TECHNISCHE UNIVERSITÄT WIEN
VIENNA UNIVERSITY OF TECHNOLOGY

Business Informatics Group
Institut für Softwaretechnik und Interaktive Systeme

Evaluation of UML Model Transformation Tools

Diplomarbeit zur Erlangung des akademischen Grades eines Magisters der Sozial- und Wirtschaftswissenschaften

Vorgelegt bei Prof. Dipl. –Ing. Mag. Dr. Gerti Kappel
mitbetreuender Assistent: Dipl. –Ing. Dr. Gerhard Kramler

Wensheng Wang

Wien, 21. Juni 2005

Eidesstattliche Erklärung

Ich erkläre an Eides statt, dass ich die vorliegende Arbeit selbständig und ohne fremde Hilfe verfasst, andere als die angegebenen Quellen nicht benützt und die den benutzten Quellen wörtlich oder inhaltlich entnommenen Stellen als solche kenntlich gemacht habe.

Wien, 20.01.2005 Wensheng Wang

Acknowledgments

I would like to thank Prof. Dipl. –Ing. Mag. Dr. Gerti Kappel and Dipl. –Ing. Dr. Gerhard Kramler who give me the opportunity to work in this field and correct my Works.

I own a large debt to my family that help me to finish my study.

Thanks for the people who have supported me working on this thesis!

Abstract

MDA(Model Driven Architecture) defines a software development process. Three models build the core of the MDA: PIM (Platform Independent Model), PSM (Platform Specific Model) and Code. The developers of software are only required to concentrate on developing models in PIM format. The transformation to PSM and Code, which is the traditional understanding for UML model transformation can be automatically achieved by using general transformation tools.

The traditional way of developing UML transformations is usually to develop a specific one to one transformation using a fixed set of rules and profiles. The development of the transformation is usually complicated and time consuming. Due to the many different forms of UML, the transformation between these UMLs has been proved to be a very tough and inefficient.

Since the introduction of MDA, MOF(Meta Object Facility) and QVT (Query, View and Transformation) concepts from the Object Management Group(OMG), UML transformations come to a new era. Many working groups have presented their concepts and made their implementation using these definitions.

This diploma thesis gives an introduction to the principle of UML transformation according to MDA, MOF and QVT. It also gives an evaluation of the up to date model transformation tools based on these concepts to show their strength and weaknesses. These state of the art transformation tools are built according to OMG MOF 2.0 Query / View / Transformation. The evaluated tools are UMT, MTL, ATL, GMT, BOTL and OptimalJ.

In this diploma thesis the analysis methods described in a review of OMG MOF 2.0 QVT [Gardner03]are used. The evaluation of the transformation tools has been done in a set of aspects, such as scalability, simplicity and bi-directionality of mappings.

Kurzfassung

MDA(Model Driven Architecture) Definiert einen Software Entwicklungsprozess. Drei Modelle bilden den Kern der MDA: PIM (Platform Independent Model), PSM (Platform Specific Model) und Code. Die Softwareentwickler können sich auf die Entwicklung von Model in PIM Format konzentrieren. Die Transformation in PSM und Code, welche das traditionelle Verständnis für UML Model Transformation ist, kann durch Benutzung von generellen Transformations- Werkzeugen automatisiert werden.

Die konventionelle Methode, um UML Transformation zu entwickeln ist eine spezielle eins zu eins Transformation mit fixen Regeln und Profilen. Die Entwicklung einer Transformation ist meistens kompliziert und Zeit intensiv. Wegen der speziellen Eigenschaften der unterschiedlichen UMLs hat sich die Transformation zwischen diesen UMLs als eine sehr schwierige und nicht effizient Aufgabe herausgestellt.

Seit der Einführung von MDA, MOF(Meta Object Facility) und QVT (Query, View and Transformation) Konzepten durch die Object Management Group(OMG), kommen die UML Transformationen in eine neue Ära. Viele Arbeitsgruppen haben ihre Konzepte präsentiert und ihre Implementierungen unter diesen Definitionen gemacht.

Die vorliegende Diplomarbeit gibt eine Einführung in die Prinzipien der UML Transformation nach MDA, MOF und QVT. Es gibt auch eine Bewertung aktueller Werkzeuge für Model Transformationen, welche mit diesen Konzepten gebaut sind, und zeigt ihre Stärken und Schwächen. Die untersuchten Tools sind UMT, MTL, ATL, GMT, BOTL und OptimalJ.

In dieser Diplomarbeit sind die Analyse-Methoden, beschrieben in einem review von OMG MOF 2.0 QVT[Gardner03], angewendet. Die Transformationswerkzeuge wurden nach einer Reihe von Aspekten bewertet, wie Skalierbarkeit, Einfachheit and Bidirektionalität des Transformation.

Content

Figure and Table list

Chapter 1

Introduction

In the 21st century software is pervasive, the software industry has become one of the largest industries on the planet, and many of today's most successful companies are organizations built around the production of software and related services.[MDSD04]

Software is a critical part in the "engine room" of all technology-based and many service-based industries today. High software development costs have a huge economic impact, and poorly designed software that restrains user productivity possibly has an even larger impact. One result of these costs is significant pressure to shift problems to low-cost locations, usually referred to as off-shoring or outsourcing.[MDSD04]

Many business software vendors have been side-tracked by keeping pace with the constantly changing set of implementation technologies. Neither off-shoring nor the next product release from tool vendors that provide infrastructure such as integrated development environments, middle ware, databases, operating systems, etc. will solve productivity problems that are caused by crumbling architectural integrity of applications, poor dependency management within enterprise systems, and dysfunctional software development processes.[MDSD04]

Software Development under Model Driven Architecture (MDA) is a new software development paradigm which is introduced by OMG. It represents an emerging paradigm for industrialized software development that provides effective methods for dealing with the root causes of escalating software development costs and with the complexities of distributed software development environments.

The concept of Model-Driven Software Development is not new, and in particular the roots of Software Product Line Engineering (Domain Engineering) can be traced back to the mid 1970s. The concept reinvented now by OMG in 2001 is first the concept of Platform Independent Models (PIM), typically in UML, and the concept

of Platform Specific Models (PSM) for platform of choice like SOAP, XML, Java, EJB, CORBA, .NET, Web Services...

Unified Modeling Language(UML) UML (Unified Modeling Language) is a Standard language of OMG (http://www.omg.org/uml). UML defines several model types not only for application structure, behavior, and architecture, but also for business process and data structure. UML establishes the foundation of Model Driven Architecture (MDA). In fact, Every MDA is based on a UML model that can be either platform-independent or platform-specific, as we choose, and the MDA development process uses both of these forms.

The software development process is now defined: PIM -> PSM -> Code, the transformation should be supported automatically by tools. Model Transformation become a key role in the MDA concept. There are a lot of kinds of model transformation tools from numerous vendors. But all these tools are still suffering from a lack of mature and practical standards for model transformations. Model transformation techniques are mainly divided in three directions: Program transformation and compiler techniques, Meta-programming techniques, Graph rewriting techniques. In order to support the MDA concept in model transformation aspect, on April, 24, 2002, the OMG issued a Request for Proposals (RFP) for MOF 2.0 Query, Views, and Transformations (QVT) to enhance the OMG Meta Object Facility MOF 2.0.

Query: an expression that is evaluated over any MOF 2 compliant model, which results in a selection of the model elements

View: is a projection on a parent model, created by a transformation

(Model) Transformation: is the process of converting one model to another model of the same system.

The motivation of this thesis is to attempt making tools evaluation under the OMG /QVT concepts, an other motivation is, during the evaluating the transformation tools, to find a common and general way to make model to model transformation. five open source and one commercial transformation tool have been selected in this thesis.

In chapter 2 the problems in traditional software development is shortly described.

In chapter 3 the UML is shortly described. In chapter 4 the MDA architecture is shortly described. In chapter 5 the Metamodeling described. In chapter 6 the QVT is shortly described.

In chapter 7 are the criteria for transformation tools described. There are a number of MDA transformation tools on the website(http://www.modelbased.net/). The MDA transformation tools: UMT, MTL, ATL, GMT, BOTL, OptimalJ are chosen, described and evaluated.

In chapter 8 is an overview table for all the evaluated tools in the thesis. In chapter 9 is the conclusion of transformation tools evaluation. In chapter10 is the appendix where the important used URLs are listed, the abbreviation and the index.

Chapter 2

The problem in traditional Software Development

2.1 General Process in Software Development

Software is since the invention of computer more than 50 years, but it is still very hard to say that software is a real industry. The statement that any product which has software inside can not be a real industry product is still valid. Software development in some developers eyes are handwork and art. These has given software development many new ideas and make many progress in software development. But the small workshop concept has prevent software becomes a real industry which means standardization and division of works.

So, either we use our time in the first phases of software development building high-level documentation and diagrams, or we use our time in the maintenance phase finding out what the software is actually doing. Neither way is directly productive in the sense that we are producing code.

There are a few of Problem that we will describe following in software development. [Fran03]

2.2 The Productivity Problem

The software today has been developed mainly in the process with Figure 2-1, the process contains mainly the following phase[Fran03]:

1. Conceptualization and requirements gathering

2. Analysis and functional description

3. Design

4. Coding

5. Testing

6. Deployment

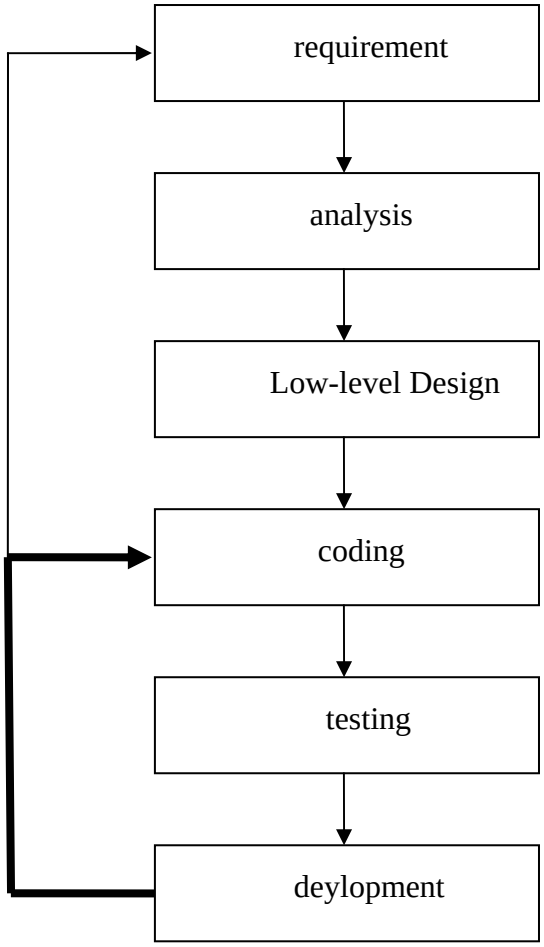

Figure 2-1 Traditional software development life cycle[Fran03]

UML, the unified Model language is mainly used in phase 2 and 3. From the coding phase the UML is in many cases not interesting any more. The Code will mostly reinvented and redesigned again.

2.3 The Portability Problem

Since the software development is relative to traditional industries new, every year will be new technology introduced. New programming language like Java, Linux, XML, HTML, SOAP, UML, J2EE, .NET, JSP, ASP, Flash, Web Services, and so

on. Many companies have problems porting their product from this language to other languages. In extreme situations the old code can not be used any more, the new version of software will be written in the new technology again.

2.4 The Interoperability Problem

In today situation software is normally not isolated. The development of software is sometimes not created from the very beginning. Especially when building large scale software applications and systems, integration of other part of software components are needed. In order to reuse the software components the interface and the behavior should be standardized or at least have the same reaction.

2.5 The Maintenance and Documentation Problem

Documentation has always been a problem in software development. Well documented software need a lot of energy and patience of the developer, depends only on individual.

Maintenance of software product depends on coding techniques, the comments with code as documentation part and good documentation. Well documented software makes a good precondition for the maintenance.

One general problem with documentation is the update. The changes made because of testing and redesign must be followed also in documentation in time. And this required can not be done by many developers.

2.6 The transformation problem

UML, the unify model language, is a specification from OMG. The UML 1.x is a wide spreader model description language in the world. The exchange between different models using UML 1.x has been proved to be very difficult, especially the transformation of different UML tool vendors. The transformation has to be done using one to one specific profile which means a lot of efforts.

Chapter 3

UML

3.1 UML History

The UML history began in 1990s. the object oriented method developed by Grady Booch, Ivar Jacobson and James Rumbaugh was accepted in software programming. The three pioneers as “three amigos” merged to create UML.[Fran03]

In year 1994 Rational Software Corporation successfully built the first draft of the unified Model language. In early 1995 in Object Management Group(OMG) UML became an international standard. In this draft not only notations was used but also the diagram and State charts were introduced.

In January 1997, UML 1.1 response was produced after a lot of software company worked with OMG. UML 1.5 released in 2003 is current official version. Version 2.0 of UML is nearly finished.

3.2 UML Model

The Unified Modelling Language (UML) is a standard language and graphical notation for specifying, visualizing, constructing, and documenting the artifacts of software systems, as well as for business modelling and other non-software systems. [Carl01] The UML represents a collection of best engineering practices that have proven successful in the modelling of large and complex systems. The UML is a very important part of developing object oriented software and the software development process. The UML uses mostly graphical notations to express the design of software projects. Using the UML helps project teams communicate, explore potential designs, and validate the architectural design of the software.

UML defines an abstract language for describing the structure and behaviour of software systems. A standard graphical notation is defined for creating views of the

model elements in this language. Standard diagrams in the UML are followed Diagrams. [Carl01]

Structure Diagrams

Class Diagram

Show the static design view of a system, including packages, classes, interfaces, collaborations, and their relationships.

Object Diagram

Show static snapshots of instances of things found in the class diagrams, especially from the perspective of real or prototypical cases.

Component Diagram

Show the static implementation view of a system, showing a set of components and their relationships. A component represents a physical implementation of the logical abstractions in a model, such as classes and their interactions.

Deployment Diagram

Show the connectivity of physical nodes in an architectural view of the system. A node is a computational resource that provides a physical operating environment for executing one or more components.

Behavior Diagrams

Use Case Diagram

Show the behavior of a system, subsystem, or a class

Activity Diagram

Show the flow of activity within a system, including the sequential or branching flow from activity to activity and the objects that act or are acted upon by those activities.

State Diagram

Show a state machine, consisting of states, transitions, events, and activities. These diagrams are most often used to model the event ordered behavior of an object.

Sequence Diagram

Show the interaction among objects by emphasizing the time-ordering sequence of messages. The objects are typically instances of classes but may represent other classifiers such as actors, components, or nodes.

Collaboration Diagram

Show the interaction among objects but emphasize the structural organization of the objects that send and receive messages.

3.3 Strengths and weaknesses of UML in MDA

3.3.1 Strengths

There is a number of features of UML for MDA very useful. [Fran03]

Abstract Syntax is separated from Concrete Syntax

The model of UML is defined as the meta model of UML. We call meta model as a model of a model. A formal model of UML is focused mostly on abstract syntax. Concrete syntax is the graphical notation and specifying the semantics of shapes and lines that express the abstract syntax. This separation between abstract syntax and concrete syntax is a key requirement of MDA.

Extension Mechanisms

UML has built-in extension mechanisms, called UML Profiles, that allows the creation of specialize. It is also one of key requirement of MDA that the ability to define specialized languages for various aspects of systems. UML profiles can define customized language specifically geared to platform-independent modeling.

Platform-Independent Modeling

UML supports the platform-independent modeling. Platform independent means independence from some specific execution and development domains. For example,

it independents of information-formatting technologies, such as XML DTD and XML Schema. it independents of distributed component middleware, such as J2EE, CORBA, and .NET. It independents of 3GLs and 4GLs, such as Java, C++, C#, and Visual Basic.

Standard Model-language

UML is not proprietary technology. It is other strength for MDA. Because generators use standard representations of UML models as XML documents, Java objects, and CORBA objects to read models ,they can plug into UML environments as well. A profile can plug into a UML tool with the UML profiling mechanism. Standards can also make pluggable modeling framework reality.

3.3.2 Weaknesses

UML has also some of limitations to affect MDA. [Fran03]

Large and weak Partitioned

UML meta model is large and the supported activity models is not well separated . It is difficult the interdependencies to trace. That means, it is hart to use just one part of UML without the interdependencies pulling in other parts by reference. XML DTD that based on XMI reflects this problem.

No direct support for Viewpoints

The meta model provide no direct support for viewpoints, for example, a modeler indicate which viewpoint or viewpoints include a particular model element. It is possible to create different viewpoints of a UML model, but not for filtering in a standardized model.

Limitation of Profile

The extension mechanism of UML is restrictive. It is very limited, that mechanism is defined for the new language constructs by profiles and relationships among such constructs.

More Robust in Components and Patterns

UML can support more abstract component concepts, but results are not satisfactory. UML support patterns to some UML templates and collaborations, but more robust.

Vagueness about Relationships

For example, UML has little semantics of aggregation and whole-part relationships.

Misalignment of UML and MOF

OMG define both standard UML and MOF. They are subtly out of synch with each other. It makes problems for venders to build MDA tools.

No Standard Diagram Interchange

Because UML has no meta model for diagrammatic information. It defines no standard way for tools to exchange diagrams. If a UML model from one tool to another tool export, it can export only the semantic properties in a standard way but not the diagram properties.

