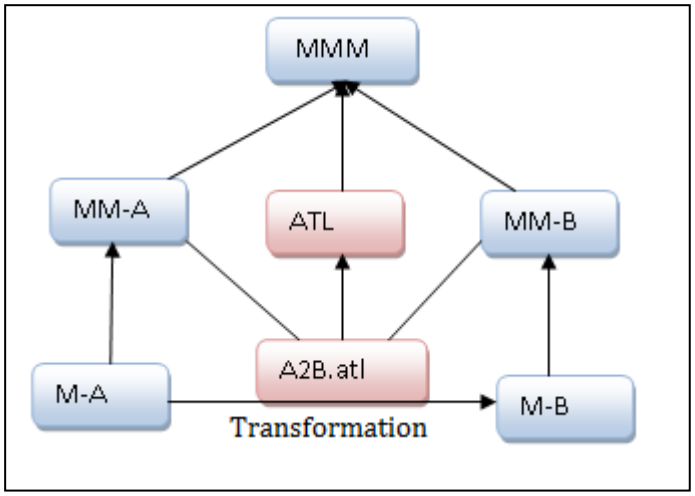
CHALMERS



Investigating Model Transformation Technology For Architecture Description Languages

Nigsti Berhanu Ayele

Chalmers University of Technology
University of Gothenburg
Department of Computer Science and Engineering
Göteborg, Sweden, October 2011

SYSTEMITE
Keeping complex systems consistent, correct and complete.

Model Transformation for Architecture Description Languages

NIGISTI BERHANU AYELE

Examiner: Dr. Miroslaw Staron, Associate Professor and Director of Software Engineering and Technology, and Software Engineering and Management

Supervisors: Professor Jörgen Hansson, Professor and Head of Division Software Engineering
Mr. Söderberg, CTO of Systemite

Chalmers University of Technology
University of Gothenburg
Department of Computer Science and Engineering
SE-412 96 Göteborg
Sweden
Telephone + 46 (0)31-772 1000

Cover Picture: It represents ATL based model transformation architecture. In order to transform source model to the target model, the source, target, and ATL meta-models are required.

Abstract

This thesis studies Model Transformation for Architecture Description Languages (ADLs) in the contexts of automotive electrical and electronics. Embedded Automotive SysTems ADL (EAST-ADL) is an architecture description language, which captures automotive electrical and electronic systems with sufficient detail to allow modeling for documentation, design, analysis and synthesis, which is currently, implemented using different tools such as SW (SystemWeaver), MetaEdit+, Enterprise Architect and Papyrus. EAST-ADL XML (EAXML) and SW XML (SWxml) are two different implementations of EAST-ADL. Investigating the possibility of using Model Driven Transformation to transform EAXML to SWxml and vice versa has been the main part of this study.

The investigation includes discovering the architectural and structural relationships between EAXML, which is an AUTOSAR based representation, and SWxml, which is SW based implementation. The architectural patterns of these two implementations are defined and are used to drive the transformation requirements. Hence, to investigate the architectural and structural relationships between SWxml and EAXML, we developed and validated meta-models of these two implementations. According to the architectural pattern, we derived transformation requirements and stated the mapping rules between EAXML and SW accordingly.

Based on the analysis made on EAST-ADL, EAXML and SW, a prototype has been developed to prove that model driven transformation is a possible way to realize model transformation. According to the mappings, we have conducted analysis on the meta-models to drive the rules. These rules are implemented using the Atlas transformation language (ATL). Because of the different levels of abstraction between EAXML and SW model, complementary and main transformations are implemented. In addition to ATL, eclipse based frameworks such as AtlanMod MegaModel Management (AM3), Eclipse Modeling Framework (EMF), and Graphical Modeling Framework (GMF) are used to implement the model transformation.

A prototype of bidirectional transformation has been implemented as a proof-of-principle. We transformed the SW model to EAXML and then we transferred the EAXML back to SW and imported it to the SW platform. Since the acceptance tests made to perform bidirectional transformation using the prototype worked without problem, model driven transformation, specifically ATL, has indicated to be a promising solution for the bidirectional transformation between SW and EAXML.

Acknowledgments

Next to GOD, I would like to express my deepest gratitude to my supervisors, Prof. Jörgen Hansson, Professor and Head of Division Software Engineering, and Mr. Söderberg, CTO of Systemite. I believe both of them have supported me in the following ways: Prof. Jörgen, he has been an inspiration since the first course he gave me, and yet regardless of how busy he is, he accepted my idea and encouraged me to work with him. It is because of his enthusiastic class and encouragement that I am in this position. Jan, he gave me the opportunity to work with an interesting topic of study and also he gave me the help and support I needed to accomplish this work. I am grateful for the experience I gained from working under his supervision. I am also thankful to him for acknowledging my work by throwing a party in his company. The support and help I received from both made the study easier.

I owe my deepest gratitude to all Systemite staff members for their cheery helpful attitudes they gave me during 6 months stay with them, and to INSA higher administrators and the rest of the staff members for their encouragement. Special thanks go to Mr. Tsehaye Kidane and Mr. Abi Ahmed.

I am thankful to the department of Computer Science and Engineering, and Chalmers library for their help and fast response for every help I needed during my study in CTH.

I would like to thank my classmates of Software Engineering and Technology field, for making my study in Chalmers and in Sweden of great experience.

I am also grateful for the continuous support I received from my bf Asmelash Tsegay and my uncle Mr. Solomon Tedella. It is also my pleasure to thank all my Ethiopian and International friends for the sisterly and brotherly care they provided me with.

Lastly and most importantly, I would like to take this opportunity to thank my beloved family, my mother (Mrs. Tsirihu Abreha), my father (Mr. Berhanu Ayele), and my sister (Abeba Berhanu). I am so thankful to the lessons I received from my mam throughout my life. She is the greatest person in my life who taught me to respect people, friendship, Love and job. I would not be in this position without their courage and lessons. Thanks to all for setting the best example to my life.

Contents

Abstract iii
Acknowledgments iv
Contents v
List of Tables vi
List of Figures vii
Abbreviations viii
1. Introduction 1
1.1. Report Structure 1
2. Background 2
2.1. EAST-ADL Implementation in SW 3
2.2. EAXML 5
2.3. Why Transformation? 5
3. State-of-the-Art 7
3.1 Introduction to MDE 7
3.2 Model transformation 7
3.2.1 ATL 7
3.2.2. QVT 8
3.2.3. Kermeta 9
3.2.4. ETL 10
3.2.5. Dually 10
3.2.6. oAW 11
3.3. Model Transformation Languages Metrics 11
4 Problem Statement 12
4.1. Problem Definition 12
4.2. Research Questions 13
4.3. Research Methodology 14
5. SWxml and EAXML 16
5.1. SWxml vs EAXML 16
5.2. Meta-Model Representation Views 20
6. The Model Transformation 25
6.1. Model Transformation Languages Evaluation 25
6.1.1. ATL 25

6.1.2. oAW 32
6.1.3. Result of the Transformation Language Investigation 34
6.2. Prototype Implementation 34
6.2.1. Meta-model implementation 34
6.2.2. Transformation Logic 35
6.2.3. The ATL Implementations 38
6.3. Contribution of the Solution 41
7. Summary 42
Future Work 43
References 45
Appendix A 47
SW.xml 47
Appendix B – XML Schema meta-models 50
Appendix C – The EMF meta-models 52
SW Meta-model 52
XML Meta-Model 56
Appendix D - ATL Transformation Files 57
XML2SW.atl 57
SW2EAXML.atl 62
EAXML2SW.atl 68
SW2XML.atl 75

List of Tables

Table 1: ATL handling capacity 8
Table 2: Metrics of different transformation languages 11
Table 3: Feature modeling implementation analysis 19
Table 4: SW2EAXML 38
Table 5: EAXML2SW 38

List of Figures

Figure 1: Structure of an item 4
Figure 2: SWxml 5
Figure 3: ATL model transformation [4] 8
Figure 4 : relationships between QVT meta-models [39] 9
Figure 5: Model transformation using Kermeta [38] 10
Figure 6: Simple structure 13
Figure 7: Complex Structure 13
Figure 8 : Research methodology 15
Figure 9: Feature modeling from EAST-ADL Specification 17
Figure 10: Sample EAXML user model 20
Figure 11: Sample SW object model 20
Figure 12: EAXML in XML Schema format 21
Figure 13: EAXML meta-model in EMF 21
Figure 14: The partial view of EAXML using GMF 22
Figure 15: SW meta-model in XML format 22
Figure 16: SW in XML Schema format 23
Figure 17: SW met-model in EMF format 24
Figure 18: The partial view of SW meta-model in GMF 24
Figure 19: Composite transformation (34) 29
Figure 20: SW2EAXML main transformation 30
Figure 21: EAXML2SW main transformation 30
Figure 22: SW2EAXML complementary transformation 31
Figure 23: EAXML2SW complementary transformation 31
Figure 24: Complete bidirectional transformation between SW and EAXML 32
Figure 25: SW and EAXML transformation using oAW 33
Figure 26: The relationship between meta-models and ATL 35
Figure 27: The bidirectional transformation flow and logic 36
Figure 28: The bidirectional transformation high level view 39
Figure 29: SWXML2SW.atl header 39
Figure 30: SW2EAXML.atl header 39
Figure 31: EAXML2SW.atl header 39
Figure 32: SW2XML.atl header 40
Figure 33: Tool integration logic 42

Abbreviations

ADLs – Architecture Description Language
AM3 - AtlanMod MegaModel Management
ATL – Architecture Transformation Language
AUTOSAR – AUTomotive Open System Architecture
DI – Dependency Injection Pattern
DSL – Domain Specific Language
EAST-ADL – Embedded Automotive SysTems ADL
EAXML – EAST-ADL XML
EMF – Eclipse Modeling Framework
EAXMLMM – EXML XML meta-model
ETL- Epsilon Transformation Language
KM3 – Kernel Meta Meta Model
MDD – Model Driven Development
MDE – Model Driven Engineering
MDSD – Model Driven Software Development
MDT – Model Driven Transformation
MOF – Meta Object Facility
MTT –Model Transformation Technology
M2C – Model to Code Transformation
M2M – Model to Model Transformation
M2M2T – Model to Model to Text Transformation
M2T – Model to Text Transformation
OCL – Object Constraint Language
OMG – Object Management Group
oAW – open Architecture Ware
QVT – Query View Transformation
RSQ – Research Question
SW – SystemWeaver
SW-M – SystemWeaver Model
SW-MM – SystemWeaver meta-model
T2T – Text to Text transformation
XSD - XML Schema Definition
XMI – XML Metadata Interchange
XML – eXtensible Mark-up Language
UML – Unified Modeling Language

1. Introduction

This study is part of the maenad project (www.maenad.eu), and is undertaken in Systemite. Its purpose is to investigate model transformation technologies for ADLs.

EAXML and SWxml are two different implementations of the EAST-ADL. To increase reusability of information, Two-way transformation implementation between SWxml and EAXML is required. This study covers the investigation of model transformation technologies and the transformation between SWxml and EAXML. ATL, which is found to be the research based solution, is described in detail towards solving the transformation between SWxml and EAXML.

Upon maenad project objectives, the main objective of investigating this model transformation is to later integrate SW with other different tools which are used to implement EAST-ADL. For example, if integration between SW and MetaEdit+ is required then the integration can be realized through transformation of SW to EAXML and then transforming EAXML to MetaEdit+. This kind of transformation enables us to perform tool integration with less resource and fewer transformations than tool integration though explicit transformation, which is a transformation without using EAXML.

Generally, transforming the SW's concept to the EAXML and vice versa supports systematic reuse of information.

By reviewing a range of research papers and internet sites this paper investigates different model transformation technologies and examines their performance against four requirements: compatibility, usability, performance and complexity. The report then considers selecting the promising model transformation language by proving the principles through prototype implementation.

1.1. Report Structure

To clearly define the problem definition and present the solution the report is structured as it is shown below.

Background: this chapter defines important concepts used throughout the research.

State-of-the-art: this chapter describes related fields of studies which are available.

Problem Statement: this chapter elaborates the main problem that is solved by this research study. It also describes the research questions that have to be answered and the research methodology that is used to solve it.

SWxml and EAXML: this chapter describes both SWxml and EAXML. It also describes the analysis between these two implementations.

The Model transformation: this chapter describes the solution of the research study. It describes the MTT selection and the prototype implementation.

Generally, the analysis conducted and the complete solutions along with the logics used to solve the problem are presented in this chapter.

Summary: in this chapter, we summarized the overall research and suggested the feature work that we believe has to be done in the future.

2. Background

Any software systems development starts from requirements. These requirements can be gathered using different mechanisms according to the product type, the market and the current circumstances. For successful development and maintenance, the requirements should be well documented in such a way that developers and users can understand them. The specification document is the main center of communication among different parts which are directly or indirectly involved in that project. Documenting the requirements using a model based development technique increases correctness, unambiguity, verifiability of the document and improves understanding of the user. For this reason, MDSD process has been a powerful process in the area of developing safety critical systems. It is a vital software development process which can also enable us to generate an executable code as a byproduct form. Such system development process decreases the level of misunderstanding that could happen during implementation. This is of great importance for testing the system against its test-cases which are derived from the use-case or other types of models of the system. In order to develop the desired system successfully, considering all conditions at early design stage is the most important part. OMG fosters MDA [30][42]. MDA has been evaluated by many companies as an approach used to specify and develop applications where systems are represented as models and transformation functions [30]. The transformation functions are used to map between these models as well as to generate automatically executable code [17]. The idea of model driven software development can be realized using these MDD approaches.

The term model is defined by Seidewitz as "a set of statements about some system under study" [13]; it is a set of affirmations, constraints or rules to get a higher abstraction of a problem. It is also defined as "measure, rule, pattern, example to be followed" [14]. Meta-model is the structural and organizational formal description of a model, which is specified using the semantics introduced by meta-meta-models.

ADLs are used to design both software and hardware. In the sense of software, they are used for analyzing and representing software architecture. The ADLs for software capture the behavioral specifications of components and their interactions that comprise the software architecture. However, the ADLs for hardware capture the structure (hardware components and their connectivity) and behavior (instruction-set) of programmable architectures consisting of the processor, coprocessor, and the memory subsystem. Based on the above definition, EAST-ADL is an ADL for a System.

DSL is a language tailored to a specific domain for solving a wide range of different problems [20]. It is the opposite of general purpose programming languages such as c, java and general purpose modeling languages such as UML. The main objective of DSL is to reduce time-to-market by enabling development and using concepts closer to the problem domain at hand, rather than those offered by programming languages. Because of this, DSL is the basis for model transformation. Models created using domain specific language can be transformed into other models developed using different modeling tools which use different implementation concept.

EAST-ADL is a language used to capture automotive electrical and electronic systems with sufficient details by refining the requirements in such a way that all parts which are related to the system can understand it with minimum level of ambiguity [3]. The fact that we use different tools to implement EAST-ADL leads to model transformation and tool integration. It enables modeling through introduction of different layers. These layers allow reasoning of the overall system requirements by dividing them at several levels of abstraction. These layers are called Vehicle, Analysis, Design, Hardware and Implementation levels. According EAST-ADL Specification [3], these different levels are described below:

- Vehicle level describes the decomposition of system characteristics organized as a software product line.
- The analysis level includes the Functional Analysis Architecture (FAA) which is built from an abstract definition of the system to capture analysis support of what the system shall do.
- The design level includes the Functional Design Architecture (FDA) which represents a decomposition of functionalities denoted in the FAA, including behavioral description but excluding software implementation constraints. The decomposition has the purpose of making it possible to meet constraints regarding non-functional properties such as allocation, efficiency, reuse, or supplier concerns and the final level refers to the system element in an AUTOSAR model.
- The Hardware Architecture which includes Electronic Control Units, communication links, sensors and actuators and their connections are also considered as analysis level.
- Implementation level refers to the System level in the AUTomotive Open System Architecture (AUTOSAR) model
    - AUTOSAR is a standard for software implementations in the automotive industry. This is stated as the main argument for the AUTOSAR-compliance of the EAST-ADL exchange format.

Tool availability is one of the goals to achieve safety through modeling techniques [8]. SW is one of the tools EAST-ADL has implemented. SW considers the classes and references of EAST-ADL as an Item and part respectively.

