid multiple creation of same data definitions. The synonym problem is solved by using alias between attributes (data) and through re-definement in different contexts, typically inheritance. The problem concerning homonyms is suggested to be solved by using the same name and context. The representation is defined independent of semantics. It can be defined several representations for data with one semantic type. By defining standard values in each form model the default value conflict is supposed to be solved. To avoid data lacking it is suggested to exchange partial models (some parts of a model) between the departments. When a department requests information, the information shall be collected from several departments holding the information. This is supposed to be invisible for the department requesting the information. The information model is platform

independent, and it is suggested to use JMI to connect to internal systems and arbitrary platforms. These suggested solutions are supposed to simplify data integration between departments. At the time of writing, the TOR approach has not created a tool to support data integration.

4.1.4 Evaluation of the TOR approach

To achieve the goal about providing for the possibility to do overlap for, the TOR approach suggests to divide semantic and format. It proposes how to separate syntactic- and semantic data by adding new tools; TORmodell and TORdesign. At the same time, this has given the possibility to complete the domain models that are supposed to make it possible for departments to integrate data without adding proprietary solutions. According to [60] interoperability is maintained at two levels. How they handle semantic interoperability is described above, but at the technical level they suggest to use XML as a file format.

The TOR project uses models to describe the information reported from the departments. Further, in their solution approach, models are the core basis for describing data and metadata. The TOR project specifies how models can be used to solve interoperability problems. From their deliverables we understand how TOR intends to structure data and detect overlapping information. By realizing the TOR approach, OR would possibly be able to find overlapping information and which departments that need to collaborate. It is proposed to use a message model as a exchange format between departments to solve data integration problems. An evaluation of the TOR approach is presented in Table 4-2.

Table 4-2 Evaluation of the TOR approach

Requirements	Evaluation
Metadata enrichment	TOR project uses models to enhance semantic and syntax information of the data.
Synonyms	This problem is suggested to be solved by using alias between attributes.
Homonyms	By using same name and context, this problem should be solved.
Data representation conflicts	The representation is defined independent of semantics.
Differences in properties	Under development
Data precision conflicts	Under development
Default value conflicts	Default values can be defined in each form model
Attribute integrity constraint conflicts	Not handled in TOR
Data lacking	The data which is not relevant to the other department should not be sent as regards to personal information protection.
Platform independent data model	UML is used to describe the data, and the information model is platform independent. The document model is a segment of the information model and the message model is enriched document model.
Tool support	TOR project does not propose at the time of writing any tool to support data integration between the departments.

4.2 The ATHENA approach

In chapter 2 we introduced the ATHENA project, and additionally we presented our student scenario case: Automotive scenario (recall section 2.4). In this section we present the ATHENA architecture and analyse it as a proposed solution to the problem described in Automotive scenario.

4.2.1 ATHENA architecture

The ATHENA project is divided into three categories called *action lines*. These action lines have different research areas. Table 4-3 gives an overview of the ATHENA project. It is made up of six research and development projects, that are coordinated under the name *Action Line A*, and six community building activities called *Action Line B*. With infrastructure business and technical support functions, the entire ATHENA project is managed and provided through *Action Line C*.

Table 4-3 ATHENA project

PARTS	**Action Line A**	**Action Line B**	**Action Line C**
FOCUS	Technology	Socio-Economic	Project Governance
FIELD	Research and Development	Community Building	Management
	PROJECTS	**ACTIVITIES**	**C1:** Program administration **C2:** Scientific Co-ordination **C3:** Exploitation **C4:** IT Infrastructure
	A1: Enterprise Modeling in the context of Collaborative Enterprises. **A2:** Collaborative Business Process Execution **A3:** Knowledge Support and Semantic Mediation Solutions **A4:** Interoperability framework and Services for Networked Enterprises **A5:** Planned and Customizable Service-Oriented Architectures **A6: Model-driven and adaptive interoperability architectures**	**B1:** Community Creation **B2:** Knowledge Sharing **B3:** Business Interoperability Research **B4:** Dynamic Requirement Definition **B5:** Piloting including Technology Testing Coord. and Pilot Infrastruct. **B6:** Training	

The interest area in our thesis is the research and development project in Action line A that supports scientific and technological objectives of ATHENA. Figure 4-5 illustrates the architecture of ATHENA concentrating on the projects in Action line A. Project A4 represents the way to compromise and complement the approaches, methodologies and results of the projects A1-A6. We will focus on project A6 which is represented in bold in the table above. For more elaborate description of Athena and the other projects in Action line A see [31] or Appendix D.

Figure 4-5 shows two enterprises proprietary systems, A and B that are meant to collaborate by exchanging information. The red circle represented the A6 projects focus area. MDA is used as a strategy to achieve the collaboration between the two enterprises.

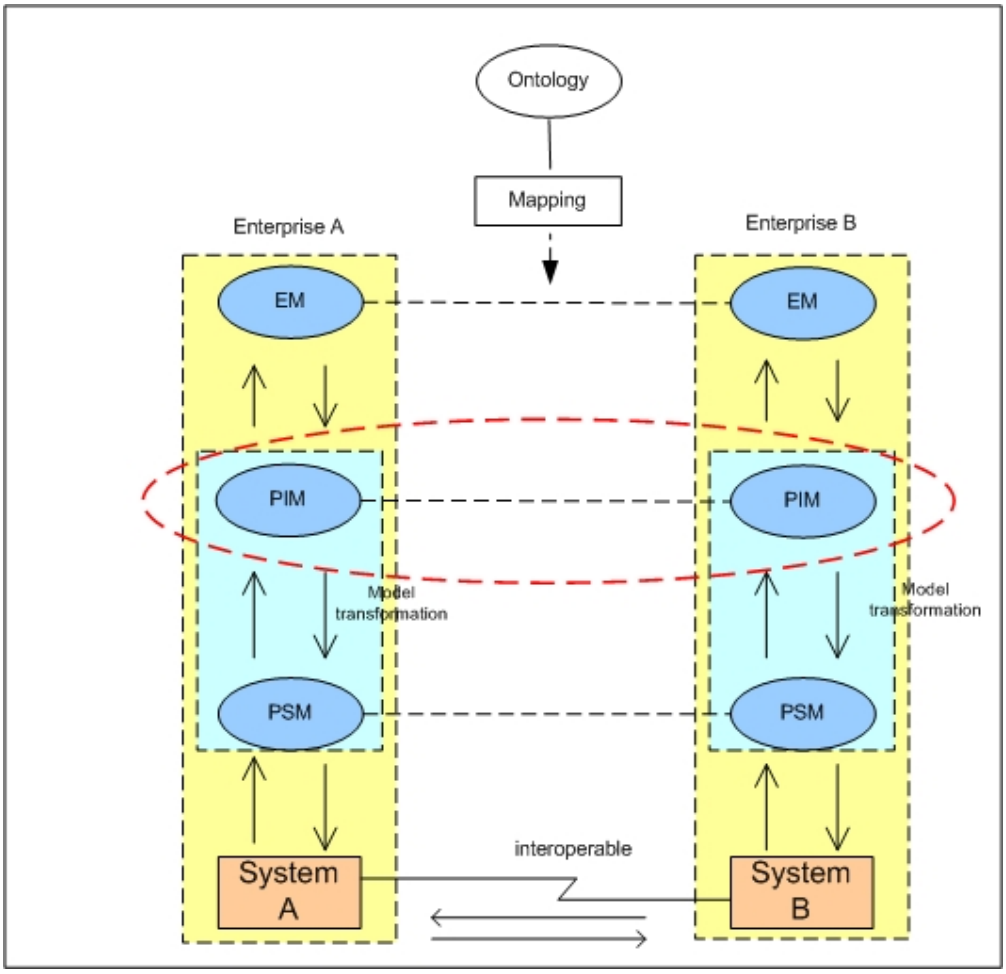


Figure 4-5 ATHENA's Action line A overview

Project A6: *Model-Driven and Adaptive interoperability Architectures* focuses on among other factors applying the principles of model-driven and platform-independent architecture specifications. Therefore we consider that currently results of this project can contribute to solve the problems described in the Automotive scenario.

4.2.2 Project A6: Model-Driven and Adaptive interoperability Architectures

Project A6: "Model-Driven and Adaptive interoperability Architectures" is being lead by SINTEF [62]. The main objective is how model-driven and adaptive architectures can be combined in developing dynamic and adaptive interoperability software solutions or architecture approaches. The objective of A6 is to provide new and innovative solutions for the problem of sustaining interoperability through change and evolution.

According to the deliverables, project A6 has the following goals:

- To support requirements and validate solutions for the involved sectors from projects B4, B5 and A4.

- To provide metamodels & methodologies for interoperability architecture solutions.
- To evaluate and extend multiple adaptive autonomous and federated architecture approaches, including those based on Agent and Peer-to-Peer (P2P) Technologies and the Model-Driven Architecture approach.
- To provide support for non-functional interoperability aspects, through a model-driven approach.
- To apply the use of ontologies and semantics to model and service Registry/Repositories for better semantic interoperability.
- To provide semantic mapping and mediation technologies.
- To provide executable frameworks and support for active models.

These objectives and goals are supposed to be achieved by integrating principles of model-driven interoperability architectures and adaptive interoperability architectures.

- *Model-driven interoperability* architectures deal with *design-time* aspects of system engineering. Model-driven development methodologies describe how to develop and utilise (visual) models as an active aid in the analysis, specification, design and implementation phases of an *information and communication technology* (ICT) system.
- *Adaptive interoperability* architectures centre on *run-time* aspects of system engineering. P2P technologies enrich an ICT systems with dynamic and adaptive qualities.

The A6 project emphasize on technical interoperability, looking into model interoperability and execution interoperability between two systems. It is an interoperability framework structured according to the *ATHENA Interoperability Framework* proposed in A4. A6 contains three reference models to describe and support the application of MDD of software systems. Figure 4-6 shows how this framework is refined in A6 with focus on ICT interoperability, and is taken from [2].

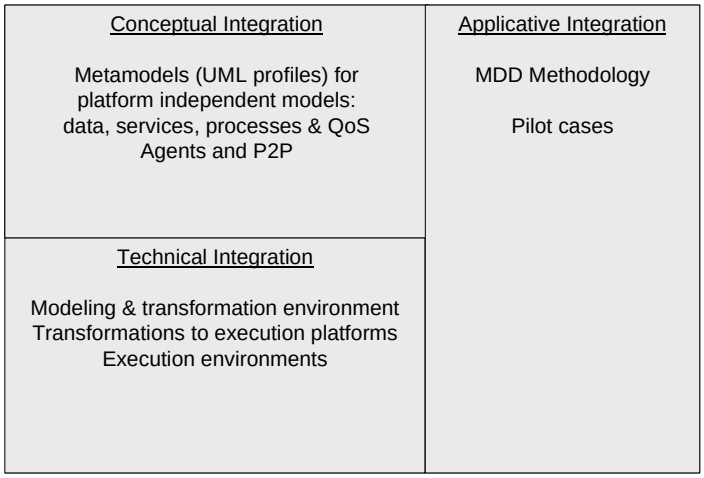


Figure 4-6 Athena Interoperability Framework focusing on ICT

Conceptual integration is concerned about concepts, metamodels, languages and model relationships. The reference model for conceptual integration has been developed from an MDD point of view focusing on the enterprise application and software system. Figure 4-7 shows the reference model for conceptual integration. A *computational independent model* (CIM) describes the business context and business requirements for the software system under consideration. The figure is taken from [2]. PIM describes software specification independent of execution platforms. Further, PSM describes the realisation of software system in a specific execution platform. The figure also shows how MDA and AMD approach can be used as a "top-down" or "bottom-up" approach to software development and integration. Models are used to describe different concerns of a software system. The models at the various levels may be semantically annotated using ontologies which help to achieve mutual understanding on all levels. Reference ontology will also help ATHENA to do model transformations and mappings between and across the three model levels. The ATHENA project proposes to use ontologies as a solution to solve the data integration problems [8]. An ontology is aimed at identifying and describing semantics of what exists in the real world. It is intended to use a shared ontology which captures consensual knowledge and is accepted by a group [31]. Semantic annotation of data to be exchanged is the foundation of the ontologies.

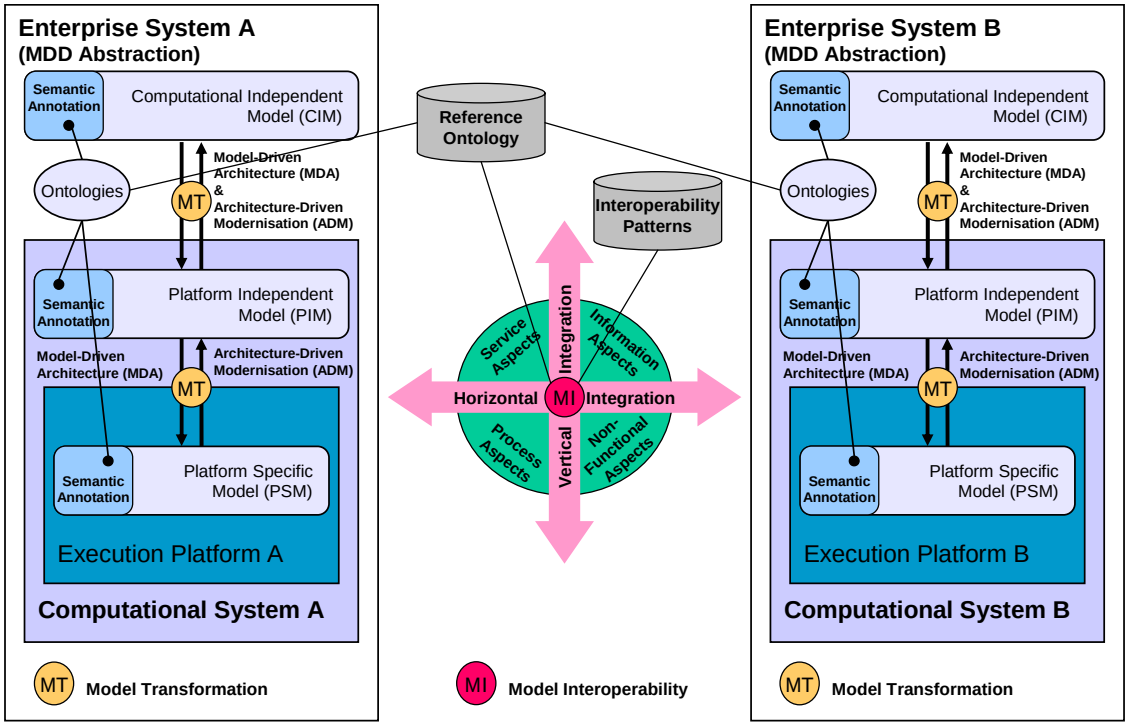


Figure 4-7 Reference model for conceptual integration

Technical integration focuses on the software development and execution environments. It provides development tools for developing software models and execution platforms for executing software models. Figure 4-8 shows the reference model for technical integration. The figure is taken from [2]. The software system is connected to a service bus which provides the necessary communication infrastructure that is required to deploy a distributed system. Further, a registry and repository will play an important role in integrating software systems. Service bus is used as an architectural pattern for handling technical integration of software systems.

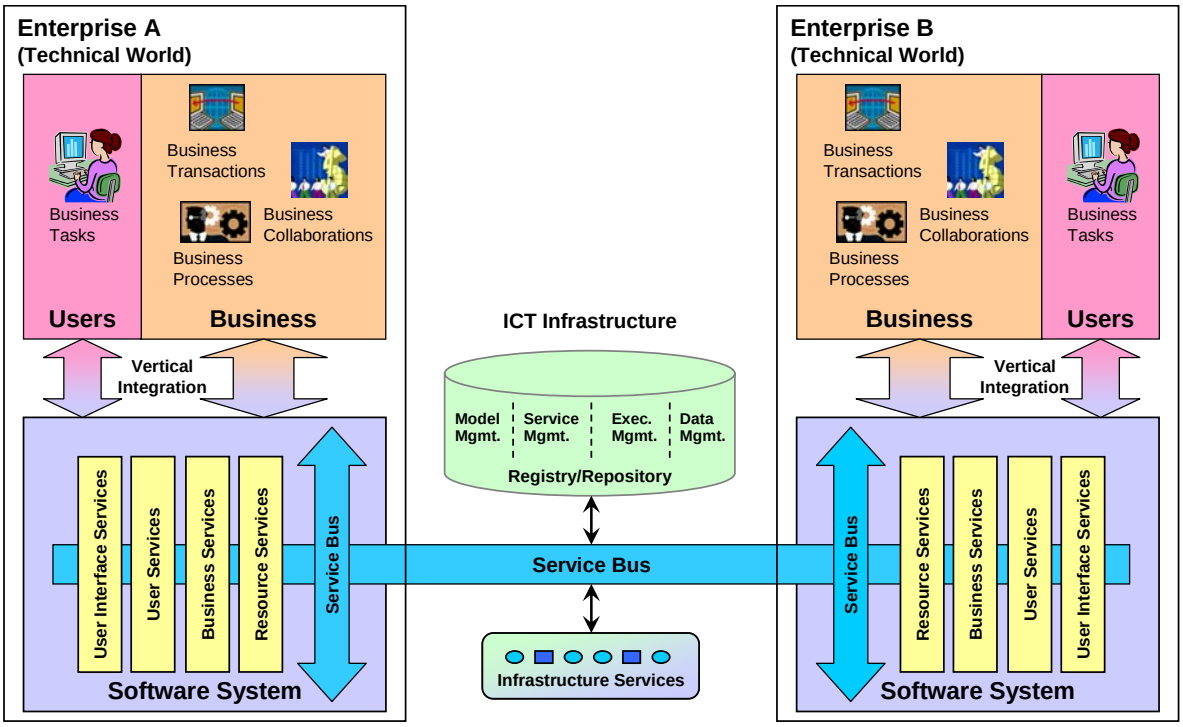


Figure 4-8 Reference model for technical integration

Applicative integration focuses on methodologies, standards and domain models. It provides us with guidelines, principles and patterns that can be used to solve software interoperability issues. The achievement with this reference model is how enterprise models and software models prescribed by enterprise modeling and software modeling approaches can be integrated into the overall framework. Figure 4-9 shows the reference model for applicative integration. The figure is taken from [2]. An enterprise model describes a set of enterprise aspects which includes descriptions of business operations which are referred to business models. These business models provide a context for the software solutions that needs to be developed and integrated, and thus needs to be reflected in the software model. Software models describe how software systems are used to support the businesses of an enterprise. The software model refines the business models in terms of software specification and

software realisation models. The software models can be classified as CIM, PIM or PSM models according to a MDD abstraction.

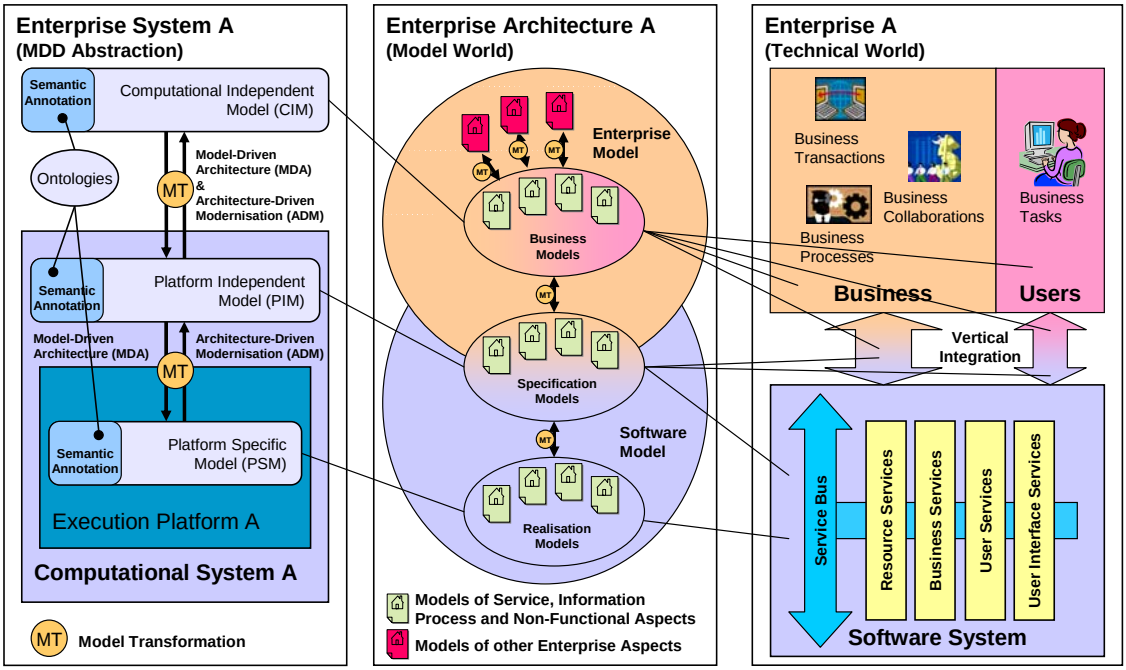


Figure 4-9 Reference model for applicative integration

Further, A6 proposes the use model transformation and model integration to achieve model-driven interoperability. The integration can either be done vertical or horizontal. Vertical integration follows the MDA and ADM approaches, and horizontal integration deals with mapping and transformation between models in the same abstraction level to ensure model interoperability. According to the deliverables model mappings can be defined using the metamodels and supported ontology.

4.2.3 Evaluation of the ATHENA approach

ATHENA project uses MDA where models are very central. ATHENA focuses on enterprise interoperability where organisational-, business- and technical interoperability is considered. They want to solve all of these interoperability areas. ATHENA deliverables proposes solution where models are being used to achieve interoperability. The A6 project specifies how model-driven and adaptive architecture can be combined in developing new interoperability software solutions.