No Meta model for Object Constraint Language

UML has no meta model for Object Constraint Language(OCL). OCL has only concrete syntax, but no abstract syntax. It is impossible for tools to exchange abstract syntax for OCL expressions in a standard model.

3.4 UML for building PIM

3.4.1 Semantics of UML

Semantics of UML is comprised of UML Class Diagrams and State Models that could generate data structures and control flow, but there is no standard way to describe the low-level behavior of a given state. Because of this, Action Semantics(AS) for the UML standard is needed. It provides a semantic set of the operations required to specify the behavioral aspects of a UML model to a level of detail such that a self-contained and completely executable application can be generated from that model. Unfortunately, the AS for the *UML* is only a semantic standard, not a syntactic standard. [Mell02]

3.4.2 Executable UML

Executable UML(Mellor and Balcer 2002) is a profile of the UML combined with the dynamic behavior of the execution semantics.

Executable UML can be used as PIM language, because it is computationally complete. An executable UMI model can be directly executed.

The difference between the Programming languages and UMI action language is the different levels of language abstraction. The Programming languages details of the hardware platform without worrying about registers on machine, stack and so on. UML action language details of software platform without worrying about structure, remote procedure calls and so on.

3.5 XMI

XMI is an interchange format for metadata that is defined in terms of the Meta Object Facility (MOF) standard. It defines a approach for generating an XML DTD from a meta model definition and generating XML documents from model.

The XML-based Metadata Interchange (XMI) has two major components[XMI 1.1, OMG Document]:

• ***XML DTD Production Rules*** if XMI is applied to the UML meta model, it produces a DTD for exchanging UML models. XMI defines a set of rules to validate the generated DTDs for producing XML documents. The DTD production rules specify only the syntax of the representing models.

• ***XML Document Production Rules*** if document production rules are applied to a model or model fragment, the result is an XML document. The inverse of these rules can be applied to an XML document to reconstruct the model or model fragment. The rules are applied in both cases in the context of the specific metamodel for the metadata being interchanged. The document production rules specify the semantics of the XML.

Rules for Document Production XML are necessary because the DTD Production Rules specify only the syntax of the XML for the models.

With these both rules for DTDs and documents it is possible not only to generate DTDs from a meta model automatically, but also to generate implementation code.

The problem in this scenario is that it is difficult to develop and exploit best practices across the consulting group without being able to exchange model and design data between different tool sets.

The XML Schema is approved as the successor to the XML DTDs in 2001. Then the OMG made a new XMI specification to define a mapping of MOF to XML Schema.

XMI hat followed these principles:

Automated translation

The transformation from UML model definitions and instances to XML schema and documents is used automatically processor.

Repeatable translation

XMI compliant produce the same XML document schema or instance from UML model or instance.

No multiplicity

During the transformation is all multiplicity information lost in UML model.

Data interchange

XMI specification focus on data exchange rather than human-edited documents.

Low priority on compactness

XMI DTDs and Schema are not decelerated to be written and read by human. The basic tendency of machine-generated XML to be less compact than hand-coded XML is not the main cause of complexity.

Generators take XMI documents as input and produce various artefacts such as code and relation data models. But some of these products require a representation of a UML model as input, such as a Rational Rose UML model file.

The UML meta model does not cover the UML graphical notation. The XMI DTD for UML dose not specify a format for encoding the graphical information contained in UML diagrams. A generator that produces code and other artefacts

from a UML model dose not need the diagram information that contain elements such as box, line, coordinate, and so on.

Chapter 4

The Model Driven Architecture

The Model Driven Architecture(MDA) is a framework for software development defined by the Object Management Group(OMG). Simply said, MDA is an effort of OMG. Its purpose is to use modeling language as a programming language, not only as design language. Executable UML is the end target of MDA. In order to realize the target, OMG has created several standards like UML 2.0, OCL, MOF 2.0 (QVT), XMI and CWM. MDA is not a new technology like Java and CORBA and also not new programming theory. It is a collection of many standards and a framework for developing software systems using model driven development concept. Using MDA means using model as a central point in software development.

MDA provides an approach for, and enables tools to be provided for[MDA Guide Version 1.0.1]:

specifying a system independently of the platform that supports it,

specifying platforms,

choosing a particular platform for the system, and

transforming the system specification into one for a particular platform.

4.1 The MDA Development Life Cycle

The MDA has changed the role of modeling in software development. The MDA development life cycle, which is shown in Figure 4-1, does not look very different from the traditional life cycle. But the model plays an important role in the development cycle. The following three models are at the core of the MDA. [Klep03]

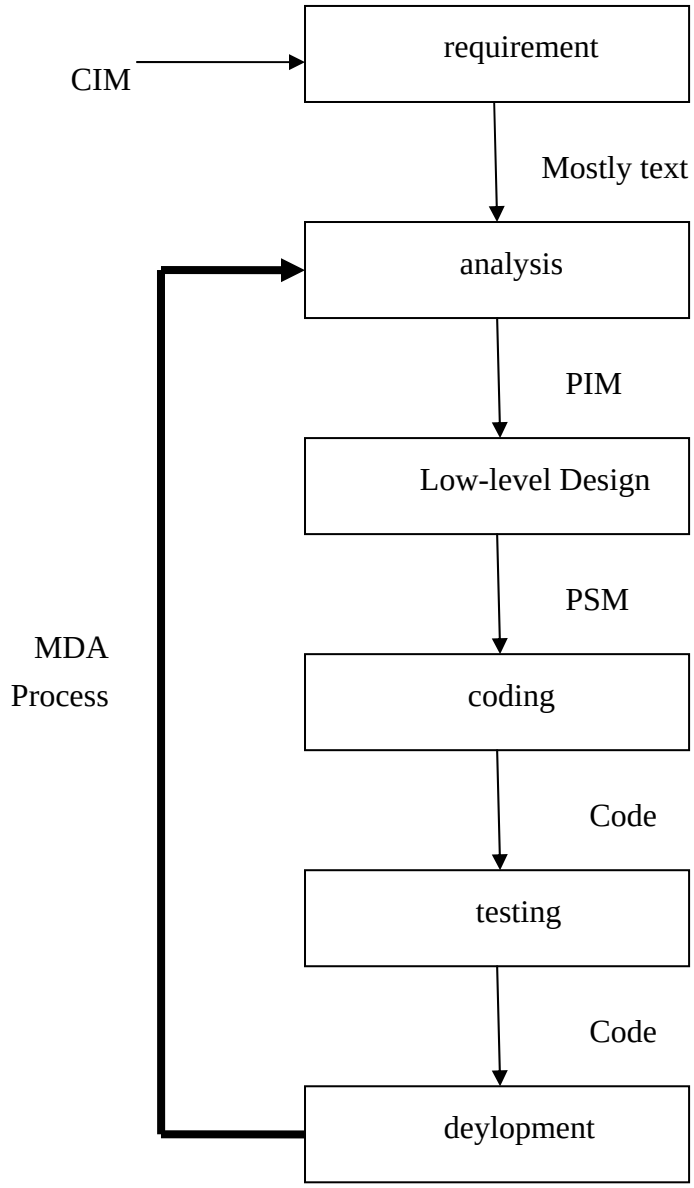

Figure 4-1 The MDA development Life Cycle[Klep03]

The Computation Independent Model (CIM)

This model is independent from computation. CIM is the description of system in view of expert domain which shows no detailed structure of the system. CIM specify only the system domain characterizes. CIM is sometimes also called as domain model.

Platform Independent Model (PIM)

PIM is the first model defined in MDA as the highest level of abstraction that is independent of any implementation technology. PIM specify a system to support the

business logic. In PIM how the model is implemented in which technology is not specified. This independence in the PIM can guarantee the model used with a number of different platforms of similar type to represent abstraction on more platforms.

A common technique for achieving platform independence is using the technology-neutral virtual machine concept. The virtual machine can be defined as a set of services. The services can be defined independently of any specific platform. And the virtual machine can be then realized in platform-specific ways on different platforms.

Platform Specific Model (PSM)

In the next step, the PIM will be transformed into one or more Platform Specific Models(PSMs). The PSM is a model specific for one platform or one technology. One PIM can be transformed to several PSM. PSM use the specific technology to describe the system. The technology can be a framework like EJB or relational database model. This transformation will be done by specific tools which support a full transformation from PIM to PSM automatically.

Code

In the final step, the PSM will be transformed to code. A direct transform from a PIM directly to deployable code, without producing a PSM is also possible. In such case the transformation tool might also produce a PSM, for use in understanding or debugging that code. Also the transformation from PSM to code will be supported by tools which enable the transform to be automatically.

4.2 Automation of the Transformation Steps

The main revolutionary part of MDA is the transformation will be all supported by tools which will enable the transformation automatically. Figure 4-2 shows MDA transformations, which are executed by tools. In traditional software development the transformation from PSM to code are already realized. What is new in MDA is that the transformation from PIM to PSM is automated as well. This approach is totally new. This does allow a developer to have immediate feedback on the PIM that is under development, because a basic prototype of the resulting system can be generated on the fly. The step from PIM to PSM is also the most difficult part in MDA. PIM requires the system designer to have enough knowledge of the whole

system and foundation knowledge of end produced code: for debugging etc. For the transformation to PSM, PIM must also give enough information to allow the PSM to be established. [Klep03]

Figure 4-2 Three major steps in the MDA development process

4.3 Productivity

The gain in productivity is obviously. The developers of software will only focus on the business logic problems. The developing work is clearly divided. One group of people is concentrating on model establishing and the other can only concentrating on transformation. The coding work is not the main activities any more for the development. The code are produced by tools. The tools can be bought then from other software vendors who are specified on the transformation.

4.4 Portability

The portability in MDA is archived by PIM. Every model using PIM can be automatically transformed into multiple PSMs for different platforms and technologies. For new technologies and platforms that will appear later, the vendor of the new technology or platforms will have to come together with companies for transformation tools to deliver the corresponding transformation for PIM. [Klep03]

4.5 Interoperability

In MDA these are definitions for bridges, which means the relation between PSM and Code, like the Figure 4-3 shows here. The bridges can be also supported by automation tools. These bridges definition is useful for some of the existing transformation programs which are now available. These old transformation programs can be used just like before, but they play an the role as the bridges in the MDA. Such a bridge can be a Java to relational database or C++ to Java etc. Bridges

localize the interfaces, which can be changed and subsequently propagated through the code. [Klep03]

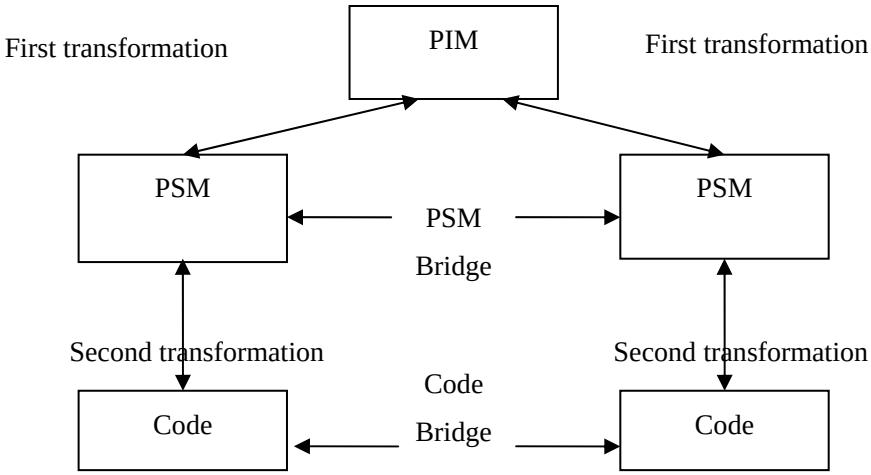

Figure 4-3 MDA Interoperability using bridges[Klep03]

4.6 Maintenance and Documentation

According to the MDA definition, the model in PIM is an exact representation of the code. PIM is now at a higher level of abstraction for the code. Thus, the PIM is already the high-level documentation that is needed for any software system. The PIM is "code" and documentation at the same time. In MDA the model description, PIM, is not thrown away after designing is finished, if there is something need to be changed, the changes only need be made in PIM and the traditional code will be generated automatically. This means at the time of changing in the new form of "code"(PIM), the documentation is updated at the same time.

Of course other documentation like CIM at a high level of abstraction will naturally be used. Any additional information, which cannot be captured in a PIM, will remain in this level of documents.

4.7 MDA Building Blocks

As already said, MDA is a framework of many OMG standards to support its functionalities. The building blocks of the MDA framework and the OMG standards which will support the blocks a listed in the following: [Klep03]

High-level models language, will be supported by OMG UML 2.0.

High-level models, the basic features which such a model for model will be supported by OMG MOF.

Definitions of how a PIM is transformed to a PSM.

A language must be a formal language.

Tools that implement the execution of the transformation definitions. Preferably these tools offer the users the flexibility to tune the transformation step to their specific needs.

Tools that implement the execution of the transformation of a PSM to code.

4.8 The application of MDA

The applying of MDA in software development depends on mainly two preconditions, first the software product is under very rapid changes of new technologies. Actually the changes have been made in every region, it can be the middle ware or the database techniques or even the programming language. the second condition is the business will remain in a relative long time. This will granites the investment for the modeling will remain also for a long time.

The advantage of using MDA are:

The investment for the modeling will be protected for a long time. The model using MDA will remain relative long independent of any changes made in new implementation technologies.

Flexible for new implementation technologies. The PIM can be ported to different PSM on different technologies and platforms.

Flexible also for new business changes, in MDA any changes made in PIM will be easily transformed to PSM and code which lead more quickly to a product.

The disadvantages of using MDA are:

The challenge for the developers, they will mainly working with modeling and must also have a lot of knowledge of coding at the same time. The late one will be needed for debugging and tuning activities.

The investment for modeling will be surely large than for a traditionally software process.

The potential risk of immature of the OMG standards and the tools which support the OMG standards.

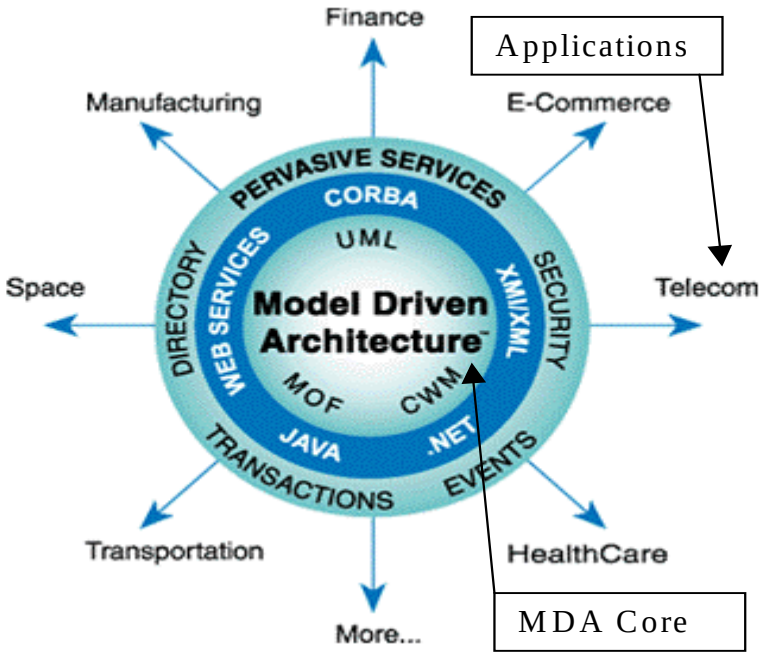

Figure 4-4 The Application of MDA[OMG MDA]

The applying of MDA has been taken place in many fields, mainly in the following as the Figure 4-4 shown by OMG: Bank, Insurance, Financial management, Embedded Systems, Telecommunication.

4.9 Development Process

The process model is not defined by OMG for using MDA. The users of MDA are free of choice in process model. In traditional process the design and model writing usually are used only as documentation, in a process using MDA, the process should surely emphasize the model development and maintenance. [Klep03]

Chapter 5

Metamodeling

The motivation of meta modeling is caused by MDA and also from the many form of the modeling language. Since Object Oriented programming has been introduced in software development, modeling has become a important role. And due to the rapid changes and progresses in many business requirements, many modeling notations and tools corresponding have been created in order to catch up the changes, each of the modeling tools has some benefits over the other. The problem here is most of these modeling notations are not exchangeable. Specially build transformation tools have to be developed. So the basic idea of meta modeling is to make a constraints and provide some basic features which all modeling notation should have. According to this the exchange tools can be easily built.

5.1 Meta modeling in general

A model is a representation of the structure and/or behavior of an application or system. A representation is based on a language that has a well defined form (”syntax”), meaning (”semantics”), and possibly rules of analysis, inference, or proof for its constructs. The syntax may be graphical or textual. For building a concrete model some infrastructure are needed, for example how should the semantics looks like etc. Meta model is needed to make a language for model. Meta modeling is the process of designing languages through meta and meta-meta notations. They help to ensure syntactically correct specifications as well as in the construction of customizable modeling. The idea behind meta modeling is to provide Tool and Data interchange between many different tools.

5.2 MOF and Modeling Layers of the OMG

The state of art meta modeling is using a fix meta model for a specific area. With the rapid changes the need for freely created of meta model is growing up with the rapid

changes in business logic. In answer of this demand on modeling the OMG has introduced the concept of MOF in framework of MDA.