2.1. EAST-ADL Implementation in SW

SW Concept: the main concept of SW is model based development where the actual building blocks or components act as information carriers. The concept also includes a powerful system data model which describes logical and physical components and their interactions, inheritance or relations, status, requirements, version and variants, and access to components [37].

SW platform: which is part of the implementation of SW concept, is a multi-tier architecture with support for access control, secure communication, server and client side cache as well as support for powerful configuration management. It is generally an infrastructure used for maintaining consistency in design information, distribution of consistent design information and integration of design process [37].

Models in SW are built by Item, representing the suitable building blocks that the models are built by. All items have a unique identifier so that the item can be distinguished from other items, and so that it can be located and referenced in a SW database.

(SW identifiers are represented by hexadecimal strings like x0000001D04000023, or complete URLs like url: swap: //localhost:1768/x0000001D04000023 but in this description we use shorthand notations to illustrate the IDs). An item has a number of standard *properties*, like a name, creation date, description and more.

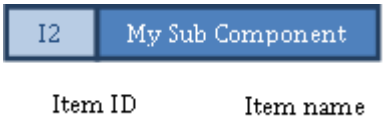


Figure 1: Structure of an item

SWxml: in SW, EAST-ADL is implemented using Items rather than objects and is exported in XML file format. We call the exported XML file SWxml. A class or a reference is identified as an item. From this, it is easy to recognize that the XML file is represented using sets of Items. The Item can be one of the following two.

itemType – this is the item which corresponds to classes in EMF.

partType – this is the item which corresponds to references in EMF.

For further understanding of the SWxml, look in Figure 2. The concept of item is the result of SW component based implementation.

```
<?xml version="1.0" ?>
- <SystemWeaver3>
  - <Info>
      <Description>feature</Description>
      <ExportedBy>admin</ExportedBy>
      <ExportDate>2011-03-23</ExportDate>
      <ExportedItem>x0000001D04000027</ExportedItem>
    </Info>
  - <Meta>
    - <Types>
      + <ObjTypes>
      + <ItemTypes>
      + <PartTypes>
      </Types>
    + <AttributeTypes>
    </Meta>
  - <Items>
    - <Item id="x0000001D04000027" sid="ss0" ancestor="x0000001D04000027">
        <Name>Feature</Name>
        <VersionText />
        <VersionNumber>1</VersionNumber>
        <Status>I</Status>
      - <Description>
        - <Binary>
            <![CDATA[  ]]>
          </Binary>
          <Plain />
        </Description>
        <Attributes />
      - <PartGroups>
        - <PartGroup sid="ss05">
            <Name>requiredBindingTime</Name>
          - <Parts>
```

Figure 2: SWxml

2.2. EAXML

It is an AUTOSAR based representation of EAST-ADL and is transformed to XML Schema. EAST-ADL is first implemented using UML objects and classes and then later transformed to AUTOSAR representation of an XML Schema file. According to Dominguez et. al. [21], the EAXML represented in XML Schema is resulted from M2M2T transformation and the following mappings are made during the M2M2T transformation.

a. All classes are transformed into ComplextType and Element
b. Root class is mapped to an Element and a ComplexType
c. Elements are declared for every class and they are organized in a substitution group

2.3. Why Transformation?

Representing a system using different implementation concepts lead to the concept of model transformation. Using different concepts to implement EAST-ADL results to a system represented using different syntax and semantics. This issue leads to the concept of model transformation where the interoperability of the meta-models [15], reusability, maintainability and scalability [17] can be attained. In order to accomplish the goal of bidirectional

transformation, it is important to apply bidirectional transformation approach [12]. Even if it is not possible to implement the bidirectional transformation at once, it is important to perform the Two-way transformation using different approaches/mechanisms. For xml based model transformations, XML-support language and meta-model based transformations are two important constraints that should be considered during implementation of this transformation [16].

Generally, in MDD, there are two types of transformations called M2M and M2C transformations. They are also sometimes called as horizontal and vertical transformations [4]. Most of the time, M2M transformation is horizontal. However, vertical transformation is a transformation made within a model to generate another model, meta-model or code of a specific model.

Model transformation technologies are useful for reusability of models in cases where a system is described using two overlapping models [3]. In Maenad project SW, MetaEdit+, and Papyrus are three tools that have to be integrated to each other using the common implementation called EAXML. According to [3], EAST-ADL is a large language with at least 283 classes and 564 associations, where every class has a transformation rule taking care of attributes. For this reason, none of the tools which implement EAST-ADL makes full implementation of it. According to [Mr. Söderberg] about 90% of EAST-ADL is implemented. Each tool has different implementation concepts. Due to the high complexity caused by size and rapid evolution, developing transformation from one meta-model to another implementation of EAST-ADL has been difficult [4].

Since there are many types of modeling tools, one of the most important things during modeling is to identify the kind of objects that should be modeled. Objects can be static or dynamic and the objects in AUTOSAR are static objects. Keeping this in mind, there are many tools which enable us to model objects, such as UML, EMF, SW, etc. Machine or vehicle objects are static; it is mostly difficult to model machines using different available software modeling tools. UML 2.0 is second-generation ADL which is used for modeling software intensive systems and it has more than 13 viewpoints. It mostly covers technology and business aspects of the architecture during modeling. Due to those many viewpoints, UML diagrams can be interpreted in different ways and the different interpretations of diagrams leads to ambiguity. This is one reason for representing AUTOSAR using schema rather than the objects in UML. However, tools like SW enable modeling of static objects through component based modeling.

There are not many clear explanations to answer the question “why is it not possible to represent AUTOSAR objects using UML objects?” However, based on [Mr. Söderberg’s] point of view, the main reason AUTOSAR objects cannot be represented using UML objects is that UML is designed to represent dynamic creation of objects which implies to software modeling rather than vehicles or machines. Therefore EAXML uses AUTOSAR M3 rules for the meta-model by using element, association, type, prototype, is-of-type and instanceref constructs. Because of this, AUTOSAR defines the structure and semantics of the data using UML class diagrams and semantics using common data exchange language [7].

3. State-of-the-Art

3.1 Introduction to MDE

MDE is a software development approach which uses models as constructive elements for the implementation of the system. In the history of model driven engineering, the level of abstraction to develop systems has always been increasing. Many approaches have evolved using MDE concepts. MDA, which is realization [30] of MDE, is one of these approaches. Different modeling tools, model transformations and code generations are the results of model driven engineering. MDE is also the base for DSL [31] and model transformation.

3.2 Model transformation

Model transformation is one of the rapidly growing and demanded technologies [24]. Many companies and organizations find it enormously useful. Information reusability, tool integration, and model reusability are some of the merits that can be obtained from model transformation. Due to complexity and implementation differences, many researchers have described that the basic challenge to implement model transformation for specific models, which are developed using different modeling tools, is choosing the right model transformation tool [25]. However, the demand for model transformation is increasing due to the high demand of tool integration, configurability feature and information reusability.

3.2.1 ATL

ATL is a popular M2M transformation tool. According to (Jouault et al. incited in Tolosa et al.) [12], ATL model transformation adopts partially declarative transformation approach which merges both declarative and imperative transformations. The hybrid feature enables us to deal with conditions of domain specific scenarios, simpler formal syntax and mathematical foundations. The fact that this tool has declarative behavior helps to have error free code due to the no side effect feature [15].

ATL complements declarative behavior over imperative and vice versa in case of impossibilities. On situations such as impossibility of transformation using declarative language, ATL allows users to use imperative language in order to enable complex transformations. ATL is one among the different model transformation tools which has the capability to automatically create target model using the information represented in source model, source meta-model and target meta-model.

The ATL model, which is basically the main transformer, uses source meta-model, target meta-model and source model as inputs during transformation. The ATL model consists of import, rules and helpers. The helpers mostly describe global variables derived from the input meta-model. These helpers are used later in the rules to implement the real transformation. As it is shown inFigure 4, the ATL model transformation has easy and clear architectural structure. It supports unidirectional model transformation. However, it is possible to implement bidirectional transformation through explicit implementation of both side transformations. In ATL transformation, the source model has read-only access whereas the target model has write-only access. This kind of transformation execution is called “Source-target-execution” [10].

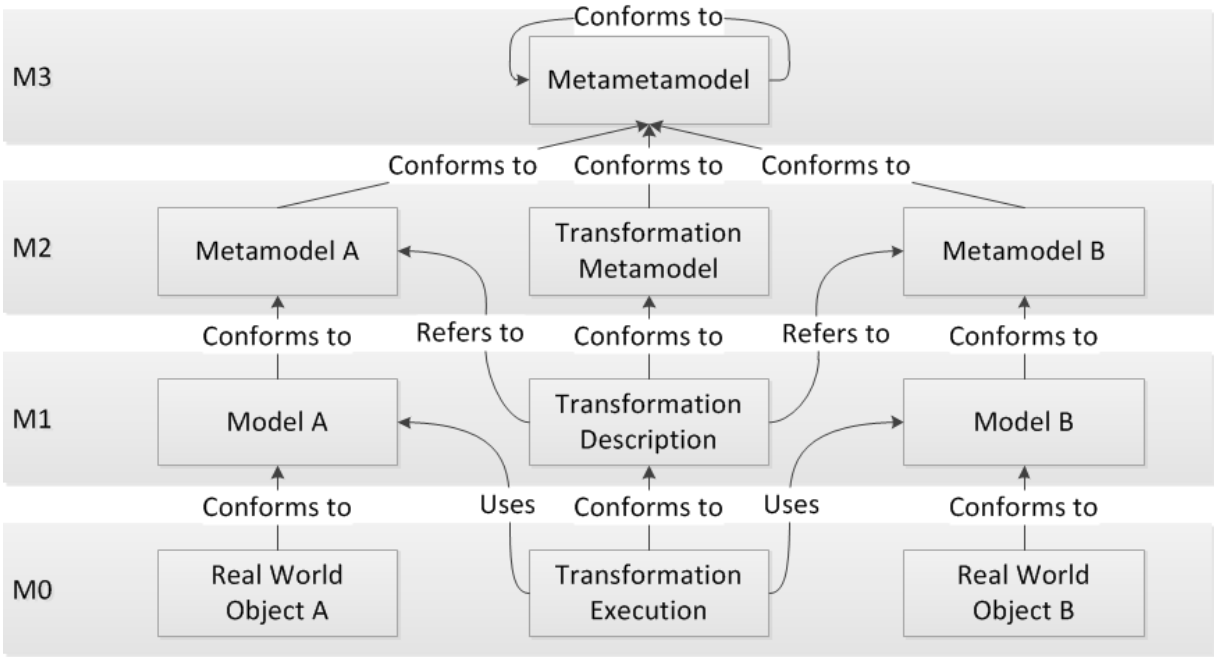


Figure 3: ATL model transformation [4]

According to Gronmo et al. [16]'s study, ATL fulfills the following requirements:

Requirement	ATL
Commonly-used language	No
Inheritance	Yes
Graphical annotation	No
Lexical Notation	Yes
Declarative	No
Bidirectional	No
XML support	No
UML-to-UML	Yes
Text-to-UML	No
UML-to-Text	Yes
UML tool independence	Yes
Meta-model/MOF-Based	Yes
Merged Imperative and Declarative	Yes

Table 1: ATL handling capacity

3.2.2. QVT

QVT has both declarative and imperative nature. It is a M2M transformation tool through Query, View and Transformation. As it is described below these three structures have their own responsibilities.

- **Queries:** are applied to describe the source model as instances of the source meta-model.
- **View:** is the description of the target model by stating what the target model should look like
- **Transformation:** is responsible for the final model to model transformation by making sure the queries are projected on the view, creating the target model.

Language and interoperability dimensions are two orthogonal dimensions of QVT conformance. The valid QVT conformance point is specified by the intersection of these two dimensions. Syntax Executable, XMI Executable and XMI Exportable [26] are the items that should be conformed with the items in Language dimension [39]. Language dimension contains Core, Relations and Operational items [40]. Using this model transformation tool, it is possible to make more than two [16] directions of transformation. However, two directions is the common transformation.

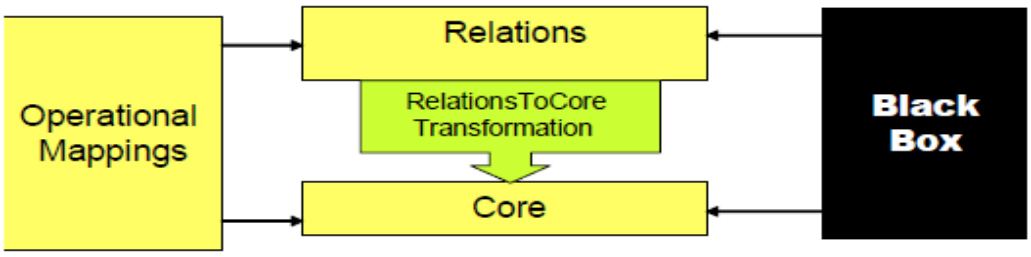


Figure 4 : relationships between QVT meta-models [39]

3.2.3. Kermeta

Kermeta is an executable meta-language for modeling. It is an imperative object-oriented language which is both an executable meta-language and a kernel, upon which to build other languages. Generally, Kermeta is a transformation language which is specifically shaped based on the concepts used in Xion and MTL [27]. Xion is a language which is platform-independent and was originally developed in the context of the Netsilon environment [27] and is used for the development of model-driven web information systems. MTL is an object-oriented model transformation language. It provides APIs to allow manipulating models from various repositories such as EMF, Netbeans MDR [27] in a unified manner.

Kermeta which is distributed as an Eclipse plug-in is an open source project from INRIA. It includes parser, type-checker, and an interpreter. Kermeta provides packages, classes, operations and methods, inheritance and late bindings which can be used to encapsulate transformations. Operation calls or method overloading are two among many of the methods used during successful model transformation. Kermeta is not a transformation language, but it is a general-purpose language that can be used to implement mechanisms to support model transformation [38]. It is an imperative language for modeling, with a basic syntax inspired from Eiffel. The execution is made by an interpreter and the code is statically type checked. Kermeta uses lambda expressions in order to implement and statically type check OCL-like iterators.

The main goal of Kermeta, with its workbench, is to provide a support for all language driven engineering activities. It is typically used to build tools useful to build software, which includes model checkers, simulators, model transformations including model weavers and compilers.

Kermeta includes MOF compliant, Model oriented, Imperative, Object-Oriented, Aspect-Oriented, and is statically typed. In addition to these characteristics, it also includes some model oriented concepts such as associations, multiplicities, or object containment management [38]. Figure 5 shows how Kermeta works. The constraints use to implement the meta-model semantics and QVT is used as part of it to transform the models.

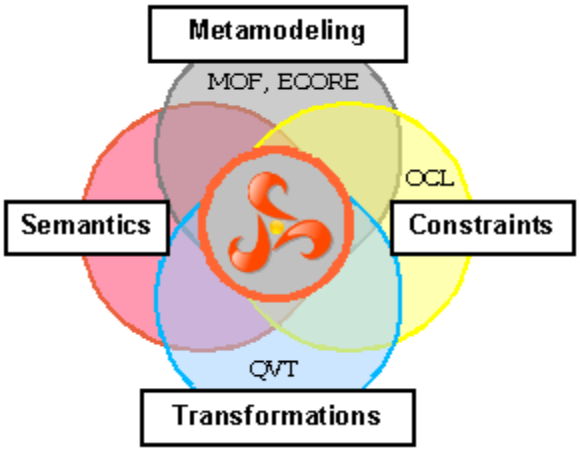


Figure 5: Model transformation using Kermeta [38]

3.2.4. ETL

ETL is a rule-based and task-specific [41] model transformation language which has a declarative signature, hybrid and imperative body expressed in EOL [28]. Each of the programming supports demonstrates particular advantages and disadvantages. Declarative is limited to a simple mapping where the source and target meta-models are similar to each other in terms of structure. However, they fail to solve complex mappings. The imperative transformation languages are capable of solving complicated transformation scenarios. However, they operate at a low level of abstraction which means that the user needs to manually address issues such as tracing and resolving target elements from their source counterparts and orchestrating the transformation execution. It consists of transformation rules that can translate elements in the source model into elements in the target model and can query/navigate/modify both source and target models. ETL, as a transformation language, has its own features as listed below.