The deliverables from ATHENA at the time of writing propose a solution to data integration by using ontology and interoperability patterns. They specify the data integration problems and proposes a solution to the problems, but are still developing techniques and tools to

support their solution approach. Models and ontology will together be a central part of the solution. The problems specified in the Automotive scenario case are intended to be solved by ATHENA's solution approach. Evaluation of ATHENA is specified in
Table 4-4.

Table 4-4 Evaluation of ATHENA

Requirements	Evaluation
Metadata enrichment	ATHENA proposes use of OMG's metadata architecture to achieve metadata enrichment. In addition to achieve correct semantic they use ontology.
Synonyms	ATHENA proposes use of models and ontologies to solve these data integration problems. The solution approach is under development.
Homonyms	
Data representation conflicts	
Differences in properties	
Data precision conflicts	
Default value conflicts	
Attribute integrity constraint conflicts	
Data lacking	
Platform independent data model	The models can be integrated at a platform independent level. ATHENA has decided to use UML 2.0 as the common language for describing the implementation neutral models.
Tool support	ATHENA , is at the time of writing, developing a tool support for mapping models between enterprise systems,

4.3 Summary

In this chapter we examined and evaluated two existing solution approaches. These are solution approaches to the two problems introduced in chapter 2. At the time of writing, the TOR project focuses on how to solve the metadata problem and does not concern on how to solve the data integration problems between departments. On the other hand, ATHENA is a solution to solve all integration problems between two or more enterprises. Both solution approaches direct in defined directions, and characterizes that a model shall be a central part of the solutions. To solve the interoperability challenges, models should be used to the extent they are being applied to represent metadata for semantic enrichment. In the next chapter we specify our proposed solution.

5 MODI Framework

Based on the underlying issues identified in the previous chapters, we propose a model-based approach to data integration based on MDA. The approach is intended to be independent of any particular implementation environment, but still intended to support implementation of a platform independent data model. The aim is to support information interoperability concerning data integration between heterogeneous enterprise systems.

The approach is an *interoperability framework*, called the *MODI Framework*. MODI stands for a MOdel-based approach to Data Integration, which means enabling heterogeneous systems to integrate data by using models. According to the European Communities [3] the term interoperability framework is defined as: "A set of standards and guidelines which describe the way in which organisations have agreed, or should agree, to interact with each other. An interoperability framework is, therefore, not a static document and may have to be adapted over time as technologies, standards and administrative requirements change."

The MODI framework, or MODI for short, provides a set of guidelines which specify how to develop tools that support data integration between heterogeneous enterprise systems. Also, the guidelines describe how the enterprises can use the tools, and the way in which the enterprises should agree to interact with each other.

The target group for the framework consists of two roles; namely a *developer* and *user*. The *developer* is a person who shall implement the tools to support data integration. This person should have a detailed understanding of the application domain, UML, MDA, XML technologies, platforms needed in order to produce quality software, transformation rules between source and target models. The developer also needs to know the transformation language, QVT. Also, the developer's responsibility is to define relationship between PSMs and PIMs, between PSMs and between PIMs. A *user* is a person (in an enterprise) using the tools for reverse engineering and model mapping. It is not intended that both cooperating enterprises use the tools for the same integration process. In addition, the user must have basic knowledge about UML.

Since it is a base for different modeling standards, we choose to use the UML as a modeling language for describing the platform independent data models. UML diagrams such as class diagrams, use case diagrams, activity diagrams, sequence diagram, collaboration diagram and state diagrams can be used to express or make PSMs and PIMs. The scope is business documents, expressed in UML class models and exchanged between enterprise systems. The main reason for selecting class diagrams is that these can be used to create a conceptual description of a software system. That is why we find it appropriate to use class diagrams to define PSMs and PIMs.

5.1 Principles of MODI

The way in which the enterprises should agree to interact with each other is through a PIM. These PIMs will have different structure, but at the same time independent of any technology. The aim is to integrate enterprises' (e.g. EnterpriseA and EnterpriseB) PIMs, then transmit EnterpriseA's instances to EnterpriseB's, and represent the instances in EnterpriseB's platform specific format. For this reason, EnterpriseA should be the tool user, since they are transmitting the instances. However, we have not considered the case where enterprises are transmitting instances both ways; from EnterpriseA to EnterpriseB and from EnterpriseB to EnterpriseA.

MODI guides the developer through implementation of a reverse engineering and model mapping tool. The reverse engineering tool (MODI Reverse) shall enable the user to parse the enterprises' platform-specific code into PSMs, and then transform the PSMs into PIMs. The model mapping tool (MODI Mapper) is supposed to allow the user to load PIMs, and then map the PIMs by using mapping rules (same as transformation rules). Further, the tool shall enable the user to generate code, and transform the instances. An engine (Mapping engine) shall generate the code and transform the instances. Figure 5-1 shows the process which is executed by using MODI Reverse and MODI Mapper.

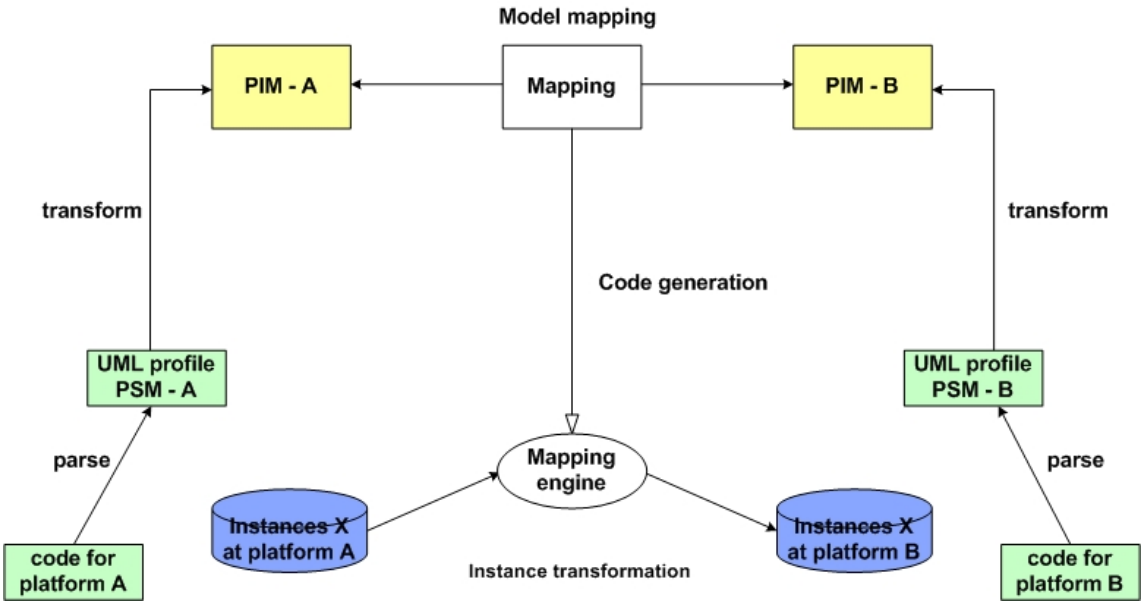


Figure 5-1 MODI process

In which order the tools are implemented is up to the developer. However, it is important to note that the model mapping tool is dependent on the reverse engineering tool. In particular, we assume that a user does not have their code represented in PIM, and need to use MODI Reverse to create a PIM before mapping with MODI Mapper. .

5.1.1 Eclipse – platform for tool integration

The MODI Reverse and MODI Mapper tools should be plugged into a platform for tool integration, such as Eclipse [63]. Our reason for choosing Eclipse as a platform is its plug-in based framework, and that it is a universal platform for tool integration. This way, Eclipse makes it easier to create, integrate and utilize software tools. Besides, Eclipse-based tools give freedom of choice in a multi-language, multi-platform and multi-vendor environment. Eclipse is language-neutral, which means that it permits unrestricted content types.

Eclipse architecture

Figure 5-2 shows the Eclipse architecture [64]. This plug-in based architecture makes it easy to extend the Eclipse platform or integrate tools, including own-developed tools, into the Eclipse platform by writing plug-ins. The core piece of Eclipse is a small kernel, and except for this part everything is written as a plug-in. The Eclipse Platform's functionality is a result of interactions between plug-ins and the kernel. A plug-in which is included with the standard Eclipse *Software Developer Kit* (SDK) for Java Development is a *Java Development Toolkit* (JDT). The Eclipse SDK is the combination of Eclipse Platform, JDT and *Plug-in Development Environment (*PDE) into a single download. The tools shown at the right in the figure are examples of tools that can be plugged into Eclipse. It is also possible for tools to be dependent of each other, like shown in the figure (Your Tool and Their Tool).

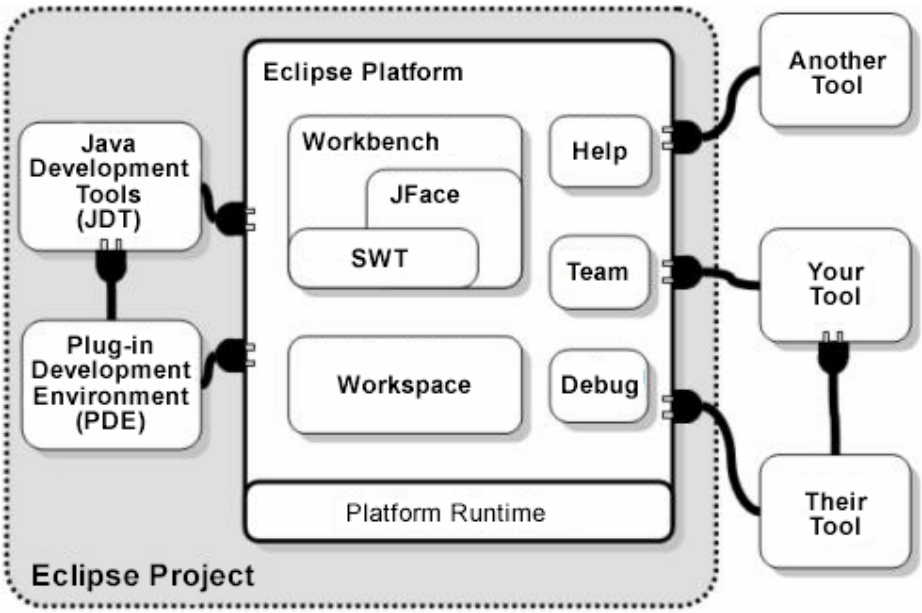


Figure 5-2 Eclipse plug-in architecture

A more compelling effect of the architecture is its meaning for open source in general. This implies that developers are no longer tied to a specific tool, product or closed license. In addition, it provides an industry platform for the development of highly integrated tools.

5.2 MODI Architecture

Figure 5-3 shows the MODI architecture in a component structure model which illustrates components. MODI Reverse and MODI Mapper are the tools we specify to support data integration between heterogeneous enterprise systems. The *Component Infrastructure* consists of components that constitute the basis for the tools. MODI Reverse tool component has a connection to two BusinessServices; namely *ParseService* and *TransformModelService*. These services are further connected to the ResourceService *CodeInfo*. The BusinessServices related to the MODI Mapper tool are *MappingService* and *EngineService*. The ResourceService *ModelInfo* has a connection to these services.

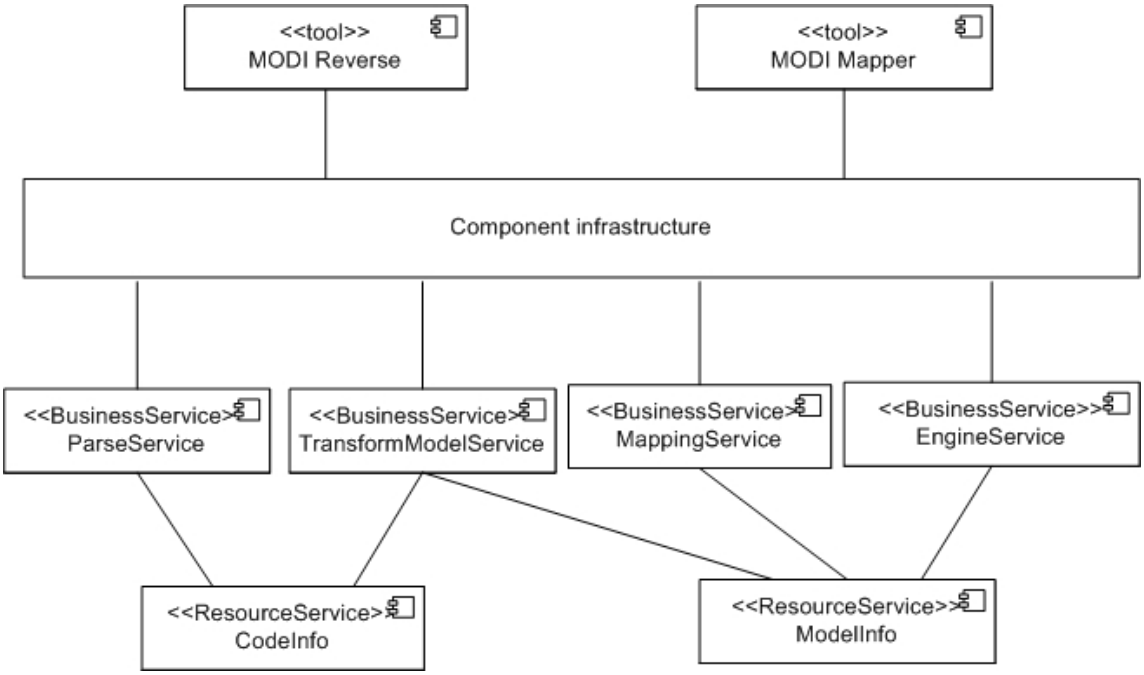


Figure 5-3 MODI architecture

5.2.1 Component Interface specification

The interfaces between the components defined in Figure 5-3 are illustrated in a component interface model shown in Figure 5-4. The component interface model shows through which interfaces the components interact with each other.

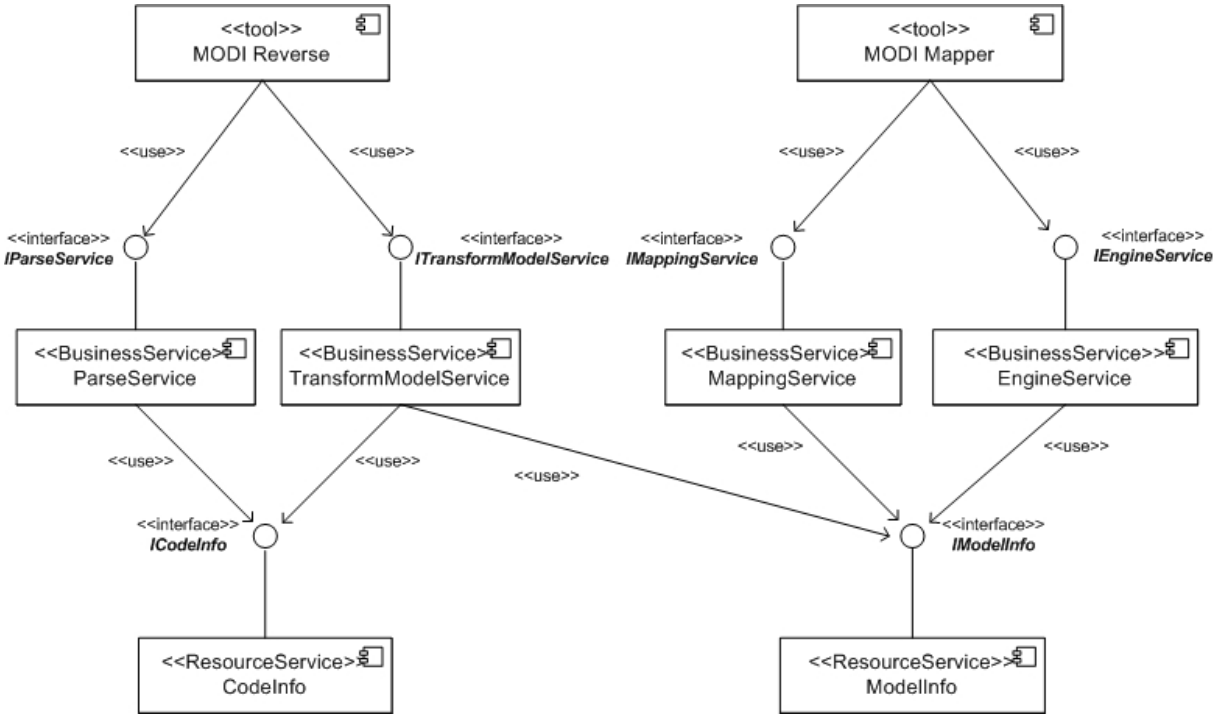


Figure 5-4 Component interface model

5.3 Reverse engineering – implement MODI Reverse

At the time of writing, OMG has not standardized any tool supporting the ADM approach. However, there are available reverse engineering tools, such as *UML Model Transformation Tool* (UMT) [51]. This tool has the ability to specify reverse engineering transformations from text/code to UML models and the other way around. An important component of a reverse engineering tool is a parser. A parser is a component which analyses, and reads the structure of a language, and then parses it into a particular format. Text-to-UML, UML-to-UML and UML-to-text are three main kinds of transformations supported by UMT. The transformers which have been specified and implemented within UMT are e.g. UML to J2EE, UML to and from *Web Service Definition Language* (WSDL). UMT uses transformations implemented in XSLT or Java. However, most of their transformations have been written in XSLT.

MODI Reverse is supposed to support text-to-model transformation, and allow enterprises using any proprietary format to achieve platform independency. The target users of this tool are those not having system code represented in PIMs. It is recommended to follow the reverse engineering approach ADM which extracts concepts from code, renders it into a PSM and then abstracts the PSM into a PIM. The PIMs should be on M1 level in OMG's metadata architecture. The components used in the MODI Reverse tool are described next.

5.3.1 MODI Reverse components

MODI Reverse tool component is a graphical user interface. Through this interface a user should be able to parse code, and transform models. This tool component is dependent on the components ParseService, TransformModelService and CodeInfo to make this possible.

The *ParseService* is a component that analyses, and reads the grammatical structure of a platform specific language. Typically a parser will contain a grammar component which searches through the language's grammar. The grammar can be expressed in *Extended BNF* (EBNF) [65]. Further, the parser shall produce an XMI file for this language, and a PSM (UML diagram) shall be created from this file. XMI facilitates the transmission of UML diagram data, and many UML tools support this interchange capability. However, only the data "behind" the diagram is currently transmitted, not the visual representation itself, so users of these tools still need to import transmitted XMI and then recreate the UML diagram.

The PSM will be an input to the *TransformModelService* component. This component shall abstract the PSM into a PIM. The platform specific data types shall be transformed into platform independent datatypes (expressed in UML). The output which is produced shall be a PIM. The code for platform specific format will be contained in the ResourceService component *CodeInfo*. It is from this service the ParseService shall get the input. CodeInfo is a repository for the specific platform.

5.3.2 MODI Reverse component interaction

Figure 5-5 shows these components' inter-dependencies, and how they collaborate in an interaction diagram. The user has an interface to the tool component MODI Reverse. Through this component the user should be able to choose platform-specific code used by the enterprise (EnterpriseA). The code should be retrieved by ParseService from *CodeInfo*. After CodeInfo returns the code, the ParseService reads the code structure. Then the user shall have the ability to define structure. By this we mean structure of the code that is being transformed. The reason is because transformation from code to PSM to PIM can be complicated in some cases and also the later transformation on instance level. It is important to consider cases that are not straight on like translation from a relational model to UML. Both a relational model and UML uses association/relation as a concept. For instance, both a 1-N and N-M relation in UML will be transformed into a table in SQL. The definitions of the structure that has been chosen and defined by the user, has to be saved in a data structure to transform correctly the from PIM to PSM to code. In addition, this has a meaning for how to transform at instance level, since one has to consider this to transform correctly.