The Meta-Object Facility (MOF) is the OMG's adopted technology for defining metadata and representing it as CORBA objects. The MOF 1.3 specification was finalized in September 1999 (OMG document ad/99-09-05). A MOF metamodel defines the abstract syntax of the metadata in the MOF representation of a model. The MOF model itself describes the abstract syntax for representing. MOF metamodels can be represented using a subset of UML syntax. The MOF Model is made of two main packages (in MOF 2.0) Essential MOF (EMOF) and Complete MOF(CMOF).

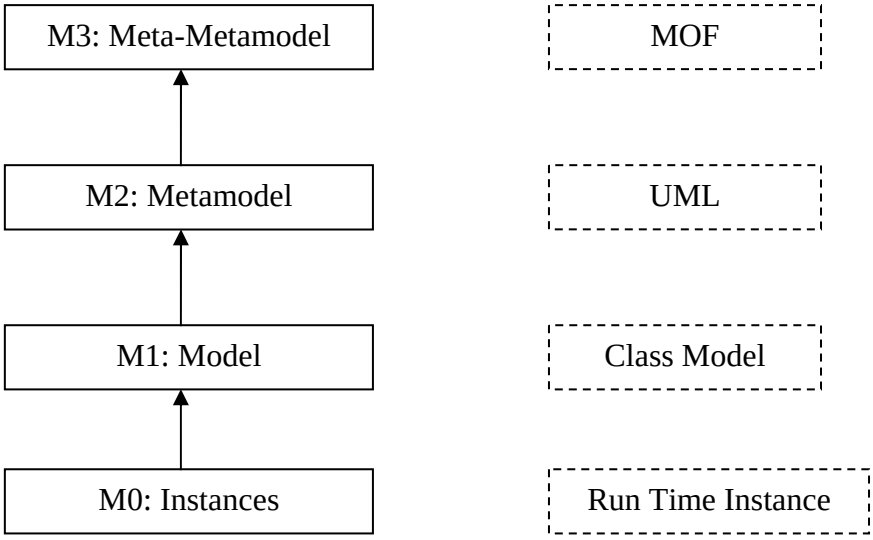

Figure 5-1 The 4 Layer architecture of MOF in OMG

The standards defined in MOF by the OMG use four layers are called M0, M1, M2 and M3. The Figure 5-1 shows the 4 layer architecture of meta modeling in MOF. [Klep03]

5.2.1 Layer M0: The Instances

The M0 layer contains the run-time instances of the user. The instances have data which is needed. The instances can be in different form, for example as database.

5.2.2 Layer M1: The Model of the System

The M1 layer contains models, for example, a UML model of a software system. Each element at M0 layer is always an instance of an element at the M1 layer. M1 elements directly specify what instances in the M0 world look like. The M1 Level is the traditional understanding of Model.

5.2.3 Layer M2: The Model of the Model

The elements that exist at the M1 layer (class, attributes, and other model elements) are themselves instances of classes at M2, the next higher layer. An element at the M2 layer specifies the elements at the M1 layer. The same relationship that is present between elements of layers M0 and M1 exists between elements of M1 and M2. Every element at M1 is an instance of an M2 element, and every element at M2 categorizes M1 elements.

The model that resides at the M2 layer is called a meta model. UML and CWM are examples of such languages.

5.2.4 Layer M3: The Model of M2

Every element at M2 is an instance of M3 element, and every element at M3 categorizes M2 elements. Layer M3 defines the concepts needed to reason about concepts from layer M2.

Within the OMG, the MOF is the standard M3 language. All modeling languages (like UML, CWM, and so on) are instances of the MOF.

Chapter 6

Query, View, Transformation(QVT)

Model-to-model transformation is a key technology for OMG's MDA concept. The need for standardization in this area led to the MOF 2.0 Query, Views, Transformations Request for Proposals (RFP) from OMG. The RFP is up to this thesis in construction. This standardization should support the Meta modeling in two folds, one for querying the Meta model and the second for the exchange between the Meta models.

6.1 UML Model transformation

UML(Model)transformation has been an important issue in software design and development. For problem of transforming the UML to code or from different Meta models, the usually way for the designer and programmer would be using profilers. For example the profiles UML for CORBA or UML for C++ and can then use these dialects of UML to prepare for the transformation between a UML design model and IDL or C++ code, with the help of some limited facilities provided by the UML CASE tool vendors. The transformation engine itself may be built on any technology like the XSLT tools and of course using any other traditional programming language C++ and Java. A second way of model transformation is to use graph-based transformations, this method is not in the scope of this thesis.

The question of model transformation also lies at the center of the MDA approach. In fact Model transformation is an important MDA success factor.

- need meta model language and framework that enables transformation between different abstraction levels
- must be able to maintain traceability
- need good tools with high usability

6.2 MOF 2.0 QVT

Automated transformations play a key role in MDA. It is important that transformations can be developed as efficiently as possible. A standard syntax and execution semantics for transformation can enable the transformation tools very much. On April, 24, 2002, the OMG issued a Request for Proposals (RFP) for MOF 2.0 Query, Views, and Transformations (QVT) to address a technology part of the OMG Meta Object Facility MOF 2.0 in order to have a way to manipulate the MOF models: [OMG MOF 2.0 QVT]

1. Queries on MOF 2.0 models,

2. Views on MOF 2.0 meta models,

3. Transformations of 2.0 MOF models.

The followings are the definitions of the terms query, view, and transformation used in this RFP: [OMG MOF 2.0 QVT]

Query: A query is an expression that is evaluated over a model. The result of a query

is one or more instances of types defined in the source model, or defined by the query

language. An example of a query over a UML model might be: Return all packages that do not contain any child packages.

View: A view is a model that is completely derived from another model (the base model). A view cannot be modified separately from the model from which it is derived. Changes to the base model cause corresponding changes to the view. If changes are permitted to the view then they modify the source model directly. The meta model of the view is typically not the same as the meta model of the source.

Views are typically not persisted independently of their source models (except perhaps for caching). Views are often read only. Where views are editable a change made via the view results in the corresponding change in the base model. It is therefore necessary for an editable view to have a defined reverse mapping back to the base model. A view may be partial, that is based on a subset of the source model. A view may be complete and have the same information content as the source, but

reorganized for a particular task or user. A query is a restricted kind of view. Views are generated via transformations.

Transformation: A transformation generates a target model from a source model. Transformations may lead to independent or dependent models. Transformations may be *one-way* (unidirectional), Transformations may be *two-way* (bi-directional), in which case each model may be modified after the application of the transformation; changes must be propagated in either direction.

The common transformation scenarios can be summarized in the following:

Simple transformations: a simple transformation is one that transforms single elements in the source model into single elements in the target model.

Expressions: Transformations may have to handle string expressions in the source or

target model. Where a meta model for the expression language exists, expressions can be treated as instances of that meta model and require little special support. However, in many cases it may be unnecessary to fully parse the expression, and some simple support for transforming text is sufficient.

Naming: a common situation requiring text handling occurs when the source and target models have different restrictions on naming, or different naming conventions.

For example, UML allows many characters in names that Java does not allow. This must be handled in some way when generating Java from UML.

Complex transformations: This type of transformation builds structures in the target model which do not directly correspond to any individual element in the source model. The transformations may be based on complex algorithms and heuristics.

Regeneration and reconciliation: Having used a transformation to generate a target model, it is likely that the user will want to modify the generated output. When subsequently the source model is also changed, it would be desirable if the transformation can be reapplied while maintaining any changes the user has made. Trying to maintain user changes may lead to conflicts, which must be resolved—perhaps by asking the user what should be done.

Transformation from partial source models: it is often useful to be able to generate a partial target model from a partial source model.

Resilience to errors: The occurrence of an exception during transformation execution should not halt the transformation, i.e., instead of simply aborting, it should be possible to generate a partial model. Rules that are not affected by the error in the source model should be executed as usual, resulting in a partial target model. This approach allows multiple errors to be detected in a single pass. The requirement of transactional behavior of transformation rules is directly related to this feature.

M-to-N transformations: it cannot be assumed that there is a one-to-one correspondence between source and target models. Support for one-to-many transformation is particularly valuable in cases where the resulting models will be interdependent and must refer to elements created during the transformation. Transformations combining models that represent various aspects of a problem are likely to be many-to-one. In the general case, many-to-many transformations must be supported.

Chapter 7

Evaluation of UML transformation Tools

Today many UML tools are available on the market, sorting through them all requires a lot of time and effort. Each tool has its strengths and weaknesses, so how does one compare them and find the best fit? In this thesis will try to make a structured Comparison of some UML Tool on the Market. The evaluation Criteria for the Tools are mostly taken from the requirements in OMG MOF 2.0 Query, View, Transformation and from the criteria used in [Gardner 03] in order to assess the power of each tool.

7.1 MDA transformation Tools

From the MDA aspect transformation tools which support transformation of models can be categorized in the followings: [Klep03]

PIM to PSM Transformation Tools

A high level architecture PIM are transformed by this type of tool into one or more PSMs. Some tools offer only minimal functionality in this area.

PSM to code Transformation Tools

PSM to code transformation tools are black-box transformation. They have one predefined type of model as source and produce predefined type of model or code as target and transformation definition. The source model is a PSM and the target model is code model. Transformation definition take predefined the type of model(PSM) as source into another predefined type(code) as target.

PIM to code Transformation Tools

Both the PIM to PSM and the PSM to code transformation are supported by this type of tools. Source model produce target model as a black box. PIM will transform direct to code.

Tunable Transformation Tools

This type of tools allow for some tuning or parameterization of a transformation.

Access to the transformation definition to adjust it to your own requirements is usually not available. The best one can get today is a transformation definition written in an internal tool-specific scripting language.

Most tools only work for a predefined PIM language, which is often required to be a restricted variant of UML. Although UML diagrams are used to model a PIM, internally the tools do not use the UML language definition, but their own tool-specific definition of UML.

Transformation Definition Tools

With transformation definition Tools we can create and modify the transformation definition. This type of tool support more flexible and allow own language definition to be plugged in and to be used in a transformation.

7.2 Evaluation Criteria of transformation Tools

The following criteria will used in the thesis to describe and compare the system characteristics of the transformation tools: [OMG MOF 2.0 QVT]

Self containing: whether the documentation is complete enough to understand how the transformation is done.

Scalability: whether a transformation can be configured to control the transformation result to be small or large.

Simplicity: whether a transformation definition is easy to understand and write depends also on personal preferences. Assuming that transformations will be written by programmers, any programming-like solution using Action Semantics could be assumed to be (fairly) simple.

Bi-directional mappings: whether a transformation can be done only in straight forward direction or be done in both direction.

Ease of Adoption: whether a transformation can be adopted more or less equally hard to any of the other transformation tools.

Rich Conditions: whether a Query language to formulate conditions in transformations is used.

Provide an abstract syntax for the transformation language: The transformation

language should have a defined abstract syntax for composition, declarative, and imperative parts. Whether it is based on an existing OMG standards. Example Declarative representation in MOF.

Adopt common terminology: whether in the documentation and GUI the common terminology's are used and understandable.

Support composition and reuse: It is often valuable to be able to construct a complex transformation from multiple intermediate transformations.

Support complex transformation scenarios: whether some complex transformation like M-N transformation can be made. Example Multiple models transformed into each other or source and target models are same model.

Provide complete examples: whether with the tool package complete examples are provided. Example use additional transformation data.

Robust on transformation executions: whether the transformation executions can be continue in case of some errors occur during the execution.

Tooling aspect: whether the tool has a strong tooling support. Example Simple source-to- target direct transformation. Example traceability of transformation or reuse and extension of generic transformations.

In below paragraph are the following tools shortly described and evaluated: UMT, MTL, ATL, GMT, BOTL, OptmialJ. UMT(UML Model Transformation Tool) is a free UML/XMI-based tool for model transformation and code generation tool. MTL(Model Transformation Language) constitutes an Implementation of Model Transformation Engine. But the compiler is only Java compiler. ATL(ATLAS Transformation language) is an QVT-based transformation language. ATL provide open source under Eclipse GMT project. It is developed by the INRIA Atlas team as a set of Eclipse plugins. GMT(Generative Model Transformer) is an Eclipse project that is provide model transformation technology for the eclipse platform. FUUT-je

tool is a code generator tool. BOTL(Bi-directional Object oriented Transformation Language) is a graphical language. It allows to specify transformations of object oriented models.. OptmialJ is a commercial product from Compuware. It supports model-to-model and model-to-code transformation and uses a notation of patterns to achieve PSM transformations.

7.3 Tool UMT(UML Model Transformation Tool)

7.3.1 Vendor and Version

UMT-QVT (UML Model Transformation Tool) is a prototype developed within the scope of the DAIM, CAFÉ, COMBINE, and ACEGIS projects. The version v0.81 from *June 2004* was used for evaluation in this thesis.

UMT-QVT requires Java run-time environment and Java 2 Platform. UMT has been tested with jdk 1.3 and jdk1.4. In this Thesis jdk 1.4 was used.

UMT uses Apache XML libraries, xerces and xalan. These are bundled with the UMT distribution.

7.3.2 Installation

UMT is download under address http://umt-qvt.sourceforge.net/. After downloading you need to do the following installation steps:

Installed Java JSDK 1.4 and Make sure that Java runtime in your execution path.

Unzip the UMT zip file to a directory, which you choice, e.g. *c:*\. It will create a subdirectory *umt*, where the UMT application is placed.

Find the file **run.bat** in the *install directory* of UMT, which was the directory you chose (e.g. *c:\umt*)

Execute the batch file **run.bat**.

You have now started UMT.

7.3.3 Introduction

The following contents are taken from website "http://umt-qvt.sourceforge.net/ ".

UMT tool support model transformation and direct code generation. The tool has its own UML models in the form of XMI. It provides also an environment in which new generators can be plugged in. The tool environment is implemented in Java. Two Generators are implemented in XSLT and Java. UMT is bundled with some generators also, which may be used or modified at the user's own wishes. The formats GML, WSDL, XML Schema, SQL, Java interfaces, IDL, EJB w. Xdoclet tags, PBEL are supported.

XMI models are the intermediate format in UMT, so the models are in an XML format. It is the basis for validation and generation towards different target platforms. Intermediate format in UMT is called XMI Light, which has the PIM position in UMT. UMT tool-chain is shown in Figure 7-1.

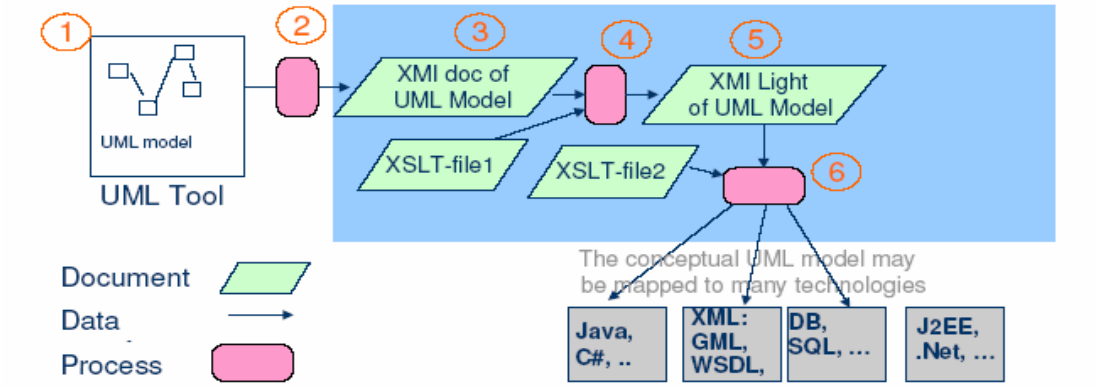

Figure 7-1 UMT tool-chain

7.3.3.1 UMT transformation process

The transformation process of UMT supports the general MDA concepts with a modelling tool, the transformation process is described in below:

(1) The source model for using UMT can be an exporting XMI produced from UML model in some UML tool. XMI provide tool-independence with respect to the UML modelling tool. Representation of XMI in the model is imported by UMT.

(2) XMI will be transformed to a internal format by UMT, called XMI Light. XSLT is used to transform XMI to XMI Light. This transformation is executed in order to simplify later transformations.

(3) UMT use XMI Light as the internal model representation. With this format transforms UMT to the different target code or format, like Java, C#, XML-based formats like WDSL, GML, database schemas, application server technologies, like J2EE, .Net.

(4) In UMT is normally transformer XSLT. It may be a Java-implementation. XMI Light is transformed to the target code, for instance J2EE, WSDL, GML, etc.

7.3.3.2 source model

XMI is used as source model which is UML integration format in UMT. Currently OMG supported XMI version 1.2 for modeling tools based on OMG-UML and metadata repositories (OMG-MOF) in distributed environments. Another version is XMI version 2.0 for XML schema.

7.3.3.3 intermediate model

UMT use XMI Light as the internal model. XMI Light is as a source meta-model to generate target code by the UMT transformer. XMI Light format is used by the tool for browsing and editing.

7.3.3.4 Transformer

Two transformer are available in UMT, one is implemented in XLST and the other one is in Java.

XSLT transformer

XSLT Style sheet is as XSLT transformer of UMT, which support two kinds of XSLT transformers; single-file and multi-file.

Single-file XSLT transformers provide one single output file from the source model, e.g. a WSDL or SQL file. The single-file transformation process is produced just by an XSLT file and the result is the single file.