- Transform many input to many output models
- Ability to query/navigate/modify both source and target models
- Declarative rules with imperative bodies
- Automated rule execution
- Lazy and greedy rules
- Multiple rule inheritance
- Guarded rules

3.2.5. Dually

Dually is a model transformation framework used to create interoperability among ADLs themselves as well as UML [29]. It has only two levels of abstraction which are M1 and MM1. As it is shown in Figure 3, each model M1 conforms to its own meta-model MM1. The transformation is then from concepts of architecture M1 into semantically equivalent architecture model M2. The transformation made using dually allows software architect [29] to generate the target model from the source model which conforms to their respected meta-models.

Similar to the rest of model transformations mentioned in this thesis work, Dually is also an Eclipse plug-in. The available version of dually is based on ATLAS Model Management Architecture (AMMA) and it extends the ATLAS Model Weaver.

3.2.6. oAW

oAW is a very powerful framework [19] for MDA and MDD, which is suitable for generating code, transforming and checking models. This framework is also known as Xtend/Xpand/Check. A validation language CHECK, which is equivalent to OCL, is provided by oAW for specification of tests. Each meta-class is checked by the constraint and then follows the keyword WARNING or ERROR. These keywords specify what kind of action should be taken in case the constraint fails. ERROR is executed if the constraint fails, the specified message is printed and all further process is cancelled. WARNING is executed if the constraint fails, the specified message is printed and the workflow execution continues working. After the models are tested using validation language, it is possible to generate code, and to transform model.

a) Workflow:

The workflow is a simple xml-based configuration language based on the DI with which all kinds of generator workflows can be described. It consists of workflow components that are executed sequentially in a single JVM. A Workflow component represents a part of a generator process such as model parsers, model validation, model transformers and code generators. oAW is shipped with different workflow components which can be used where suitable, but it is also possible to implement your own workflow upon the requirements. It is possible to recognize the output based on its ID in the log. This can be made by implementing the workflow interface called "*WrkflowComponentWithID"* and "*setID()"* operations.

b) Check

Check file has .chk extension and is full of guard sentences. The check file makes sure whether the constraint on the meta-model is fulfilled by the model or not. First, the meta-model is imported and then the context is specified for which the constraint applies. After this, the name of the meta class that is going to be checked by the constraint should be called; then the ERROR or WARNING keywords are specified to define what kind of action should be taken in case the constraint fails.

c) Xtend

Xtend is a language delivered with oAW. The file resides in the java class path of the used execution context and its file extension is .ext. The actual M2M transformation is realized through this language.

3.3. Model Transformation Languages Metrics

ATL	**QVT**	**Epislon**	**Xtend**	**Kermeta**	**oAW**
Helpers	Mappings	Rules	Extensions	Rules	CHECK
Rules	Helpers	Pre/post blocks	check	Query	workflow
Ecore and XMI dependent	XMI and Ecore dependent	Xmi dependent	Xml, uml, xmi etc	XMI and Ecore dependent	XML, XMI, XML Schema, UML

Table 2: Metrics of different transformation languages

4 Problem Statement

This research study is intended to solve a problem of model transformation in the field of MDE. In order to have clear understanding of the problem, this chapter defines the problem, discusses the list of research questions that need to be answered to solve the problem, and the research methodology that is designed to answer the questions.

4.1. Problem Definition

Each tool has its own inherent concepts. Due to this reason, SW, as a tool, has its own syntax and semantics. The SW based implementation of EAST-ADL is represented in SWxml, which is structurally different from EAXML. EAXML is an AUTOSAR-based representation of EAST-ADL. This model is available in the XML Schema format. These two xml files are called model files. Since models are the basic artifacts of system development, it is important to use model transformation so that a model represented using different semantics and syntax can be used by different tools of different concepts.

Specifically, this study is concerned with finding suitable model transformation technology for Two-way transformation between EAXML and SWxml. The research has to be proved through the implementation of a model transformation prototype. Identifying the differences between these two EAST-ADL implementations is thus also an objective of this research study.

Based on the maenad project [32], implementation of this bidirectional model transformation is one of its objectives. However, this thesis work, which is undertaken at Systemite, is the first study for the SW based implementation.

SW, MetaEdit+, and Papyrus are tools where EAST-ADL is implemented [4]. In order to have high interoperability [17] and reusability [4] of models, these tools should be integrated to one another through the common EAXML. For this transformation to fulfill the requirements, the import and export of newly transformed model is verified through the prototype transformation. Generally, these two models are represented in different levels of abstraction. SW uses Simple and Complex structures to implement EAST-ADL.

Simple Part Structure

The concept of items and parts in SW is useful for implementing most part of the EAST-ADL by capturing all the needed information. This structure has a simple conceptual mapping between the target syntax and semantics offered by SW. The usual case for this structure is when there is no distinction between the instance of a type and the type itself. That is the instance holds no additional information compared to the type. FeatureGroup and childFeatures are part of EAST-ADL implemented using simple structure.

Note that some of these cases could include minor complications, like a type that should only occur in a single instance, but these are usually due to limitations of the concepts of the target meta-model (confusion between types and values, like class properties and object properties, or when real life use cases have not been considered).

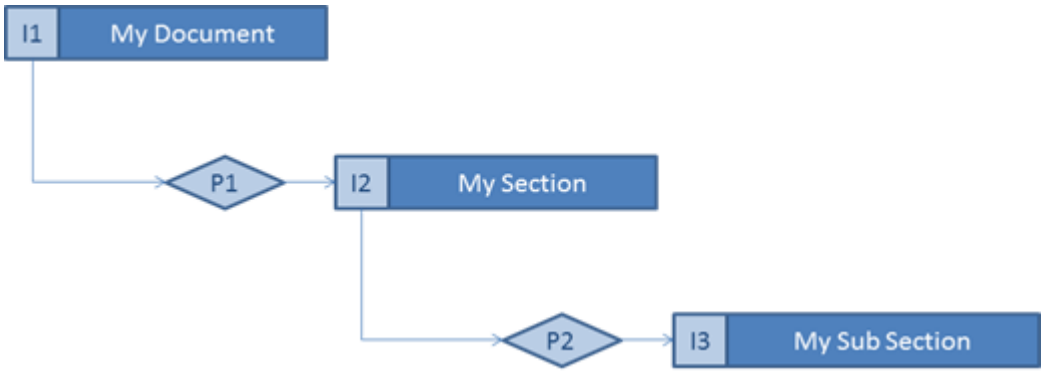


Figure 6: Simple structure

Complex Structures - are used when simple structures are not enough to represent a system, like electronic architectures, where the same component may appear multiple times in the same architecture and when references are needed deep into a specific structure. Example: The **Type/Prototype** pattern structured models in general in SW

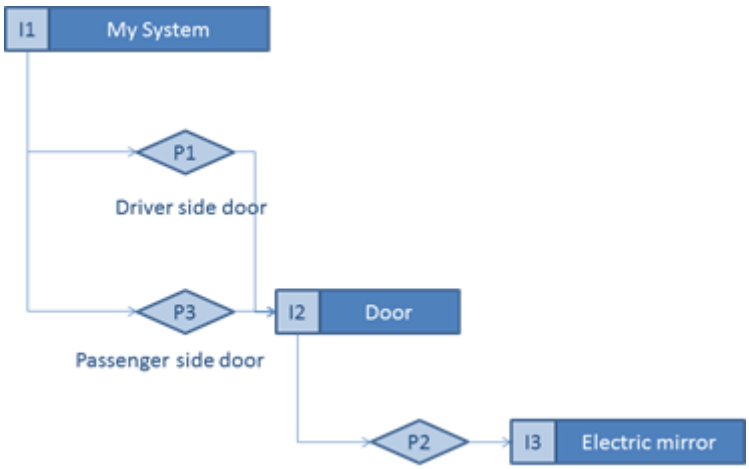


Figure 7: Complex Structure

Keeping the SW-based EAST-ADL implementation in mind, the research based solution should be able to include the following outputs:

1. Propose suitable model transformation language
2. Bidirectional M2M transformation between EAXML and SWxml
3. Reliable or easy upgrade of import/export feature
4. Generation of important error message as part of import or export
5. Support incremental import/export according SW exchange cases, versions, support for automation of transformation in one or both directions, given a defined mapping

4.2. Research Questions

For easy understanding of the problem, we have stated different research questions that we have believed answering them solves the problem. Considering the above problem definition, we designed the following research questions that are answered through the research study.

RSQ1: What are the structural and architectural relationships between SW and EAXML?

Since the main objective of this thesis work is to select suitable model transformation technology which solves the defined problem, the first thing that must be examined is the structural and architectural relationships between these two implementations.

RSQ2: How is it possible to combine the views of these two implementations and MDT to realize MDE?

Model transformation is a view combination of the input and the output meta-models. Due to this reason examining the possibilities of view combination to realize MDE is vital.

RSQ3: What are the available M2M transformation technologies that are applicable for the transformation of SW and EAXML and how can we select the suitable one?

Identifying the available model transformation technologies gives the chance to choose one transformation technology, which is final solution to solve the problem. The available M2M transformation languages must be tested against the high level requirements. However, this does not mean only the high-level requirements are the only criteria that have to be fulfilled to select the suitable one among the existing model transformations.

RSQ4: How can we achieve bidirectional transformation between SW and EAXML?

We aim to formulate the architectural patterns between SW and EAXML. Therefore these patterns are used to drive the transformation specifications. These specifications are used as test cases for the bidirectional transformation.

RSQ5: How to verify the selected M2M transformation language of the transformation between SW and EAXML?

There are many ways to transform models. The feasibility of the selected transformation technology has been evaluated to determine whether it is the right solution or not. For example: if a M2M transformation technology can only fulfill the high level requirements but not the detailed requirements, then the solution is considered as invalid.

RSQ6: How is it possible to realize tool integration through the chosen model transformation technology?

According to the maenad objectives the main purpose of conducting studies on model transformation is to further realize tool integration.

4.3. Research Methodology

We have designed the research in four stages called Problem Analysis, Solution Design, Solution Validation and Evaluation of the Prototype. We have adopted this approach from the research and design methodology principles proposed by (Wieringa 2008; Wieringa 2009). The overall research methodology design is shown in Figure 8.

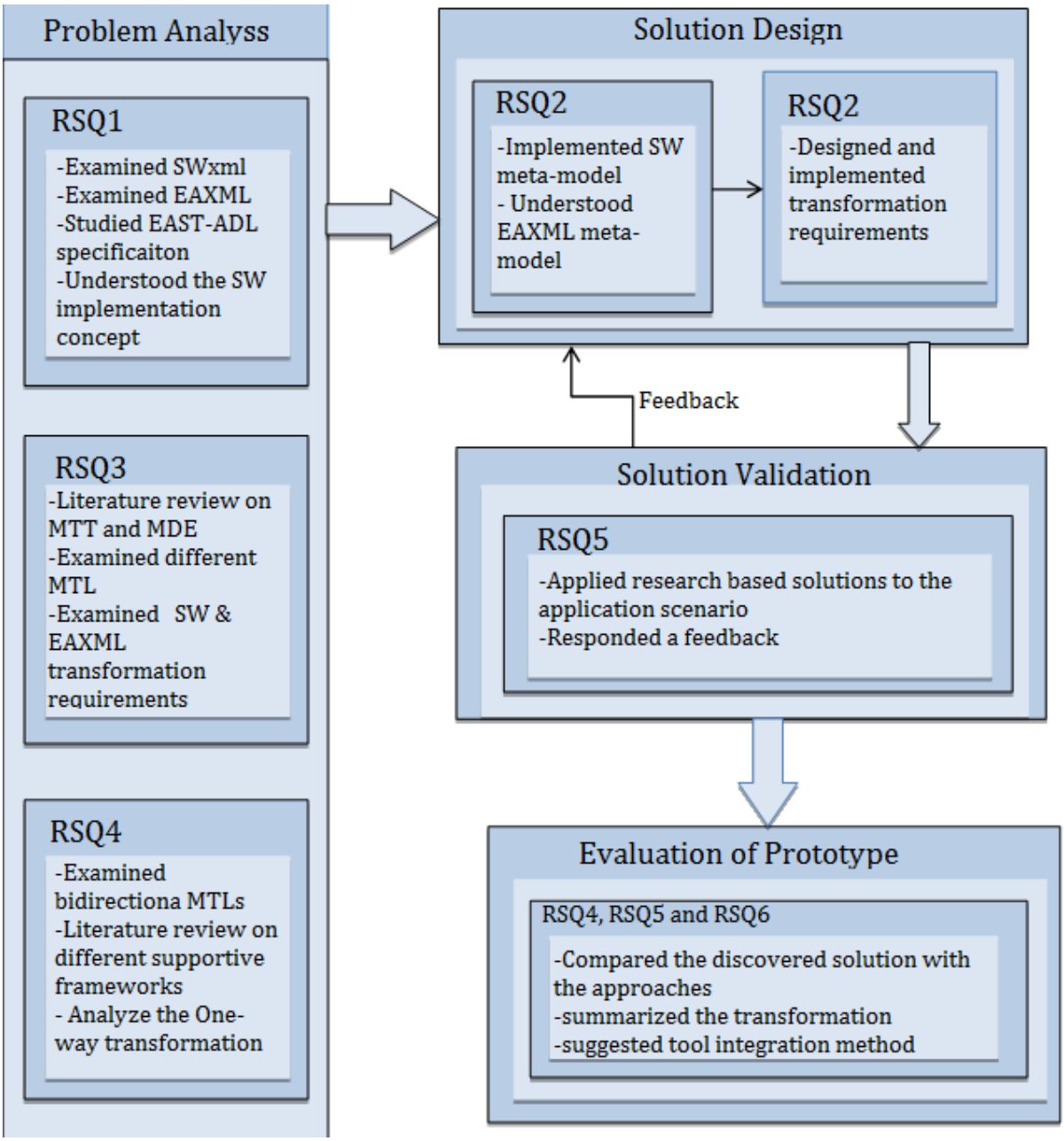


Figure 8 : Research methodology

Problem Analysis: In order to answer the research questions, we have reviewed different related research publications and examined the models which are represented in XML and XML Schema files. Since there is no available documentation about SW made in Systemite, Mr. Södberg has been used as the main source of the required information.

Solution Design: The solution is made by developing both SWxml and EAXML meta-models. Then we designed the views and derived the transformation requirements.

Solution Validation: in this stage we checked the research based solutions to the transformation requirements. According to the transformation requirements implementation

made on solution design, we have checked if the solution can still be valid and sent feedback to the solution design. If the validation fails then we get back and check the design and implementation of transformation requirement and for different transformation language. If the validation succeeds then we continued with the next stage.

Evaluation of prototype: at this stage we have implemented a prototype and compared the actual solution with the expected one. Then after getting the expected result, the research is summarized.

5. SWxml and EAXML

In this chapter, the differences between SWxml and EAXML are analyzed.

The transformation for the class feature of the EAST-ADL specification has been implemented using ATL model transformation. The feature modeling in EAST-ADL is used to illustrate the differences and similarities between SWxml and EAXML. Section 5.1 discusses the differences between Swxml and EAXML and Section 5.2 discusses the different meta-model representations views of SWxml and EAXML.

5.1. SWxml vs EAXML

Taking feature modeling from EAST-ADL specification as an example, we have described the differences among EAXML, SWxml and EAST-ADL. We have used EAST-ADL as a common specification to analyze the differences between SWxml and EAXML. The feature modeling in EAST-ADL is shown in Figure 9.

Since there was no model transformation or similar works done in Systemite, this is the first study conducted on this area. There are no artifacts that tell the differences between these two implementation concepts. To compromise the lack of information on this area, EAST-ADL specification is used to increase understanding, during conducting an analysis between SWxml meta-model and EAXML meta-model.