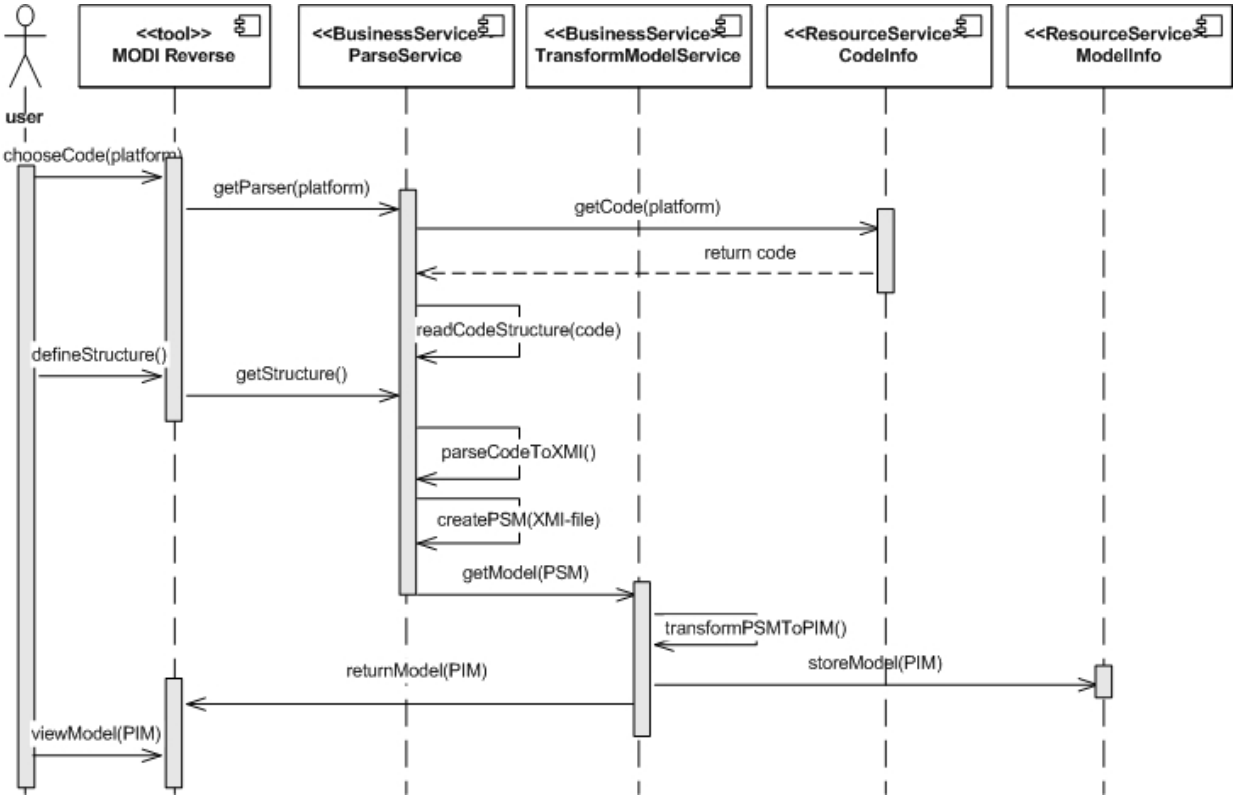


Figure 5-5 MODI Reverse process

Further, the ParseService parses it into an XMI file. The ParseService's next task will be to create a PSM from the XMI file. This PSM should be saved in the *ModelInfo* ResourceService component, and will be an input to the TransformModelService which is supposed to transform the PSM into a PIM. For PSM-to-PIM transformation the transformation language ATL (QVT) can be used. The transformation should include rules for transforming PSM datatypes into PIM datatypes. An example of a generic transformation rule would be: *PSM.datatype =: PIM.datatype*. Before the PIM is returned to the MODI Reverse, it is saved in ModelInfo. Now the user should be able to view the PIM. This process is repeated for Enterprise B.

In the case an enterprise already has a PSM, it should be possible for them to skip the parsing part, and directly transform the PSM into a PIM.

5.3.3 Transformation example (PSM-to-PIM)

The following transformation example illustrates how a transformation from PSM-to-PIM will look like. The transformation is expressed in the transformation language ATL [49]. We only show one transformation, as the other transformations would almost be the same. The example is a rule for transformation from VARCHAR (SQL datatype) to String (PIM datatype). The example is shown in the frame below:

```
rule VARCHARtoString {
      from
           d: SQLDatatype!VARCHAR
      to
          out: PIMDatatype!String {
               String.length = d.length
           }
}
```

5.3.4 Interfaces for MODI Reverse

The interfaces that are defined for the MODI Reverse tool are: *IParseService, ITransformModelService, ICodeInfo* and *IModelInfo.* The *IModelInfo* interface is shared with the MODI Mapper tool. Figure 5-6 shows these interfaces with their belonging methods. It is possible to add more methods to the interfaces as needed.

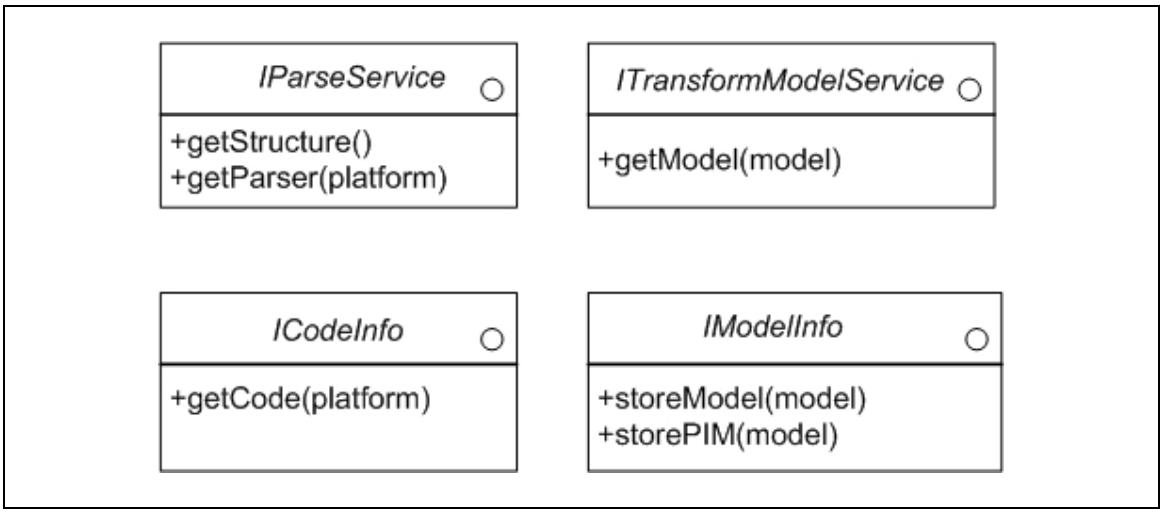


Figure 5-6 Interfaces for MODI Reverse

5.4 Mapping rules

Before describing the components needed for implementation of MODI Mapper, we describe the generic mapping model. This model is supposed to facilitate mapping between PIMs representing data with different syntaxes, structure and semantics by defining mapping rules.

A *mapping rule* is a description of how one or more constructs in a source model (PIM_A) can be transformed into one or more constructs in a target model (PIM_B). One mapping rule applies to one data integration problem case. In addition, there can be many solutions related to one mapping rule dependent on different problem cases. We have taken into account some occurrences. The developer can add functionality to the mapping rules as needed. The

mapping rules should be defined in an XML file, where the developer has the ability to add, remove etc. the functionality. The rules shall be dynamic since they will not be the same for every case. The mapping rules take into account *syntax, structure* and *semantic*.

The semantic part in the mapping rules refers to semantic equivalence, semantic incompatibility, semantic relationship, semantic relevance and semantic resemblance. This part is concerned about identifying data by comparing their meaning, and solving these issues. By comparing the metadata of the terms in PIM_A and PIM_B, semantics concerning synonyms and homonyms can be identified. At first glance, semantic mapping may seem very simple, yet it requires solving deep ontological problems, such as deciding on the correspondence between words and concepts in the world. However, we define mapping rules between models without involving the use of an ontology.

The syntactic and structural part in the mapping rules refer to comparing the actual structure and representation of the data. The structure of a model is manifested through its concept names, data types, concept relationships and constraints.

The model mapping activity includes comparing, and mapping the enterprises' PIMs. In particular, comparison should be performed as part of the model mapping. A mapping model is used to describe how the mapping shall be performed. The mapping model shown in Figure 5-7 is a generic metamodel at M2 level of OMG. This metamodel is presented in the modeling language UML, and contains the metaclasses UML Attribute, UML Class, and Mapping. The Mapping metaclass is not a part of the UML metamodel [66], but it is part of the generic mapping model described in this framework. The attributes for the metaclasses is not shown in the figure below.

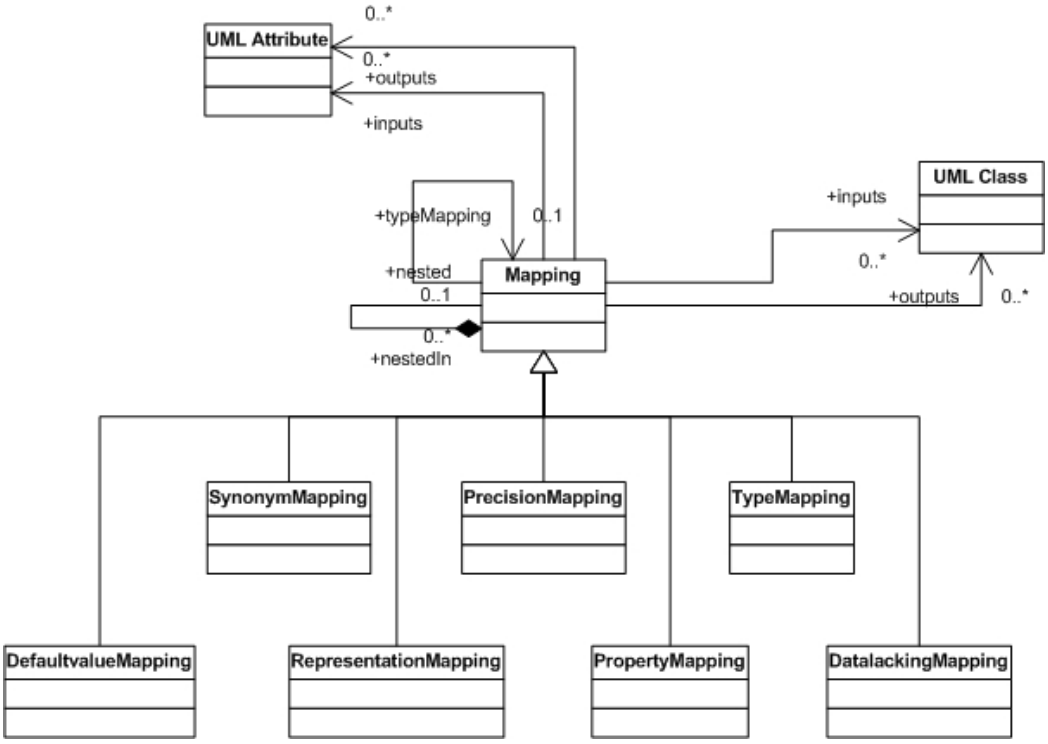


Figure 5-7 Generic mapping metamodel

UML Attribute: This metaclass is used to describe which elements at M1 level are attribute instances. This metaclass contains several attributes such as name, visibility, multiplicity etc. These attributes are metadata of the UML Attribute instance.

UML Class: This metaclass is used to categorize which elements at M1 level are class instances. This metaclass contain several attributes (same as UML Attribute) such as name, visibility, multiplicity etc. These attributes are metadata of the UML Class instance.

Mapping: This metaclass is used to describe mapping between UML Classes and UML Attributes in metamodels. This metaclass is a super-class which consist of sub-classes. These sub-classes contain different types of mapping rules dependent of a problem case. The sub-classes that are considered are: *SynonymMapping, PrecisionMapping, TypeMapping, DefaultValueMapping, RepresentationMapping, PropertyMapping and DataLackingMapping.* These sub-classes reflect the data integration problems defined in section 2.4. It is not suggested a sub-class for the *Homonym* problem, since this is a problem that cannot be solved by mapping between the attributes that are affected. There is a possibility for adding other sub-classes as new data integration problems are identified.

5.4.1 Strategies for executing mapping rules

One attribute might be affected by several data integration problems. However, we specify rules for one specific case at the time. In the case where one attribute is subject to more than one integration problem, then two or more mapping rules apply.

1. **SynonymMapping:** This sub-class contains a mapping rule applicable in the case where different attributes in PIM_A and PIM_B refer to the same semantic meaning, such as *brougham* and *cartype*. The SynonymMapping class will have a source attribute as input, and a target attribute as output. The value in the source attribute in PIM_A shall be transmitted uncritically to the target attribute in PIM_B. The aim is representing this value in the target attribute. The strategy for executing this rule is illustrated in Figure 5-8 and can be defined by the following formula:
   PIM_A.UML Attribute =: PIM_B.UML Attribute

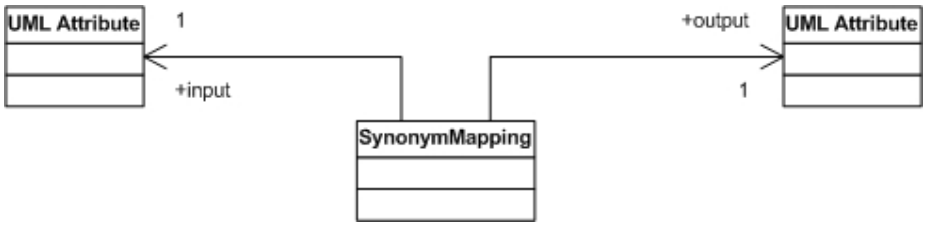


Figure 5-8 SynonymMapping metamodel

2. **Homonym:** In the case where the same attribute name is used to denote two different concepts in PIM_A and PIM_B, they cannot be mapped to each other because their semantic meanings are different. One strategy to solve this problem is to examine and compare the metadata for the affected attributes in PIM_A and PIM_B. However, it is not always sufficient to examine metadata since the semantic information is not always well described, or is lacking.

3. **RepresentationMapping:** This sub-class contains a mapping rule applicable in the case where enterprises represent attributes in PIM_A and PIM_B differently, such as when a time attribute is represented as *minutes* in PIM_A, and *hours* in PIM_B. The RepresentationMapping class will have a source attribute as input and a target attribute as output. A solution might be to use a function converting from one representation format to another. The source attribute representation will be converted into the target attribute representation. Then the source attribute's value can be transmitted, and represented in the target attribute format. The strategy for executing this rule is illustrated in Figure 5-9 and can be defined by the following formula:
   convertRepresentation(PIM_A.UML Attribute) =: PIM_B.UML Attribute

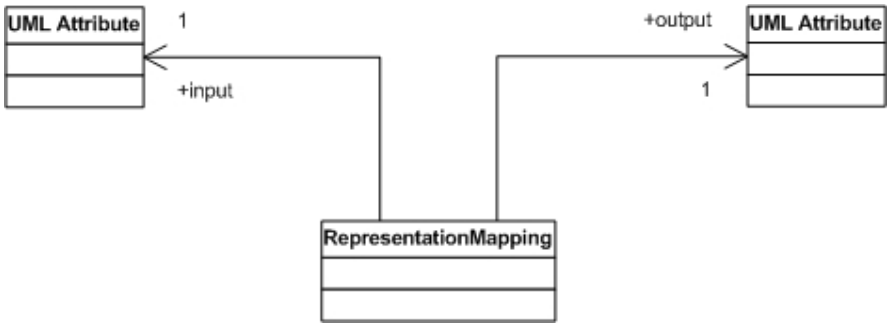


Figure 5-9 RepresentationMapping metamodel

4. **PropertyMapping:** This sub-class contains a mapping rule applicable in the case where enterprises model properties in PIM_A and PIM_B differently, such as when a persons name is represented by *firstName* and *lastName* in PIM_A, and by *name* in PIM_B. The PropertyMapping class will have one or more source attribute(s) as input and one or more target attribute(s) as output. This mapping rule provides several methods for solving this problem:

One solution may be to use a *concatenation* function in the case where there is a need to combine attributes. In addition, have a *constant* function supplying a character to separate the items, e.g. a space. The strategy for executing this can be defined by the following formula:
concatenate(PIM_A.UML Attribute + constant)$^+$ =: PIM_B.UML Attribute

The $^+$ sign at right indicates that there can be more than two attributes to concatenate, and the constant function is added between the attributes. The aim is to concatenate the source attributes in PIM_A e.g. *firstName* and *lastName* into one attribute. After they are concatenated, the constant, e.g. a space separates the *firstName* and *lastName*. The value in the source attributes can now be transmitted to the target attribute in PIM_B. The result would be: *firstName lastName = name*

Another solution is to use a *split* function in the case where there is a need to split attributes. This function chooses which character to split on, e.g. a space. The strategy for executing this can be defined by the following formula:
split(PIM_A.UML Attribute)$^+$ =: PIM_B.UML Attribute$^+$

In this solution the $^+$ sign with the split function indicates that an attribute can be separated and represented in several attributes. The other $^+$ sign indicates the number of attributes the divided attributed shall be represented in. The purpose is to split the source attribute, e.g. *name* by searching for a character like space in the source attribute, and place each divided part in the target attributes, e.g. *firstName* and *lastName*. The result would be: *name = firstName lastName*.

The strategy for executing these rules is illustrated in Figure 5-10. Both solutions are methods provided in the PropertyMapping rule.

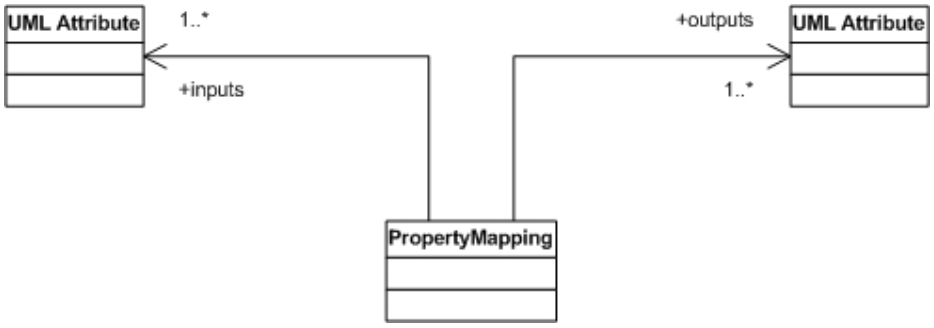


Figure 5-10 PropertyMapping metamodel

5. **PrecisionMapping:** This sub-class contains a mapping rule applicable in the case where elements in PIM_A and PIM_B have been modelled at different aggregation levels, such as modeling *address* as an attribute in PIM_A and a *class* in PIM_B. In case the source attributes match the target attributes in the target class, a mapping can be performed. However, several cases can be difficult to manage. There can be several attributes included in the target class in PIM_B that are not available in PIM_A. Another case is if an attribute includes collected information, such as *address* containing streetname, streetnumber and postnumber separated by, e.g. a space. Since attributes are not equal to a class, these elements can not be mapped directly.

In case attributes in PIM_A are matching class attributes in PIM_B the PrecisionMapping class will have one or more source attribute as input(s) and output(s). The strategy for executing this rule is illustrated in Figure 5-11, and can be defined with the following formulas: **$(PIM_A$.UML Attribute =: PIM_B.UML Class.UML Attribute)$^+$**

The $^+$ indicates that several source attributes can be mapped to the target classe's attributes. The next formula is an opposite case:
$(PIM_A$.UML Class.UML Attribute =: PIM_B.UML Attribute)$^+$

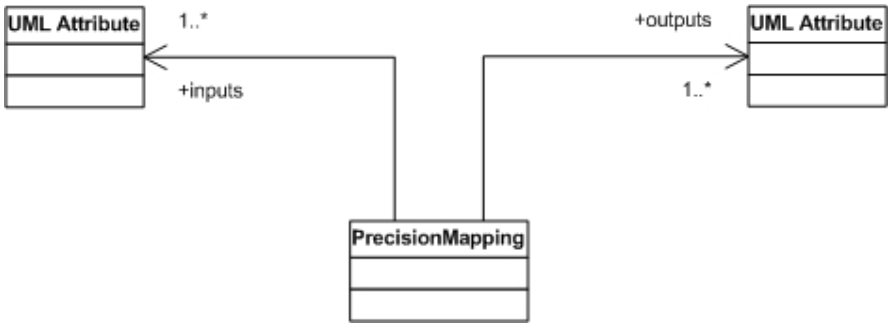


Figure 5-11 PrecisionMapping metamodel

6. **DefaultvalueMapping:** This sub-class contains a mapping rule applicable in the case where default values are used in PIM_A and PIM_B differently, such as having 15 %

discount in PIM_A and 10 % in PIM_B. The DefaultvalueMapping class will have a source attribute as input and a target attribute as output. In this case it is vital to have metadata about the target attributes default value to transform it. The calculation that should be used to manage this kind of conflict is to apply the default value on the source attribute which is to be transformed. The strategy for executing this rule is illustrated in Figure 5-12 and can be defined with the following formula:
applyTargetDefaultValueOn(PIM_A.UML Attribute) =: PIM_B.UML Attribute

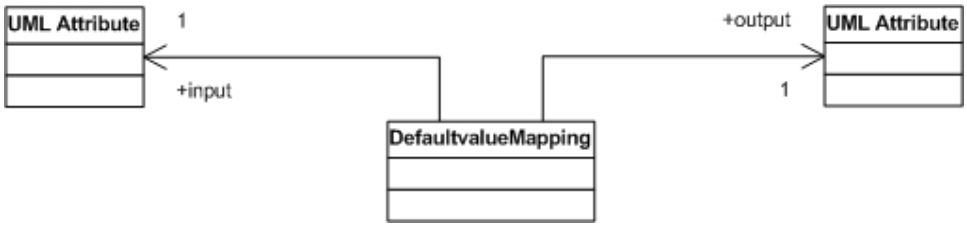


Figure 5-12 DefaultvalueMapping metamodel

7. **TypeMapping:** This sub-class contains a mapping rule applicable in the case where enterprises use different data types for the same attributes in PIM_A and PIM_B, such as *telephone* as datatype string in PIM_A and as datatype integer in PIM_B. The TypeMapping class will have a source attribute as input and a target attribute as output. A solution might be to use a function which converts from one datatype to another.. The strategy for executing this rule is illustrated in Figure 5-13 and can be defined with the following formula:
toDatatype(PIM_A.UML Attribute) =: PIM_B.UML Attribute

The aim would be to transform the source attribute's datatype, e.g. *string* into the target attribute's datatype, e.g. *int*. The result would be: *toInt(PIM_A.telephone) =: PIM_B.telephone*

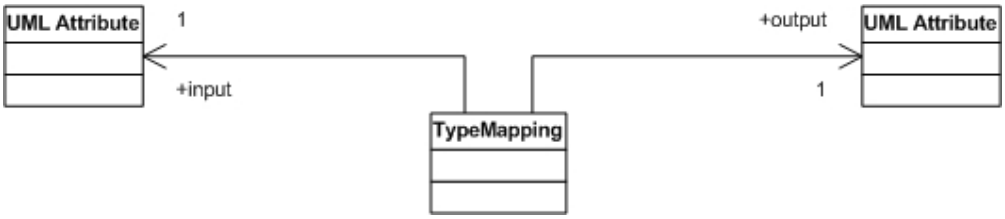


Figure 5-13 TypeMapping metamodel

8. **DatalackingMapping:** This sub-class contains a mapping rule applicable in the case where enterprises are missing attributes or classes, such as PIM_A contain the attribute *enginefuel* and PIM_B does not contain this attribute. In this case the enterprise with PIM_B cannot receive this attribute, unless they agree in between to add an attribute in the PIM_B. In this case the DatalackingMapping class will have a source attribute as input, and create a target attribute. Another case may be when PIM_A have more attributes in a class which is being mapped to a matching class in PIM_B. A solution is to use a *filter* function to only pass data that are used by both PIM_A and PIM_B and

ignore the remaining attributes. The strategy for executing this rule is illustrated in Figure 5-14 and can be defined by the following formula:
filter(PIM_A.UML Class.UML Attribute)$^+$ =: (PIM_B.UML Class.UML Attribute)$^+$

The $^+$ sign at left indicates that several source attributes can be filtered out, and the $^+$ sign at right indicates that several target attributes can be matched. In case a target attributed is expecting to get input from a source attribute that does not exist, a solution may be to add a default value to the target attribute.