Multi-file XSLT transformers provide several output files from the source model, e.g. a set of java files. The multi-file transformation process is produced by an XSLT file and a UMT Java Transformer class.

Output files defined in an XML structure of packages and files by XSLT transformer. They produce physical files.

Java transformer

A UMT Java emitter a Java-class that implements the UMT Transformer interface

"transformer.TransformationEngine", which is shown below.

```
public interface TransformerEngine
{
        public void setInputSource (java.io.Reader input);
        public void setTransformationImpl (Object impl);
        public void setOutputDir (String dir);
        public void doTransformation ();
        public void addTransformationResultListener
        (TransformationResultListener listener);
}
```

The simplest way of implementing this interface is to subtype the class *"transformer.AbstractTransformer"*. The class implements all methods but the *"doTransformation"* method should be implemented by the user. The default input that is sent in a *"setInputSource"* call is an instance of a model on XMI Light form. The transformation itself must be implemented in the *"doTransformation"* method. An example is given in the "*DefaultJavaTransformer*" class in the documentation of UMT.

Both types of XSLT and Java transformers need the same XMI Light input to generate XML result.

In order to add a new transformer (install it) in UMT, the UMT transformation configuration tool can be used.

UMT transformation configuration tool can be used to add a new transformer, through the "*Settings/Transformations*" menu. Transformations can be added dynamically.

7.3.4 Test and Evaluation

Category: UMT is PIM to PSM Transformation Tool, PSM to code Transformation Tool, Tunable Transformation Tool and Transformation Definition Tool.

The following criteria are checked in order to have a rough imagination of the system characteristics of the transformation tools UMT:

Self containing: there is only one document available. In the document there is a complete description of GUI of UMT, but only a little description about the intermediate meta model XMI light used in UMT.

Scalability: in the Java transformer there is one possibility to support multiple class file output.

Simplicity: the transformation is simple in UMT, it support the concept MDA light which the PSM level is spared. The input model is directly transformed into target code. UML Activity graphs can be read by UMT in the form of XMI for Activity Graphs. XMI 1.1 for UML1.3/UMT1.4 is currently supported.

Bi-directional mappings: in UMT the transformation can be done in both direction. Currently, two main different types of transformers can be plugged in: Java transformers and XSLT-transformers.

Ease of Adoption: the XSLT and Java code can be separately adopted by other tools, The Java part uses the general XML parser to decode the input source.

Rich Conditions: Query and View for the intermediate model, the XMI. Light can be supported through Xpath and the transformation can be supported by XSLT.

Provide an abstract syntax for the transformation language: The syntax of XMI Light is available as XMI format in the documentation.

Adopt common terminology: the GUI windows and Menu items are written in English and understandable.

Support composition and reuse: not supported jet in UMT.

Support complex transformation scenarios: not supported jet in UMT.

Provide complete examples: A simple transformation is illustrated in the document but for complex transformation there is no example available in UMT documentation.

Robust on transformation executions: not supported jet in UMT.

Tooling aspect: not supported jet in UMT.

7.3.5 Transformation Examples

Input format: XMI, XMI Light, XMI or XMI Light as Project
Output format: GML ,WSDL ,XML Schema, SQL, Java Interfaces, IDL, PBEL, ACEGIS Workflow, J2EE Xdoclet, J2EE servlet Xdoclet, Example 'Java based', EMF Ecore, BPEL(Business Proc.Exec Lang)

One Example is shown here using UMT to transform a XMI input to Java code.

Input:

The example below shows how the XML-based representation of the XMI Light looks like. The details of the format are specified in an XML schema file (XMI Light.xsd).

```
<model xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:noNamespaceSchemaLocation="XMI Light.xsd">
<package id="rootPack_ticket" name="ticket">
<class abstract="false" id="class_TicketService" name="TicketService"
stereoType="service">
<implements id="class_ITicketService" />
<implements id="class_IAuthorisaton" />
<attribute cardinality="1..1" name="owner" type="dt_string" />
<attribute cardinality="1..1" name="version" type="dt_string" />
<association aggregationType="none" cardinality="0..*" collectionType="set"
name="reservations" targetClass="class_Reservation" />
<operation name="buyTicket" returnType="class_Ticket" />
<operation name="pay">
<parameter name="amount" type="dt_integer" />
</operation>
<taggedValue tag="persistence" value="transient" />
</class>
```

```
<class abstract="false" id="class_Ticket" name="Ticket"
stereoType="entity">
<attribute cardinality="1..1" name="price" type="dt_integer" />
<attribute cardinality="1..1" name="issueDate" type="dt_date" />
<attribute cardinality="1..1" name="ticket_id" type="dt_string" />
<taggedValue tag="persistence" value="transient" />
</class>
<class abstract="false" id="class_Reservation" name="Reservation"
stereoType="entity">
<attribute cardinality="1..1" name="reservation_id" type="dt_string" />
<association aggregationType="none" cardinality="1..*" collectionType="set"
name="reservation_for" targetClass="class_Ticket" />
<taggedValue tag="persistence" value="transient" />
</class>
<class abstract="false" id="class_ITicketService" name="ITicketService"
stereoType="interface">
<operation name="payReservation" returnType="class_Reservation">
<parameter name="ticket_id" type="dt_string" />
</operation>
<operation name="getTickets" returnType="dt_list_Ticket_" />
<operation name="reserveTicket">
<parameter name="ticket_id" type="dt_string" />
</operation>
<taggedValue tag="persistence" value="transient" />
</class>
<class abstract="false" id="class_IAuthorisaton" name="IAuthorisaton"
stereoType="interface">
<operation name="login" returnType="dt_string">
<parameter name="username" type="dt_string" />
<parameter name="password" type="dt_string" />
</operation>
<taggedValue tag="persistence" value="transient" />
</class>
<datatype id="dt_string" name="string" />
<datatype id="dt_integer" name="integer" />
<datatype id="dt_date" name="date" />
<datatype id="dt_list_Ticket_" name="list(Ticket)" />
<package id="pack_TicketFactory" name="TicketFactory">
```

```
<class abstract="false" id="class_TicketFactory" name="TicketFactory">
<attribute cardinality="1..1" name="session_id" type="dt_string" />
<operation name="createTicket" returnType="class_Ticket" />
<taggedValue tag="persistence" value="transient" />
</class>
</package>
<package id="pack_TicketShop" name="TicketShop">
<uses target="pack_TicketFactory" />
<class abstract="false" id="class_TShop" name="TShop">
<attribute cardinality="1..1" name="shop" type="dt_string" />
<taggedValue tag="persistence" value="transient" />
</class>
</package>
……
</package>
</model>
```

in XMI file class is equivalent to a UML class, operation is equivalent to a UML operation, attribute is equivalent to a UML attribute, and association is equivalent to a UML association.

Output:

The output file(ticket.idl) shown here is in IDL format:

```
module ticket {
    interface ITicketService {
            public reservation payReservation(string ticket_id);
        public list(ticket) getTickets();
        public void reserveTicket(string ticket_id);
   };
    interface IAuthorisaton {
            public string login(string username, string password);
   };
};
```

7.3.6 Summary

Strength of the UMT:

The UMT was implemented rather before MOF, QVT. The implementation reflects the concept of MDA light, in which no PSM model is used. This makes the implementation simple. And also makes the use of the tool easy. Documentation is well for the GUI user part.

Weakness of the UMT:

Documents are not available for intermediate XMI light model. Query/View can only be supported through theoretically by XPath, no examples are provided.

7.4 Tool MTL(Model Transformation engine)

7.4.1 Vendor and Version

MTL(Model Transformation engine) tool is developed by Triskell team, in Inria (INSTITUT NATIONAL DE RECHERCHE EN INFORMATIQUE ET EN AUTOMATIQUE).

All works in MT Engine are released under an open-source LGPL license.

The version of BasicMTL used in the thesis is the version beta v.04 from 2004.4.20.

7.4.2 Installation

MTL is downloaded under http://modelware.inria.fr/rubrique5.html. After downloading you need to do the following Steps:

Install Java 1.4.1 SDK (suggestion c:\j2sdk1.4.1)

Install Eclipse (suggestion c:\MTL\eclipse)

Extract the BasicMTL distribution in a local directory (suggestion: c:\MTL). It will create a subdirectory *mtl*, where the MTL application is placed.

Install the recommended eclipse plugins in your eclipse plugin directory

Configure ant in eclipse: in Preferences/Ant/Runtime/Classpath add/lib/tools.jar

Start the program in C:\mtl\eclipse\eclipse.exe

You have now started MTL plugins using eclipse Update Manager..

7.4.3 Introduction

The following contents are taken from website "http://modelware.inria.fr/rubrique8.html".

MTL engine is aimed to help to resolve the QVT language choice. The Triskell team propose an architecture using a pivot metamodel. The meta model language for supporting the pivot metamodel is called either MTL or BasicMTL in MT engine.

The concept of MT engine is to transfer the source model into the meta model and then to the code directly. This concept is commonly used in many places, such as virtual machine in Java. MT engine is a compiler that currently compile to Java. The user of BasicMTL is encouraged to use any other languages to implement the model transformation language BasicMTL.

MT engine main features :

- The implementation is using OO design (based on MOF concepts)
- multiple inheritance support
- allows to metamodelize a metamodel directly in BasicMTL
- user can use metamodel either from a repository or from BasicMTL in a transparent way.

7.4.3.1 Transformation process

Source model can be an XMI. There is no intermediate model.

7.4.3.2 Source model

Loading and saving model are repository dependent. Source model can be an XMI using MDR.

Two techniques are provided by MTL:

- The model as parameters from a repository specific library to a repository independent library.
- A repository independent library adding repository dependent actions.

7.4.3.3 Intermediate model

There is no intermediate model.

7.4.3.4 Transformer

Java is as transformer.

7.4.4 Test and Evaluation

Category: MTL is PIM to code Transformation Tool, Tunable Transformation Tool, Transformation Definition Tool.

The following criteria are checked in order to have a rough imagination of the system characteristics of the transformation tools MT engine:

Self containing, there is only one document available. In the document is the concept of pivot metamodel introduced. The syntax of BasicMTL and how to implement the compiler for it are also described.

Scalability, the user can build the compiler of the BasicMTL by himself. The user is totally free for the dimension of the transformation.

Simplicity, although the transformation implementation is documented and the MT engine is also showing the example of how to do it. But the user will have to study the syntax of BasicMTL and the implementation in order to realize a transformation.

Bi-directional mappings, only theoretically possible, no example is shown.

Ease of Adoption, the syntax of BasicMTL is well defined, can be used as transformation language in other project.

Rich Conditions, Query and View for the BasicMTL is spported.

Provide an abstract syntax for the transformation language, The syntax of BasicMTL is described in the documentation.

Adopt common terminology, document is written in English and understandable.

Support composition and reuse, not supported jet in MT engine.

Support complex transformation scenarios, not supported jet in MT engine, association is planned for the next step.

Provide complete examples, the example of implementation of BasicMTL compiler is available in the documentation.

Robust on transformation executions, not supported jet in MT engine.

Tooling aspect, not supported jet in MT engine.

7.4.5 Transformation Example

Input format: MTL Project, Java Project, Plug-in Project, Eclipse Modeling Framework, XMI
Output format: Application Program, XMI

One Example is shown here using MT engine to transform a XMI input to Java code.

Input:

```
<?xml version="1.0" encoding="UTF-8" ?>
-<XMI xmi.version="1.2" xmlns:UML="org.omg.xmi.namespace.UML"
timestamp="Mon Sep 08 00:03:33 CEST 2003">
-<XMI.header>
-<XMI.documentation>
  <XMI.exporter>Netbeans XMI Writer</XMI.exporter>
  <XMI.exporterVersion>1.0</XMI.exporterVersion>
  </XMI.documentation>
  </XMI.header>
-<XMI.content>
-<UML:Model xmi.id="a1" name="MercurePL" isSpecification="false"
isRoot="false" isLeaf="false" isAbstract="false">
-<UML:Namespace.ownedElement>
  <UML:Class xmi.id="a4" name="Mercure" visibility="public"
isSpecification="false" isRoot="false" isLeaf="false" isAbstract="false"
isActive="false" />
  <UML:Class xmi.id="a7" name="Engine" visibility="public"
isSpecification="false" isRoot="false" isLeaf="false" isAbstract="true"
isActive="false" />
-<UML:Association xmi.id="a9" isSpecification="false" isRoot="false"
isLeaf="false" isAbstract="false">
-<UML:Association.connection>
- <UML:AssociationEnd xmi.id="a11" visibility="public"
isSpecification="false" isNavigable="true" ordering="unordered"
aggregation="composite" targetScope="instance" changeability="changeable">
- <UML:AssociationEnd.multiplicity>
-<UML:Multiplicity xmi.id="a12">
-<UML:Multiplicity.range>
  <UML:MultiplicityRange xmi.id="a13" lower="1" upper="1" />
  </UML:Multiplicity.range>
```

```
  </UML:Multiplicity>
  </UML:AssociationEnd.multiplicity>
-<UML:AssociationEnd.participant>
  <UML:Class xmi.idref="a4" />
  </UML:AssociationEnd.participant>
  </UML:AssociationEnd>
……….
```

Output :

```
<?xml version = '1.0' encoding = 'ISO-8859-1' ?>
<XMI xmi.version = '1.2' xmlns:UML = 'org.omg.xmi.namespace.UML' timestamp
= 'Mon Jun 13 10:14:38 CEST 2005'>
  <XMI.header>
    <XMI.documentation>
      <XMI.exporter>Netbeans XMI Writer</XMI.exporter>
      <XMI.exporterVersion>1.0</XMI.exporterVersion>
    </XMI.documentation>
  </XMI.header>
  <XMI.content>
    <UML:Model xmi.id = 'a1' name = 'MercurePL' isSpecification = 'false'
isRoot = 'false'
      isLeaf = 'false' isAbstract = 'false'>
      <UML:Namespace.ownedElement>
        <UML:Class xmi.id = 'a2' name = 'Mercure' visibility = 'public'
isSpecification = 'false'
          isRoot = 'false' isLeaf = 'false' isAbstract = 'false' isActive =
'false'/>
        <UML:Class xmi.id = 'a3' name = 'Engine' visibility = 'public'
isSpecification = 'false'
          isRoot = 'false' isLeaf = 'false' isAbstract = 'true' isActive =
'false'/>
        <UML:Association xmi.id = 'a4' isSpecification = 'false' isRoot =
'false'
          isLeaf = 'false' isAbstract = 'false'>
          <UML:Association.connection>
            <UML:AssociationEnd xmi.id = 'a5' visibility = 'public'
isSpecification = 'false'
```

```
              isNavigable = 'true' ordering = 'unordered' aggregation =
'composite' targetScope = 'instance'
              changeability = 'changeable'>
              <UML:AssociationEnd.multiplicity>
                <UML:Multiplicity xmi.id = 'a6'>
                  <UML:Multiplicity.range>
                    <UML:MultiplicityRange xmi.id = 'a7' lower = '1' upper
= '1'/>
                  </UML:Multiplicity.range>
                </UML:Multiplicity>
              </UML:AssociationEnd.multiplicity>
              <UML:AssociationEnd.participant>
                <UML:Class xmi.idref = 'a2'/>
              </UML:AssociationEnd.participant>
            </UML:AssociationEnd>
            <UML:AssociationEnd xmi.id = 'a8' visibility = 'public'
isSpecification = 'false'
              isNavigable = 'true' ordering = 'unordered' aggregation =
'none' targetScope = 'instance'
              changeability = 'changeable'>
              <UML:AssociationEnd.multiplicity>
…….
```

7.4.6 Summary

Strength of the MT engine:

The MT is implemented to support MOF, QVT. The approach made in MT engine is to make a meta model language, the MTL or BasicMTL, which can be seen as the PIM model in MT engine. The MT engine mainly addresses on the development of a metamodel language for model transformation, but not the implementation.

Weakness of the MT:

The MT engine provide only an implementation of compiler of a metamodel transformation language. The tool set for supporting other features of QVT are not jet available.

7.5 Tool ATL(The Atlas Transformation Language)

7.5.1 Vendor and Version

ATL (Atlas Transformation Language) has been defined to perform general transformations within the MDA framework (Model Driven Architecture).
The version of ATL binaries used in this thesis is ATLv0.2.

7.5.2 Installation

ATL is download under "http://eclipse.org/downloads/index.php" for eclipse-SDK and "http://download.eclipse.org/tools/emf/scripts/downloads.php" for EMF.

Download the eclipse-SDK-3.0.2-{platform}.zip with platform=win32 for Windows or platform=linux-gtk for Linux.

Download emf-sdo-xsd-SDK-I200405131028.zip.

Download the latest version of adt (adt_YYYYMMDD.zip).

Unzip eclipse-SDK-3.0-{platform}.zip, this will create an eclipse directory with every Eclipse files in it.

Unzip emf-sdo-xsd-SDK-I200405131028.zip from the same directory (it also contains an eclipse directory, with the EMF additional files to Eclipse SDK).

Enter the eclipse directory then unzip adt_YYYYMMDD.zip, which does not contain an eclipse directory but plugins and features directories to be placed into the already-existing eclipse directory.

7.5.3 Introduction

The following contents are taken from website "http://www.sciences.univ-nantes.fr/lina/atl/".