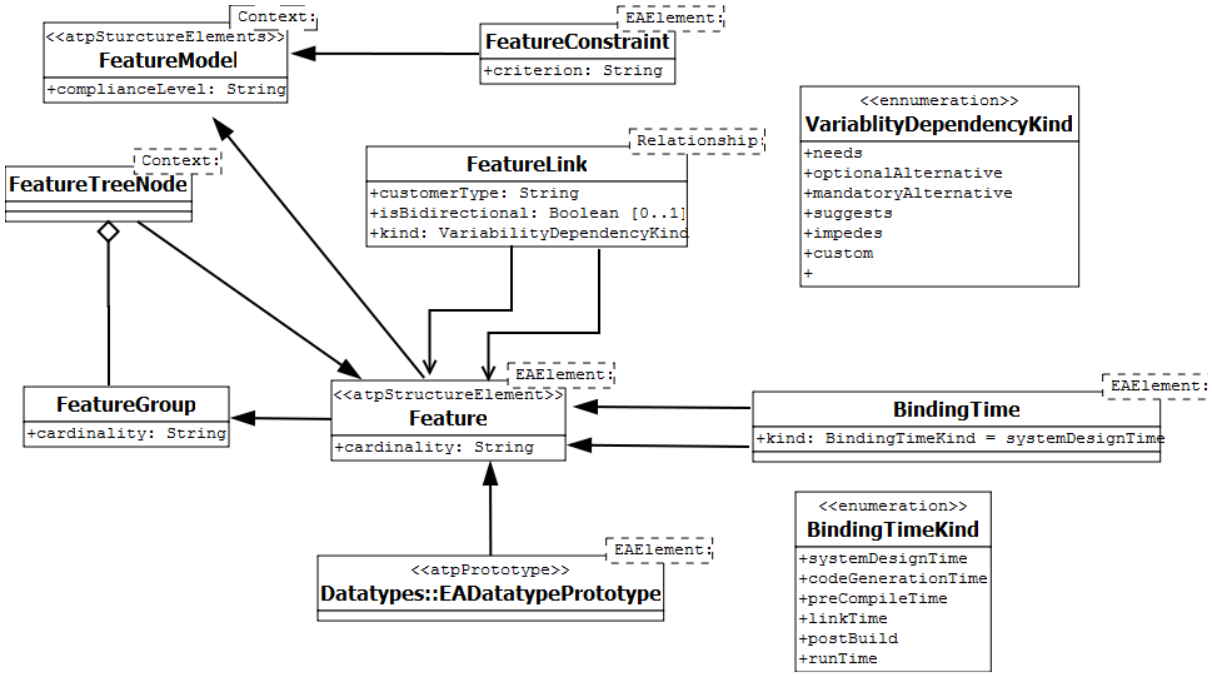


Figure 9: Feature modeling from EAST-ADL Specification

As it is described above the implementation differences for feature modeling is analyzed. To check the analyses look on Table 3. A total of two types of analysis are made prior to the model transformation technology investigation. The first type of analysis is conducted to identify the naming differences. The second type of analysis is conducted to identify more of structural differences. According to the analysis conducted we divided the differences as Simple and Complex ones.

Simple: this is mostly naming difference. When a class in EAST-ADL is named differently in one or both of the implementations, then we call it simple difference. For example, the FeatureTreeNode class in EAST-ADL is named as childNodes in EAXML, such kind of differences are naming differences.

Complex: these differences are mostly structural differences. If a reference or attribute is implemented in different classes than the EAST-ADL specification, we call it complex difference. These kinds of references need special attention during transformation. We said it needs special attention because ATL traces references during transformation. If these kinds of differences are not recognized well, then it is less probable to get the transformation working correctly. SW and EAXML meta-models are available in different level of abstraction. Due to this reason the representation of their model is different as it is shown on Figure 10 and Figure 11.

EAXMl		EAST-ADL	SW
ChildNode		FeatureTreeNode	ItemType of name childNode
Feature		Feature	ItemType of name Feature
Featureparamter		featureParameter	ItemType of name featureParameter
EaDatatypePrototype		atpPrototype	ItemType of name atpPrototype
BindingTime		BindingTime	ItemType of name BindingTime
Class	Reference		
ownedRelationship	featureLink	featureLink	ItemType named featureLink
	Include		ItemType-ownedRelationShip
	multiLevelReference		ItemType: include
	Realization		ItemType : multiLevelReference
	Refine		ItemType: refine
	requirementsLink		ItemType:requirementsLink
	Satisfy		ItemType: satisfy
	Verify		ItemType :verfy
	Extend		ItemType: extend
featureGroup		featureGroup	ItemType with name featureGroup
EAXMl		EAST-ADL	SW
ChildNode		FeatureTreeNode	ItemType of name childNode
Feature		Feature	ItemType of name Feature

Featureparamter		featureParameter	ItemType of name featureParameter
EaDatatypePrototype		atpPrototype	ItemType of name atpPrototype
BindingTime		BindingTime	ItemType of name BindingTime
Class	Reference		
ownedRelationship	featureLink	featureLink	ItemType named featureLink
	Include		ItemType-ownedRelationShip
	multiLevelReference		ItemType: include
	Realization		ItemType : multiLevelReference
	Refine		ItemType: refine
	requirementsLink		ItemType:requirementsLink
	Satisfy		ItemType: satisfy
	Verify		ItemType :verfy
	Extend		ItemType: extend
featureGroup		featureGroup	ItemType with name featureGroup

Table 3: Feature modeling implementation analysis

EAXML Model -Figure 10Figure 11 shows the XMI user model of the meta-model EAXML.

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<eastadl:Elements xmi:version="2.0"
  xmlns:xmi="http://www.omg.org/XMI" xmlns:eastadl="eastadl">
 <feature category="Feature" name="Feature" cardinality="1..2" uuid="x0000001D04000027">
  <uaTypeRefs>
   <uaTypeRef value="The correspondence in SystemWeaver is not found" dest="USER-ATTRIBUTE-ELEMENT-TYPE"/>
  </uaTypeRefs>
  <uaValues/>
  <traceableSpecificationRefs>
   <traceableSpecificationRef dest="EA-STRING"/>
  </traceableSpecificationRefs>
  <actualBindingTime name="actualBindingTime" uuid="x0000001D040135F9"/>
  <requiredBindingTime name="requiredBindingTime" uuid="x0000001D040135F6"/>
 </feature>
</eastadl:Elements>
```

Figure 10: Sample EAXML user model

SW Model: Figure 11 shows the equivalent XMI format of the SWxml model

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<xmi:XMI xmi:version="2.0"
  xmlns:xmi="http://www.omg.org/XMI" xmlns:SWVR="SWVR">
 <SWVR:Items>
  <item name="Feature" partGroups="/1" topNode="/2" ancestor="x0000001D04000027"
    id="x0000001D04000027" sid="ss0">
   <versionText/>
   <description>
    <binary/>
   </description>
   <attributes/>
  </item>
  <item name="requiredBindingTime" topNode="/3" ancestor="x0000001D0400002A" id="x0000001D0400002A"
    sid="ss2">
   <versionText/>
   <description>
    <binary/>
   </description>
   <attributes/>
  </item>
  <item name="actualBindingTime" topNode="/4" ancestor="x0000001D0400002E" id="x0000001D0400002E"
    sid="ss2">
   <versionText/>
   <description>
    <binary/>
   </description>
   <attributes/>
  </item>
 </SWVR:Items>
 <SWVR:PartGroups>
  <partGroup name="requiredBindingTime" sid="ss05">
   <parts>
    <part name="requiredBindingTime" ancestor="x000000051500002C" id="x000000051500002C">
     <defObj id="x000000051500002C"/>
     <attributes/>
    </part>
```

Figure 11: Sample SW object model

5.2. Meta-Model Representation Views

As long as both source and target meta-models are the basis for the transformation rules, understanding and implementing the meta-models in the correct way is crucial task which has to be done prior to the model transformation language investigation and model transformation implementation prototype.

Throughout the research study we have used GMF, EMF, and XML Schema to understand and analyze the meta-models. Using different representation frameworks helps us to analyze the architectural relationship between SW-MM and EAXML-MM. Each tool and framework used to analyze the meta-models is described in this section.

EAXML Meta-Model

EAXML meta-model is an explicit implementation of EAST-ADL. Every class is modeled separately. If we look at the EAXML meta-model for the feature-modeling of EAST-ADL, we have

found all the classes modeled separately. This makes the meta-model large. Including the AUTOSAR classes, EAXML contains a total of about 600 classes.

XML Schema: Originally the EAXML is available in XML Schema as it is shown in Figure 12.

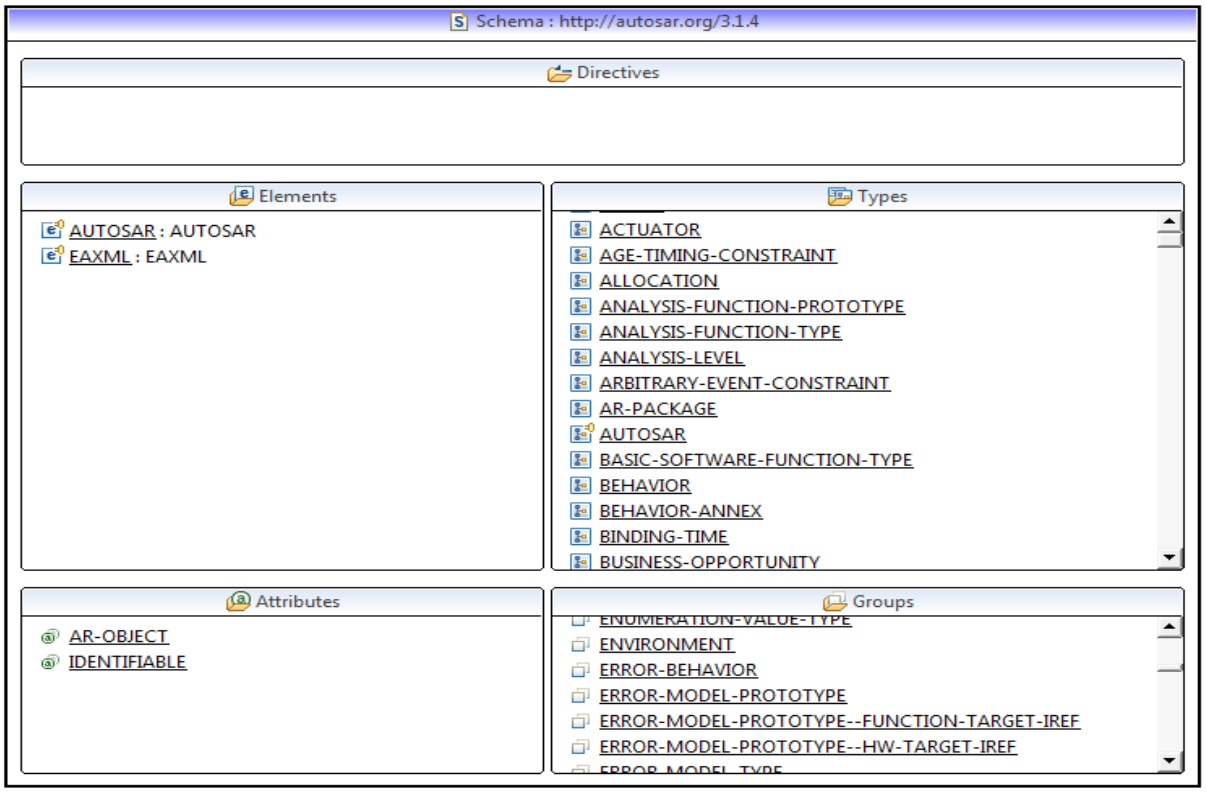


Figure 12: EAXML in XML Schema format

EMF: The EAXML meta-model is also available in EMF (Ecore) format. This representation is easier to understand than the Schema. As it is shown in Figure 13, the classes and references can be viewed easily.

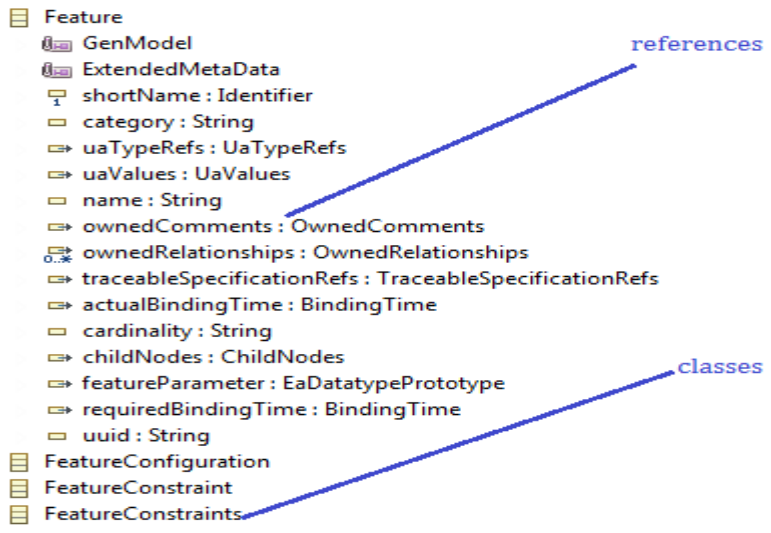


Figure 13: EAXML meta-model in EMF

GMF: This framework is used to represent meta-models in graphical representation. This representation is the easiest way to understand the meta-models. The GMF representation of EAXML meta-model is shown in Figure 14. However, it is limited to the size of the graphical displayer used to view the meta-model.

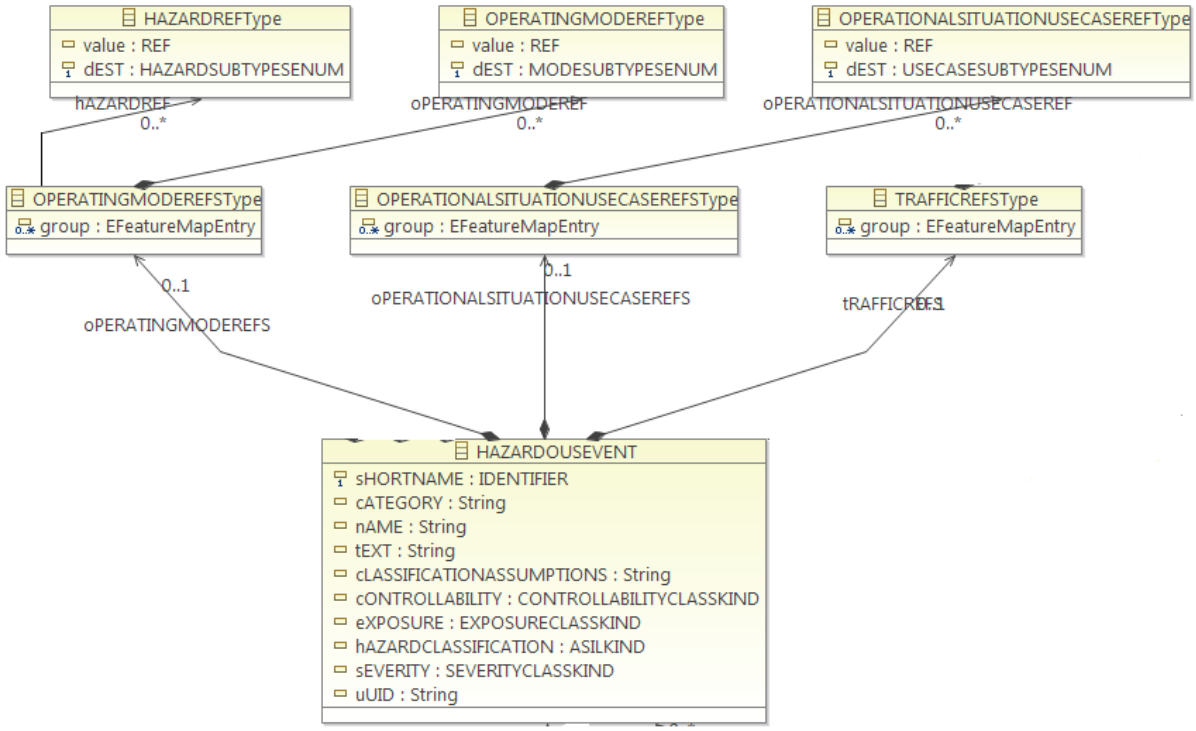


Figure 14: The partial view of EAXML using GMF

SW Meta-Model

This meta-model is available in four formats called XML, XML Schema, EMF and GMF. As it is shown in Figure 15, SW meta-model is implemented in an implicit way using ItemTypes, PartTypes, ObjTypes, and AttributeTypes. SWxml is originally available in XML file format.

```
<?xml version="1.0" ?>
- <SystemWeaver3>
  - <Info>
      <Description>feature</Description>
      <ExportedBy>admin</ExportedBy>
      <ExportDate>2011-03-23</ExportDate>
      <ExportedItem>x0000001D04000027</ExportedItem>
    </Info>
  - <Meta>
    - <Types>
        <ObjTypes>
      + <ItemTypes>
      + <PartTypes>
        </Types>
    + <AttributeTypes>
    </Meta>
  + <Items>
  </SystemWeaver3>
```

Figure 15: SW meta-model in XML format

XML Schema: The Schema of SW is created as a result of XML conversion to XML Schema using Oxygen. For easier understanding and further model transformation, it is mandatory to transform the

XML meta-model to its Schema. The XML Schema of SW is much more difficult to understand than the other formats. The newly created schema from SWxml is shown in Figure 16.