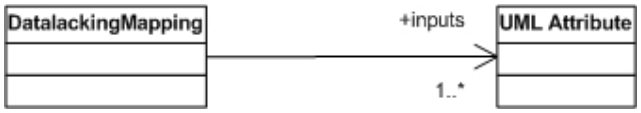


Figure 5-14 DatalackingMapping metamodel

A mapping rule for models including data lacking is difficult. The reason is because the collaborating models do not have matching data. This leads to leaving out mappings or adding missing attributes. However, it is not always permissible or easy to create attributes in the target models. One reason may be that it is not support for creation of a new attribute in the original format.

These mapping rules defined above shall be functionality of the MODI mapper tool. The metamodels (M2) representing the mapping rules shall be metamodels to the PIMs (M1) that are supposed to be mapped in MODI Mapper.

5.5 Model mapping – implement MODI Mapper

In chapter 3, section 3.5 and section 3.6 we examined the data integration tools BizTalk Mapper and Altova MapForce 2005. These tools allow users to integrate data represented in different formats. The BizTalk Mapper maps between XML and Altova Mapforce 2005 also maps between XML specifications and other formats as well. In this section we specify how to implement the model mapping tool MODI Mapper. This is supposed to be similar to the data integration tools examined, but the difference shall be to enable mapping between PIMs. Another difference is that metadata is not available in the examined tools, but MODI Mapper shall have support for this to enhance mapping between data with different semantic. MODI Mapper is supposed to allow a user to map between PIMs. This tool is supposed to support model-to-model and model-to-text transformation. The target users of this tool are enterprises having system code represented in PIMs. The components used in the MODI Mapper tool are described next. MODI Mapper will have some similar qualities to the UMT tool, such as providing for model-to-model transformation, model-to-text transformation. The difference will be that UMT's model-to-model transformation is from PSM-to-PIM and PIM-to-PSM, while MODI Mapper will contain PIM-to-PIM mapping.

5.5.1 MODI Mapper components

MODI Mapper tool component is a graphical user interface. Through this interface a user should be able to load the PIMs to be mapped, and perform mapping on the PIMs by using mapping rules. In addition to loading and mapping models, the user shall be able to view and store models. The user chooses the platforms that are to be transformed from this component. This tool component is dependent on the components MappingService, EngineService and ModelInfo to make this possible.

The *MappingService* is a component which shall create mappings from the mapping performed by the user, and then create a resulting PIM from these mappings. The resulting PIM will contain the mappings performed on the data integration problems. It documents the data transformation and integration workflow. The importance with documenting data transformation and integration workflow is to better understand the data-interrelationship and data semantics. M1 models of the generic mapping model (recall Figure 5-7) are a part of this component.

The *EngineService* component shall translate the mapping model into an executable mapper. It includes a generator which generates code so the mapping can be executed. The generator deals with automatically execution of code generated from the mapping model. The responsibility of this service will be to handle transformation between platform-specific formats. Examples of transformations include Java-to-SQL, XML-to-SQL, Java-to-XML, XSLT etc. The ResourceService *ModelInfo* shall be a component containing PIMs and PSMs.

5.5.2 MODI Mapper component interaction

Figure 5-15 shows the inter-dependencies between these components, and how they collaborate. The user has an interface to the tool component MODI Mapper. Through this component the user should be able to load both PIMs at the same time. Further, the user is supposed to start performing the mapping. To facilitate the mapping activity for the user, there should be a service reading metadata automatically from the PIMs. The metadata should be loaded and available by double-clicking on the classes. From the figure we see a *Loop* label around the methods *mapPIMs(mappingRule)*, *getMappingRule()* and *createMapping()*. This indicates that the user should be able to perform a number of mappings as needed. For each mapping the MappingService shall create a mapping. After mappings are performed, the MappingService shall create a resulting PIM, and store the model in ModelInfo. The user shall have the ability to view this PIM. Next, the MODI Mapper component enables the user to choose the source and target platform for transformation. The resulting PIM will next be an input to the EngineService, and the platforms are set. The EngineService shall create a code skeleton for the mapping from the resulting PIM and generate the code. To generate code from platform A to platform B, the developer can use QVT to specify transformation between the PSMs metamodels. Finally, the instances must be transformed. Also, it is important to use the information saved during the reverse engineering process to transform back from PIM to code. The aim is to represent an enterprise's instances into another enterprise's specific format. By using the generated code, the instances shall be transformed to the target platform.

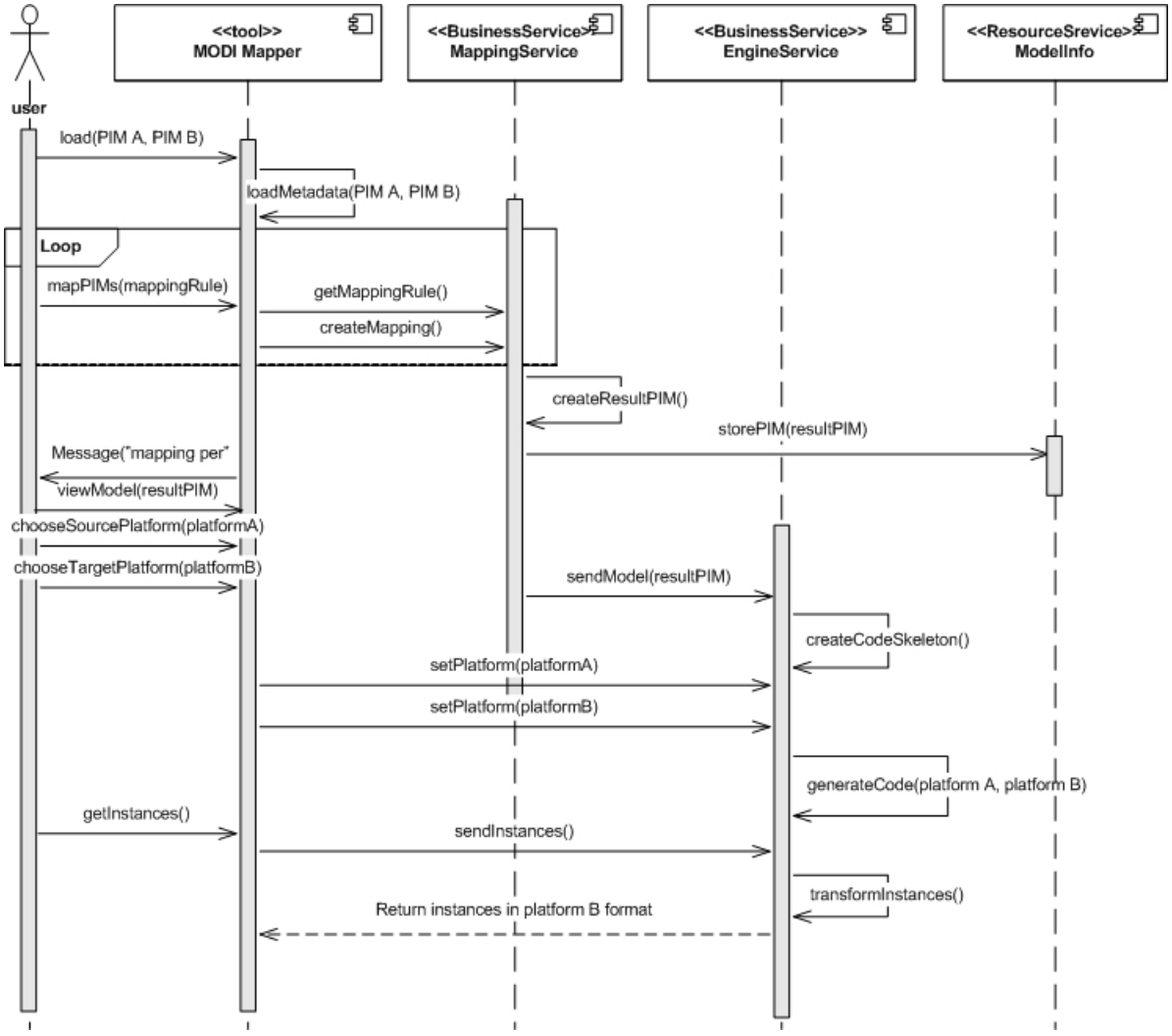


Figure 5-15 MODI Mapper process

5.5.3 Interfaces for MODI Mapper

The interfaces that are defined for the MODI Mapper tool are: *IParseService, ITransformModelService,* and *IModelInfo.* The *IModelInfo* interface is shared with the MODI Reverse tool (recall Figure 5-6). Figure 5-16 shows these interfaces with their belonging methods.

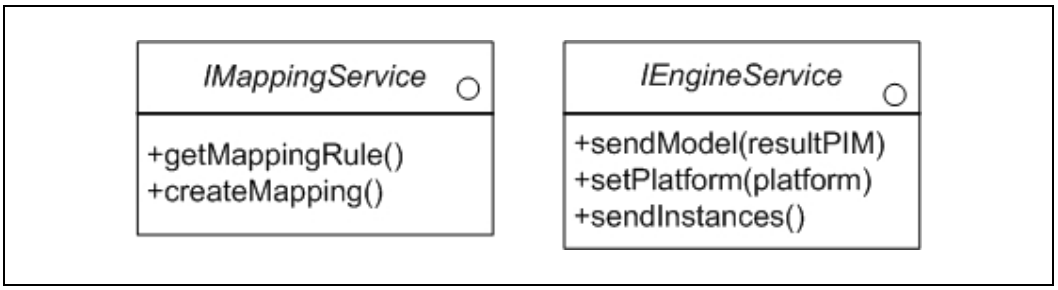


Figure 5-16 Interfaces for MODI Mapper

5.5.4 Functionality of MODI Mapper

Functionality required for the MODI Mapper tool is specified in this section. General buttons are required to manage some part of the functionality:

- The *Load model* button shall allow loading models that are supposed to be mapped to each other. The user will load a model from the model repository ModelInfo, which can be local for the source enterprise, or shared.
- The *Store model* button shall allow storing models in a repository.
- The enterprise shall have the ability to view the result of the mapping by using the *View model* button. When the result mapping is shown, the user should have an option to accept the performed mapping. If the mapping is accepted, the user shall choose the platform from where instances are taken, and the platform to which the instances are transformed.

Figure 5-17 illustrates what the MODI Mapper tool may look like, and how it can be plugged into an Eclipse editor. The figure shows four panels; namely *Package explorer, Mapping grid, Property for mapping rule* and *Property for selected class*. These panels are specified in detail further.

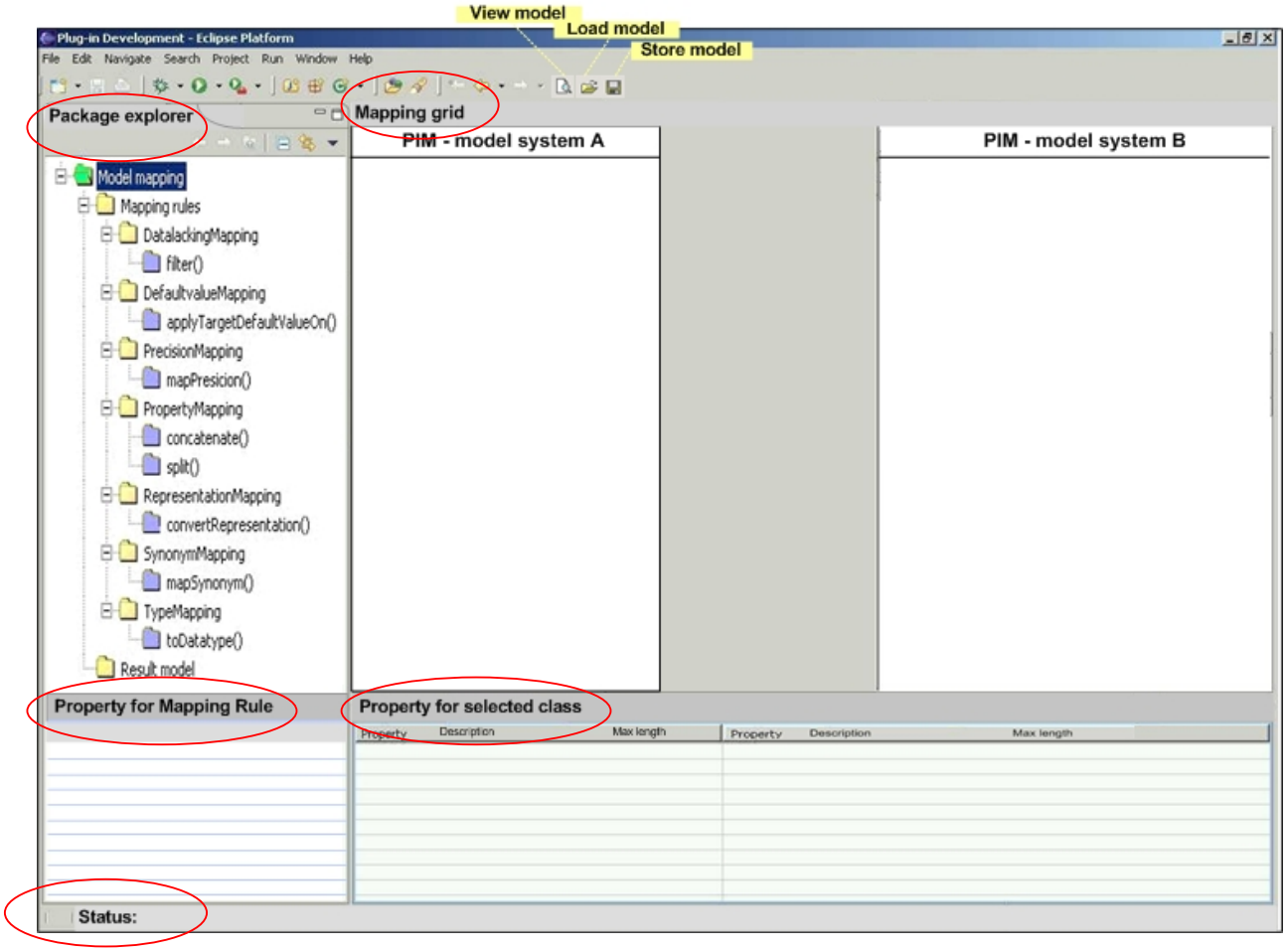


Figure 5-17 MODI Mapper tool

Package explorer

This pane shall contain a mapping library containing mapping rules which users shall be able to choose from. From the package explorer we can see a library containing some mapping rules with different options. An example is *PropertyMapping* rule with *concatenate* and *split* function. The user shall be able to drag and drop these functions, e.g. by dragging the concatenate function into the Mapping grid the user can start performing the mapping. Also, the developer should have the option to add mapping rules and more functions.

Mapping grid

The Mapping grid comprise of three panes. The source model shall be displayed at the left pane, and the target model shall be displayed at the right pane. When the models are loaded in these panes, the classes in the models shall contain connectors for each attribute in the class. The user shall use these connectors to create links between attributes and mapping functions. The middle pane is a mapping grid that shall display lines that links source attributes to target attributes where mapping shall be performed. Simple relationships between attributes should be mapped by dragging an attribute from the source model to the appropriate attribute in the target model. More complex mappings may be performed by using the mapping functions.

Property for mapping rule

Further, the user shall have the possibility to manipulate properties for mapping rules. For every mapping function, the user should get different properties related to the mapping. For SynonyMapping no properties should be displayed in this pane, since it is a direct mapping.

The *RepresentationMapping* rule has a convert function called *convertRepresentation*. The Property for mapping rule pane shall display the connecting attributes representation format. The user shall be able to choose the target attributes format.

The PropertyMapping rule has two functions: namely *concatenate* and *split*. When the user connects attributes from the source model to the concatenate function, the user shall be able to click on the mapping function in the Mapping Grid, and then see the chosen attributes and choose in which order these attributes shall be concatenated in the Property for mapping rule pane. In addition, the user should be able to choose a constant used to separate the attributes. In the case where the user connects an attribute from the source model as input to the split function and target attributes as output, the user shall be able to choose in which order and the number of parts the source attribute should be divided in this pane.

The *PrecisionMapping* rule has the function *mapPresision*. The Property for mapping rule pane displays which group of source attributes are equal to a target class or invert. Some mappings in this case can be performed by using the PropertyMapping rule, but the difference is that the PrecisionMapping rule will display which source attributes that constitute a target class or which source class that constitutes which target attributes.

The *DefaultvalueMapping* rule has the function *applyTargetDefaultValueOn*. The user shall be able to see the target attributes default value. In the case there the default value is not available, the user can choose *null.*

The *TypeMapping* rule has the function *toDatatype*. The Property for mapping rule pane shall display which datatype is being used by the target attribute.

The *DatalackingMapping* rule has a function called *filter*. The user shall be able to choose which attributes to filter out.

The suggestions above are example of which properties can be displayed in the Property for mapping rule pane. The developer can choose which properties are necessary to be displayed and how much the user can contribute with.

Property for selected class

By double-clicking on the classes, the users should have the ability to see metadata relevant to the chosen class. Metadata for the chosen class is supposed to be presented in the Property for selected class pane (see Figure 5-17). From this information the user can see which

vocabulary is used, with a detailed description. This is among other factors convenient as mentioned when managing synonyms and homonyms.

The MODI Mapper tool should also contain a status bar. This is shown lowest at left in Figure 5-17 at the previous page. This bar shall contain state information to the user, such as *"models loaded"*, *"mapping completed"* etc.

5.6 Summary

In this chapter we have presented our solution, MODI Framework. We have specified how to develop two tools to facilitate data integration between enterprises. Mapping rules are defined to solve the different data integration problems defined in chapter 2. The next two chapters present the applications of MODI to the problem cases described in chapter 2. Further, it is assumed that the tools are developed, and the problem examples are test cases to illustrate the use of the tools.

6 MODI Framework applied to NDR

In this chapter it is suggested how data integration problems between various departments can be solved by using the tools specified in the MODI Framework. As the proposed solution TOR is dealing with the metadata problem, we assume that the metadata problem is solved. It is important to precise that the data definitions in OR separates syntax and semantics as proposed in TOR project. We present an example where two departments (department A and department B) are integrating data. In this case, department A is the user of the tools since they are sending their instances to department B. It is assumed that department B sends their system code file to department A. This chapter is divided in two parts, the first part deals with use of MODI Reverse, and the second part deals with use of MODI Mapper.

6.1 Use of MODI Reverse

Department A is using SQL, and department B is using XML. To do mapping between the two systems they need to transform their system code to PIM. In Table 6-1 a description of department A's system code is expressed in DDL.

Table 6-1 SQL code for the customer

SQL code for customer expressed in DDL	
CREATE TABLE Person(personID INTEGER(10), name VARCHAR(40), address VARCHAR(30), telephonenr INTEGER(8), martialStatus VARCHAR(2),) CREATE TABLE Enterprise(organisationNr INTEGER(10), enterpriseName VARCHAR(25), yearOfestablishment INTEGER(4), visitAddress VARCHAR(20), postalAddress VARCHAR(20), postalCode INTEGER(4), enterpriseType VARCHAR(10),)	CREATE TABLE EntAnimal(organisationNr INTEGER(10), animalname VARCHAR(10),) CREATE TABLE Animal(animalname VARCHAR(10), animaltype VARCHAR(10),) CREATE TABLE Renting(houseNr INTEGER(10), nameOfRenter VARCHAR(30), startDate DATE, endDate DATE, rented_completely_partially_costFree VARCHAR(3),)

Figure 6-1 shows an *Entity-Relationship* (ER) model for department A. The user can choose between the system code and ER model for parsing to a UML PSM.