The ATLAS group is an INRIA research team located at the University of Nantes and linked to the LINA research laboratory (Laboratoire d'Informatique de Nantes-Atlantique).

Under the direction of Patrick Valduriez, the ATLAS group is mainly interested in the area of complex data management for distributed systems.

The Atlas Transformation Language (ATL) is a hybrid language (a mix of declarative and imperative constructions) designed to express model transformations as required by the MDA approach to answer the QVT RFP issued by OMG. It is described by an abstract syntax (a MOF meta-model), a textual concrete syntax and an additional graphical notation allowing modelers to represent partial views of transformation models, shown in Figure 7-2. A transformation model in ATL is expressed as a set of transformation rules. The recommended style of programming is declarative. Transformations from Platform Independent Models (PIMs) to Platform Specific Models (PSMs) can be written in ATL to implement the MDA.

One central principle of the ATL project is *"Transformations are models"*, the transformation model is also treated as input for the transformation engine.

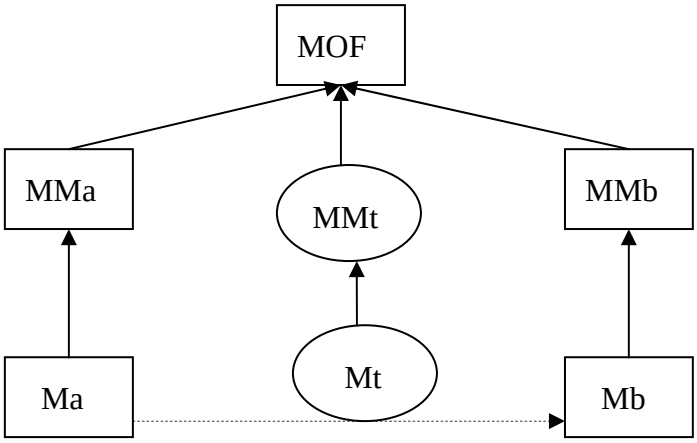

Figure 7-2 Transformation Engine

ATL v0.2: a hybrid transformation engine. ATL v0.2 integrates both declarative and imperative styles of transformation and is ported to Eclipse/EMF.

The engine has been implemented in the Java programming language. Consequently, the Netbeans MDR meta-data repository is choose as metamodel repository. Input and output models and meta-models are loaded, handled and serialized through this repository. ATL transformation models are first read using an ANTLR-generated parser and loaded into a Java-implemented ATL-specific meta-data repository. Netbeans MDR was not used to deal with transformation models for development reasons. Then, every model and meta-model needed (typically: input model plus input and output meta-models) is read from its XMI definition. The transformation is

then executed, rule after rule and the resulting model is eventually serialized to an XMI file.

The ATL framework is a repository of models supporting the creation of a library of transformations ranging from simple examples to fully reusable components. A prototype repository already exists that stores transformation components in compressed archives (ZIP files) including a meta-data description in the form of an XML file (contents.xml). Components can be: transformations (written in ATL, written in another language or composite transformations using others), meta-models (a transformation depends on the transformation meta-model, the input and output meta-models) and models (input/output samples). The ATL framework also integrates the notion of "technological spaces" Although mainly intended to deal with MDA models (based on MOF meta-models and accessible via XMI or JMI), this framework should also handle other kinds of models from different technological spaces (e.g. Java programs, XML documents, DBMS artifacts, etc.). To this end, what we need is a collection of adaptable "injectors" and "extractors" to complement the library of MDA transformation components. Models, meta-models, transformations, injectors and extractors are examples of MDA components that will be handled as uniformly as possible by the ATL repository.

ATL project a part of the Eclipse GMT project is going to be used as an IDE for ATL, with advanced code edition features (syntax highlighting, auto-completion, etc.). ATL will provide a context in which transformation-based MDA tools can be designed and implemented for Eclipse.

The ATL project, part of the Eclipse GMT project, will progressively provide a complete environment for developing, testing and using model transformation programs through the following items:

A front-end of the ATL framework repository.

A source code editor for transformations adapted from the Eclipse source code editor. Several levels of implementation are possible, from a simple text editor to a full-featured one (including syntax highlighting, auto-completion, etc.). Users will be able to launch transformations from the IDE, which could interpret error messages and use them to point out the problems.

A debugger to complete the Eclipse IDE for ATL.

And finally a transformation engine (for ATL v0.2) will be released as part of the GMT project .

7.5.3.1 Transformation process

The source model for using ATL is UML model. UML model will be transformed to a internal format, called XMI.

7.5.3.2 Source model

UML model (XML metamodel) is as source model.

7.5.3.3 Intermediate model

XMI is as intermediate model.

7.5.3.4 Transformer

Java is as transformer.

7.5.4 Test and Evaluation

Category: ATL is PIM to PSM Transformation Tool, PSM to code Transformation Tool, Tunable Transformation Tool, Transformation Definition Tool.

The following criteria are checked in order to have a rough imagination of the system characteristics of the transformation tools ATL:

Self containing: many documents are available. In the documents are the concept of ATL introduced. The syntax of ATL abstract Language and how to make transformation for model and metamodel are also described.

Scalability: the user can not freely decide for the dimension of the transformation.

Simplicity: Transformations are made automatically for the user.

Bi-directional mappings: only theoretically possible, no example is shown.

Ease of Adoption: the syntax of ATL is well defined, can be used as transformation language in other project.

Rich Conditions: Query/View/Transformation is fully supported in the ATL.

Provide an abstract syntax for the transformation language: The abstract syntax of ATL is described in the documentation.

Adopt common terminology: document is written in English and understandable.

Support composition and reuse: is supported in ATL by model repository.

Support complex transformation scenarios: is supported in ATL

Provide complete examples: the example of model transformation for web service is available.

Robust on transformation executions: not supported yet in ATL engine.

Tooling aspect: traceability is supported in ATL.

7.5.5 Transformation Example

Input format: ATL Project

Output format: XMI

One Example is shown here using MT engine to transform a XMI input to Java code.

Input: ATL project

Filename: uml2java

The atl file, input xmi file and output xmi file are shown in Figure 7-3 .

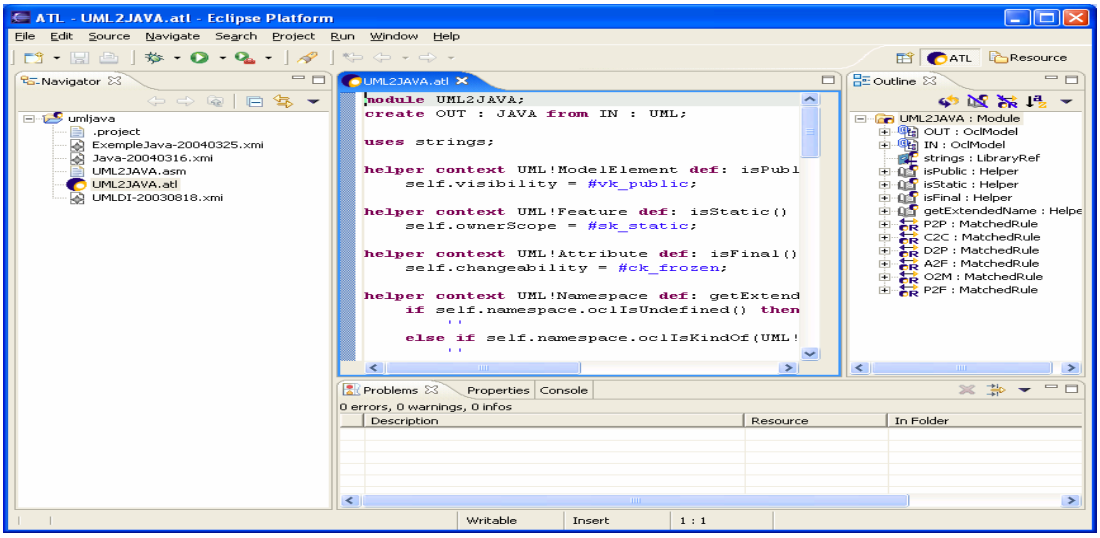

Figure 7-3 Example ATL

An ATL file is created in menu "File > New > File ...". the name of your ATL file will been following by the ".atl" extension.

The structure of an ATL file is described in three parts :

the *headers* declare the name of the module, which meta-model we will create from which one, and what other modules we need to import ;

the *helpers* define methods for a specific context and a node in the ingoing meta-model ;

the *rules* apply transformations from the ingoing meta-model to the outgoing one.

For example:

Headers

The headers report global information for the ATL file.

module UML2JAVA;

create OUT : JAVA from IN : UML;

uses strings;

The *module* has to match the name of the file, as for Java. It thus specifies the name of the current transformation.

The *create* line defines two variables :

The first variable, here "OUT", sets the meta-model to produce ;

The second variable, here "IN", sets the meta-model to be parsed.

The *uses* line tells which extern modules to use there. This corresponds to the *import* term in Java. Here we specify to use the extern string module.

Rules

Rules are methods allowing transformation from a model to another one. The name should be explicit because it's not used by the ATL builder but gives a simple information on the rule.

```
rule C2C {
	from e : UML!Class
	to out : JAVA!JavaClass (
		name <- e.name,
		isAbstract <- e.isAbstract,
		isPublic <- e.isPublic(),
		package <- e.namespace
	)
}
```

A rule takes one ingoing model, here *UML!Class*, and at least one outgoing model, here *JAVA! JavaClass*. So you understand we try to transform an UML class into a Java class.

Properties (like attributes, references, ...) are filled with the assignement "<-". The right members are ingoing model properties, OCL expressions or helpers. In this example, we call the helper which is described below.

Helpers

The helpers, like functions, make the development more easier for the developer by providing extended abilities for most rules. This is also a way to factorize the code, that is more esthetic.

helper context UML!ModelElement def: isPublic() : Boolean =

self.visibility = #vk_public;

Then for this helper, we specify the context *UML!ModelElement* for the helper *isPublic* that returns a *Boolean*. We find in the context the ingoing meta-model UML specified in the headers and a node in this model. The helper can take parameters. The body can contain OCL expressions above ATL ones.

Output: xmi file

```
<?xml version="1.0" encoding="ISO-8859-1" ?>
```

```
- <XMI xmi.version="1.2" xmlns:Model="org.omg.xmi.namespace.Model"
    timestamp="Mon Mar 22 15:07:45 CET 2004">
- <XMI.header>
- <XMI.documentation>
  <XMI.exporter>Netbeans XMI Writer</XMI.exporter>
  <XMI.exporterVersion>1.0</XMI.exporterVersion>
    </XMI.documentation>
    </XMI.header>
- <XMI.content>
  <Model:Class xmi.id="a1" name="int" annotation="" isRoot="false"
    isLeaf="false" isAbstract="false" visibility="public_vis"
    isSingleton="false" />
  <Model:Class xmi.id="a2" name="void" annotation="" isRoot="false"
    isLeaf="false" isAbstract="false" visibility="public_vis"
    isSingleton="false" />
- <Model:Tag xmi.id="a3" name="javax.jmi.packagePrefix" annotation=""
    tagId="javax.jmi.packagePrefix">
  <Model:Tag.values>org.gaetan</Model:Tag.values>
- <Model:Tag.elements>
  <Model:Package xmi.idref="a4" />
    </Model:Tag.elements>
    </Model:Tag>
- <Model:Package xmi.id="a5" name="PrimitiveTypes" annotation="" isRoot="false"
    isLeaf="false" isAbstract="false" visibility="public_vis">
- <Model:Namespace.contents>
  <Model:PrimitiveType xmi.id="a6" name="String" annotation="" isRoot="true"
    isLeaf="true" isAbstract="false" visibility="public_vis" />
  <Model:PrimitiveType xmi.id="a7" name="Boolean" annotation="" isRoot="true"
    isLeaf="true" isAbstract="false" visibility="public_vis" />
    </Model:Namespace.contents>
    </Model:Package>
- <Model:Package xmi.id="a4" name="JAVA" annotation="" isRoot="false"
    isLeaf="false" isAbstract="false" visibility="public_vis">
- <Model:Namespace.contents>
- <Model:Import xmi.id="a8" name="PrimitiveTypes" annotation=""
    visibility="public_vis" isClustered="true">
- <Model:Import.importedNamespace>
  <Model:Package xmi.idref="a5" />
```

```
    </Model:Import.importedNamespace>
    </Model:Import>
- <Model:Class xmi.id="a9" name="JavaElement" annotation="" isRoot="false"
    isLeaf="false" isAbstract="true" visibility="public_vis"
    isSingleton="false">
- <Model:Namespace.contents>
- <Model:Attribute xmi.id="a10" name="name" annotation="" scope="instance_level"
    visibility="public_vis" isChangeable="true" isDerived="false">
- <Model:StructuralFeature.multiplicity>
  <XMI.field>1</XMI.field>
  <XMI.field>1</XMI.field>
  <XMI.field>true</XMI.field>
  <XMI.field>true</XMI.field>
    </Model:StructuralFeature.multiplicity>
- <Model:TypedElement.type>
  <Model:PrimitiveType xmi.idref="a6" />
    </Model:TypedElement.type>
    </Model:Attribute>
    </Model:Namespace.contents>
    </Model:Class>
- <Model:Class xmi.id="a11" name="ClassFeature" annotation="" isRoot="false"
    isLeaf="false" isAbstract="true" visibility="public_vis"
    isSingleton="false">
- <Model:Namespace.contents>
- <Model:Attribute xmi.id="a12" name="isFinal" annotation=""
    scope="instance_level" visibility="public_vis" isChangeable="true"
    isDerived="false">
- <Model:StructuralFeature.multiplicity>
  <XMI.field>1</XMI.field>
  <XMI.field>1</XMI.field>
  <XMI.field>true</XMI.field>
  <XMI.field>true</XMI.field>
    </Model:StructuralFeature.multiplicity>
- <Model:TypedElement.type>
  <Model:PrimitiveType xmi.idref="a7" />
    </Model:TypedElement.type>
    </Model:Attribute>
    </Model:Namespace.contents>
```

```
- <Model:GeneralizableElement.supertypes>
  <Model:Class xmi.idref="a9" />
    </Model:GeneralizableElement.supertypes>
    </Model:Class>
```

7.5.6 Summary

Strength of the ATL:

The ATL is implemented to support MOF, QVT. The approach made in ATL is very strong. The abstract syntax transformation language is available and also the transformation engine is available in ATL v0.2. Model and meta model repository is also supported by ATL.

Weakness of the ATL:

The user of ATL is not freely to scale the transformation, M-N transformation is not supported by ATL yet.

7.6 Tool GMT (Generative Model Transformer)

7.6.1 Vendor and Version

Eclipse, open source software development project, is the vendor of GMT.

GMT version 0.1 uses Ecore as a foundation.

GMT version 0.2 uses GMT.gcore enabled MDA tool orchestration.

GMT version 0.3 provide web-based specification tools that are suitable for use in a distributed team environment.

GMT version 0.4 provide specification tools for visual domain-specific modeling languages.

GMT version 0.4 is used in this thesis.

7.6.2 Installation

Download homepage is http://www.eclipse.org/gmt. TH GMT project uses the results archived in project ATL. The GMT is still in constructing. FUUT-je(Fantastic, Unique, UML Tool for the Java Environment) is the first downloadable component for GMT. It is prototype of PSM supporting in GMT for Java environment. The feature of FUUT-je is mostly taken from its predecessor, a Business Component Prototype for IBM SanFrancisco .

7.6.3 Introduction

The following contents are taken from website “http://www.eclipse.org/gmt”.

The goal of the Generative Model Transformer project is to construct/assemble a set of tools for model driven software development with fully customizable Platform Independent Models, Platform Description Models, Texture Mappings, and Refinement Transformations.

The project will result in

> A tool that fulfils the MDA promise for faster/more accurate/better maintainable application development.

A tool for industrial use that can be used with a Model-Driven Software Development paradigm.

MDA related research - which is encouraged and needed. Results will be factored into the project where applicable.

The GMT uses the ATL as described in Chapter 5.3 as its meta model language.

7.6.3.1 Transformation process

FUUT-je optimizes Java environment. First XML schema is imported as source model. Then it generates applicative code using Swing GUI.

7.6.3.2 Source model

UML class diagram(XML) is as source model in FUUT-je tool.

7.6.3.3 Intermediate model

There is no Intermediate model.

7.6.3.4 Transformer

Java is as transformer.

7.6.4 Test and Evaluation

Category: GMT is PIM to code Transformation Tool, Tunable Transformation Tool, Transformation Definition Tool.

The following criteria are checked in order to have a rough imagination of the system characteristics of the transformation tools GMT, FUUT-je:

Self containing: many documents are available. In the documents are the concept of GMT and FUUT-je introduced. The syntax of ATL abstract Language is used as meta model language in GMT. The documents in ATL can be used also in GMT.

Scalability: the user has the possibilities to use plugins for generate the Java code.

Simplicity: Java code are generated automatically for the user.

Bi-directional mappings: Not supported.

Ease of Adoption: the template principle for generating Java code can be adopted by some transformer.

Rich Conditions: Query/View/Transformation is fully supported in the ATL.

Provide an abstract syntax for the transformation language: The abstract syntax of ATL is described in the documentation.

Adopt common terminology: document is written in English and understandable.

Support composition and reuse: is supported in ATL by model repository.

Support complex transformation scenarios: is only supported in ATL, not in FUUT-je yet.

Provide complete examples: the example is not available yet in FUUT-je.