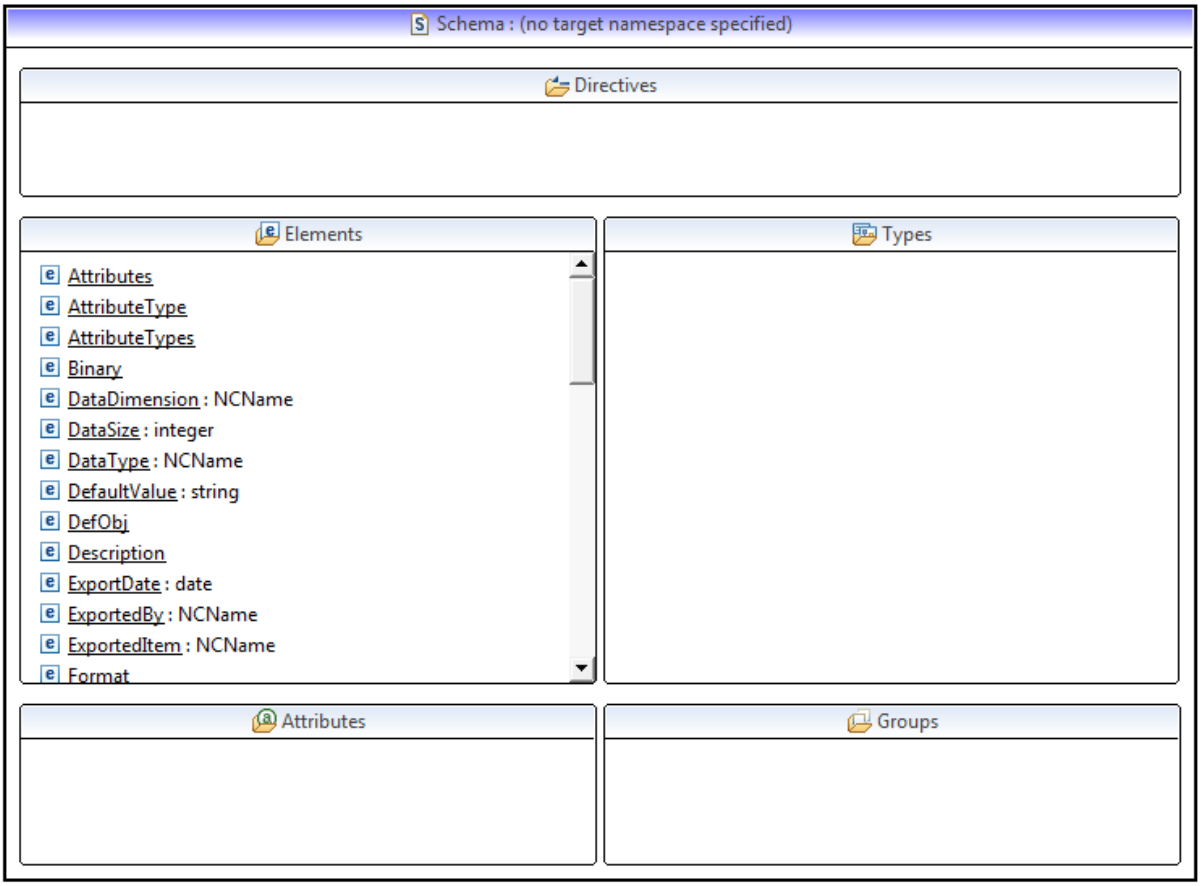


Figure 16: SW in XML Schema format

EMF: The SW Meta-model is also available in EMF (Ecore) format. The Eclipse Modeling Framework is used to transform the SW Schema to its EMF based meta-model. This representation is easier to understand than the Schema. It also used as input to the M2M transformation. As it is shown in Figure 17 the classes, attributes and references can be seen easily.

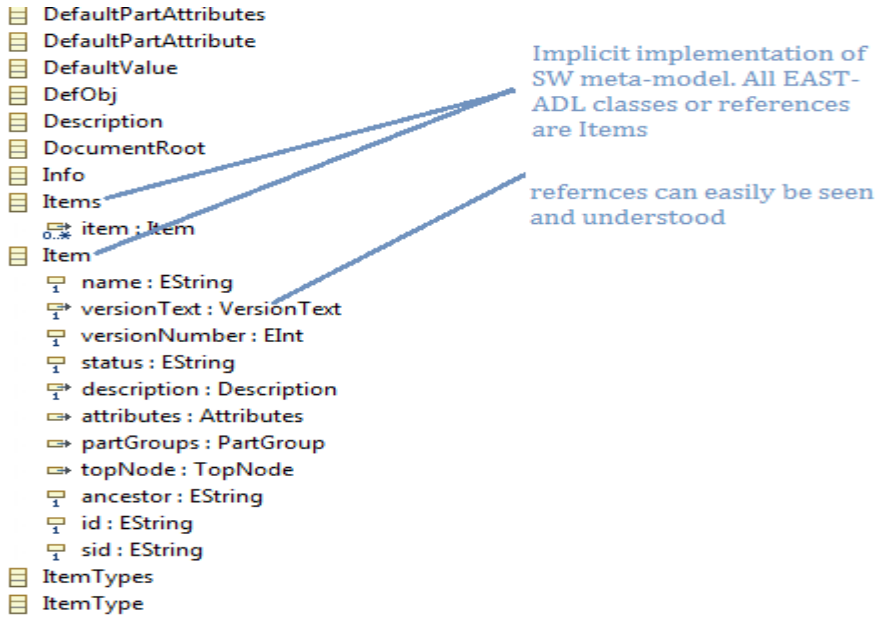


Figure 17: SW met-model in EMF format

GMF: Using GMF, meta-models can be represented and viewed graphically. This representation is the easiest way to understand the meta-models. It is because GMF enable us to see all the classes, attributes and references easily. However, this does not mean it does not have any drawbacks. The level of graphics that can be seen at a time depends on the size of electronic visual display. The GMF representation of SW meta-model is shown in Figure 18.

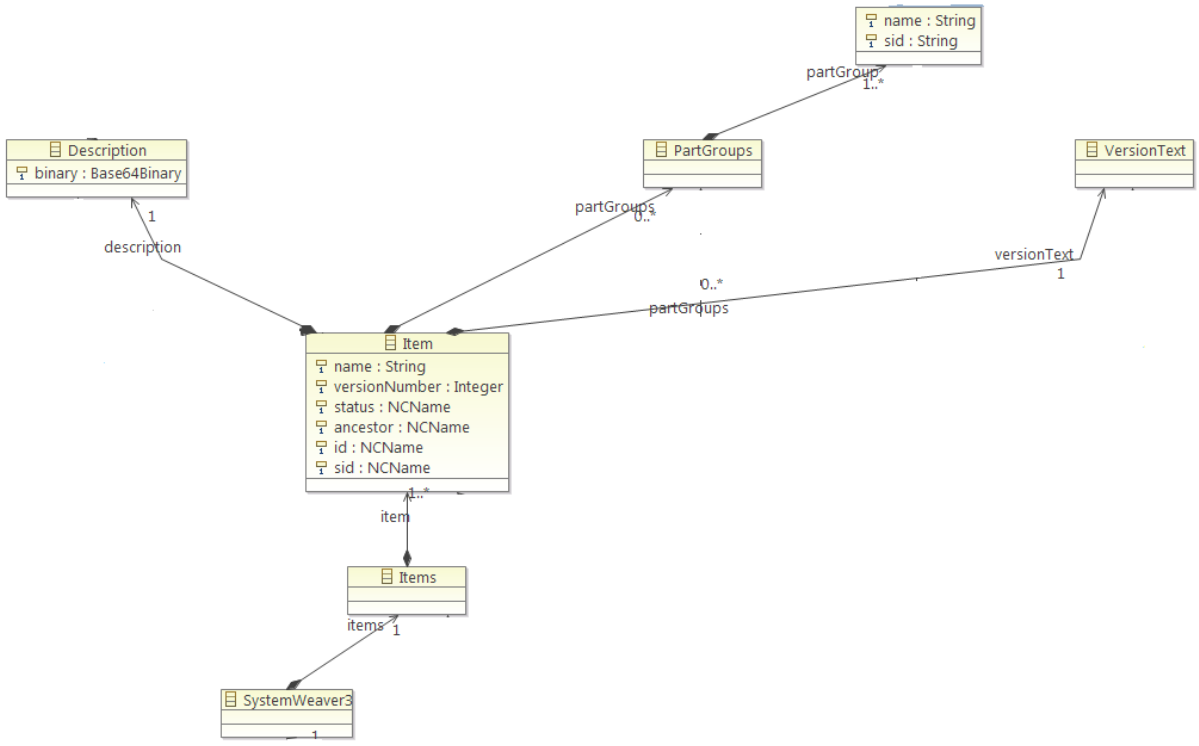


Figure 18: The partial view of SW meta-model in GMF

6. The Model Transformation

In order to solve the problem, we have investigated different transformation languages in the field of MTT. We investigated 6 transformation languages called **oAW**, **Dually**, **ETL**, **Kermeta**, **QVT**, and **ATL**. However we have found two of them suitable languages for the transformation between SWxml and EAXML. Since one language must be chosen we evaluated ATL and oAW against all the requirements and selected ATL to perform the transformation between SWxml and EAXML.

This chapter describes the investigation, and research evaluations of the research solution through prototype implementation. Section 6.1Model Transformation Languages Evaluation describes the evaluation conducted between ATL and oAW. Section Prototype Implementation describes the prototype implementation including the rules derived to complete the transformation. Section 6.3 Contribution of the Solution discusses the contribution of the study.

6.1. Model Transformation Languages Evaluation

In MDE, the scope of model transformation technologies is to create target model from source model**.** Based on the type of models and meta-models that need to be transformed, model transformation languages have their own weaknesses and strengths [25]. All transformation technologies have their own specific application. One transformation technology can be used to implement horizontal transformation, vertical transformation, code generation or M2M transformation. This shows that model transformation technologies have their own applications and handling capacities. Therefore one among the reasons why this investigation is required is to discover the transformation technology which is suitable to implement the transformation between SWxml and EAXML. This shows that no transformation technology is better than the other. Because of this the best way to select the suitable technology is to set the conditions and requirements that need to be fulfilled for the transformation between SWxml and EAXML to succeed. The following high level requirements have been set as criteria to investigate the suitable model transformation technology for the transformation between EAXML to SWxml.

1. The chosen transformation language should support XML or XML Schema.
2. The chosen transformation language shall produce XML model
3. The output model of EAXML to SW transformation should be imported to and exported from SWExplorer.
4. The transformation language shall handle both complex and simple structures of SW.

6.1.1. ATL

ATL [1], which is an Integrated Environment Developed on top of Eclipse platform, provides syntax highlighting, debugger and is mostly used in MDE [2].

Using MDE concept, software specification should be translated into executable programs automatically [12]. For this reason, the source code is represented as an executable text model which conforms to its meta-model. ATL is one of the tools which adequately enable us to transform models in a horizontal transformation approach [12].

ATL includes three kinds of units called modules, library and queries. Each unit has its own functionality as follows:

- **Module**: corresponds to the classical ATL transformation, which contains header, helper and rules.
- **Query**: enables us to return a primitive value from a model.
- **Library**: defines a set of ATL functions. The library model mostly includes general definitions which are used all over the transformation.

During M2M transformation, the source model, source meta-model, and target model and target meta-model are configured in a certain hierarchies as it is shown in Figure 20. As it is shown in Figure 20, the source and target models conform to their respective meta-models. The ATL transformation model uses the source meta-model, target meta-model and source model as inputs.

Even though in [43], endogenous and exogenous are specified as types of transformations, we have used these terminologies to define the relationship among models. The relationship between SWxml and EAXML is categorized as exogenous as it is defined below.

- **Exogenous models:** are models implemented using different tools. If the source model is implemented using a different tool than the target model's tool, then the transformation between these two models is called Exogenous. Since SWxml is implemented using SW and EAXML is implemented using UML 2.0, they are called exogenous models. Respectively the transformation between SWxml and EAXML is an exogenous transformation.

- **Endogenous models:** are models developed using the same tool. The transformation between such models is called endogenous transformation. Models which are implemented using UML are called endogenous.

We have implemented the SWxml meta-model using Eclipse Modeling Framework. During our implementation, we have discovered that there are many methods to implement EMF based meta-models and we have divided these methods into two approaches.

Approaches used to create EMF based SWxml meta-model

i. XML Schema Mapping

This approach is used to automatically create meta-models from their XML Schema through mapping. The meta-model classes, references, attributes and data types are mapped to EMF Eclass, Ereference, Eattribute and EMF data types.

SWxml is available in XML file format whereas EAXML is available in XML Schema. EMF supports direct mapping from Ecore, UML, Rational Rose and XML Schema. Therefore SWxml meta-model cannot be mapped to its respective EMF based meta-model directly. It is vital to generate the SWxml's schema prior to the mapping. In order to generate the SWxml Schema from the SWxml, we investigated XML Spy, and Oxygen. As the result of the investigation, we have found Oxygen as the tool to support the transformation between XML and XML Schema. We then generated the EMF based meta-models from the newly generated XML Schema.

However, the EMF based meta-models generated through mapping has to be reviewed before use. Because of the information loss such as multiplicity, we have found it important to review the meta-models and validate them before use.

To implement SWxml EMF based meta-model, we have used this approach. However, we re-implemented the meta-model to make sure that the information loss is correctly fixed. The partial view of the EMF based SWxml meta-model which we have implemented is shown in Figure 17.

a. Merits of using XML Schema mapping approach

This approach minimizes the resources used to implement meta-models. The resources are minimized through the automatic mapping from the XML Schema to Ecore meta-model. Since the EMF meta-models drive the ATL transformation, the availability of these meta-models is vital. It is of great advantage to use this approach if the meta-models are not available in a clear documentation. Mapping the meta-models from XML Schema increases the understanding of the meta-model.

b. Limitation of using XML Schema mapping approach

During XML Schema to Ecore mapping, some of the XML Schema specific constructs are not represented directly in the created Ecore; such information is added to Ecore using Ecore annotation called Extended Metadata. This API is provided by EMF to query, modify or create the additional metadata. Losing important information such as multiplicity leads to dramatic change in the meta-model. Because of the loss of such information the transformation based on such meta-models might not even work as it is intended. However, modifying the meta-model manually can solve the problem. We have fixed the information loss issue on mapping XML Schema to Ecore according the explanation given in [11].

ii. Manually implementing a meta-model

This approach can be used if the meta-models are available in a clear model diagram or other representation other than xml or XML Schema. It is possible to write a meta-model using Ecore or GMF manually. For this approach to succeed, it is important to use three to four eclipse based frameworks called ATL, AM3 and EMF or GMF.

Generally, this approach is more tiresome and time consuming than the first approach. The development cost is almost double of the first approach. The reason is that if there is an automated way to directly map the existing meta-model to Ecore meta-model, the time and cost decrease. However, in case of this approach the complete meta-model has to be implemented manually using EMF or MOF.

Since the available meta-models are not presented in a clear model diagram, we have not considered this approach as an option. However, since this approach is a piece of work to produce correct meta-models, we have found it important to include it in the report. Generally, there are many ways that Ecore meta-models can be implemented; the following listed methods are some among the many ways to generate Ecore meta-model from different formats.