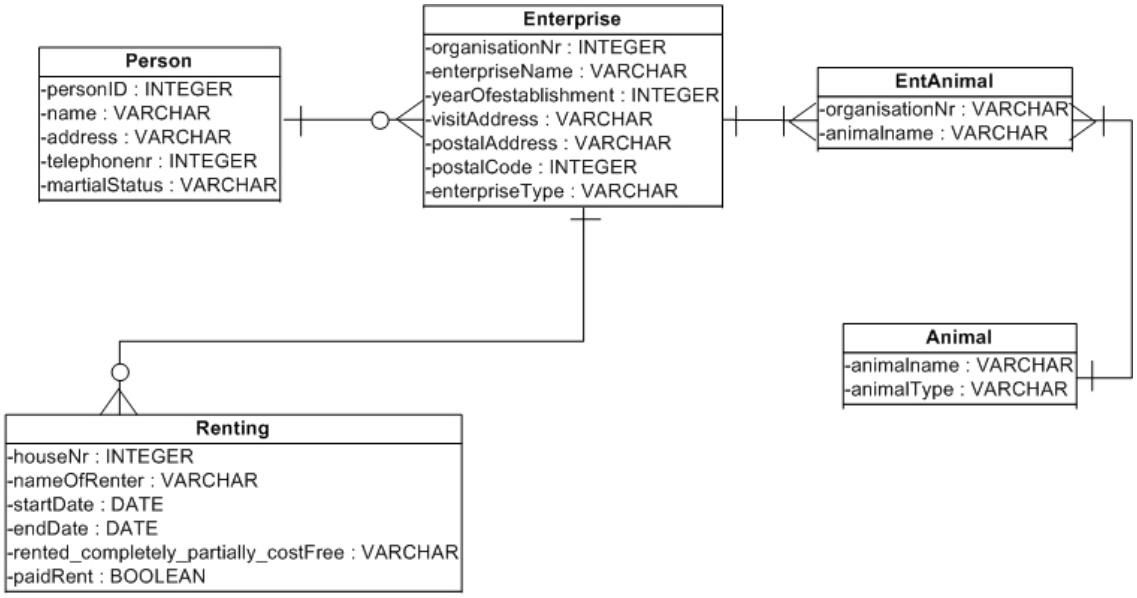


Figure 6-1 ER model for enterprise A

Table 6-2 shows the department B's data expressed in XML Schema.

Table 6-2 XML Schema for supplier

XML Schema for supplier

```
<?xml version="1.0"?>
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema"
targetNamespace="http://www.ndr-case.com"
xmlns="http://www.ndr-case.com"
elementFormDefault="qualified">

<xs:element name="Owner"  minOccurs="1" maxOccurs="1">
  <xs:complexType>
    <xs:sequence>
          <xs:element name="personNr" type="xs:integer"/>
          <xs:element name="firstName" type="xs:string"/>
          <xs:element name="lastName" type="xs:string"/>
          <xs:element name="addressRoad" type="xs:string"/>
          <xs:element name="addressNr" type="xs:integer"/>
          <xs:element name="phone" type="xs:string"/>
    </xs:sequence>
  </xs:complexType>
</xs:element>

<xs:element name="Enterprise">
  <xs:complexType>
    <xs:sequence>
          <xs:element name="NR" type="xs:integer"/>
          <xs:element name="name" type="xs:string"/>
          <xs:element name="postalAddress" type="xs:string"/>
          <xs:element name="phoneNr" type="xs:string"/>
    </xs:sequence>
</xs:complexType>
</xs:element>
```

```
<xs:element name="PostalCode">
   <xs:complexType>
     <xs:sequence>
          <xs:element name="postalCode" type="xs:integer"/>
          <xs:element name="city" type="xs:string"/>
     </xs:sequence>
</xs:complexType>
</xs:element>

<xs:element name="Animal">
   <xs:complexType>
     <xs:sequence>
          <xs:element name="animal" type="xs:string"/>
          <xs:element name="category" type="xs:string"/>
     </xs:sequence>
</xs:complexType>
</xs:element>

<xs:element name="AnimalLine">
   <xs:complexType>
     <xs:sequence>
          <xs:element name="Comment" type="xs:string"/>
     </xs:sequence>
</xs:complexType>
</xs:element>

</xs:schema>
```

From the XML Schema we can se that attributes are also coded at department A, but represented differently. The code for department A contains in addition a table *Renting* which is not considered in department B. The user needs to parse the code for department A and department B into PSMs, and then the PSMs into PIMs. Figure 6-2 shows the process for department A using MODI Reverse.

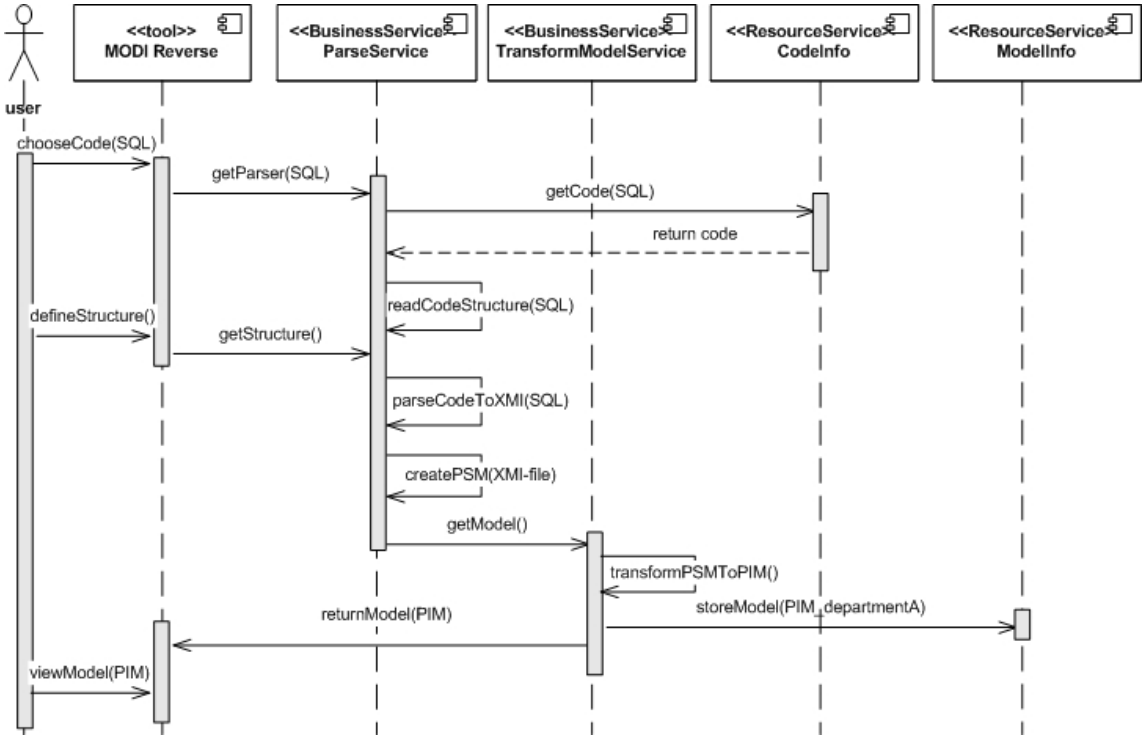


Figure 6-2 MODI Reverse process for department A

In the process for department A, it is performed a transformation from the relational model to UML. In this case, all of the tables will not be transformed into classes. The *EntAnimal* table for department A will be transformed into a relation in UML PSM. This is because it is an many-to-many relationship between the tables *Enterprise* and *Animal.* As this is not allowed in SQL but permissible in UML, the table *EntAnimal* will become an relation. The user has a dialog with the MODI Reverse where he defines entity-tables and relation-tables. This information needs to be saved. This information is further used in MODI Mapper when transforming back, from PIM to code.

Figure 6-3 shows the MODI Reverse process for department B.

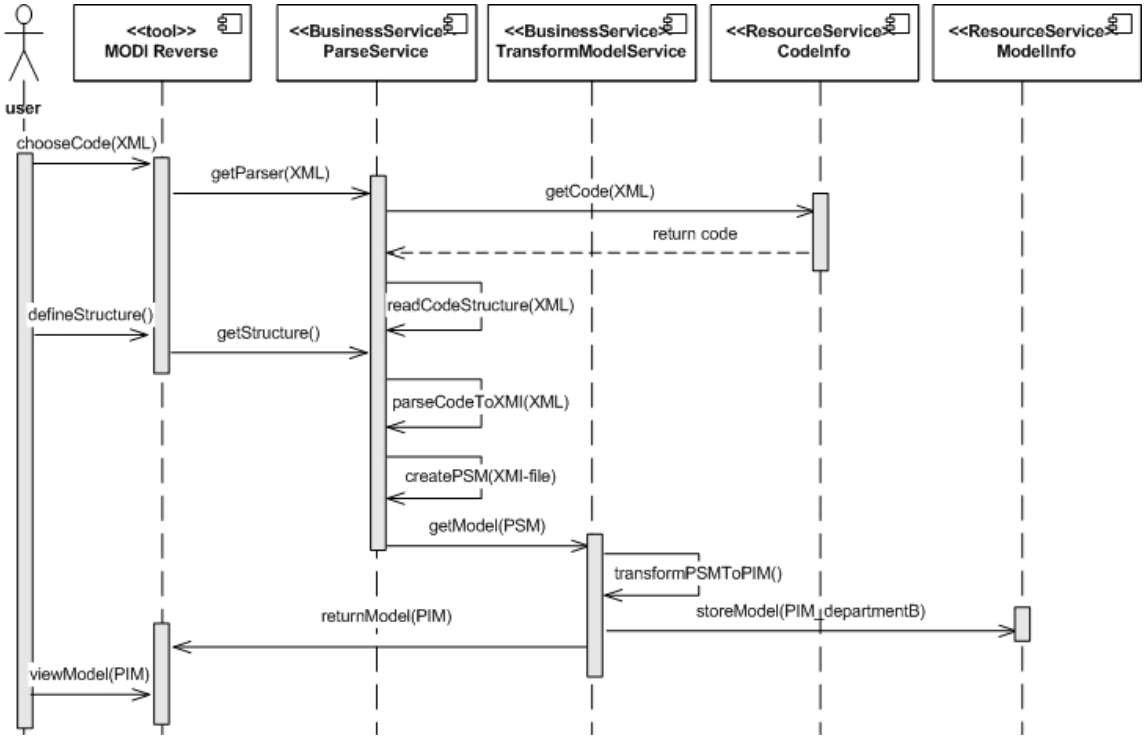


Figure 6-3 MODI Reverse process for department B

In Figure 6-4 the PIM for department A is presented as a result from the MODI Reverse tool.

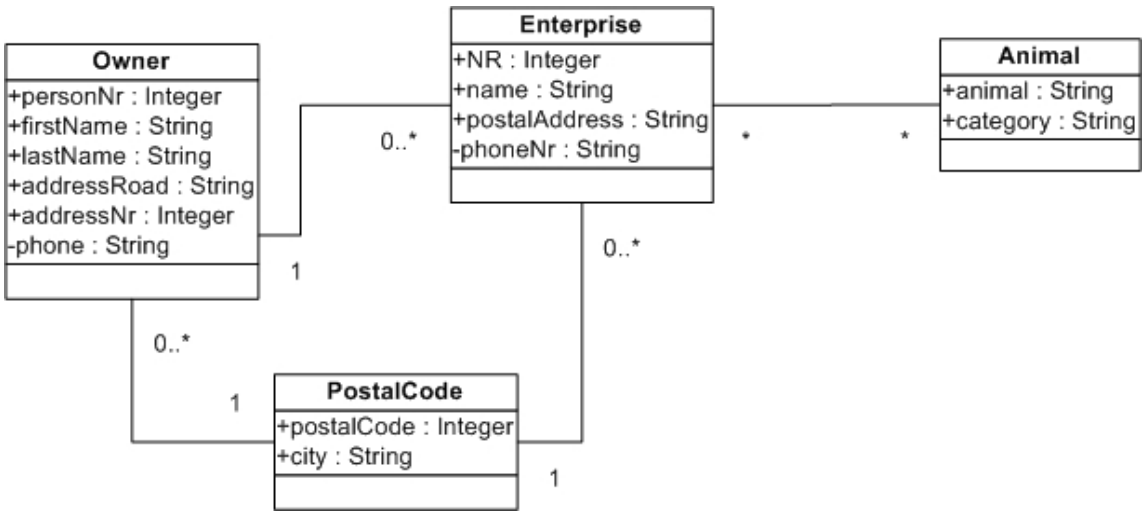


Figure 6-4 PIM for enterprise A

Figure 6-5 shows the PIM for department B.

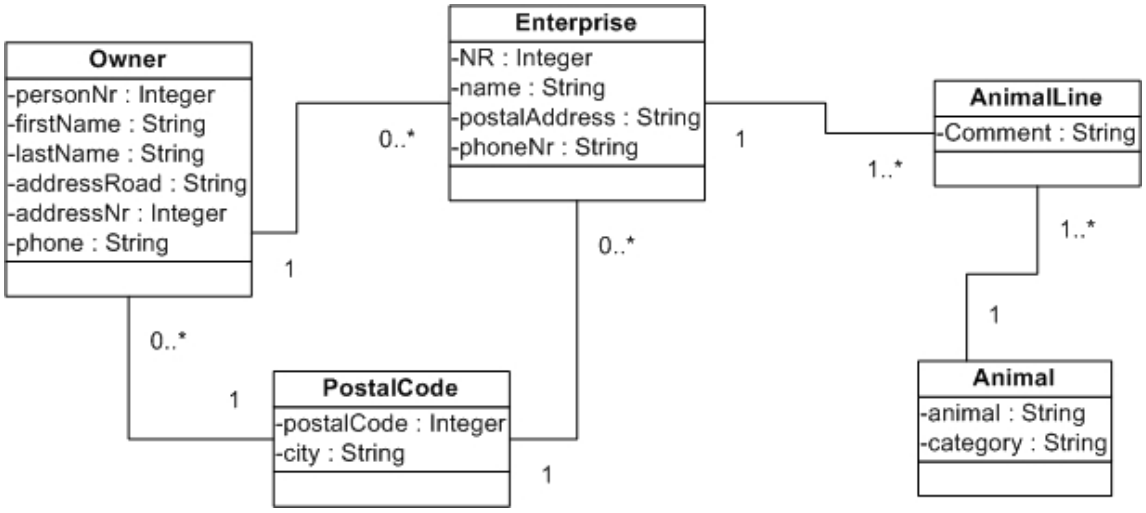


Figure 6-5 PIM for enterprise B

In Table 6-3 the mapping from code to PIM is described for department A. And in Table 6-4 the mappings for department B is given. We will only consider a selection of the code/PSM since it would be mostly the same for the remaining part.

Table 6-3 Mappings from code/ PSM to PIM for department A

Attribute	Code / PSM	PIM	Description
Person			
personID	INTEGER(10)	Integer	When transforming integer with length of 10, the integer in PIM will be transformed to maxInt. This is done to ensure that nothing is excluded when instances are transformed
name	VARCHAR(40)	String	VARCHAR is transformed into a String. The string will have the same length as in code/ PSM.
address	VARCHAR(30)	String	
telephonenr	INTEGER(8)	Integer	When transforming Integer with length 8, the Integer in PIM will be transformed into maxInt.
martialStatus	VARCHAR(2)	String	VARCHAR is transformed into a String. The String will have the same length as in code/ PSM.
Renting			
houseNr	INTEGER(10)	Integer	When transforming Integer with length 10, the Integer in PIM will be transformed to maxInt.
nameOfRenter	VARCHAR(30)	String	VARCHAR is transformed into a String. The String will have the same length as in code/ PSM.
startDate	DATE	Date	The datatype DATE in SQL for department A is represented as *yy.mm.dd,* and as *dd.mm.yy* in XML.
endDate	DATE	Date	
rented_completely_partially_costFree	VARCHAR(3)	String	VARCHAR is transformed into a String. The String will have the same length as in code/ PSM.

Table 6-4 Mapping from XML to PIM for department B

Attribute	PSM	PIM	Description
Owner			
personNr	Integer	Integer	XML Integer datatype will be transformed into PIM datatype Integer. The Integer in PIM will have max length.
firstName	String	String	The datatype String will remain unchanged.
lastName	String	String	The datatype String will remain unchanged.
addressRoad	String	String	The datatype String will remain unchanged.
addressNr	Integer	Integer	XML Integer datatype will be transformed into PIM datatype Integer. The Integer in PIM will have max length.
phone	String	String	The datatype String will remain unchanged.
PostalCode			
postalCode	Integer	Integer	The Integer in PSM has a length 4, but in PIM the Integer will get max length.
city	String	String	The datatype String will remain unchanged.

6.2 Use of MODI Mapper

The PIMs are ready to be uploaded by the MODI mapper tool. The user uses the *load model* button to load the two models. In Figure 6-6 both of the PIMs are shown after they have been loaded.

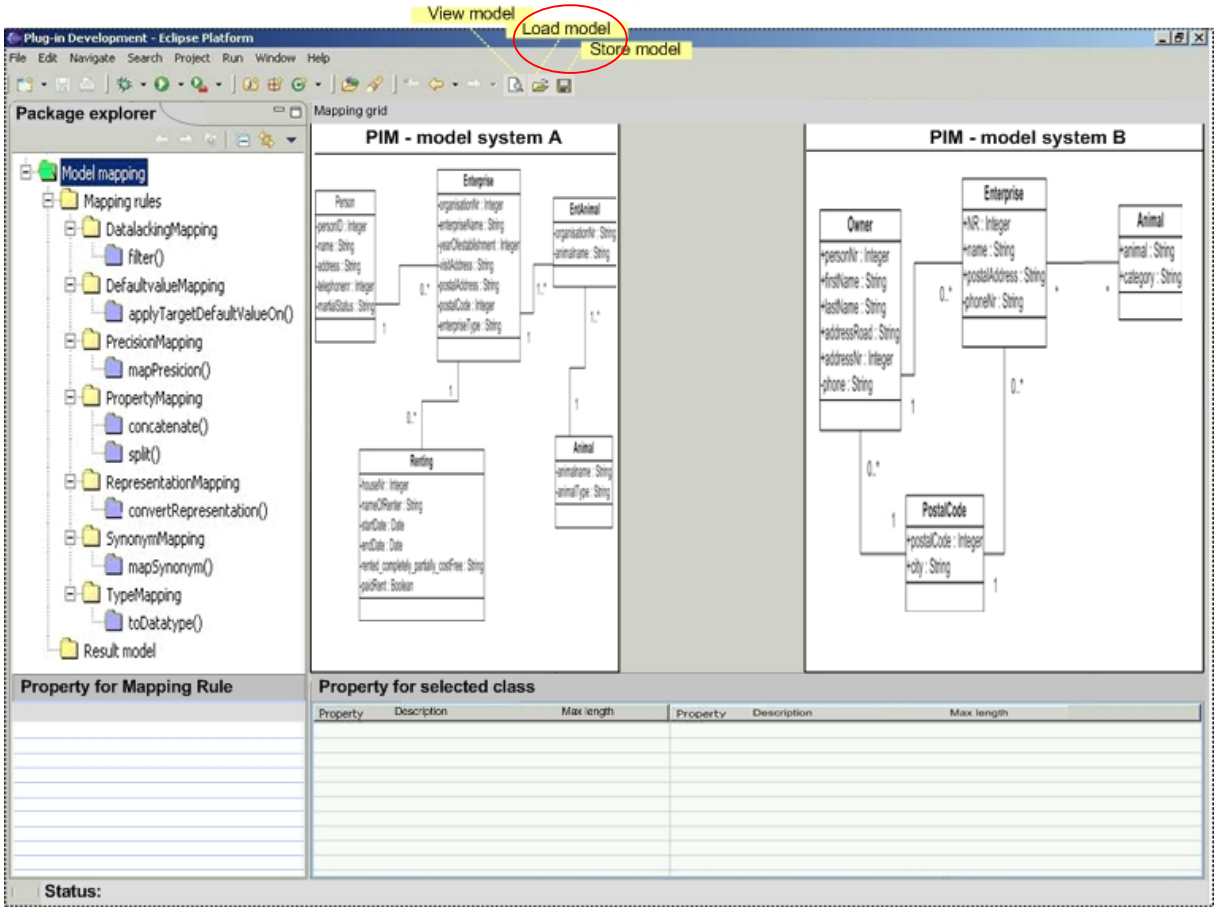


Figure 6-6 MODI mapper with PIM_A and PIM_B

Before doing the mapping, problems that occur when integrating the data need to be identified. In Table 6-5 the problems are described. We only describe two classes from the models in the table to illustrate how the mapping is done. In the case where there is data lacking the user does not need to do anything since the information is not relevant to the department who is receiving the information. From the two models we see that there are many similarities between the classes, but they describe their attributes in different level of abstraction.

Table 6-5 Mapping table

PIM_A	PIM_B	Data integration problem	Mapping rule
Person	***Owner***		
personID	personNr	- synonyms	1.SynonymMapping
name	firstName	- differences in properties	4.PropertyMapping
Not handled	lastName		
address	addressRoad	- synonyms - differences in properties - data precision conflict	1.SynonymMapping 4.PropertyMapping 5.PrecisionMapping
	addressNr		
	postalCode		
	city		
telephonenr	phone	- synonyms - Attribute integrity constraint	1.SynonymMapping 6.TypeMapping
martialStatus	***Not handled***	- data lacking	7.DatalackMapping
Enterprise	***Enterprise***		
organisationNr	NR	- synonyms	1.SynonymMapping
enterpriseName	name	- synonyms	1.SynonymMapping
yearOfestablishment	***Not handled***	- data lacking	7.DatalackMapping
visitAddress	***Not handled***	- data lacking	7.DatalackMapping
postalAddress	postalAddress	No problem	
postalCode	postalCode	- data precision conflict	5.PrecisionMapping
	city		
enterpriseType	***Not handled***	- data lacking	7.DatalackMapping
Not handled	phoneNr	- data lacking	7.DatalackMapping

After identifying the different data integration problems the user can map those attributes and apply the related mapping rule to solve the problem. Figure 6-7 illustrates different mappings created between the two PIMs. The yellow boxes in the middle pane shows the mapping function which is applied to. All of the mappings identified above are not shown in Figure 6-7, only a selection of the mappings are shown.