Robust on transformation executions: not supported yet in FUUT-je.

Tooling aspect: not supported yet in FUUT-je.

7.6.5 Transformation Example

Input format: DTD, Schema, GME Project, Java source file, Java class file, XML, Object Diagram, XML as Project, XMI(FUUT-je)
Output format: Java code(FUUT-je), Application Program, XML as Project

One Example is shown here using MT engine to transform a XMI input to Java code.

Input:

UML diagram save in the XML file. Figure 7-4 shown the input uml diagram.

Figure 7-4 Example MT Engine, Input

This example project can save as XML model.

File name: address.xml

```
- <!--
 FUUT-je model saved on: Thu Dec 02 13:53:09 CET 2004  -->
- <ClassMap>
-<company>
- <![CDATA[
yourcompany
  ]]>
  </company>
-<frameworkRoot>
- <![CDATA[
prototypes
  ]]>
  </frameworkRoot>
-<name>
- <![CDATA[ AddressBook Example ]]>
  </name>
-<outDir>
- <![CDATA[[c:\fuut\src
  ]]>
  </outDir>
- <outPath>
- <![CDATA[ c:\fuut ]]>
  </outPath>
- <owner>
- <![CDATA[ You]]>
  </owner>
- <packageName>
- <![CDATA[ address ]]>
  </packageName>
-<ModelItem type="Class" name="AddressBook">
-<ClassHeader>
```

```
-<description>
- <![CDATA[ <p> </p>]]>
  </description>
  <line1Pos>177</line1Pos>
  <line2Pos>214</line2Pos>
-<name>
- <![CDATA[ AddressBook]]>
  </name>
-<position>
- <![CDATA[ 184 @ 148]]>
  </position>
-<title>
- <![CDATA[ ClassHeader Properties]]>
  </title>
-<xmlTag>
- <![CDATA[ AddressBook]]>
  </xmlTag>
  </ClassHeader>
-<ClassBody>
  <attributesVisible>true</attributesVisible>
- <frameworkType>
- <![CDATA[ Entity]]>
  </frameworkType>
  <generateCode>true</generateCode>
  <isAbstract>false</isAbstract>
  <methodsVisible>false</methodsVisible>
  <persistenceRoot>true</persistenceRoot>
-<scope>
- <![CDATA[ public]]>
  </scope>
  <selected>true</selected>
-<stereotype>
- <![CDATA[ Class]]>
  </stereotype>
  <upDownButtons>true</upDownButtons>
-<version>
- <![CDATA[ 1]]>
  </version>
```

```
  <visibleInModel>true</visibleInModel>
  <xml1stLowercase>false</xml1stLowercase>
-<Attribute>
- <dbType>
```

Output:

Produce multi Java code and application-program. This example is a address book, which shown in Figure 7-5.

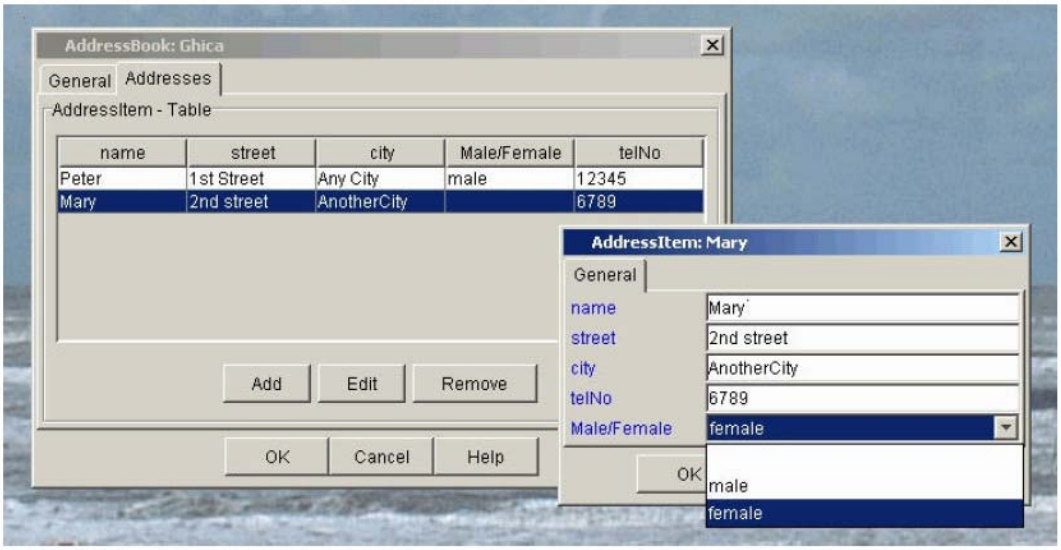

Figure 7-5 Example MT Engine, Output

7.6.6 Summary

Strength of the GMT engine, FUUT-je:

A prototype transformer is available. FUUT-je also provide a Java GUI code generator.

Weakness of the GMT, FUUT-je:

GMT is still under construction. Many features are still need to be implemented.

7.7 Tool BOTL(Bi-directional Object oriented Transformation Language)

7.7.1 Vendor and Version

ArgoUML4BOTL - Editor for BOTL meta models and rules. It produces BOTL-XML specifications.

7.7.2 Installation

Download home page is http://www4.in.tum.de/~marschal/botl/download.htm.

Run download jar file.

7.7.3 Introduction

The following contents are taken from website "http://www4.in.tum.de/~marschal/botl".

BOTL is a rule based model transformation language. It supports to specify transformations of object oriented models. The Bidirectional Object-oriented Transformation Language (BOTL) is based on a formalization of UML class diagrams. Currently BOTL serves mainly as a specification mechanism for the description of tool chains and the integration of development models.

Its main Features are:

BOTL is based upon a precise, formal foundation and a comprehensible, graphical notation.

BOTL comes with techniques that allow to verify

- whether a BOTL specification is applicable,
- whether it will produce models that are conform to a given meta model, and
- whether specifications are bijective.

It means, users are allowed to build/use transformation rules base on graphics to transform class diagrams from one model to an other model.

The transformation rules are specified using an easily understandable UML-like notation. Currently the prototypical model transformation tool is based on ArgoUML.

Figure 7-6 gives an overview over the BOTL tool infrastructure. One can specify rules and meta models with the ArgoUML extension. A verification component (not yet realized) verifies whether the rules will produce valid (meta model-conform) output or not. BOTL transformation specifications are stored as XML documents and can be consumed by a transformer component that transforms BOTL representations of object models. Adaptors can be used to import and export object models from different technical notations, like e.g. Java object structures or XMI representations. Currently there is an adaptor for Java objects, an XMI adaptor is under development.

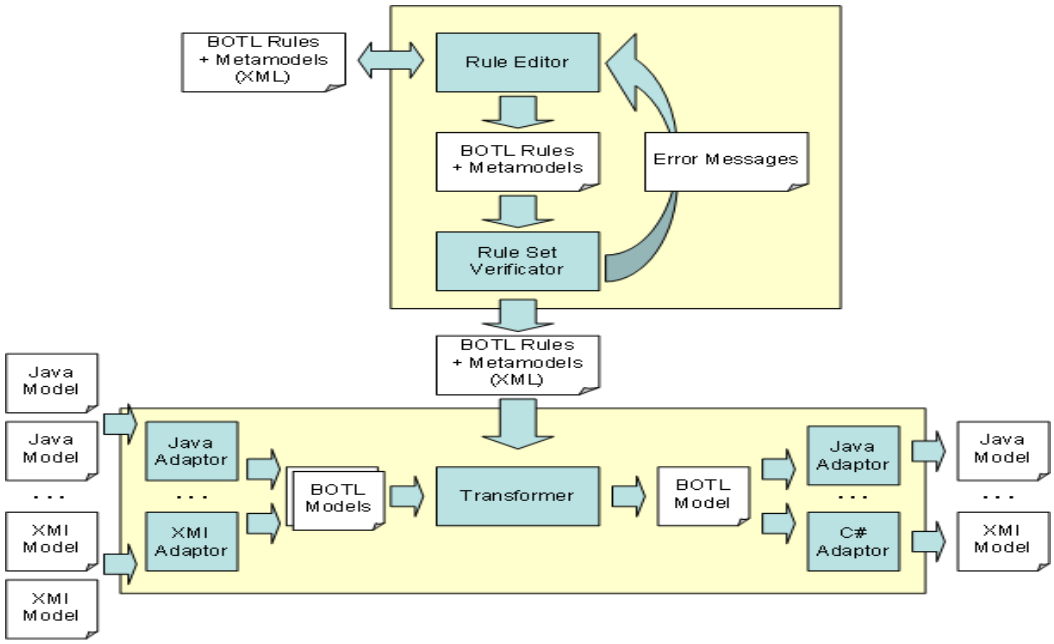

Figure 7-6 Overview of the BOTL tool support

7.7.3.1 Transformation process

UML model is used as source model for an object in BOTL. A BOTL rule set consists of a number of rules that create model fragments. The fragments of source model are translated with rules into target model fragments and merged the newly created fragments into a target model.

7.7.3.2 Source model

Because BOTL transforms always object models, source model is class models, which can be mapped to UML or MOF meta class.

7.7.3.3 Intermediate model

BOTL model is used as the internal model.

7.7.3.4 Transformer

BOTL use BOTL rules and XML metamodel as Transformer. A more detailed explanation of BOTL rules is provided in Braun, Marschall (2003).

7.7.4 Test and Evaluation

Category: BOTL is PIM to PSM Transformation Tool, PIM to code Transformation Tool, PSM to code Transformation Tool, Tunable Transformation Tool, Transformation Definition Tool.

The following criteria are checked in order to have a rough imagination of the system characteristics of the transformation tools BOTL:

Self containing: many documents are available. In the documents are the concept of BOTL introduced. The working principles of BOTL how to make transformation for model and meta model are also described.

Scalability: the user can not freely decide for the dimension of the transformation.

Simplicity: Transformations are made automatically for the user.

Bi-directional mappings: only theoretically possible, the works on bijective is still under construction.

Ease of Adoption: the BOTL rules work mechanism is well defined, can be used as transformation tool set in other projects.

Rich Conditions: Query/View/Transformation is not supported in the BOTL.

Provide an abstract syntax for the transformation language: will be not supported by BOTL.

Adopt common terminology: document is written in English and understandable.

Support composition and reuse: No repository support in BOTL.

Support complex transformation scenarios: is supported in BOTL.

Provide complete examples: many examples for class diagrams transformation are available.

Robust on transformation executions: not supported yet in BOTL.

Tooling aspect: not supported yet in BOTL.

7.7.5 Transformation Example

Input format: Object Diagram(XMI), XMI, XML, Botl file
Output format: XML, Botl file, java file

One Example is shown here using XML input to Java code, shown in Figure 7-7. But here are only part of the code. If you want to know the details, you down load the example "uml2uml.botl "on website "http://www4.in.tum.de/~marschal/botl/download.htm ".

Input: file: acme-sample

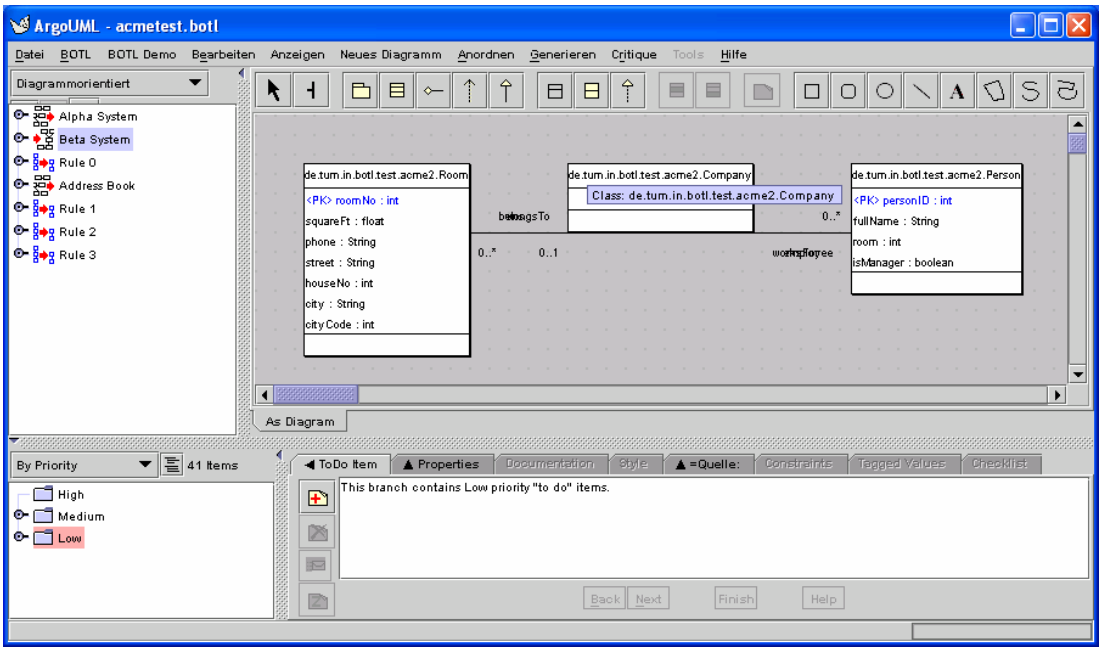

Figure 7-7 Example BOTL, Input

There are two applications in this example: the Alpha Information System and the Beta Application. The both process data are about employees and offices. The structures of the two applications, which use object model, are determined by UML class diagrams.

```
<?xml version="1.0" encoding="UTF-8" ?>
- <RuleSet name="acme">
- <Rule name="Rule 0">
- <SourceModelVariable>
```

```
-<ObjectVariableAssociation cardinality="1">
-<OVAEnd0 id="id17715814">
  <ObjectVariable>ov0</ObjectVariable>
  <ClassAssociationEnd>id842753</ClassAssociationEnd>
    </OVAEnd0>
- <OVAEnd1 id="id5310299">
  <ObjectVariable>ov2</ObjectVariable>
  <ClassAssociationEnd>id17667041</ClassAssociationEnd>
    </OVAEnd1>
  <ClassAssociation>id15555956</ClassAssociation>
    </ObjectVariableAssociation>
- <ObjectVariable id="ov0">
  <Class>id10753836</Class>
- <AttributeVariable>
  <Term value="o" />
  <Attribute>id15556236</Attribute>
    </AttributeVariable>
- <AttributeVariable>
  <Term value="m" />
  <Attribute>id30578471</Attribute>
    </AttributeVariable>
    </ObjectVariable>
- <ObjectVariable id="ov2">
  <Class>id23826254</Class>
- <AttributeVariable>
  <Term value="id" />
  <Attribute>id6340542</Attribute>
    </AttributeVariable>
- <AttributeVariable>
  <Term value="f" />
  <Attribute>id1359399</Attribute>
    </AttributeVariable>
- <AttributeVariable>
  <Term value="s" />
  <Attribute>id30408657</Attribute>
    </AttributeVariable>
    </ObjectVariable>
  <Metamodel>id1546945</Metamodel>
```

```
    </SourceModelVariable>
- <DestinationModelVariable>
- <ObjectVariableAssociation cardinality="1">
- <OVAEnd0 id="id1210203">
  <ObjectVariable>ov3</ObjectVariable>
  <ClassAssociationEnd>id31456955</ClassAssociationEnd>
    </OVAEnd0>

    ……..……..
- <SourceMetamodel name="Alpha System" id="id1546945">
  <Type it="int" id="id9259939" />
  <Type it="String" id="id30176509" />
  <Type it="float" id="id31084639" />
  <Class name="de.tum.in.botl.test.acme.Enterprise" id="id28375046" />
- <Class name="de.tum.in.botl.test.acme.Employee" id="id23826254">
- <Attribute name="personID" type="id9259939" id="id6340542">
  <DefaultValue />
    </Attribute>
- <Attribute name="firstName" type="id30176509" id="id1359399">
  <DefaultValue />
    </Attribute>
- <Attribute name="secondName" type="id30176509" id="id30408657">
  <DefaultValue />
    </Attribute>
  <PrimaryKey>id6340542</PrimaryKey>
    </Class>

    ……..……..
```

Output:

The Alpha Information System(AIS) should be transformed to correspond to the Beta Application(BA)’s meta model. Inter operation between the two applications that try to express the same or similar concepts in a common application domain, either has to be converted in a format according to one of the partners’ metamodel format or a common metamodel might be chosen. in this example the AIS is to be extended to be capable of communicating with the BA.

```
- <DestinationMetamodel name="Beta System" id="id3628285">
  <Type it="int" id="id32132273" />
```

```
  <Type it="float" id="id8815770" />
  <Type it="String" id="id29226651" />
  <Type it="boolean" id="id1431333" />
- <Class name="de.tum.in.botl.test.acme2.Room" id="id12192270">
- <Attribute name="roomNo" type="id32132273" id="id22091307">
  <DefaultValue />
    </Attribute>
- <Attribute name="squareFt" type="id8815770" id="id23839420">
  <DefaultValue />
    </Attribute>
- <Attribute name="phone" type="id29226651" id="id9007193">
  <DefaultValue />
    </Attribute>
- <Attribute name="street" type="id29226651" id="id19857159">
  <DefaultValue />
    </Attribute>
- <Attribute name="houseNo" type="id32132273" id="id6906552">
  <DefaultValue />
    </Attribute>
- <Attribute name="city" type="id29226651" id="id13655511">
  <DefaultValue />
    </Attribute>
- <Attribute name="cityCode" type="id32132273" id="id29426954">
  <DefaultValue />
    </Attribute>
  <PrimaryKey>id22091307</PrimaryKey>
    </Class>

    ……..
```

7.7.6 Summary

Strength of the BOTL:

The BOTL is aimed at class diagrams transformation for UML. It is base on graphical, rule based principle. The transformation for the users is very comfortable.