- **Injecting it from km3** - km3 is a model file where meta-models can be implemented through coding. After implementing the meta-model it is possible to generate its

respective EMF based meta-model by using a functionality called inject which is provided by ATL.

- **Manually implementing Ecore meta-model**– creating Ecore meta-model using this method is as tiresome as Injecting from km3. However, this is a good approach to develop the correct meta-model.

As it is shown in Figure 19, there must be defined source and target meta-models which can be used as a basis and input for the transformation rules. These two meta-models are used by ATLModel during the transformation.

During the transformation, EAXML and SWxml are the target and source models, whereas EAXMLMM (EAXML meta-model) and SW-MM (SWxml meta-model) are the meta-models. Each model conforms to their respective meta-models and the meta-models conform to their meta-meta model.

Bidirectional M2M transformation is achieved through an explicit One-way transformation. During EAXML2SW transformation the resulted model need to be imported to and exported from SWExplorer. The import or export functionalities of SWExplorer works only if the resulted model is represented in the correct XML file format. Therefore as the result of this transformation an XML model file must be created as a target model. Unfortunately, ATL does not have the ability to take an XML model as input or to create an XML model file as an output. Taking the high level requirements mentioned in section 6.1 into consideration, we investigated different frameworks that support to make the transformation complete. We used AM3 to inject and extract SWxml. The inject functionality supports ATL by creating the Ecore equivalent of the SWxml. The extract functionality supports ATL by creating the xml equivalent of SW.ecore. However, this does not mean a single transformation makes the import and export possible. The Inject and Extract functionalities, provided by AM3, lead to extra transformations named as Complementary transformations.

Generally, in order to achieve the transformation using ATL, three different eclipse frameworks called GMF, EMF, and AM3 are used as auxiliary frameworks. GMF is used to represent and view the meta-models graphically and makes it easier to understand the differences and similarities of these two meta-models for further transformation. EMF is used to create the meta-models from their respective Schemas. AM3 is used to handle the model file format issue.

The transformations used for this study are divided in to two categories called Main and Complementary transformations. The Main transformation is the transformation between EAXML and SWxml meta-models and the Complementary transformation includes the rest of the transformations made in order to complement the lacked features of ATL. These two different types of transformations are described below. According to J.E. et al [34], series of transformations to achieve one transformation is called composite transformation.

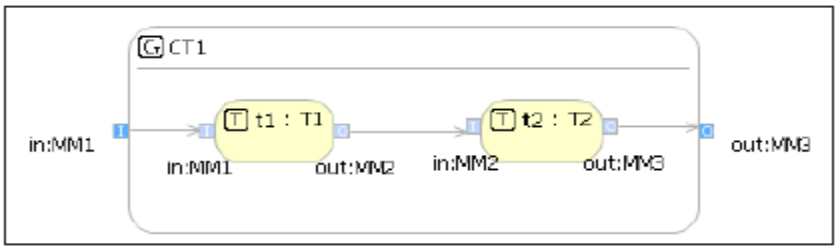


Figure 19: Composite transformation (34)

We have used the same legends to illustrate the different transformations made during the implementation of the bidirectional transformation. To avoid repeat definition of the legends, we have defined them prior to the figures.

Legends used to illustrate the transformation

- **Inject –** is a function which is provided by the AM3 and is used to transform .xml model file format to .ecore model format which conforms to the general XML model.

- **Extract-** is a function provided by AM3 and is used to extract .ecore model file to its specific purpose .xml format.

- **ConformsTo –** This shows that the model is constructed based on its meta-model's semantics construction. During ATL transformation the models are always checked against their meta-models.

- **Uses –** is a terminology used to show that the ATL file uses the correspondent input and output meta-models.

The Types of Transformations Used

I. Main Transformation

The main transformation is considered to be the main part of the transformation. This transformation is the critical one. Complementary transformation is used to make the Main transformation to be complete solution. The Main transformation covers the transformation between SWxml and EAXML meta-models and is illustrated in Figure 20 and Figure 21.

- **SW-to-EAXML transformation:** is the transformation between SW-MM and EAXML-MM. Since the primary model transformation is between these meta-models, we call the transformation between these two meta-models Main transformation. As it is illustrated using Figure 20, the SW-M represents the SW model, the SW-MM represents the SW meta-model, EAXML-MM represents the EAXML meta-model, the EAXML-M represents the EAXML model and the ATL represents the ATL model transformation file. The ATL file uses SW-MM and EAXML-MM to drive the transformation rules. Using the rules derived from source and target meta-models, the ATL file finally transforms the SW-M to EAXML-M.

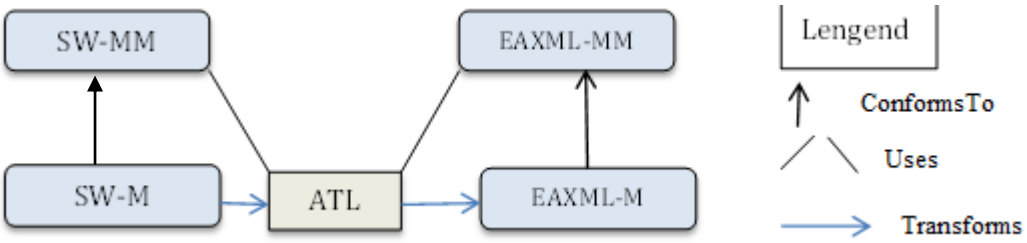


Figure 20: SW2EAXML main transformation

- **EAXML-to-SW transformation:** during the bidirectional transformation, there are two Main transformations. The first one is the transformation between SW-MM to EAXML-MM and the second one is the EAXML-to-SW. It is illustrated in Figure 21 that the transformation is between EAXML-MM-to-SW-MM. The ATL file is constructed according to the information used in EAXML-MM and SW-MM. Using this information, the ATL file is used to transform different EAXML user models to SW models.

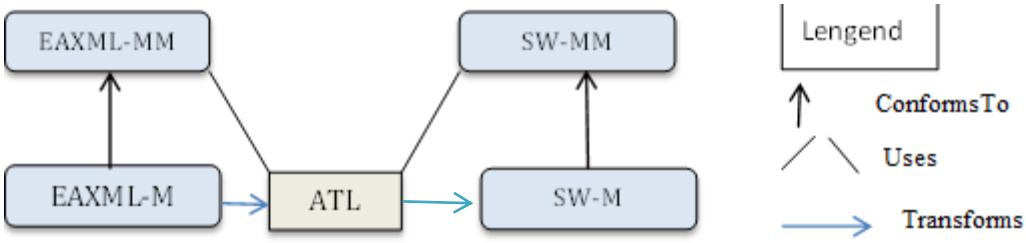


Figure 21: EAXML2SW main transformation

II. Complementary Transformation

ATL cannot handle all transformation requirements listed in section 6.1 by its own. In order to complement the lack of ATL, different Eclipse frameworks called AM3, EMF, and GMF are used. AM3 is used to inject and extract xml object models. EMF is used to create the EMF meta-models form the XML Schemas. The Complementary transformations include ATL transformation and AM3 Inject or Extract functions.

- **SW to EAXML Complementary Transformation:** includes injecting the SWxml using AM3 framework. This injection action creates SWxml.ecore, which conforms to the general XML meta-model, from the SWxml, which is exported form SWExplorer. Then SWxml to SW transformation takes place to transform the SWxml.ecore to SW.ecore model. Figure 22 illustrates the SW to EAXML Complementary transformation by using the Inject functionality to transform the SW.xml to SW-M. Then the ATL file is used to transform XML-MM to SW-MM.

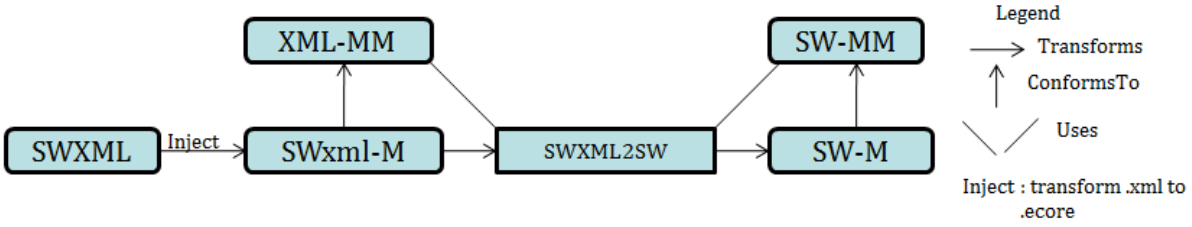


Figure 22: SW2EAXML complementary transformation

- **EAXML to SW Complementary Transformation:** it takes place right after the Main transformation of EAXML to SW. The SW.ecore is created as an output model of the Main transformation between EAXML to SW. However, since the main objective is to obtain the output model in xml format, we need to run Complementary transformation. We have used this Complementary transformation to transform SW.ecore model to SWxml.ecore model using ATL transformation. Then the SWxml.ecore model is extracted to SWxml using AM3 framework. Figure 23 illustrates the transformations used to perform the complementary transformation from EAXML to SW. The ATL is used to transform the concept in EAXML-MM to SW-MM and transform EAXML-M to SW-M. However, the SW-M is not the expected output. Therefore the Extract functionality from AM3 is used to extract the SW.xml from its SW.ecore model file.

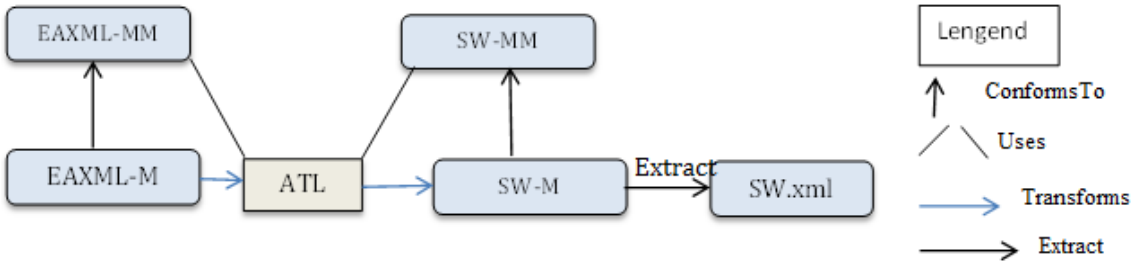


Figure 23: EAXML2SW complementary transformation

For general understanding of the transformation Figure 24 shows the complete bidirectional transformation. We put the above different transformations to illustrate the complete bidirectional transformation.

As it is illustrated in Figure 24, both the SW-MM and SW-M are represented in XML file format. Since it is not possible to use XML files with ATL, we need to transform both the model and the meta-model to the ATL supportive formats. Therefore the SWxml meta-model is first transformed to SW.xsd using an XML editor called Oxygen. Then the SW.xsd is mapped to its EMF based meta-model. The SW.xml model is transformed to SWxml.ecore format using AM3 framework. The SWxml.ecore model conforms to the general XML-MM. Then the first ATL is implemented to transform the concept of XML-MM to SW-MM. These meta-models are EMF based meta-models.

The EAXML meta-model is available in XSD format. We have mapped the EAXML.xsd to its EMF based meta-model. After the EMF based EAXML-MM is developed, the Main transformations

takes place. The second ATL file uses the information of both SW-MM and EAXML-MM and transforms the SW-M to EAXML-M. The word ecore is used as a suffix of the models and meta-models to show that they are of type ecore.

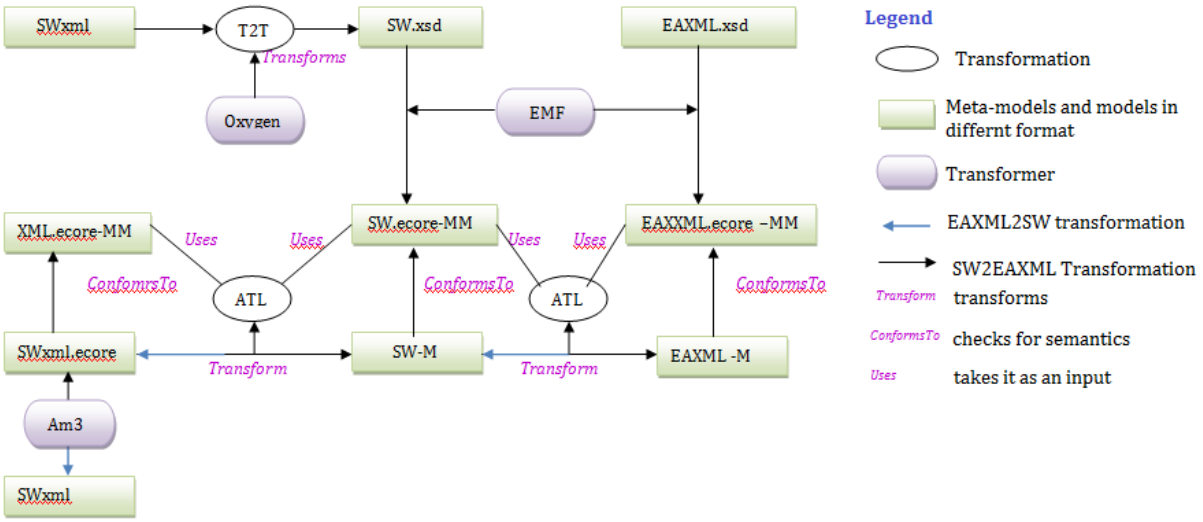


Figure 24: Complete bidirectional transformation between SW and EAXML

Since, being able to import the target model to its respective tool is one proof to show that the integration can work; and that the model transformed using ATL is possible to import it to SWEplorer, we took ATL transformation language and EMF, and AM3 frameworks as the tools that can fully solve the problem.

6.1.2. oAW

Based on oAW the bidirectional transformation between EAXML and SWxml is considered as M2M2T transformation. The target model is represented using XMI format; in order to get the preferable format which is xml, another model to text [21] transformation is needed.

Generally, oAW is a framework which provides textual languages that are useful in different contexts in MDSD process (e.g. checks, extensions, code generation, and model transformation). Xtend, Xpand and Check are built based on common expression language and type system. This means they can operate on the same model, meta-models and meta-meta-models. oAW uses functional programming language EXTENDS to make M2M transformation. It contains different component classes for transformation and code generation and the specific component class which supports M2M transformation is called Xtend. M2M transformation is controlled by the central part of an oAW project which is the so called workflow. The actual transformation is realized by the Xtend extension.

It is possible to use XML Schema and XML both as an input and output models using this transformation. However, the complexity increases with the size of the meta-models. The fact that this transformation language does not support rule based transformation makes it more difficult to use it for this study. The lack of rules in its implementation makes it infeasible to apply it for large meta-models which respectively can have large models. Since this study covers large meta-models, we argue that choosing such tools makes the transformation complex and difficult to maintain.

The oAW based model transformation between SW and EAXML is illustrated using Figure 25. Figure 25 shows that the SWxml is first exported from SWExplorer. Then using the xml editor called Oxygen the SWxml meta-model is transformed to SWxml Schema. The SWxml Schema is transformed to its EMF equivalent meta-model.

When we look at the other side of the transformation, the EAXML.xsd is transformed to its EMF equivalent meta-model. Both SW-MM and EAXML-MM are instances of Ecore. Then after these two meta-models are available in EMF, the transformation between SW-M, which is the SWXML, and EAXML-M, which is the EAMXML.xsd, is performed using Xtend.