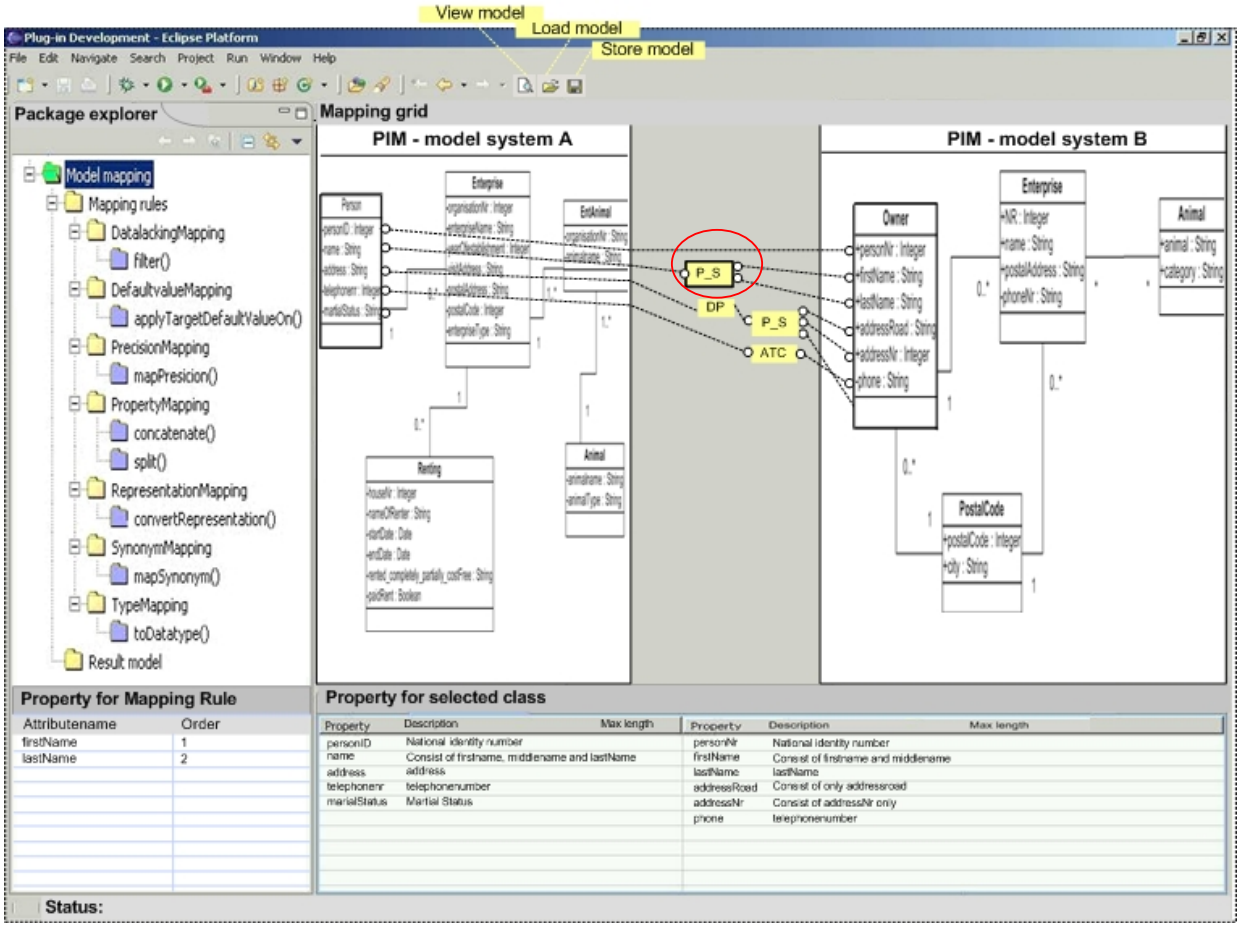


Figure 6-7 MODI mapper with mapping rules

The mapping rules that are used in this example are: PropertyMapping with split function (P_S), Precision mapping with mapPrecision function (DP) and TypeMapping with toDatatype function. Further, we explain the P_S function in detail. Input to the P_S function is the *name* attribute from the source model's class *Person.* The function has two output links connected to the target attributes *firstName* and *lastName* from the target model's class *Owner.* The P_S function splits the source attribute in two. From the figure we can se that the classes that are being mapped and the P_S function have a bold frame. The classes' bold frame indicates that these classes have metadata presented in the *Property for selected class* pane. The functions' bold frame indicates that this function's properties are displayed in the *Property for Mapping Rule* pane. From this pane the user can see the target attributes and chooses in which order the source attribute should be split. The other mappings are performed in the same way. When the user has performed the mapping it should be a possibility to see the mapping result and approve the mapping. The next step is to generate code.

After generation of the result mapping model, code can be generated to execute the mappings. If the user accepts the result mapping model, he can choose the platforms the transformation is performed on. In this case the user will choose SQL2XML transformation.

Department A is now able to send instances through the mapping engine to solve the data integration problems identified between them. The mapping engine will also transform the instances from SQL platform to XML.

7 MODI Framework applied to Automotive scenario

In this chapter it is suggested how the data integration problems defined in the Automotive scenario can be solved by using the tools specified in the MODI Framework. This chapter is divided in two parts. The first part deals with the user using the MODI Reverse tool. The second part deals with the user using the MODI Mapper tool. To simplify the case, we assume that the customer is performing the mapping from their model to the supplier's model.

7.1 Reverse Engineering

The customer's (Fiat) proprietary format is SQL and the supplier's (Bosch) proprietary format is XML. To transform the code for Fiat and Bosch to PIMs the MODI reverse tool is used. To do mapping between the two systems they need to transform their system code to PIM. Figure 7-1 shows the process for Fiat using MODI reverse.

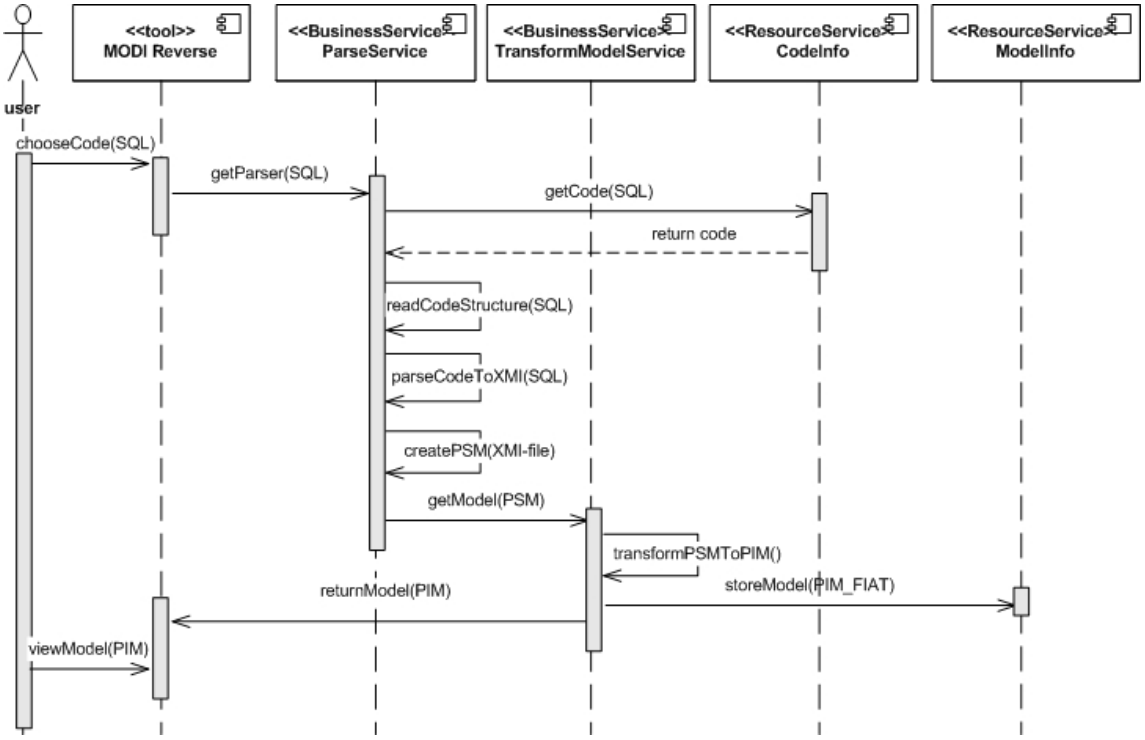


Figure 7-1 MODI reverse process for Fiat

Figure 7-2 shows the process for Bosch using MODI reverse.

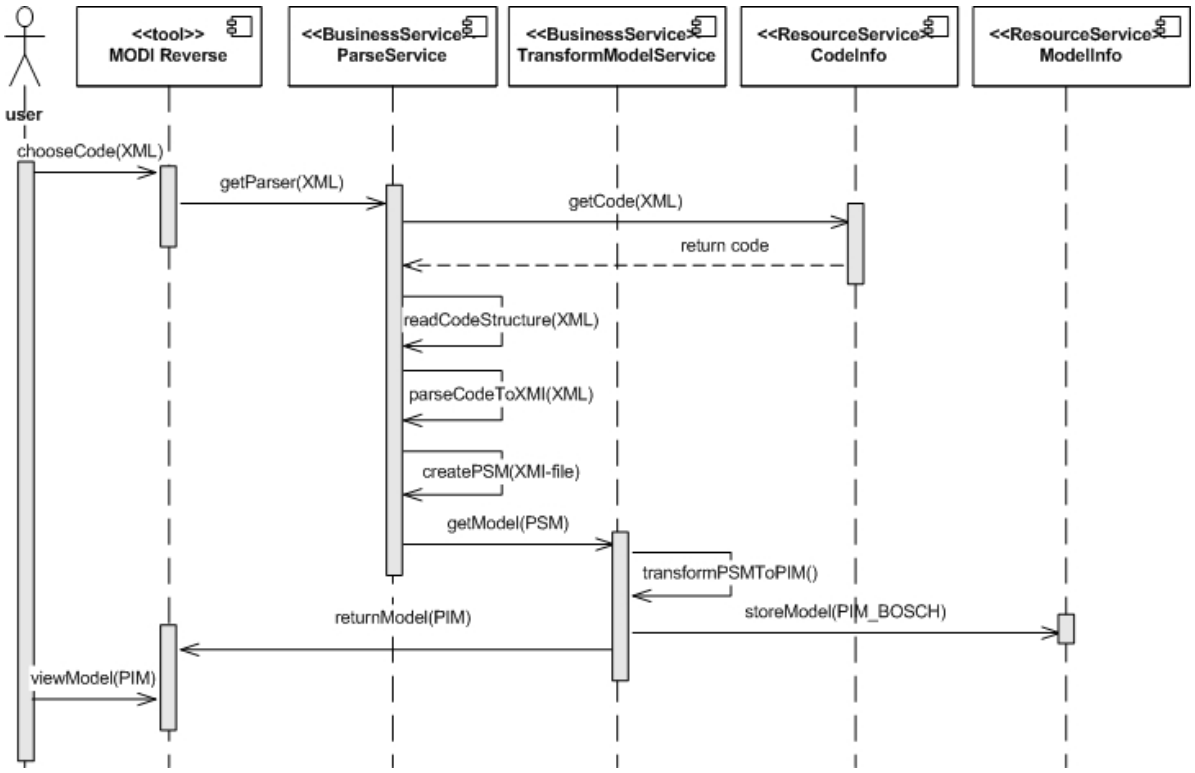


Figure 7-2 MODI reverse process for Bosch

Figure 7-3 shows the PIM (PIM_A) produced using MODI reverse on the code for Fiat.

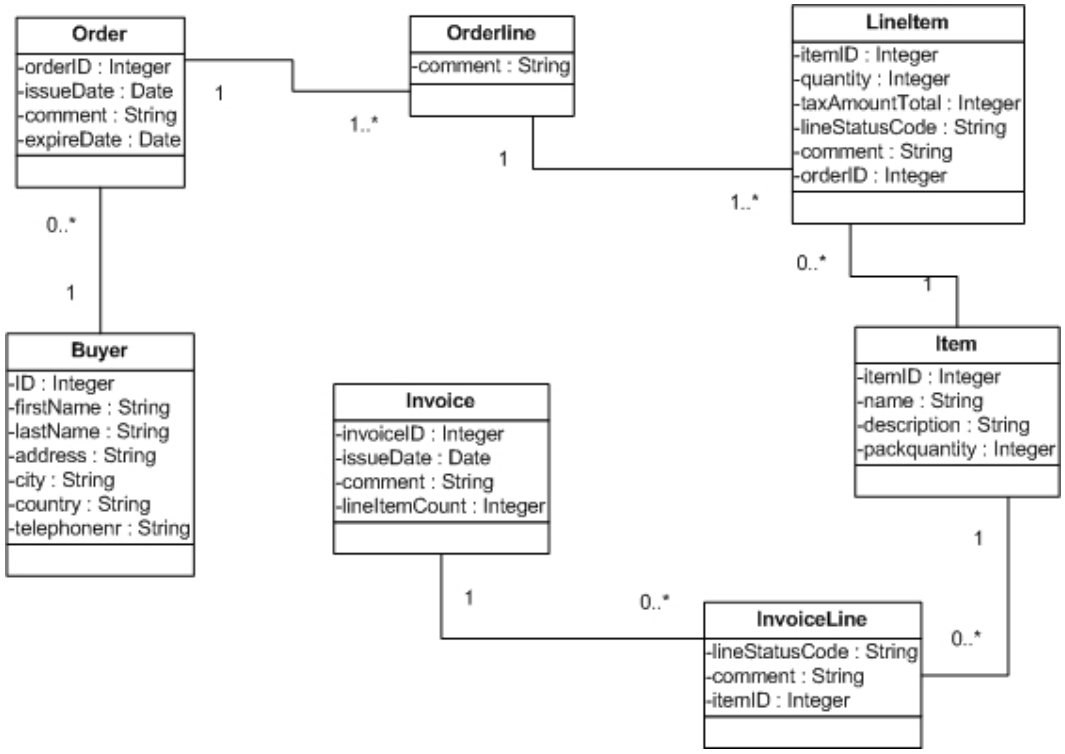


Figure 7-3 PIM_A for customer Fiat

Figure 7-4 shows the PIM (PIM_B) produced using MODI reverse on the code for Bosch.

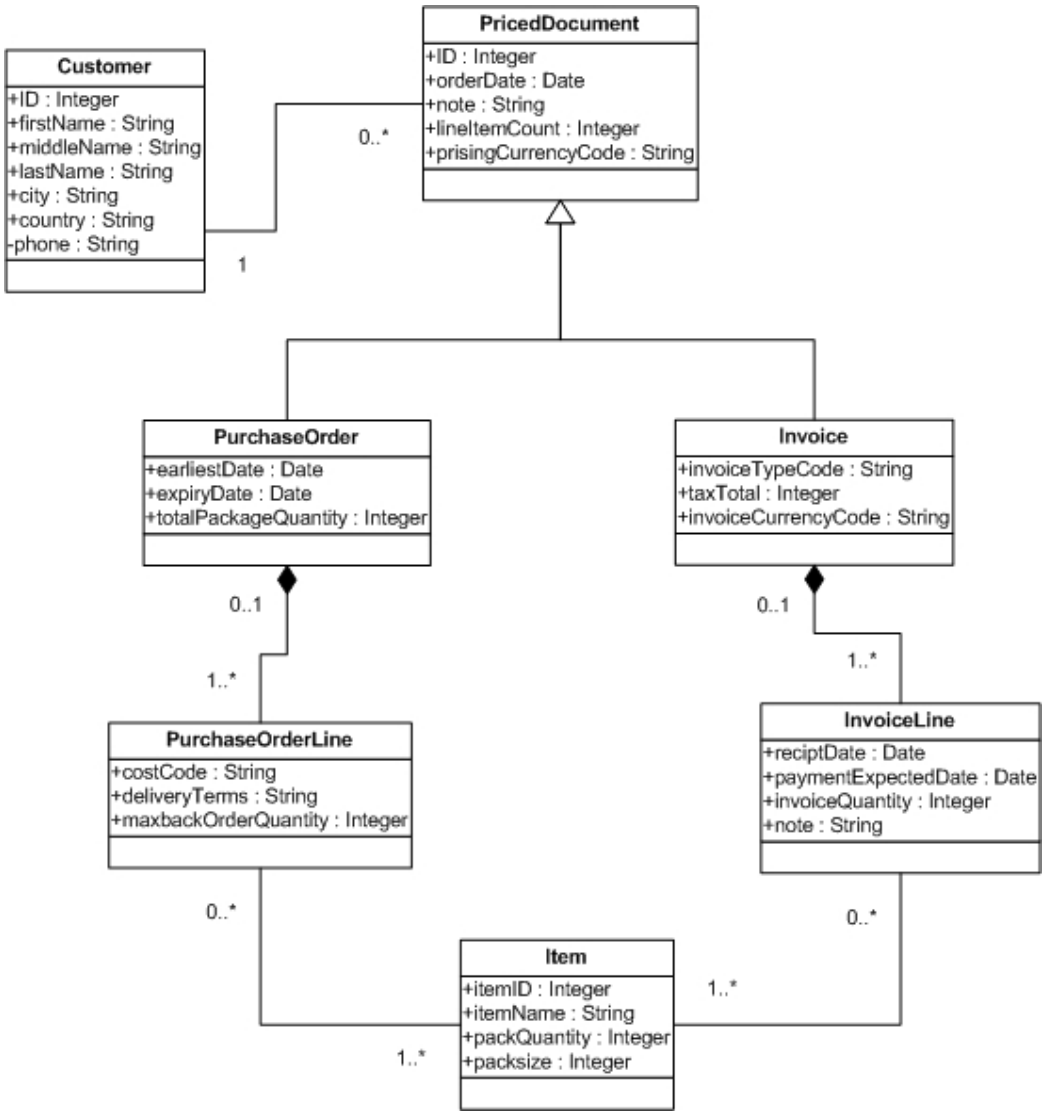


Figure 7-4 PIM_B for supplier Bosch

Both of the models are now represented as PIMs, and stored in a model repository. Next step is to map PIM_A and PIM_B.

In Table 7-1 the differences from PSM to PIM is described for department A. We will only consider a selection of the classes in PSM.

Table 7-1 Differences from PSM to PIM for Fiat

Attribute	Data type Code / PSM	Data type PIM	Description
Order			
orderID	INTEGER(10)	Integer	When transforming Integer with length of 10, the Integer in PIM will be transformed to maxInt. This is done to ensure that instances are transformed as whole.
issueDate	DATE	DATE	The datatype DATE in SQL for Fiat has the same representation as in PIM; namely dd.mm.yy
comment	VARCHAR(50)	String	VARCHAR is transformed to String in PIM. The String will have the same length as in PSM.
expiryDate	DATE	DATE	The datatype DATE in SQL for Fiat has the same representation as in PIM; namely dd.mm.yy
ID	INTEGER(5)	Integer	VARCHAR is transformed to String in PIM. The String will have the same length as in code/ PSM.

7.2 Use the MODI Mapper

The user from Fiat can start mapping with the aid of the data integration tool, MODI Mapper. Both of the PIMs are loaded into MODI Mapper, see Figure 7-5. In addition the user can double-click on the classes and the metadata information about the models is shown. This information is shown in the properties pane below the models.

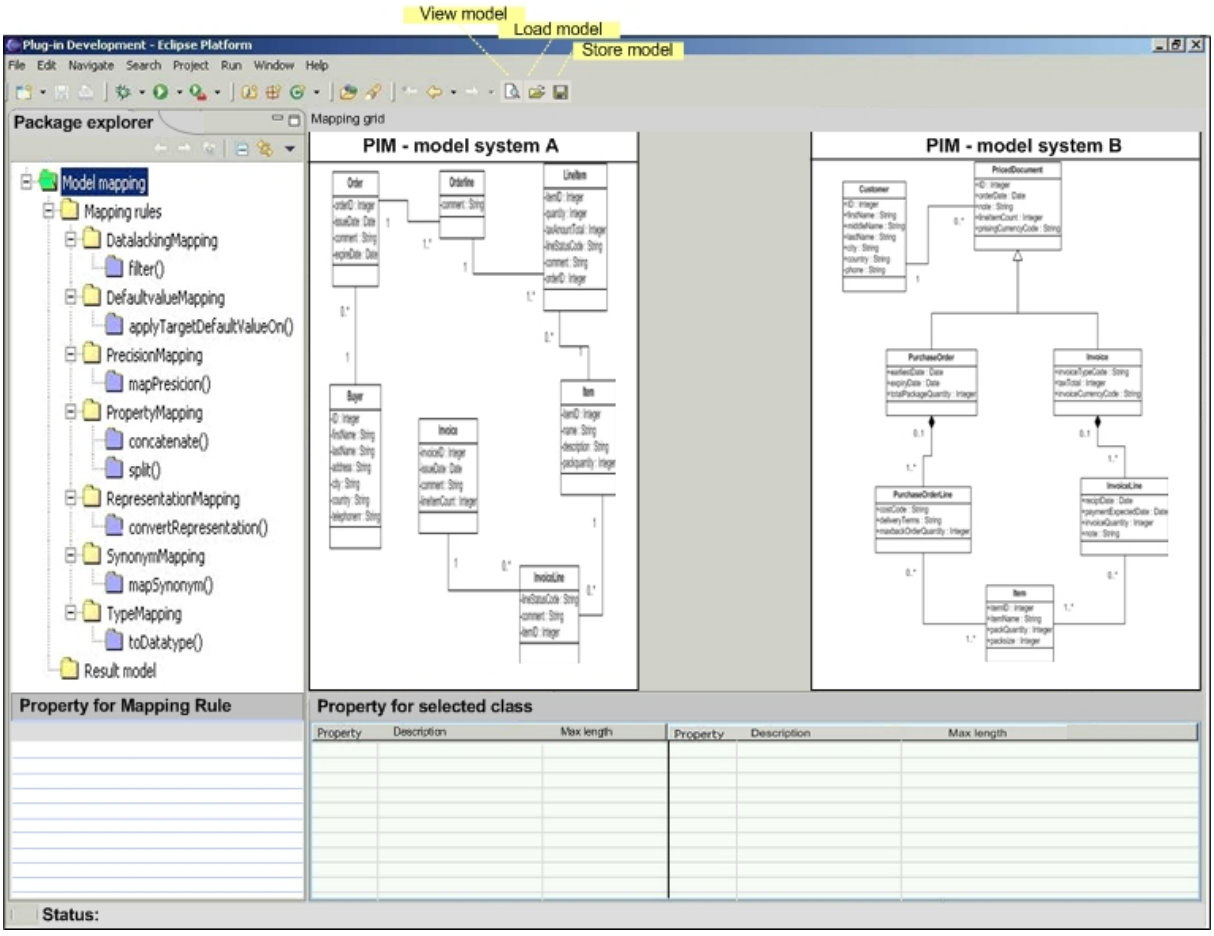


Figure 7-5 PIMs loaded in MODI Mapper tool

It is suggested how the data integration problems defined between class Order in PIM_A and class PurchaseOrder in PIM_B, and class Buyer in PIM_A and class Customer in PIM_B can be solved. Table 7-2 lists a mapping table that shows the differences between PIM_A and PIM_B attributes, and which one of them need to be mapped with the aid of which mapping rule.