Weakness of the BOTL:

The BOTL has not full filled some of the requirements defined in MOF 2.0, QVT. It is still only concentrating on the transformation of class diagrams in Object oriented design. The input is only in Java, XMI input is still under construction.

There are a few wrong of this Tool. For example, if a attribute is deleted, the whole class is deleted.

7.8 Tool OptimalJ

7.8.1 Vendor and Version

OptimalJ is a commercial modeling software product from Company Compuware. A trial version of OptimalJ Professional Edition free for 30 days is available.

The version of OptimalJ used in the thesis is Compuware OptimalJ BE 3.2.

7.8.2 Installation

Free download from http://javacentral.compuware.com/download.htm.

Run the jar file.

7.8.3 Introduction

The following contents are taken from website " http://javacentral.compuware.com " and book "MDA Explained the Practice and Promise of the model driven archtecture" [Klep03].

OptimalJ is an advanced development environment that provides design, build, modify and deploy distributed Java applications. This is the approach of Compuware for the Object Management Group's (OMG) Model Driven Architecture (MDA) in Java. The OptimalJ model levels include the Domain Model, the Application Model and the Code Model, which corresponding the PIM, PSM and code level in the OMG MDA. Figure 7-8 shows the architecture of OptimalJ. It address mainly to the development in J2EE area, can accelerate J2EE development by generating working applications directly from visual models.

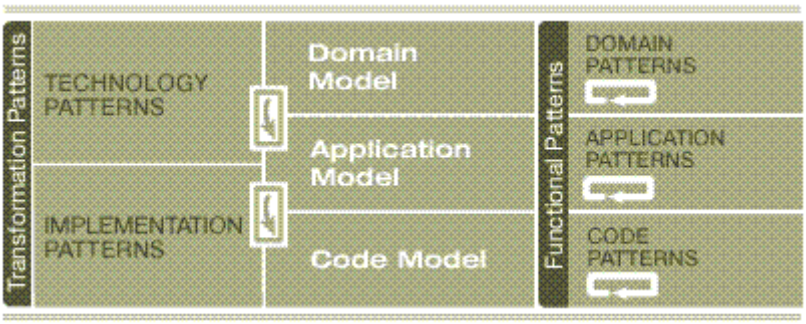

Figure 7-8 OptimaJ Architecture[Klep03]

OptimaJ uses a notation of patterns to achieve PSM transformations which is a slightly different notation (structural) for the MDA-part of the tool.

OptimalJ provides the following main functionalities:

The shortest path to greater J2EE application productivity

OptimalJ provide a model-based environment for developing applications in J2EE. The developers is allowed to work in a declarative way in the domain model level, The domain model is high-level and object-based, containing the structure of the information that is handled by the application and relationships between different data structures. It contains domain classes and their attributes, associations, business methods and rules, but doesn't contain any implementation or coding detail. From the domain model, OptimalJ generates presentation, logic and data application models via generation patterns in line with the Sun J2EE standard.

Faster, easier Java development:

The developers only need to interact with a visual model of the application and the codes are automatically generated. The development environment provides also a source editor, class browser, form editor and debugger to enable developers to view, modify and customize the generated application. Existing classes not generated by OptimalJ can be imported and called by the application.

Business Rules Editor

Based on user-defined business rules within OptimalJ, developers can customize applications easily. Using the Business Rules Editor, you can add both static and dynamic business rules at the model level. Dynamic business rules are stored in a rules base, which allows changes to the application to be made during deployment—without touching the application. OptimalJ translates the business rules into Java code and implements them at the appropriate points in the application. By making business rules easily identifiable elements in the model, you can implement business requirement changes quickly, without extensive coding.

Pattern-driven generation

At the heart of OptimalJ is a set of design and implementation templates called patterns. OptimalJ uses patterns to generate all the code required for a running

application. The applications can use the full range of J2EE components including Session and Entity Enterprise Java Beans (EJB), Java Server Pages (JSP) and application data tables.

Active synchronization

OptimalJ synchronizes Java code with the application models so the model accurately represents the delivered application at all times. This synchronization allows to change an application by modifying elements at the model level; OptimalJ automatically regenerates only the changed code.

Integrated deployment

OptimalJ support fully the Sun J2EE production servers. An other integrated Compuware OptimalServer can be used as an option to OptimalJ. It also includes an open-source test environment that includes a web server and EJB container. The developers can directly test as they develop without worrying about the complexities of deployment.

7.8.3.1 Transformation process

(1) The source model for using OptimalJ tool is UML model. It produces UML with variant using special stereotypes as internal format.

(2)J2EE is used to transform UML to special UML. This transformation is executed in order to simplify later transformations.

(3) OPtimalJ use special UML as the internal model representation. With this format transforms OptiamlJ to the different target code or format, like SQL, Java, JSP…

(4) In OptiamlJ is normally transformer J2EE.

7.8.3.2 Source model

The source model for using OptimalJ tool is UML model.

7.8.3.3 Intermediate model

In termediate model is UML variant using special stereotypes

7.8.3.4 Transformer

OptimalJ uses patterns to generate all the code required for a running application.

7.8.4 Test and Evaluation

Category: OptimalJ is PIM to PSM Transformation Tool, PSM to code Transformation Tool, Tunable Transformation Tool, Transformation Definition Tool.

The following criteria are checked in order to have a rough imagination of the system characteristics of the transformation tools OptimalJ:

Self containing: many documents are available. In the documents are the concept of OptimalJ introduced.

Scalability: since the OptimalJ, also from the name of the product, which optimized for the Java(J2EE) application, the user can only decide in frame of J2EE and Java. For other transformation to other technologies the OptimalJ will provide transformation patterns in technology patterns.

Simplicity: Transformations are made automatically for the user. The user only need to use graphical interface to design the model.

Bi-directional mappings: not support yet by OptimalJ.

Ease of Adoption: the whole product is well constructed develop environment, but any separation of the product will be not easy.

Rich Conditions: Query/View/Transformation for the model is not supported in the OptimalJ.

Provide an abstract syntax for the transformation language: The transformation from Domain model to Application model uses so called Technology Patterns in OptimalJ. Technology Patterns are written in Java, so architects can derive new Technology Patterns by extending the ones provided by OptimalJ. It is a declarative technology. OptimalJ will provide a new Technology Pattern to do it.
The transformation to code is also based on pattern(template of Java code) is described in the documentation. The Java code generated by Code Patterns is not directly related to the component definitions in the Application Model. This will allow the developers flexibility to use specific patterns to customize the application.

Adopt common terminology: document is written in English and understandable. The model architecture are slightly different from MDA terminology.

Support composition and reuse: business rules and code template can be saved and reused in OptimalJ .

Support complex transformation scenarios: in frame of J2EE is supported in OptimalJ.

Provide complete examples: the example of establish of typical MDA application in J2EE is provided by Compuware.

Robust on transformation executions: not supported yet in OptimalJ.

Tooling aspect: not supported yet in OptimalJ.

7.8.5 Transformation Example

Input format: uml

Output format:

PSM: Entity-Relationship diagram, EJB stereotype, Stereotype for Web interface

Code: SQL, Java, JSP

One Example is shown here using OptimalJ to transform a XMI input to Java code. A concrete example of the MDA process is give, how simple PIM is transformed automatically into rather complex PSMs and code. Example Rosa's Breakfast Service is described during tow steps(PIM to PSM and PSM to code) here.

Rosa's company supplies a complete breakfast to the homes of its customers. Customers can give the hour and place of delivery and their credit card number to get one of the breakfast menus on Rosa's Web site.

PIM of this example will be generated in UML and three PSM are needed. One PSM is for database, second is for the EJB model, and third is for the web interface. Therefore we have three PIM to PSM transformations.

We need also three code model for the PSMs. Three code model are SQL, Java and JSP. Three PSM will be transformed to code model:

A relational model to SQL transformation for database: a transformation takes a input PSM model written as an Entity-Relationship model and generates a code model written in SQL.

An EJB model to Java transformation for EJB: a transformation takes a input PSM model written in the UML EJB variant and generates a code model written in Java.

A Web model to JSP and HTML transformation for the Web site: a transformation takes a input PSM model written in the UML Web variant and generates a code model written in JSP and HTML.

Input:

PIM of example Rosa's Breakfast Service is written in plain UML. This is the model that is created "by hand". The model defines the breakfast services PIM in Figure 7-9 and show in Figure 7-10.

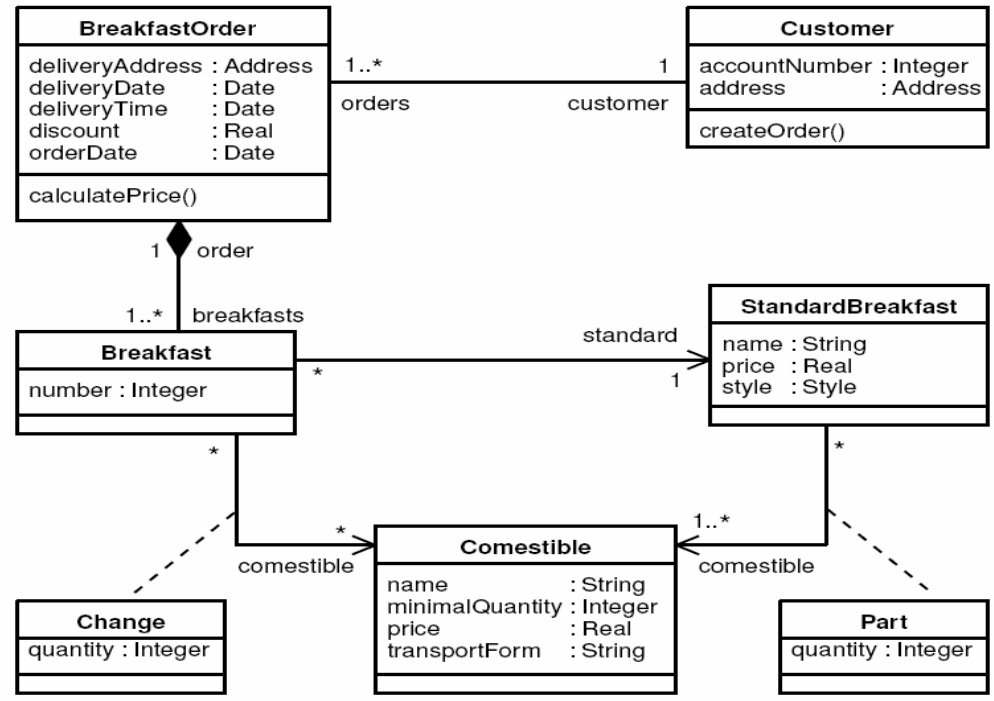

Figure 7-9 PIM of Rosa's Breakfast Service[Klep03]

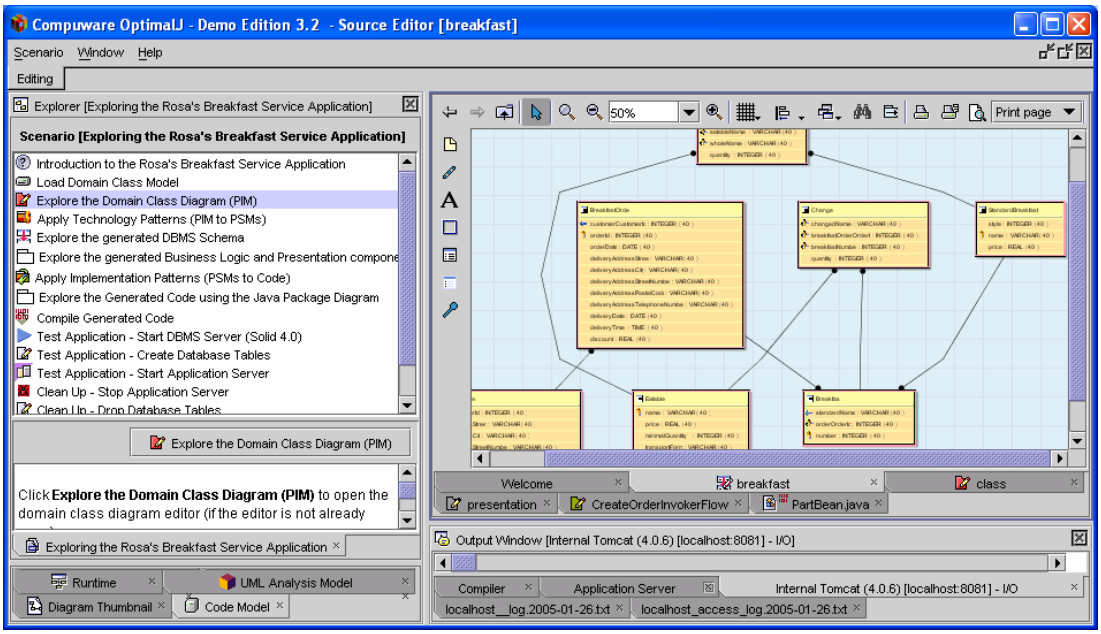

Figure 7-10 Example OptimaJ, Input

The model of the breakfast services transforms from PIM to PSM. Figure 7-11 depicts the resulting database model in the form of an Entity-Relationship diagram. Figure 7-12 shows the DBMS Schema in program.

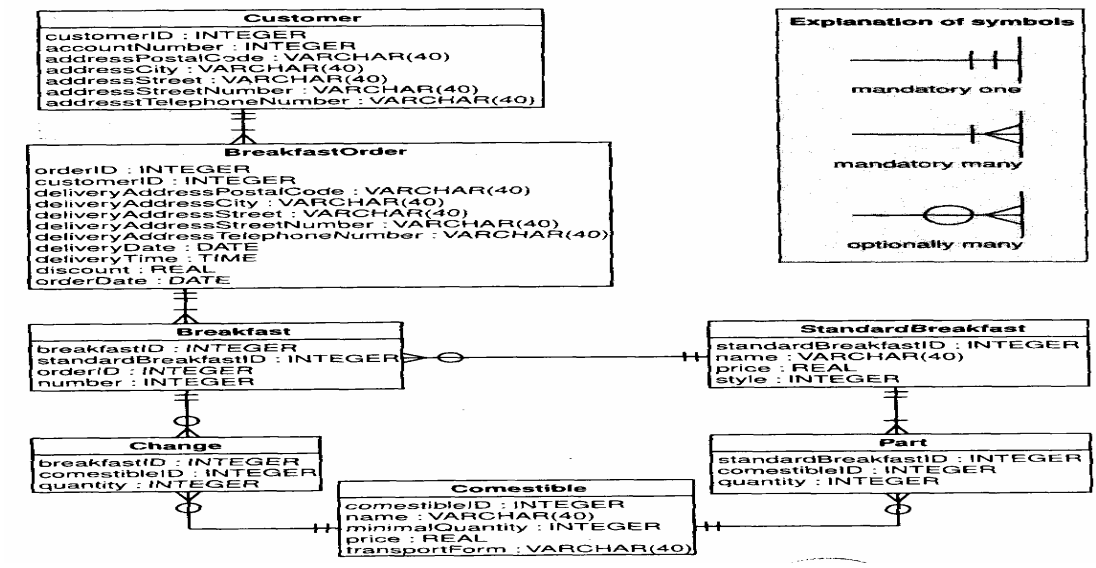

Figure 7-11 Relational PSM of Rosa's Breakfast Service[Klep03]

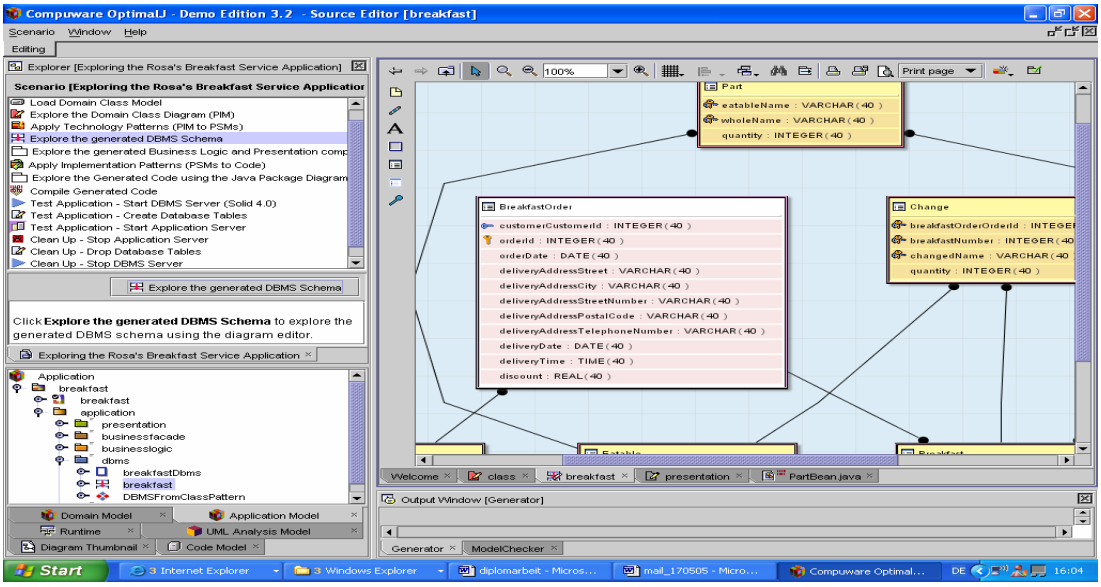

Figure 7-12 the DBMS Schema in program

Relational PSM of Rosa's Breakfast Service is only one PSM in this example. If you need more information about PIM and PSM of Rosa's Breakfast Service of OptimalJ tool, you can see the Kleppe's book "MDA Explained- The Model-Driven Architecture: Practice and Promise".