Transformation Logic

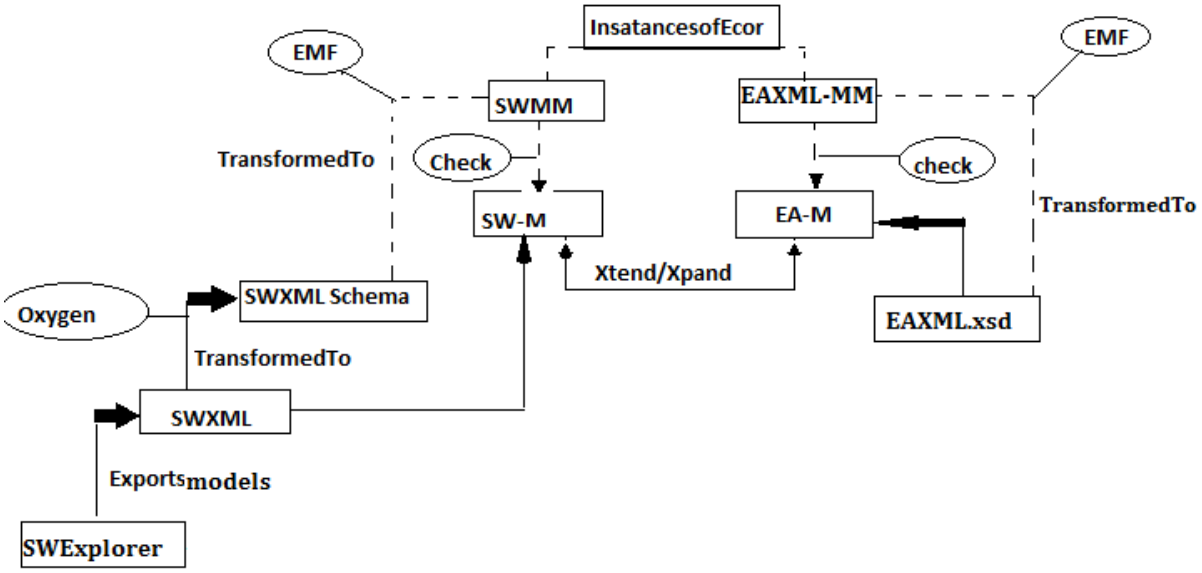


Figure 25: SW and EAXML transformation using oAW

oAW model transformation is mostly done using two extensions called Xtend and Xpand. During model transformation, there should be an input model and both source and target meta-models. The source meta-model is used as confirmation of the user model. During the transformation, the source model is read-only and the target is write-only. Then, Xtend or Xpand handles the transformation of the model. Without considering the size of the models, this concept can be applied to this study. Unlike ATL, this concept must apply observable logics to transform the models. This decreases the performance of the application. Representing EAXML and SWxml using different markup languages causes for further format transformation. Using oAW minimizes the number of transformations that should be undertaken during transformation. The only format transformation needed during this model transformation is XML to XML Schema and vice versa. Generally oAW can read and write models of format UML2, UML (Poseidon, MagicDraw, Enterprise Architect, Rose, XDE, Innovator, etc.), textual models (using JavaCC or antlr parsers as frontends), XML and Visio.

oAW performs the following actions before M2M transformation.

- Calculation to determine which one should have read or write access

- Tests the semantic correctness of the specified models through oAW validation language called CHCECK

Reasons to consider oAW

For a transformation language to be chosen as a suitable tool for this project, it needs to fulfill the transformation requirements listed in the introduction part. oAW or Xtend/Xpand/Check is used for M2M, M2T and T2T transformations and is powerful tool which can accept UML, XML etc. formats as an input model. It can also give the output model in different formats which makes it more suitable for this project. However, despite these merits, the code complexity increases as the model size increases. For a large meta-model transformation, we need to write a transformation code for each and every transformation made. Since this study covers large and complicated model transformations, it is advisable to use this transformation tool.

As it is mentioned earlier, oAW can generate different formats by using a powerful template language called Xpand. This template language can also be used to generate code at the end of model transformation. Currently, the oAW components are migrated to Eclipse Modeling and are mostly called Xtend/Xpand/Check. The fact that, oAW contains all these three extensions and that they can be used for one project at the same time, makes it a more complete package than the other transformation tools. Configurability is one of the features that make this tool suitable for this thesis work.

Drawbacks of oAW

oAW has composite implementation languages. There are multiple languages that have to be used to implement one transformation. Using these different templates, extensions and workflows leads to complexity of implementation. The drawbacks of using oAW are that it is not possible to make bidirectional transformation and that it is complicated to use. We also believe that it decreases the performance as the size of the meta-models increases.

6.1.3. Result of the Transformation Language Investigation

Taking the investigation conducted between ATL and oAW into consideration, ATL has been chosen as the suitable model transformation language for this study. Since ATL is a rule based and compatible with different Eclipse based frameworks such as AM3 and EMF, we have found it suitable to perform the bidirectional transformation between SWxml and EAXML with a good transformation quality and in a reasonable resource. It can handle transformation of complex cases where there is structural difference between the models that has to be transformed.

6.2. Prototype Implementation

A prototype has been implemented using ATL and the rest of the frameworks accordingly. The prototype implementation contains the meta-model implementation, the transformation logic used to implement the solution and the ATL implementation.

6.2.1. Meta-model implementation

The meta-models of EAXML and SW are implemented using EMF. The implementation description is discussed on the next sections.

Implementing SW and EAXML meta-models

The available model in SWxml is the model which is exported from SWExplorer. There is no available SW meta-model represented using model diagram. EMF is used to implement the meta-models. We have implemented SW-MM using the information obtained through mapping

of SW XML Schema to Ecore. We have used the developed EMF based SW-MM throughout the prototype development.

For the prototype implementation, the Ecore based EAXML meta-model which is developed by KTH is used.

6.2.2. Transformation Logic

Overall Transformation steps:

The first thing that has to be done before any transformation is implementing and checking the source and target meta-models availability. For example if SW-MM is the only available meta-model, then we must know that we cannot implement ATL transformation. As it is shown in Figure 26, if only EAXML-MM and SW-MM are available, then it is only possible to implement SW2EAXML and EAXML2SW upon the available source model. The same is true for SW-MM and XML-MM. If these two meta-models are available, then it is possible to write an ATL transformations called SW2XML and XML2SW. Figure 26 illustrates the relationship between meta-models and ATL. If the respective meta-models are available, then the transformation takes place, if one or all of the meta-models are not available then the transformation cannot take place.

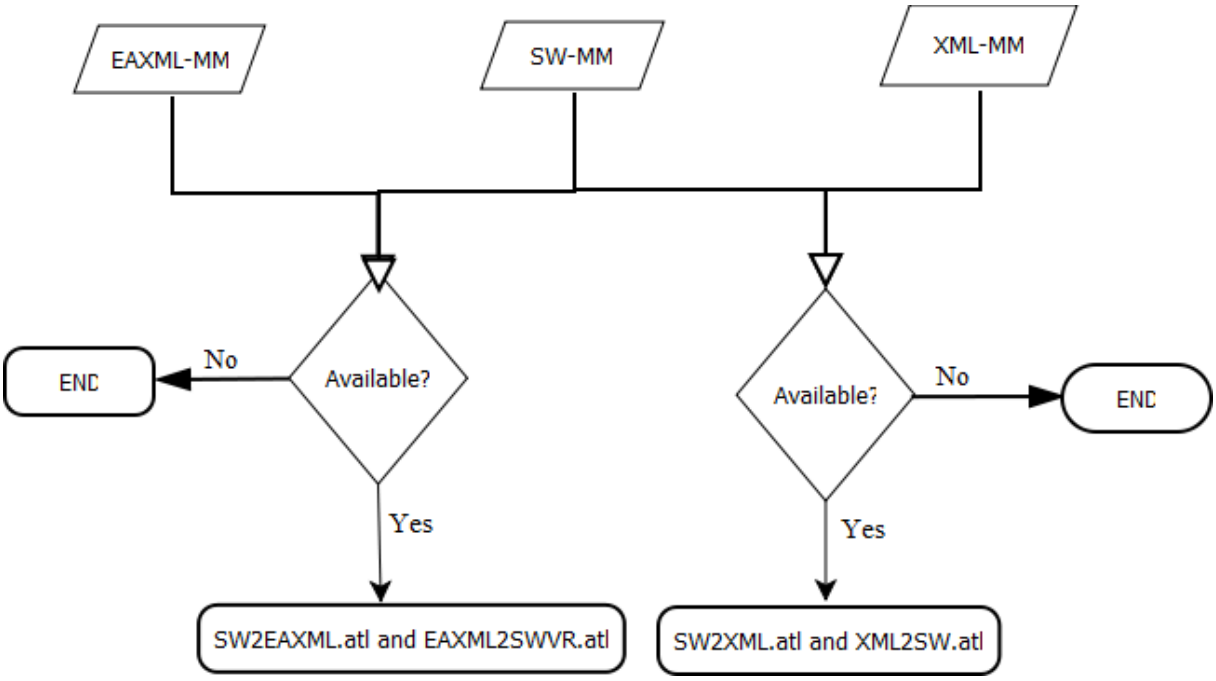


Figure 26: The relationship between meta-models and ATL

Bidirectional transformation steps:

The bidirectional transformation is a combination of Complementary and Main transformations. According to the eight steps, steps four and five are the Main transformations. The rest of the steps describe the Complementary transformation and tool integration. The numbers in Figure 27 describes the steps taken to complete the bidirectional transformation. These steps are described next.

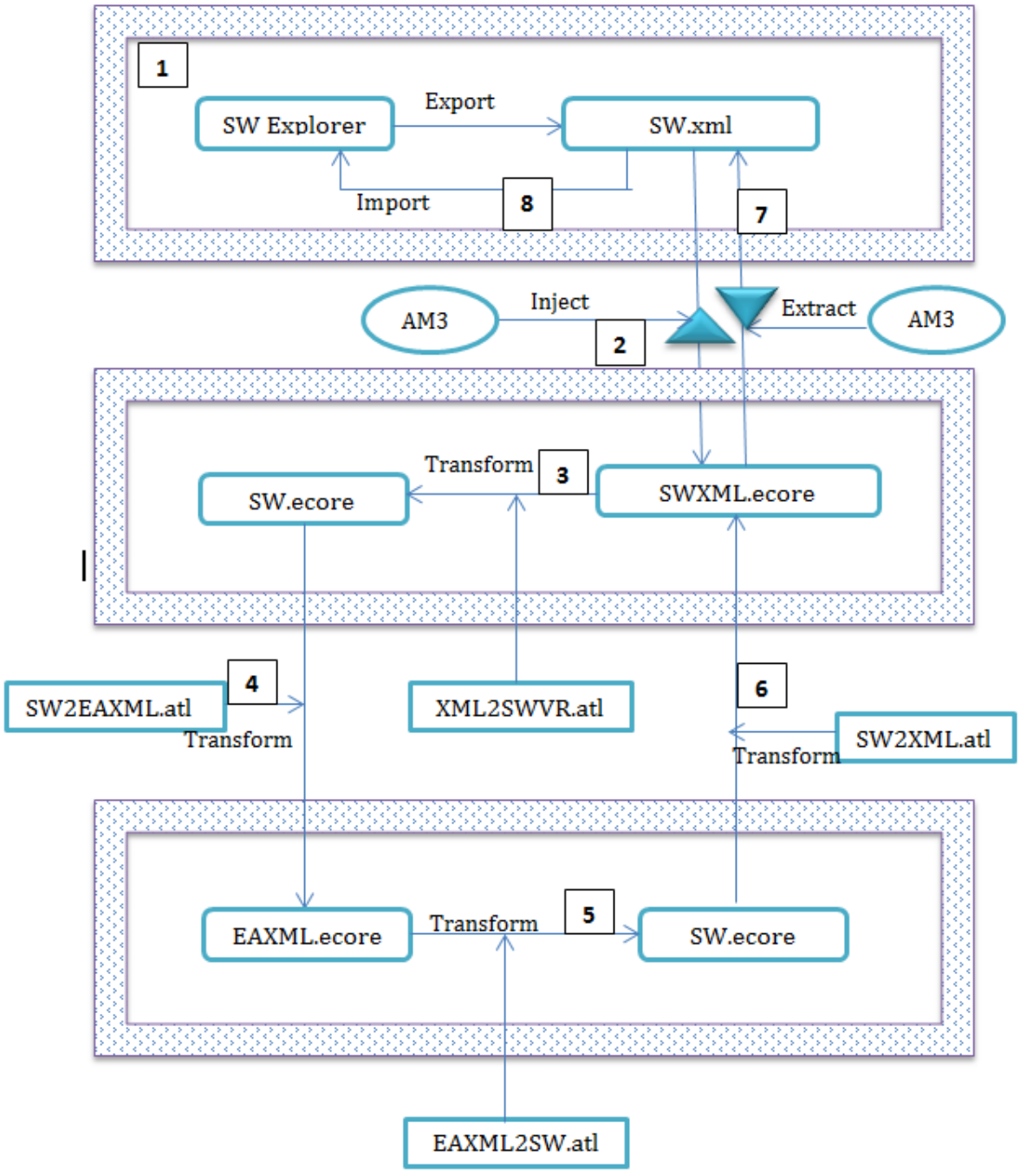


Figure 27: The bidirectional transformation flow and logic

1. The first step shows that SW.xml, which is the object model, is exported from SWExplorer. The output of this operation is SW.xml.

2. The second step injects the SW.xml in order to convert it to Ecore format of general XML model. The Inject function is provided by AM3. Therefore as the result of the injection, the SWxml.ecore is created. This is the first step of the Complementary transformation, in SW2EAXML transformation.

3. The third step is the transformation from SWxml.ecore, which conforms to the XML-MM, to the SW.ecore, which conforms to SW. To see the complete transformation code of XML2SWR , look at Appendix D - **ATL Transformation Files**.

4. The fourth step is the Main transformation. It takes SW.ecore as an input model and creates EAXML.ecore as target model. To look at the SW2EAXML.atl see Appendix D - **ATL Transformation Files**.

5. The fifth step is another Main transformation. It takes the EAXML model as an input and creates the SW model as an output. For more understanding of the EAXML2SW.atl, see Appendix D - ATL Transformation Files.

6. The sixth step is Complementary transformation. It transforms the SW.ecore to SWxml.ecore. The transformation is known as SW2XML. For further understanding see Appendix D - **ATL Transformation Files**.

7. The seventh step is part of the Complementary transformation through an extraction. This extraction function is provided by AM3 and is used to extract the final SW.xml from the general SWxml.ecore.

8. The eighth step shows the tool integration through model transformation. We have tested the implementation through import. The import shows that a model developed using different tool can be imported to the other tool.

Transformation Scenarios:

In the previous sections, we have seen how the meta-models are implemented and how the model transformation logic is implemented. Transformation scenarios are source of the ATL rules. In order to have the correct transformation solution, it is mandatory to make the transformation analysis between SWxml and EAXML. The classes, references and attributes representation have to be identified. The bidirectional transformation is analyzed in Table 4 and Table 5. Table 4 is used to analyze the SW-2-EAXML transformation. It shows the attributes, references and classes. Similarly, Table 5 describes the EAXML-2-SW transformation.

SW2EAXML

EAXML	SW	Part –SW
requiredBindingTime	requiredBindingTime	Yes
actualBindingTime	actualBindingTime	Yes
childNode	childNode	Yes
uaTypeRefs	userAttributeTypes	Yes

OwnedComments(rational, comment)	ownedComments	Yes
OwnedRelationships(Derive-Requirement, featureLink, satisfy, refine, verify, realize)	featureLink, Derived, satisfiedRequirement, refineRequirement, verify, realizes	Yes
FeatureParamete	featureParameter	Yes
TracealbleSpecificationRefs	traceableSpecificationRefs	Yes
UaValues(userAttributeValue)	userAttributevalue	Yes
Uuid	id	Attribute

Table 4: SW2EAXML

EAXML2SW

SWVR	**EAXML**
Name	(actualBindingTime, requiredBindingTime, featureroot etc.)
Ancesstor	uuid
Id	uuid
Status	Value
Description	ownedComment(comment and rational)
Attributes	
partGroups **(partGroup** **Parts** **Part**	(actualBindingTime, childNodes,requiredBindingTime ownedrelationships, Ea DataType, Traceable Specification Refs)
topNode	Not available
versionNumber	Not available
Sid	Not available

Table 5: EAXML2SW

6.2.3. The ATL Implementations

In this section the ATL transformations implemented are described. Four ATL transformations called XML2SW, SW2EAXML, EAXML2SW, and SW2XML are implemented. These four transformations are used to successfully perform bidirectional transformation. The bidirectional transformation is shown in Figure 28 : The bidirectional transformation high level view Figure 28. The transformation from SW-to-EAXML and the meta-models and ATL files involved in a transformation are elaborated. The black arrow shows the transformation from SW-to-EAXML and the green arrow shows the transformation from EAXML-to-SW. The ATL files included in Figure 28 are explained next.