Table 7-2 Mapping table

PIM_A	PIM_B	Data integration problem	Mapping rule
OrderProductsService			
Order	**PurchaseOrder**		
Ordered	ID	- synonyms - data precision conflict	1.SynonymMapping 5.PrecisionMapping
issueDate	orderDate	- synonyms - representation conflict - data precision conflict	1.SynonymMapping 3.RepresentationMapping 5.PrecisionMapping
Comment	note	- synonyms - data precision conflict	1.SynonymMapping 5.PrecisionMapping
Not handled	lineItemCount	- data lacking	
Not handled	prisingCurrencyCode	- data lacking	
Not handled	earliestDate	- data lacking	
expiryDate	expiryDate	- representation conflict - data precision conflict	3.Representation Mapping 5.PrecisionMapping
Not handled	totalPackageQuantity	- data lacking	
ID (foreign key, primary key in table Buyer)	***Not handled***	- data lacking	
Buyer	**Customer**		
ID	ID	-homonyms	
firstName	firstName	- differences in properties	4.PropertyMapping
	middleName		
lastName	lastName	No problem	No rule applied, direct linking
Address	***Not handled***	- data lacking	
City	city	No problem	No rule applied, direct linking
Country	country	No problem	No rule applied, direct linking
telephonenr	phone	- synonyms - attribute integrity constraint	1.SynonymMapping 7.TypeMapping

Further, the mapping rules are applied to the identified problems listed in the table above. The attributes in the PIMs that are not affected by any data integration problem is linked directly. The user uses the drag and drop technique to create links. Further the mapping rules are applied on the data integration problems defined in the Automotive scenario case. This is done by choosing the mapping rule for the specific data integration problem and using it on the problem. Figure 7-6 shows an selection of the mappings.

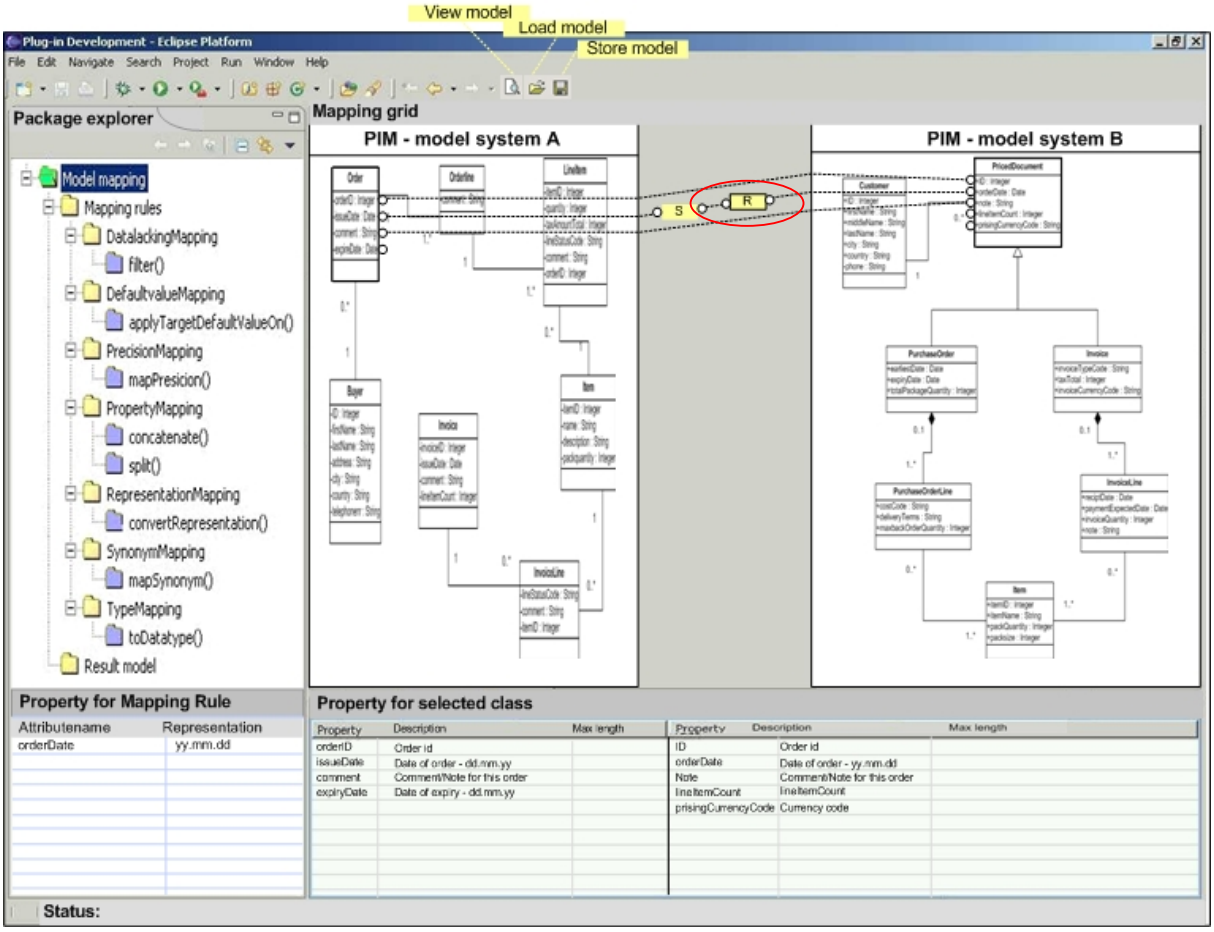


Figure 7-6 Mapping between PIMs

The mapping rules that are used in this example are: RepresentationMapping with convertRepresentation function (R) and SynonymMapping with mapSynonym function (S). Further, we explain the R an S function in detail. Input to the R function is the *issuedate* attribute from the source model's class *Order*. The function has an output link connected to the target attributes *orderDate* from the target model's class *PricedDocument*. These attributes have different representation formats and are synonyms. The R function converts the source attributes' representation format and the S function establishes a link between the source and target attribute. These functions are linked to each other. From the figure we can se that the classes that are being mapped and the R function have a bold frame. The classes' bold frame indicates that these classes have metadata presented in the *Property for selected class* pane. The functions' bold frame indicates that this function's properties are displayed in the *Property for Mapping Rule* pane. From this pane the user can see the target attribute and chooses how to represent the source attribute. When the user has performed the mapping it should be a possibility to see the mapping result and approve the mapping.

Next, a resulting PIM is produced, and the user chooses to generate code for execution of the mappings. Since the customer uses SQL and the supplier uses XML,

implementation of transformation-code from SQL2XML shall be available before generating code. The user chooses the generate option in the MODI Mapper and executable code is automatically performed.

To represent the customer’s instances in the supplier’s format, the instances are transmitted through the mapping engine.

8 Evaluation of MODI Framework

In this chapter a description of why we have based our framework on MDA is given. Next, we discuss why we choose our solution in preference to another. Finally, we evaluate MODI Framework according to the requirements.

8.1 Benefits with a model-based approach to data integration

A model-based approach fits as a solution to prevent data integration problems, since models clarify not only syntax and structure of data but also the semantics. Also, the availability of technologies for manipulating models enhances their range of use.

To solve data integration problems, models can be used to represent metadata for semantic matching. A good thing about models is that they can be used for various purposes. An example is “to capture and precisely state requirements and domain knowledge so that all stakeholders may understand and agree on them” [67]. Several models can be used to capture requirements of a software system from various aspects. This enhances understanding of what is being built among involved stakeholders. Models have the quality to capture design in a mutable form separate from the requirements.

A model can contribute with helping to explore several architectures and design solutions easily before writing code or capture business need. This enhances and simplifies the ability of exchanging information, and allows enterprises to collaborating on same level.

Another advantage is that models can deal with complexity. For example, a model may abstract to a level that is logical to humans, without getting lost in details. Further, a model can appear at various levels of abstraction where models are defined by models. In chapter 3 it was specified that MOF represents the highest level in OMG’s metadata architecture, and it is a meta-metamodel for describing abstract languages. This enables models to focus on metadata in models. Metadata enrichment can contribute with semantic matching among different data items. Semantic information is a major aspect of models, in addition to visual presentation. The meaning of an application in e.g. a network is captured by the semantic aspect. The elements in a model hold the meaning of the model. Formal documentation of system semantics through modeling will increase software quality and extend the useful lifetime of designs.

With aid of specified technology such as MOF and XMI, it is possible to generate an interchange format for models and further integrate data represented in the models.

Model can be modelled with the modeling language, UML which is supported by several modeling tools, an examples is Rational Rose.

8.1.1 Arguments for basing our approach on MDA

As MDA is platform-independent at its core, and enhance use of models we base our framework on MDA. A central aspect of MDA is the concept of *model transformation*, in which one model is transformed into another model of the same system. We have based our solution on this aspect, but consider model transformation between models of different enterprise systems. The main reason for choosing MDA is its promotion of machine-readable models, and use of the modeling standards UML and MOF. These modeling standards are independent of any middleware platform, and will represent common features. Another aspect is that MDA is independent of language. Interoperability will be most transparent within an application category. Further, it supports integration of legacy systems to MDA. They may be brought into MDA by wrapping them with a layer of code that is consistent with an MDA core model (ADM). Enterprises can define their business needs in models that are specific and independent to any platforms.

MODI Framework proposes how to map between two enterprises PIMs. Even though mapping between PIMs benefits in many ways, it comes back at the point-to-point problem. To avoid this, enterprise PIMs can be mapped to a standard PIM. MODI Mapper will be able to manage this, since it in principal is supposed to map between two PIMs. It does not matter if the other one is a standard. The condition is that enterprises have to agree upon mapping towards a standard PIM.

The OR is a metadata repository for departments. Departments is using the data definitions defined in OR. From this we can state that OR is a standard agreement for the departments about metadata. Hence, departments do not need to do mapping with each other. If OR uses PIM to describe data definitions, the departments can map their PIMs to the OR's PIM.

8.2 Evaluation of MODI Framework – a model-based approach

8.2.1 Metadata enrichment

We conclude that the MODI Framework specifies how to integrate data with different syntax, structure and semantics. This activity is done with aid of models at a platform independent level. It supports for enrichment of models metadata by integrating metadata, and making it available with the aid of the data integration tools, MODI Reverse and MODI Mapper. However, a drawback is that there could always be better annotation on data. One solution would be to have an annotation attribute in the

metaclasses. This will enable more precise meaning of the data. Another solution would be to use ontologies [8], like proposed by ATHENA (recall chapter 4) .

8.2.2 Mapping rules

The generic mapping metamodel specifies how to map between models with aid of mapping rules. The benefit with the mapping rules is that they contribute with specifying how data integration problems can be solved. MODI Framework specifies mapping rules only for the data integration problems defined in this thesis, see section 2.5. This does not mean that other data integration problems can not be supported by our solution. The mentioned problems are general examples, and specifications for other similar problems may be supported by MODI Framework as further work. Another issue is that a mapping rule can contain several options. As more problem cases are identified within one data integration problem, methods applying the different cases can be added.

8.2.3 Platform independent data model

The ADM approach is not suggested to be used for re-engineer legacy systems. However, in MODI Framework this approach is used for transforming system code or PSMs to PIMs for the purpose of having enterprises to collaborate through PIMs. By having enterprise systems data represented in PIMs, turns the attention away from use of a specific platform. In addition, this increases the ability for collaboration with enterprises' using any type of platform. The condition is that enterprises need to present their data in PIMs.

8.2.4 Tool support – MODI Reverse and MODI Mapper

The reverse engineering tool MODI Reverse is specified to transform platform specific code into a PIM. The advantage with this tool is that the developer can use written transformation rules by using ATL.

The intension of the data integration tool MODI Mapper is specified to manage data mapping. By mapping models it is easier for the user to see the structure of what is being mapped. An alternative approach is to use a tree structure, but the drawback that a tree structure does not easily display structure and relationships between elements. However, a model is better off on showing relationships between elements. Also, the elements datatype are visible in the models (at M1 level). Models give the user more information about what is being mapped against what.

With aid of a data integration tool like the specified MODI Mapper, enterprises can integrate data with syntactic, structured and semantic differences. Allowing enterprises to retrieve metadata with aid of the data integration tool enhances this opportunity.

The advantage with the code generation activity specified as an engine in the MODI Mapper is that it automatically generates code for mapping between the models. A challenge with our approach is to implement transformations between specific platforms. There is a drawback with this challenge because it demands a lot of work to implement specific transformations. The enterprises do not have to use an intermediate format to transform their instances.

8.3 Alternative solution to MODI Mapper

The MODI Mapper shall handle one transformation for the instances. In particular, the instances are not transformed into some intermediate format, but directly into the specific format. The advantage is that a lot of time is not used on transformation. However, this solution is suitable for a few amounts (between five and ten) of specific platforms. When more and more platforms are included, this solution leads to N^2 transformation between platforms. The code for instance transformation from one platform into another can exist from before and be re-used. QVT shall be used in our solution to map between M2 models.

An alternative solution to mapping models, and generate code is to use QVT on M1 models. As described earlier, QVT performs mapping on metamodels (M2 level) and transformation on models (M1 level). In case the models have same abstract language such as UML, a mapping between metamodels is not necessary. However, by performing mapping on models, in the same manner as QVT performs mapping between metamodels, it is possible to perform transformation on instances instead of executing transformation on models. Figure 8-1 illustrates this alternative solution.

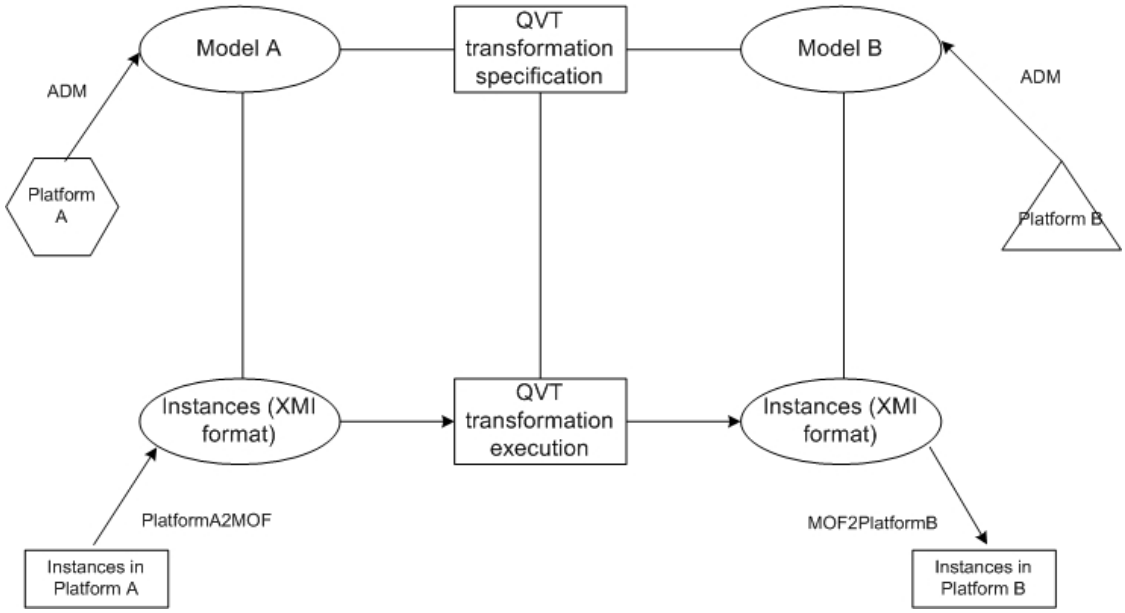


Figure 8-1 Alternative solution with use of QVT

In this solution the QVT transformation specification will create mapping between two models, and generate an engine (transformation execution) to perform the transformation of instances from platform A's XMI representation to platform B's XMI representation. The benefit is that the developer does not need to create the engine to perform transformation. The enterprises instances are suppose to be represented in a common format, XMI. This is because the QVT transformation execution is automatic generated from the mapping performed. The drawback is to transform instances from platform A to MOF and MOF to platform B since QVT transformation execution operates on XMI format. Dissimilar from our solution, this approach uses more time on transformation of instances. However, the advantage is that the instances can be represented in XMI as a platform independent format. This solution does not result in N^2 transformations of the specific platforms.

Table 8-1 shows a summary of the MODI Framework evaluations.

Table 8-1 Evaluation of MODI Framework

Requirements	Description
Metadata enrichment	MODI Framework specifies how to integrate data with different syntax, structure and semantics, by integrating PIMs and metadata.
Synonyms	MODI Framework specifies how to integrate data that are synonyms, by the metaclass SynonymMapping.
Homonyms	MODI Framework specifies how to manage data that are homonyms, by semantic enrichment and availability of semantic.
Data representation	MODI Framework specifies how to deal with data representation conflict by the metaclass RepresentationMapping
Differences in properties	MODI Framework specifies how to deal with differences in properties by the metaclass PropertyMapping
Data precision	MODI Framework specifies how to deal with data precision conflict by the metaclass PrecisionMapping
Default value	MODI Framework specifies how to deal with default value conflict by the metaclass DefaultvalueMapping.
Attribute integrity constraint	MODI Framework specifies how to deal with conflict concerning attribute integrity constraint by the metaclass TypeMapping.
Data lacking	MODI Framework specifies how to deal with data lacking by metaclass DatalackingMapping
Platform independent data model	MODI Framework supports for data integration through PIMs as a data model. The PIMs for data integration are versatile and self-explanatory.
Tool support	MODI Framework has specified how to develop the tools MODI Reverse and MODI Mapper, which can manage data mapping and integration.

From the evaluation we can conclude that MODI Framework fulfills the requirements listed above. The tools specified in the MODI Frameworks' solution approach shall be implemented at Sintef. Since MDA supports business interoperability, driven by business needs, our solution is open for extensions, e.g. address interoperability problems concerning services, processes and non-functional aspects.

8.4 Summary

In this chapter we have specified why we have based our solution on MDA, discussed why we chose our solution in preference to another and evaluated MODI Framework according to the requirements defined in chapter 2, section 2.5.

9 Conclusion and future work

9.1 Conclusion

This thesis outlines a model-based approach to data integration between heterogeneous enterprise systems. A review on interoperability is given, and an introduction of MDA to facilitate interoperability. Two problem examples specifying data integration problems are analysed, and requirements to solutions for data integration are defined. Technologies related to interoperability, integration and mapping are examined. Existing solution approaches are analysed as solutions to the problem examples. Both related technologies and the existing solution approaches are evaluated according to the requirements defined. Further, our proposed solution, MODI Framework, is based on the analysis and examinations. The main emphasis of the solution is on how to develop tools to support integration of heterogeneous data from one enterprise's format into another enterprise's format with aid of models represented on a platform independent level. The proposed approach is based on MDA and presented as an interoperability framework. In particular, we specify how to develop tools to support data integration, and how enterprises can use the tools to simplify data integration tasks by mapping between PIMs. Mapping rules are applied on some general data integration problems with aid of the mapping tool. Furthermore, MODI Framework is applied to the problem examples, and evaluated according the requirements defined in this thesis.

9.2 Future work

During the work of the framework and through evaluation, some aspects that need further investigation have been discovered. A test implementation should be constructed in order to verify that the framework is functioning as anticipated. This would include complete implementation of MODI Reverse and MODI Mapper to provide full tool support to data integration. These specified tools will be realized by developers at Sintef. Another extension would be to propose solutions for achieving process and service interoperability, and also considering non-functional aspects with reference to interoperability.

Identifying processes that are similar in spite of differences is still an area that is not much investigated. However, there exist few attempts in the area like the shared repository ebXML. Enterprises may discover each others business offerings as well as establish agreements to invoke cooperation between their respective business processes via the shared repositories.

An initiative that intends to provide a Web Service discovery framework for more efficient ontology-based and metadata driven service discovery is *Semantic Web Enabled Web Services* (SWWS) [68]. It intends to move the semantic information out of the Web Service description, semantic annotate the Web Service description and keep it separate in shared ontologies so other services can access it. A combination of model based- and ontology based approach to data integration is a discussion much emphasised on.

APPENDIX.A

A.1 Definitions

ADM, Architecture-Driven Modernization. A reverse engineering approach defined by OMG which specifies how to integrate and modernise existing legacy systems according to new business needs.