Output:

Result is compile able and executable code, shown in Figure 7-13. Costumers can order from one of the breakfast menus on the website with this program.

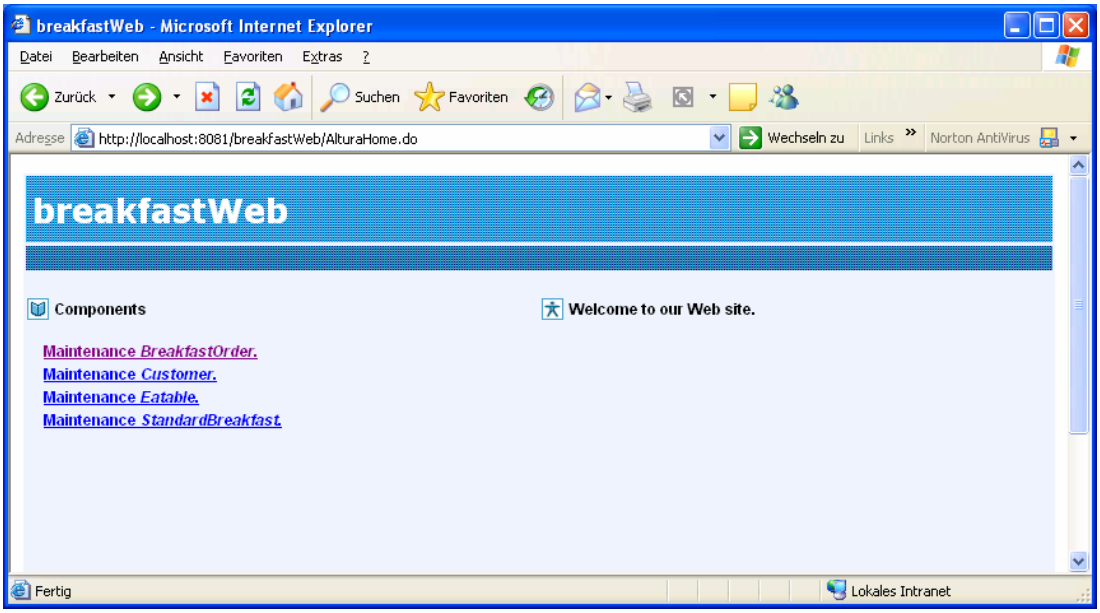

Figure 7-13 Example OptimaJ, Output

7.8.6 Summary

Strength of the OptimalJ:

Like any other commercial products the OptimalJ is well constructed and user friendly. It support fully support for J2EE development using MDA concept. The development environment is integrated. Documents are well constructed and understandable.

Weakness of the OptimalJ:

The terminology is slightly different to MDA. Reverse transformation is not possible. And because of commercial reasons the OptimalJ is scaled only use on J2EE technology. It is the main advantage and also the main disadvantage. The technology pattern which is the PIM to PSM transformation is also not adjustable for the user.

Chapter 8

Summary of the Tools

The following is a summary table for all the transformation tools used in this thesis:

	Self containing	Scalability	Simplicity	Bi-directional mappings	Ease of Adoption	Rich Conditions	Abstract syntax for the transformation language	Adopt common terminology	Support composition and reuse	Support complex transformation scenarios	Provide complete examples	Robust on transformation executions	Tooling aspect
UMT	Y	Y	Y	Y	Y	Y	N	Y	N	N	Y	N	N
MT L	Y	Y	N	N	Y	Y	Y	Y	N	N	Y	N	N
ATL	Y	N	Y	N	Y	Y	Y	Y	Y	Y	Y	N	Y
GMT	Y	Y	Y	N	Y	Y	Y	Y	Y	N	N	N	N
BOTL	Y	N	Y	N	Y	N	N	Y	N	Y	Y	N	N
OptimalJ	Y	Y	Y	N	N	N	Y	Y	Y	Y	Y	N	N

Y… The tool support the feature of MOF 2.0 QVT

N… The tool does not support the feature of MOF 2.0 QVT

P… The tool support only partially the feature of MOF 2.0 QVT

Table 8-1 System Characteristics of the state of the art transformation Tools

Summary of the Tools

tools	Input format	Output format	strength	weakness
UMT	XMI XMI Light XMI or XMI Light as Project	GML WSDL XML Schema SQL Java interfaces IDL PBEL ACEGIS Workflow J2EE Xdoclet J2EE servlet Xdoclet Example 'Java based' EMF Ecore BPEL(Business Proc.Exec Lang)	The UMT was implemented rather before MOF, QVT. The implementation reflects the concept of MDA light, in which no PSM model is used. This makes the implementation simple. And also makes the use of the tool easy. Documentation is well for the GUI user part.	Documents are not available for intermediate XMI light model. Query/View can only be supported through theoretically by XPath, no example s are provided.
GMT (FUUT-je)	DTD Schema GME Project Java source file Java class file XML Object Diagram XML as Project XMI(FUUT-je)	Java code(FUUT-je) Application Program XML as Project	A prototype transformer is available. FUUT-je also provide a Java GUI code generator.	GMT is still under construction. Many features are still need to be implemented.
MTL	MTL Project Java Project Plug-in Project Eclipse Modeling Framework	Application Program XMI	The MT is implemented to support MOF, QVT. The approach made in MT engine is to make a meta model language, the MTL or	The MT engine provide only an implementation of compiler of a metamodel transformation language. The

	XMI		BasicMTL, which can be seen as the PIM model in MT engine. The MT engine is mainly address on the development of a metamodel language for model transformation, but not on the implementation.	tool set for supporting other features of QVT are not jet available.
ATL	ATL Project	XMI	The ATL is implemented to support MOF, QVT. The approach made in ATL is very strong. Either abstract syntax transformation language is available and also the transformation engine are also available in ATL v0.2. Model and meta model repository is also supported by ATL.	The user of ATL is not freely to scale the transformation, M-N transformation is not supported by ATL yet.
BOTL	Object Diagram(XMI) XMI XML Botl	XML Botl file java	The BOTL is aimed at class diagrams transformation for UML. It is base on graphical, rule based principle. The transformation for the users is very comfort table.	The BOTL has not full filled some of the requirements defined in MOF 2.0, QVT. It is still only concentrating on the transformation of class diagrams in Object oriented design. The input is only in Java, XMI input is still under construction.
OptimalJ	uml	Entity-Relationship diagram	Like any other commercial products the OptimalJ is well	The terminology is slightly different to MDA.

		EJB stereotype Stereotype for Web interface SQL Java JSP	constructed and user friendly. It support fully support for J2EE development using MDA concept. The development environment is integrated. Documents are well constructed and understandable.	Reverse transformation is not possible. And because of commercial reasons the OptimalJ is scaled only use on J2EE technology. It is the main advantage and also the main disadvantage. The technology pattern which is the PIS to PSM transformation is also not adjustable for the user.

Table 8-2 Strength and Weakness of transformation tools

	PIM to PSM tool	PSM to Code tool	PIM to Code tool	TT tool	TD tool
UML	Y	Y	N	Y	Y
ATL	Y	Y	N	Y	Y
MTL	N	N	Y	Y	Y
GMT	Y	Y	Y	Y	Y
BOTL	Y	Y	Y	Y	Y
OptimalJ	Y	Y	Y	Y	Y

TT: Tunnable Transformation DT: Transformation Definition

Table 8-3 Tools Categories

Chapter 9

Conclusion

In our thesis, we have explained and demonstrated using MDA approach to support software development and the important concept in OMG MOF 2.0 Query, Views, and Transformations (QVT) for supporting MDA model transformation. Also a review of six Model transformation tools with respect to OMG /QVT concept has been provided. We come to the conclusion that ATL and GMT are the best realization of OMG /QVT concept among these tools during the evaluation in this thesis. In the following we will show the further works of these transformation tools which are under development could be done in the future.

The strength of the tool UMT(UML Model Transformation Tool) lays in the simplicity and the ability of using plugins concept in the tool. The weakness of UMT is the lacking of strong support of transformation language with abstract syntax. Since UMT was developed before the QVT recommendation, the concept of Query and View are not supported by the intermediate transformation language which is called as XMI Light in UMT. It seems also in the future the UMT will not be an implementation of OMG /QVT. It will remain as a light weight transformation tool. With the development of UMT, the user of the tool can expect more and more specific language transformation will be supported, the user will have strong support by editing his own transformation profile for his self defined transformations.

MTL defines an abstract syntax for model transformation, theoretically it can transform any model which defines a metamodel. MTL is an object- and view-oriented imperative language. Because the transformation language is imperative, it is more complicated to learn than the declarative languages. The implementation of MTL is a framework for model transformation. Currently MTL supports the following repositories: MDR (MetaData Repository, from Sun), ModFact (from Lip6) and EMF (Eclipse Modeling Framework, from IBM). The next approach of MTL will be the adaptation of QVT concepts.

ATL is an very strong QVT-based transformation language. The strength of the tool ATL lays in that the ATL try to be compliant with the OMG MOF/QVT (Queries / Views/ Transformation) recommendation. The future work around ATL will be more industrial applications. The future work around GMT, the tool set for ATL will be concentrated on more industrial applications. The strength of GMT is also the compliance with OMG MOF/QVT. The strength of BOTL is that it allows to specify transformations of object oriented models. The weakness of BOTL is the lacking of the OMG /QVT support, it goes different ways for model transformation. With its bi-directional model transformation, it can be integrated within an model transformation frame.

OptimalJ is the only commercial product evaluated in this thesis. It is a very sophisticated product with both built-in technology pattern transformations for the J2EE plat-form as well as tools for creating new transformation patterns or for customizing existing ones. OptimalJ represents an industrial trend of development for domain specific model transformation tool set. In OptimalJ the MDA concepts are very positive supported for implementation of applications in Service oriented Architecture. The weakness of OptimalJ is the lacking of support for OMG/QVT and it’s fix bundling to the J2EE technology.

The abstract transformation language will still be the most important work in the next step. Although most of the tools and papers have shown that a hybrid language (declarative and imperative combined) format should be used for the transformation language, but the detailed syntax has to been standardized in the near future.

The simplicity, scalability and documents are also important aspects in the implementation. For a fast and wide spread verification work all of these must be strongly supported.

For the future development of transformation tool we suggest the strong imperative aspect in MTL can be also learned during the next version of ATL. UMT can be a good transit solution before the complete QVT implementation comes or it can be a flexible light weight solution for model transformation tool in contrast to other heavy weight model transformation tools. BOTL can be used in some transformation frameworks as plug-in, the work of adaptation to the framework can be one developing direction for it. OptimalJ is now the best industrial solution for domain specific applications. The merge of the tool set such as ATL, GMT and OptimalJ can be one big step towards the end realization of MDA concept in the future.

The tools evaluated in this thesis have shown the approaches to reach the goal of the RFP made by OMG MOF 2.0 Query/View/Transformation. Many of the works have made great progress to the goal of QVT concept to support MDA. But a real implementation of QVT as defined in RFP is still a long way to go. According to both review papers after the RFP and also according to the evaluation work in this thesis, there is still many research work to be done before a industry standard can be archived. In the mean time implementations of the research works also, have to be done in order to verify the concepts in practice.

MDA is a great step towards automatic software development made by OMG recently. QVT is one of the small but important steps in MDA. Despite all the difficulties in first draft of QVT, a industrial standard has shown his first shapes in all the implementations. We hope the MDA will be continuously developed and be one of the first great success for the software industry in the 21 century.

Chapter 10

Bibliography

[Arlo04] Enterprise Patterns and MDA- Building Better Software with Archetype Patterns and UML. Jim Arlow, Ila Neustadt. 2004, ISBN 0-321-11230-X

[Klep03] MDA Explained- The Model-Driven Architecture: Practice and Promise. Anneke Kleppe, Jos Warmer, Wim Bast. 2003, ISBN 0-321-19442-X

[Mell02] Executable UML- A Foundation for Model-Driven Architecture. Stephen j. Melloer, Marc J. Balcer. 2002, ISBN 0-201-74804-5

[Oest04] Objekt-orientierte-Software-entwicklung Analyse und Design mit der UML 2.0. Bernd Oestereich. 2004, ISBN 3-486-27266-7

[Carl01] Modeling XML Applications with UML Practical e-Business Applications. David Carlson. 2001, ISBN 0-201-70915-5

[Gräs04] UML 2.0 projektorientiert Patrick Grässle, Henriette Baumann, Philippe Baumann. 2004, ISBN 3-89842-547-9

[Fran03] Model Driven Architecture Applying MDA to Enterprise Computing. David S. Frankel. 2003, ISBN 0-471-31920-1

[Mell04] MDA Distilled Principles of Model-Driven Architecture. Stephen J. Mellor, Kendall Scott, Axel Uhl, Dirk Weise. 2004, ISBN 0-201-78891-8

[Rais04] Model Driven Architecture with Executable UML. Chris Raistrick, Paul Francis, John Wright, Colin Carter, Ian Wilkie. 2004, ISBN 0-521-53771-1

[Nee02] the Object Management Group OMG/RFP/QVT MOF 2.0 Query/Views/Transformations RFP. http://www.omg.org/docs/ad/02-04-10.pdf, 2002

[DSTC 03] DSTC/IBM Response to the MOF 2.0 Query/Views/Transformations RFP(ad/2002-04-10).http://www.omg.org/docs/ad/03-08-03.pdf,2003

[Alc03] Alcatel, Softeam,Thales, TNI-Valiosys, Codagen Technologies Corp OpenQVT Response to the MOF 2.0 Query/Views/Transformations RFP. http://www.omg.org/docs/ad/03-08-05.pdf , 2003

[Com03] Compuware Corporation, SUN Microsystems. XMOF Queries, Views and Transformations on Models using MOF, OCL and Patterns. http://www.omg.org/docs/ad/03-08-07.pdf , 2003

[Tata03] Tata Consultancy Services. QVT Partners Revised submission for MOF 2.0 Query /Views /Transformations RFP.http://www.omg.org/docs/ad/03-08-08.pdf , 2003

[IOS 03] Interactive Objects Software GmbH Project Technology, Inc. IO/PT Revised Submission to MOF Query / View / Transformation RFP. http://www.omg.org/docs/ad/03-08-11.pdf, 2003

[Gard03] Tracy Gardner, Catherine Griffin, Jana Koehler Rainer Hauser. Review of OMG MOF 2.0 Query/Views/Transformations Submissions and Recommendations towards final Standard. http://www.omg.org/docs/ad/03-08-02.pdf, 2003

[Lang03] Benoit Langlois, Nicolas Farcet, THALES recommendations for the final OMG standard on Query / Views / Transformations. http://www.softmetaware.com/oopsla2003/langlois.pdf, 2003

[IBM99] XML Metadata Interchange (XMI) Version 1.1 ftp://ftp.omg.org/pub/docs/ad/99-10-02.pdf, 1999

[Miller03] Joaquin Miller and Jishnu Mukerji. MDA Guide Version 1.0.1 http://www. omg. org/ docs/omg/03-06-01.pdf, 2003

[Braun01] UML Tutorial, David Braun, Jeff Sivils, Alex Shapiro, Jerry Versteegh http://pigseye.kennesaw.edu/~dbraun/csis4650/A&D/UML_tutorial/ , 2001

[MDSD04] Model-Driven Software Development, http://www.mdsd.info/mdsd_cm/page.php?page=mdsd, 2004

Chapter 11

Appendix

11.1 Glossary

AS	Action Semantics
ATL	Atlas Transformation Language
BOTL	Bi-Directional transformation Language
CIM	Computation Independent Model
OCL	Object Constraint Language
CORBA	Common Object Request Broker Architecture
CWM™	Common Warehouse Meta-model
DTD	Document Type Definition
EJB	Enterprise Java Bean
EMF	Eclipse Modeling Framework
GMT	Generative Model Transformer
IDL	Interface Definition Language
J2EE	Java 2 Enterprise Edition
JCP	Java Community Process
JSP	Java Server Pages
MDA™	Model-Driven Architecture™
MDR	Meta data Repository
MOF™	Meta-Object Facility
MTL	Meta Transformation Language
OMG	Object Management Group
PIM	Platform Independent Model
PSM	Platform Specific Model
QVT	Query, View, Transformation
RFP	Request for Proposals
UML™	Unified Model Language
UMT	UML Model Transformation Tool
WSDL	Web Services Description Language

XML	eXtensible Markup Language
XMI	XML Metadata Interchange
XSL	eXtensible Style sheet Language
XSLT	XSL for Transformation

11.2 Description of Hardware Testing System

The evaluation works have been done using the following hardware configuration:

CPU:	Intel Centrino™ 1.5GHz
Memory:	1024MByte
Operating System:	Microsoft Windows XP™