ATHENA, Advanced Technologies for interoperability of Heterogeneous Enterprise Networks and their Applications. An integrated Project funded by the European Commission where the main objective is to remove interoperability barriers.

ATL, Atlas Transformation Language. A QVT-based transformation language.

CWM, Common Warehouse Model. A specification that describes metadata interchange among data warehousing, business intelligence, knowledge management and portal technologies

Data is information, in any form, on which computer programs operate.

Data definition is metadata.

ebXML, Electronic Business XML. An initiative started by OASIS and UN/CEFACT which is a set of specifications enabling enterprises to conduct business over the Internet, independent of their size and geographical location.

EDI, Electronic Data Interchange

EDIFACT, . A message platform for message-oriented computing.

GMT, Generative Model Transformer

Information is a collection of data which gives meaning, knowledge, instruction, etc.

Integration is transfer of data between different companies using networks, such as the Internet

Interoperability is the ability of two or more systems or components to exchange information and to use the information that has been exchanged.

JMI, Java Metadata Interface. It provides a platform-independent infrastructure for modelling, representing and querying the meaning of a data source's metadata, application, tool, and data integration can be improved.

Mapping is an operation that associates each element of a given set with one or more elements of a second set.

Mapping rules are transformation rules and techniques used to modify one model in order to get another model.

MDA, Model-Driven architecture. An approach defined by OMG to use models in software development, and aims to provide a platform-independent approach to domain-specific application development.

MDD, Model-Driven Development. An architectural business-driven approach for developing software systems based on requirements derived from enterprise and business models.

Metadata is a set of data that describes and gives information about other data.

Metamodel is a description or definition of a well-defined language in the form of a model.

Model is a description of (part of) a system written in a well-defined language.

MOF, Meta Object Facility. A model-driven framework for specifying, constructing, managing interchange, and integrating metadata in software systems. It represents metamodels and how to manipulate them.

NDR, National Data Registry. The name we use in this thesis to refer to the project concerning OR.

Middleware is software that occupies a position in a hierarchy between operating systems, whose task is to ensure that software from a variety of sources will work together correctly.

OASIS, Organization for the Advancement of Structured Information. OASIS is an international nonprofit consortium that promotes open collaborative development of interoperability specifications.

OMG, Object Management Group. A non-profit organisation with mission to help computer users solve integration problems by supplying open, vendor-neutral interoperability specifications.

OR, The Register of Reporting Obligations of Enterprises. A national infrastructure for handling reporting obligations established by the Brønnøysund Register Centre.

PIM, Platform Independent Model. A model with a high level of abstraction defined in UML. It specifies services and interfaces independent of software technology platforms.

PSM, Platform Specific Model. A model which adheres to constraints and conventions imposed by a specific software technology platforms.

QVT, Query, View, Transformation. A standard specification of a language suitable for querying and transforming models which are represented according to a MOF metamodel is at the time of writing.

SQL, Structured Query Language. A query language based on the relational model of database systems which includes statements for modifying the database, and for declaring a database schema.

Transformation is the automatic generation of a target model from a source model.

TOR, Named after a Norse God. A project initiated by the Brønnøysund Register Centre which is a continuation of OR.

UBL, Universal Business Language. Defines a standard of electronic XML business syntax documents such as purchase orders and invoices initiated by OASIS.

UML, Unified Modeling Language. A standard notation for constructing models of object-oriented software which allows an application model to be constructed, viewed, developed, and manipulated in a standard way at analysis and design time.

UN/CEFACT, Unified Nations Centre for Trade Facilitation and Electronic Business. A body of the United Nations whose mandate is to support the worldwide development in the area of trade facilitation and electronic business

W3C, World Wide Web Consortium.

XMI, XML Meta Interchange. An interchange format for models in the language defined using a metamodel described in the MOF. This interchange format is an OMG standard.

XML, Extensible Markup Language. A standard format for data representation and exchange in the Internet. It is an open and freely available document from W3C.

XSLT, EXtensible Stylesheet Language (XSL) Transformations. A language for transforming an XML document into another XML document.

APPENDIX.B

B. 1 XML and XSLT example

This example is taken from [69].

XML file

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<tool>
  <field id="prodName">
    <value>HAMMER HG2606</value>
  </field>
  <field id="prodNo">
    <value>32456240</value>
  </field>
  <field id="price">
    <value>$30.00</value>
  </field>
</tool>
```

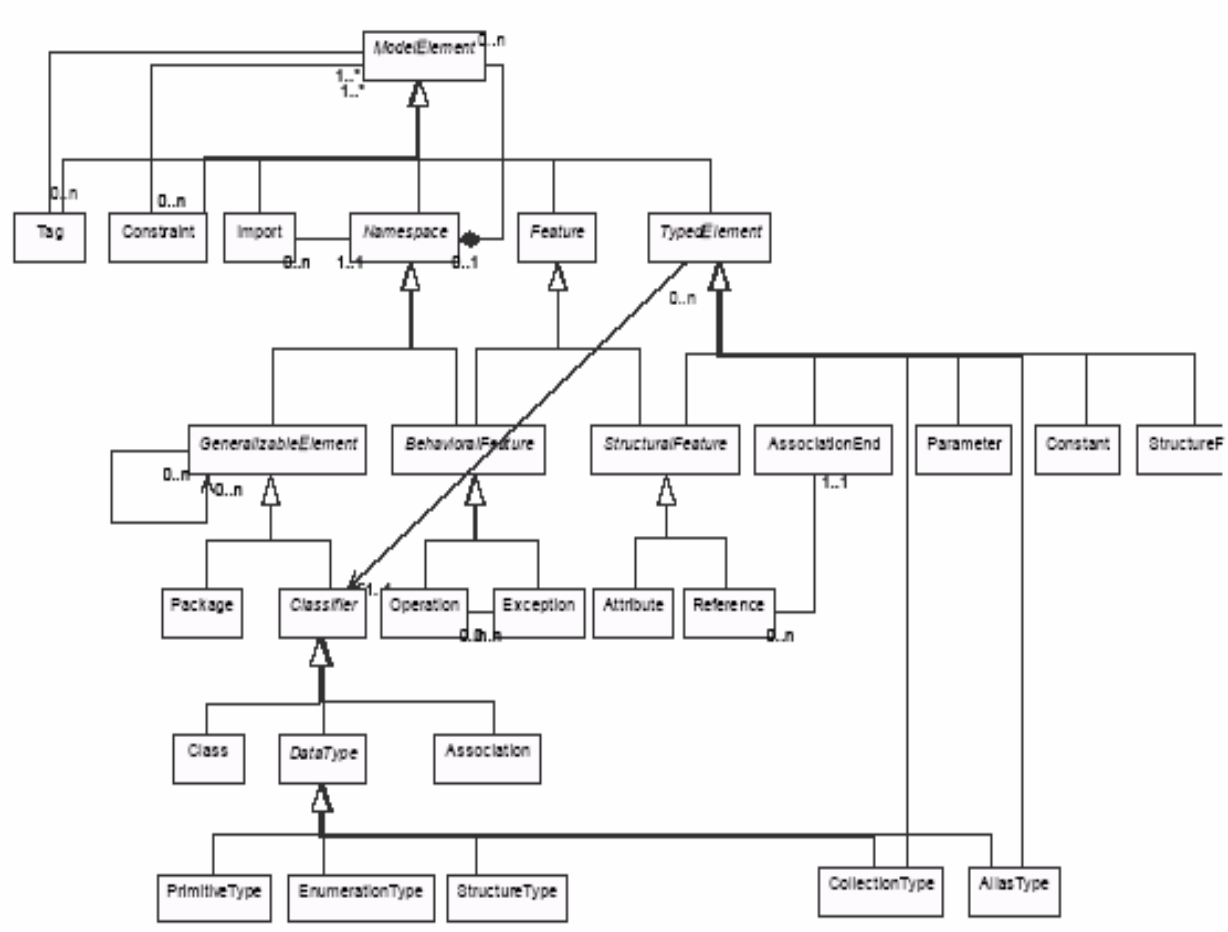
XSL file

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<xsl:stylesheet version="1.0"
xmlns:xsl="http://www.w3.org/1999/XSL/Transform">
<xsl:template match="/">
<html>
<body>
<form method="post" action="edittool.asp">
<h2>Tool Information (edit):</h2>
<table border="0">
<xsl:for-each select="tool/field">
<tr>
<td>
<xsl:value-of select="@id"/>
</td>
<td>
<input type="text">
<xsl:attribute name="id">
  <xsl:value-of select="@id" />
</xsl:attribute>
<xsl:attribute name="name">
  <xsl:value-of select="@id" />
</xsl:attribute>
<xsl:attribute name="value">
  <xsl:value-of select="value" />
</xsl:attribute>
```

```
</input>
</td>
</tr>
</xsl:for-each>
</table>
<br />
<input type="submit" id="btn_sub" name="btn_sub" value="Submit" />
<input type="reset" id="btn_res" name="btn_res" value="Reset" />
</form>
</body>
</html>
</xsl:template>
</xsl:stylesheet>
```

B.2 MOF metamodel

The figure shows the complete MOF metamodel [17].



APPENDIX.C

C.1 Description of the subset of UML

The *Class* is the central element and inheritance is allowed. However, multiple inheritances are not allowed. Further, a class can contain a number of *attributes* that are either simple attribute of a given *datatype* or it can be an *association* to another class. Also methods are not allowed, because this is an information model and not a program.

Furthermore, a datatype has a name and minimum one validation rule. In the case where a datatype already is inherited from another inheritance, the validation rule of the super-datatype adds one or several validation rules. E.g. it might be that a decimal-integer-data-type has the name *Integer* and the following validation rules:

- Characters '0' to '9' and '+' and '-' allowed
- Maximum one occurrence of '+' or '-'
- A '+' or '-' must precede all other characters

Based on this, other data types can be derived. E.g. the decimal-number-data-type with name *Number* may inherit the Integer type and then add three new rules:

- Maximum one occurrence of '.' or ','
- Maximum one occurrence of 'E' or 'e'
- A '.' or ',' must precede the 'E' or 'e'

APPENDIX.D

D.1 A description of ATHENA A projects:

In MDA, interoperability solutions are driven by business needs first and software solutions second. Based on this, models exist on different levels of abstraction. First of all, Project A1 focuses on *Enterprise Modelling* (EM). A set of enterprise aspects, such as business operations, are described in an enterprise model. Second, Project A2 is concerned about how to get business processes to interoperate. Third, semantic annotation and ontology-based reconciliation is considered in Project A3. Forth, Project A5 focuses on mapping between PSM and Project A6 focuses on mapping between PIM. And finally, Project A4 integrates all of the above mentioned A-projects.

To specify further, Project A4: *Interoperability Framework and Services for a Networked Enterprises* represents the way to compromise and complement the

approaches, methodologies and results of projects in Action Line A, A1-A6. More precisely, the results and methodologies reached in the projects are exploited by Project A4 into a conceptual, actuation and technical *ATHENA Interoperability Framework* (AIF).

References

1. IEEE (1990): *IEEE Standard Computer Dictionary: A Compilation of IEEE Standard Computer Glossaries*.
2. ATHENA (2005): *Specification of a Basic Architecture Reference Model*, (Confidencial deliverable).
3. European Communities (2004): *European Interoperability Framework for Pan-European eGovernment Services*, http://europa.eu.int/idabc/en/document/3761 (Last visited 03.03.05).
4. Sheth, A.P. and J.A. Larson (1990): *Federated Database Systems for Managing Distributed, Heterogenenous, and Autonomous Databases*. Vol. 22(3):183-236: ACM Computing Surveys.
5. INTEROP (2004): *INTEROP Portal*, http://www.interop-noe.org/ (Last visited 10.10.04).
6. Continuum Systems (2005): *Application Integration*, http://www.continuum-systems.com/enterprise-application-integration.htm (Last visited 15.05.05).
7. Microsoft (2005): *Applying Microsoft Patterns to Solve EAI Problems*, http://msdn.microsoft.com/library/default.asp?url=/library/en-us/dnbiz2k2/html/bts_eai_pattern.asp (Last visited 20.05.05).
8. Missikoff, M. and F. Taglino (2004): *An Ontology-based Platform for Semantic Interoperability*, http://www.athena-ip.org/ (Last visited 15.06.05).
9. Siegel, J. (2004): *OMG's model driven architectures*, http://www.eurescom.de/message/messageJun2002/omg.asp (Last visited: 12.01.05).
10. Object Management Group (2005): *Object Management Group Homepage*, http://www.omg.org/ (Last visited 14.02.05).
11. Kleppe, A., J. Warmer and W. Bast (2003): *MDA Explained The model Driven Architecture: Practice and Promise*: Addison-Wesley. 152.
12. Berre, A.-J., B. Elvesæter, A. Hahn and T. Neple (2005): *Towards an Interoperability Framework for ModelDriven Development of Software Systems*, (Unpublished document).
13. Hauch, R.M. (2002): *Enterprise Information Integration and the OMG's MDA and MOF*, http://www.omg.org/news/meetings/workshops/UML2002-Manual/03-3_Enterprise_Information_Integration_and_the_OMGs_MDA_and_MOF.pdf (Last visited 06.06.05).
14. Object Management Group (2003): *MDA Guide Ver. 1.0.1.*, http://www.omg.org/docs/omg/03-06-01.pdf (Last visited 14.04.05).
15. Object Management Group (2005): *OMG Model Driven Architecture™: How Systems Will Be Built*, http://www.omg.org/mda/model_driven_architecture.htm (Last visited: 03.01.05).
16. Unified Modeling Language (2005): *Unfied Modeling Language Homepage*, http://www.uml.org/ (Last visited: 04.01.05).
17. Object Management Group (2005): *Meta-Object Facility (MOF™), version 1.4*, http://www.omg.org/technology/documents/formal/mof.htm (Last visited: 05.01.05).

18. Object Management Group (2005): *Data Warehousing, CWM™ and MOF™ Resource Page*, http://www.omg.org/technology/cwm/ (Last visited: 04.01.2005).
19. Object Management Group (2001): *Model Driven Architecture(MDA)*, http://www.omg.org/cgi-bin/doc?ormsc/2001-07-01 (Last visited 15.05.05).
20. Object Management Group (2005): *Architecture-Driven Modernization (ADM)*, http://www.omg.org/adm/ (Last visited: 28.04.05).
21. Modelbased (2005): *Modelbased Homepage*, http://www.modelbased.net/mda_tools.html (Last visited 14.06.05).
22. Brønnøysund (2005): *Brønnøysund Home Page*, http://www.brreg.no (Last visited 11.11.04).
23. Altinn (2004): *Altinn Home page*, https://www.altinn.no/cms/1044/altinn/ (Last visited: 12.12.04).
24. World Wide Web Consortium (2005): *XForms - The Next Generation of Web Forms*, http://www.w3.org/MarkUp/Forms/ (Last visited: 04.02.05).
25. Marco, D. (2000): *Building and Managing the Meta Data Repository: A Full Lifecycle Guide*: Wiley Computer Publishing.
26. Brønnøysund (2005): *datadef_komplett*, http://ftp2.brreg.no/sched/ornett/datadef_komplett.data (Last visited: 23.07.05).
27. Brønnøysund (2005): *Oppgaveregisterets kodestruktur*, http://w2.brreg.no/oppgaveregisteret/hjelp_kodestruktur.jsp (Last visited: 23.07.05).
28. International Organization for Standardization (2000): *International Organization for Standardization HomePage*, http://www.iso.org/iso/en/CombinedQueryResult.CombinedQueryResult?queryString=ISO+11179 (Last visited: 12.01.05).
29. Cover Pages (2002): *UN/CEFACT ebXML Core Components Technical Specification Approved for Implementation Verification.*, http://xml.coverpages.org/ni2002-12-19-a.html (Last visited: 10.11.05).
30. Brønnøysund (2005): *Oppgaveregisterets datadefinisjoner*, http://w2.brreg.no/oppgaveregisteret/datadefinisjon_treffliste.jsp?feltnavn=organisasjon (Last visited: 23.07.05).
31. ATHENA (2005): *ATHENA Home Page*, http://www.athena-ip.org/ (Last visited 15.01.05).
32. ATHENA (2005): *ATHENA Newsletter*, http://www.athena-ip.org/newsletter/Athena_Newsletter_2.pdf (Last visited 15.02.05).
33. Garcia-Molina, Ullman and Widom (2002): *Database Systems - The Complete Book*: Prentice Hall.
34. World Wide Web Consortium (2005): *Extensible Markup Language (XML)*, http://www.w3.org/XML/ (Last visited: 03.02.05).
35. World Wide Web Consortium (2005): *World Wide Web Consortium Homepage*, http://www.w3.org/ (Last visited 13.04.05).
36. World Wide Web Consortium (2005): *The Extensible Stylesheet Language Family (XSL)*, http://www.w3.org/Style/XSL/ (Last visited 13.04.05).
37. World Wide Web Consortium (2005): *XSLT Homepage*, http://www.w3.org/TR/xslt (Last visited 14.05.05).
38. World Wide Web Consortium (2005): *XPath Homepage*, http://www.w3.org/TR/xpath (Last visited 16.05.05).
39. OASIS (2004): *ebXML Home Page*, http://www.ebxml.org/ (Last visited: 03.10.2004).

40. OASIS (2005): *OASIS Homepage*, http://www.oasis-open.org/home/index.php (Last visited 12.02.05).
41. United Nations Economic Commision for Europe (2005): *UN/CEFACT Homepage*, http://www.unece.org/cefact/ (Last visited 15.02.05).
42. Claben, M. (2004): *ebXML: Global Standard for Electronic Business*, http://www.webreference.com/xml/column46/ (Last visited 18.01.04).
43. OASIS (2005): *Universal Business Language*, http://www.oasis-open.org/committees/ubl (Last visited 13.03.05).
44. OASIS (2004): *OASIS ebXML Registry TC*, http://www.oasis-open.org/committees/regrep/faq.php.
45. Mertz, D. (2001): *Understanding ebXML - Untangeling the business Web for the future*, http://www-106.ibm.com/developerworks/xml/library/x-ebxml/.
46. SUN (2002): *Java Metadata Interface (JMI) Specification*, http://java.sun.com/products/jmi/.
47. Grose, T.J., G.C. Doney and S.A. Brodsky (2002): *Mastering XMI*: OMG.
48. Object Management Group (2002): *MOF 2.0 Query / Views / Transformations RFP*, http://www.omg.org/docs/ad/02-04-10.pdf (Last visited 03.06.05).
49. ATL (2005): *ATL Homepage*, http://www.sciences.univ-nantes.fr/lina/atl/atlProject/presentation/ (Last visited 16.06.05).
50. Eclipse (2005): *Generative Model Transformer*, http://www.eclipse.org/gmt/ (Last visited 14.06.05).
51. Modelbased (2005): *UMT-QVT Homepage*, http://umt-qvt.sourceforge.net/ (Last visited 07.06.05).
52. Microsoft (2005): *Microsoft Biztalk Server Home*, www.biztalk.org (Last visited 20.03.05).
53. Altova (2005): *Altova Homepage*, www.altova.com (Last visited 02.04.05).
54. Microsoft (2005): *Microsoft Homepage*, http://www.microsoft.com/.
55. Microsoft (2005): *Exchanging Data Over the Internet Using XML*, http://msdn.microsoft.com/msdnmag/issues/0400/cutting/default.aspx (Last visited 20.03.05).
56. Microsoft (2005): *Biztalk Mapper*, http://msdn.microsoft.com/library/default.asp?url=/library/en-us/bts_2002/htm/lat_xmltools_map_concept_mvem.asp (Last visited 20.03.05).
57. Mohr, S. and S. Woodgate (2001): *Professional Biztalk*: Wrox.
58. Altova (2005): *Altova Mapforce*, http://www.altova.com/products_mapforce.html (Last visited 02.04.05).
59. Thorbergsen, E. (2004): *Kravspesifikasjon for TOR prosjektet*, (Unpublished document).
60. Lilleng, J. and E. Thorgersen (2004): *A New Methodology for Designing Electronic Forms Promoting Reuse of Information Carriers*, http://www.idealliance.org/papers/dx_xmle04/papers/04-03-04/04-03-04.pdf (Last visited: 12.04.05).
61. Lilleng, J. (2005): *Towards Semantic Interoperability*, http://interop-esa05.unige.ch/INTEROP/Proceedings/eGovScientific/papers/5a3.pdf (Last visited: 12.04.05).
62. Sintef (2005): *Sintef Home Page*, http://www.sintef.no/ (Last visited: 15.04.05).
63. Eclipse (2005): *Eclipse Homepage*, http://www.eclipse.org/ (Last visited 20.06.06).

64. Kehn, D. (2003): *Extend Eclipse's Java Development Tools*, http://www-106.ibm.com/developerworks/opensource/library/os-ecjdt/ (Last visited 24.06.05).
65. Ghezzi, C. and M. Jazayeri (1997): *Programming language concepts*. 3rd ed: John Wiley & Sons.
66. Object Management Group (2005): *Catalog of OMG Modeling And Metadata Specifications*, http://www.omg.org/technology/documents/modeling_spec_catalog.htm#UML (Last visited 20.05.05).
67. Rumbaugh, J., I. Jacobson and G. Booch (2004): *The Unified Modeling Language Reference Manual*. Second edition ed: Addision-Wesley.
68. Information Society Technologies (2005): *Semantic Web Enabled Web Services (SWWS) Homepage*, http://swws.semanticweb.org/swws?cmd=show_entity&entity=Home+English (Last visited 15.06.05).
69. W3Schools (2005): *XSLT tutorial*, http://www.w3schools.com/xsl/default.asp (Last visited 15.05.05).