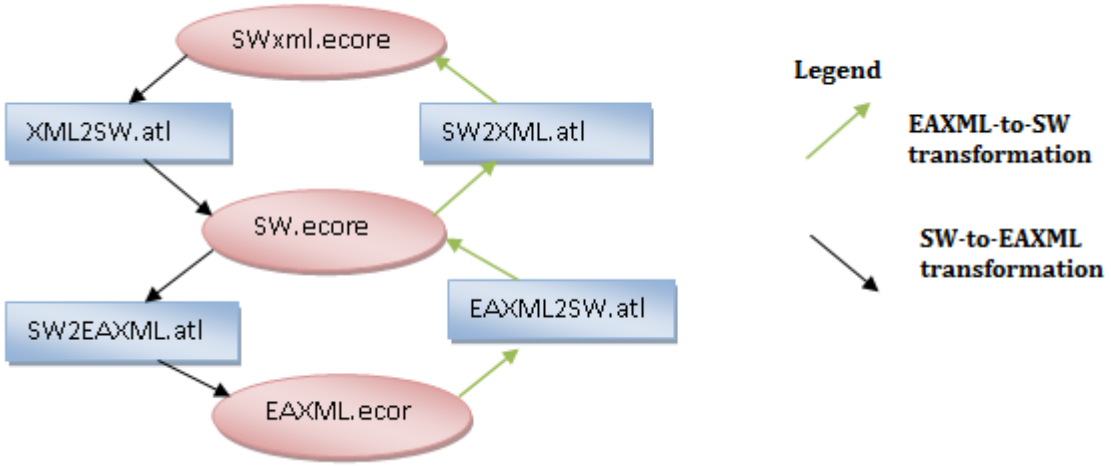


Figure 28: The bidirectional transformation high level view

1. **XML2SWVR.atl:** This transformation is performed by XML2SW.atl. It takes SWxml.ecore, SW-MM and XML MM as an input and creates the target file called SW.ecore. The header is the part of the ATL that described the input and output models and meta-models. The header for the XML2SWVR is shown in Figure 29:

```
module SWXML2SW; -- Module Template
create OUT : SWVR from IN : XML;
```

Figure 29: SWXML2SW.atl header

2. **SW2EAXML.atl:** this is the transformation from SW-to-EAXML. The header of this transformation is shown in Figure 30.

```
--path =/SWVR2EAXML/modeldiagrams/SWVR.ecore;
--path =/SWVR2EAXML/metamodels/EAST-ADL.ecore;

module SWVR2EAXML; -- Module Template
create  OUT:eastadl from IN:SWVR ;
```

Figure 30: SW2EAXML.atl header

3. **EAXML2SW.atl:** this is the Main transformation for the transformation from EAXML-to-SW.

```
module EAXML2SWVR; -- Module Template
create OUT:SWVR  from IN:eastadl ;
```

Figure 31: EAXML2SW.atl header

4. **SW2XML.atl:** this transformation is a Complementary transformation for the transformation from EAXML2SW. This is the final output from the ATL transformation. The output of this transformation is SWxml.ecore and is extracted to SW.xml using the extract projector of AM3.

```
module SWVR2XML; -- Module Template
create OUT:XML from IN:SWVR ;
```

Figure 32: SW2XML.atl header

Issues Detected During Implementation

In this section, we have presented the rules used for each transformation. The rules are the result of the architectural view comparison between SW and EAXML meta-models. The respective implementation code for each rule is found at Appendix D in their respective ATL files. The rules used in four of the transformation implementations are described under their names.

SW2EAXML: this is the complex transformation. It contains simple and complex transformations.

***Rules*:** The first rule that is implemented during this transformation is to transform a feature model of SW to feature model of EAXML.

1. SW Item with PartGroups as part of it, will be a feature on EAXML
2. The requiredBindingTime part of SW is transformed to requiredBindindTime reference, which is of type BindingTime, of EAXML. The requiredBindingTime of SW is also of type BindingTime, however it is identified as both an item and part in SW.
3. The part called actualRequiredBingTime in SW is transformed to the reference of a feature in EAXML. The reference is called actualBindingTime and has type of BindingTime. The actualBindingTime is a part and an Item in SW.
4. The part featureParameter of SW is transformed to the EaDataTypeProtoType
5. The part named as userAttributevalue in SW is transformed to the reference UValues which includes references to userAttributeValue class.

EAXML2SW: the rules are used to map between explicit and implicit concept of implementation. This makes it to be complex transformation.

***Rules*:**

1. When transforming the feature of EAXML to SW, all the Items and parts should be distinguished using rules.
2. The item should be organized inside Items parent.
3. The features in EAXML object model are transformed to the Items include partGroups as a reference within the class.
4. The parts should be organized using imperative language and must be included inside Items class
5. The names of the Item contains the names of the feature with partGroups
6. The names of the parts are transformed using imperative language

XML2SW: for this implementation, the main rules of the transformation are discussed here. The complete solution is implemented and the main points are discussed in this paper.

Rules:

1. Each Element in SWxml.ecore is transformed to Items, Item, PartGroups, PartGroup etc. In general the elements in SWxml.ecore are transformed to the classes of SW.ecore

2. Each attributes of SWxml.ecore are transformed to attributes of the SW.ecore's class's
3. Each Text is transformed to an attributes, such as Name, dataSize etc

SW2XML: this is Complementary transformation during EAXML2SW transformation. The transformation rules listed below are implemented in the ATL files listed in Appendix D.

Rules:

1. The root element is the SystemWeaver 3.0
2. Each Item is transformed to Element
3. Each part are transformed to an Element
4. Each partGroup are transformed to an Element
5. Each partGroups are transformed to an Element
6. Each Attributes are transformed to an Element
7. Each attributes are transformed to Attribute
8. Some attributes are transformed to Text

6.3. Contribution of the Solution

Generally, we have evaluated the research based solution through importing and exporting the model, which is the output of the transformation, to SWExplorer. We exported the SWxml model from SWExplorer and applied the SW2EAXML transformation. After the target model is created, we performed EAXML2SW transformation on the target model and we got similar model which we have imported it to the SWExplorer. Through this process we proved that ATL, EMF and AM3 are the right choices.

To summarize the result of this study, we have provided a solution which includes the following contributions.

- Proposed the suitable M2M transformation language and additional frameworks that helped us to perform complete M2M transformation
- Showed the feasibility of the chosen transformation language through prototype implementation
- Showed that MDT language and frameworks are possible ways of implementing bidirectional transformation.
- Implemented and reviewed SW meta-model using EMF
- Compared alternative transformation languages to choose the most suitable one
- Showed that tool integration can be possible through M2M transformation by using the output of model as an input of the respective tool
- Analyzed the model transformation specifications for EAXML and SW model transformation
- Implemented the transformation of SW meta-meta model to a model for further use

In addition to the main objective of this study which is model transformation, we have designed the logic to implement the tool integration through model transformation. The logic for tool integration is shown in Figure 33

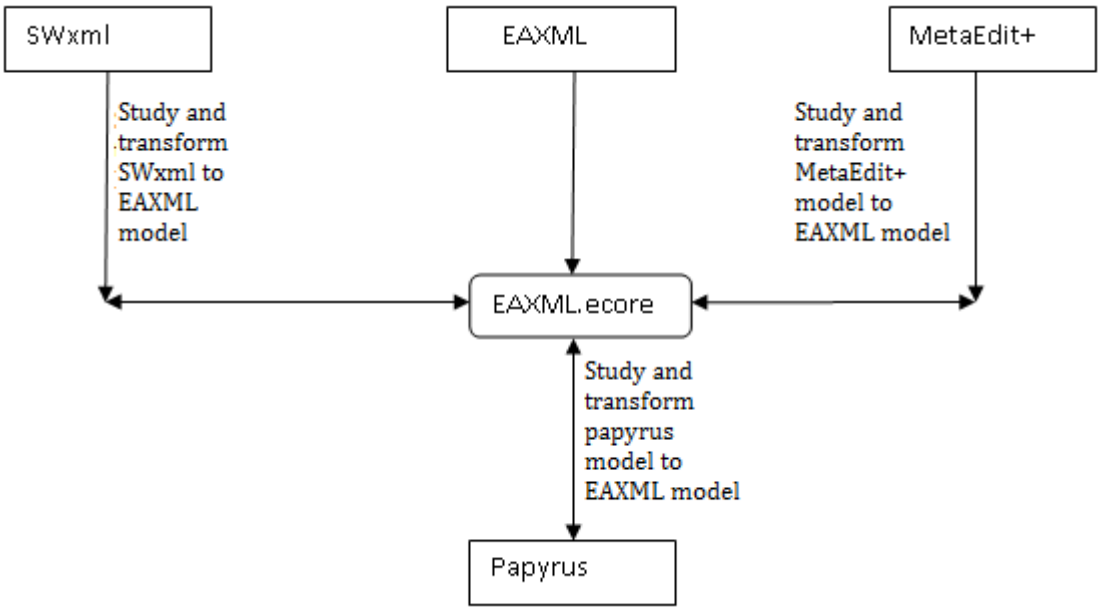


Figure 33: Tool integration logic

ATL along with EMF and AM3 have been proved that they are the promising model transformation language and supporting Eclipse based frameworks. The fact that ATL is a rule based transformation language makes it more preferable language for the transformation between SWxml and EAXML by minimizing complexities of the meta-models. To clearly define how the transformation between SWxml and EAXML is implemented, we have divided the transformations as Main and Complementary transformations.

7. Summary

We have chosen ATL as the transformation language. In addition to ATL we have used extra frameworks called AM3 and EMF to successfully implement the bidirectional transformation. AM3 is used to handle the abstraction level difference between SWxml and EAXML. EMF is used to implement both the source and the target meta-models.

Implementing model transformation requires dedication to the work and clear understanding of the meta-models. Because of this investigating the different meta-model viewer frameworks such as EMF and GMF has been one of the tasks. All the references, classes and attributes need to be clearly understood. Due to this reason understanding of these parts of the meta-models help to define the transformation rules easily. According to the investigation, the model transformation between EAXML and SWxml has been implemented. The implementation is achieved through the transformation logic illustrated in Figure 27.

According to this research study and other studies conducted, the Model driven transformation is the vital field in MDE. It is a field which is very new and needs lots of attention. By conducting this study, we are able to answer the questions listed in section 4.2. Research Questions

Answer RSQ1: the fact that both SWxml and EAXML are represented in different levels of abstractions makes identifying the architectural and structural relationships difficult. In order to understand the differences, we represented both meta-models in same level of abstraction using EMF and analyzed the relationships using the EAST-ADL specification as central document.

Answer RSQ2: understanding the architectural pattern leads to the understanding of the views. We combined the views of SW and EAXML implementations though meta-model mapping, and model driven transformation. The answer for RSQ1 becomes basic input to answer this question. We have confirmed its possibility by looking at both of the meta-models.

Using the meta-model mapping is part of model driven engineering. Hence, analyzing and implementing the meta-models and understanding the EAST-ADL specification makes the process of combining the views possible. The transformation requirements are also used as constraint to check whether SW and EAXML views can be combined using MDE and MDT.

Answer RSQ3: model transformation languages such as ATL, oAW, Epsilon, Kermeta, QVT and dually are investigated. However, both ATL and oAW are the two transformations that are investigated against all the requirements. Since ATL is found to meet most of the requirements than oAW, it is chosen as the suitable transformation language for SWxml and EAXML.

Answer RSQ4: after conducting a study on the model transformation technologies and both meta-models, we have analyzed how the bidirectional transformation can be achieved. The abstraction differences of SWxml and EAXML meta-models, makes the bidirectional transformation complex and time consuming. Because of this, different frameworks such as AM3 and EMF had to be used additionally to ATL to complete the transformation. As it is illustrated in Figure 26, we have designed the transformation logic towards the relationship between meta-models and the transformation models.

Answer RSQ5: we have tested the chosen transformation languages through prototype implementation. The prototype is checked against all the requirements and declared as pass and fail. ATL is the transformation language which passes this validation test

Answer RSQ6: tool integration is not the main objective of the transformation. However, since this study is required to later achieve tool integration, we have designed simple integration logic by considering Papyrus, SW and MetaEadit+ as the tools that has to be integrated. The logic is illustrated in Figure 33.

Future Work

Considering the research as an input to the prototype implementation, the following feature works are suggested.

- To implement the complete transformation between SWxml and EAXML, the transformation has to be organized according to the structural constructs such as SystemModeling, FeatureModeling, VehicleFeatureModeling, FunctionModeling and hardware modeling. We believe that dividing the analysis upon this division makes management and implementation easier.

- In order to get the SID, which is currently considered as data loss during the transformation, the SW meta-meta-model has to be used as an input model during EAXML-to-SW transformation.

- Completely automating the M2M transformation – this makes the transformation easy to use. Ruing ATL configuration is complex. In order to hide the configuration complexity, an automatic run application has to be developed.

- To make it more easy to use, it is even better to implement GUI and configure the transformation externally. This more user friendly than the automated transformation.

- Checking the possibility of importing the newly created model file directly from the interface through dll is also the last thing that should be considered. This makes the transformation independent from SWExplorer.

References

1. ATLAS group (LINA & INRIA) Nantes. Specification of ATL Virtual Machine. Version 1.0, 2005 [Online] [Accessed 10 June 2011] Available at: <http://www.eclipse.org/m2m/atl/doc/ATL_VMSpecification[v00.01].pdf>
2. ATL: EclipseCON 2011 [Online] Available at: <http://www.eclipse.org/atl/> [Accessed 09 March 2011]
3. EAST-ADL Domain Model Specification, version M.2.1.9 2011-01-30. [Online]Available:http://www.maenad.eu/public/EAST-ADL-Specification_M2.1.9.pdf
4. Matthias Biehl et al. – Model-based Analysis & Engineering of Novel Architectures for Dependable Electric Vehicles, MAENAD Grant Agreement 260057.
5. ATL Concepts. [Online]. Available at <: http://wiki.eclipse.org/ATL/Concepts> [Accessed 10 March 2011]
6. Jan Söderberg, founder and chief technical officer of Systemite AB
7. AUTOSAR Specification of Interoperability AUTOSAR Tools V2.0.0 R4.0 Rev 1
8. Cuenot et al. "Managing Complexity of Automotive Electronics Using the EAST-ADL" 12th IEEE International Conference on Engineering Complex Computer Systems(ICECCS 2007)
9. ATLAS group, LINA et al. ATL:Atlas Transformation Language, ATL Starter's Guide version 0.1
10. Christian et al. Model Transformation in Eclipse with ATL and EMF Henshin
11. XML Schema to Ecore Mapping June 28, 2004. [Online] Available at: <http://www.eclipse.org/modeling/emf/docs/overviews/XMLSchemaToEcoreMapping.pdf> [Accessed 10 may 2011]
12. Tolosa et al, Towards Meta-model Interoperability of Models through Intelligent Transformations S.Omatu et al (EDs.) : IWANN 2009, Part II, LNCS 5518, p. 312-322, 2009 @SpringerLink – Verglag Berlin Heidelberg 2009
13. Seidewitz, E. What models mean. IEEE Software 20(5), 26–32 (2003)
14. Ludewig, J. Models in Software Engineering – an introduction. Software and Systems Modeling 2(1), 5–14 (2003)
15. Malavolta et al. Providing Architectural Languages and Tools Interoperability through Model Transformation Technologies IEEE TRANSACTIONS ON SOFTWARE ENGINEEERING, VOL 36, NO. 1, JANUARY/FEBRUARY 2010
16. Gronmo et al. An Empirical Study of the UML Model Transformation Tool (UMT), SINTEF, Norway INTEROP-ESA-First International Conference on Interoperability of Enterprise software and Applications, February 23—25 2005, Geneva Switzerland
17. Wagelaar et al. Module Superimposition: a composition technique for rule-based model transformation languages Springer-Verlag 2009, SPEACIL SECTION PAPER – Revised: 4 Septemeber 2009, Published online: 15 October, 2009
18. Feng et al. Research on Integration of Safety Analysis in Model-Driven Software Development
19. Matthias et al. Model Based Development of Safety-Critical Systems Using Template Based Code Generation, T*echnische Universit¨at M¨unchen Garching b. M¨unchen, Germany {regensbu,buckl,knoll,schrott}@in.tum.de* 13th *EEE International Symposium on Pacific Rim Dependable Computing, 2007*
20. Liegl et al. A Domain Specific Language for UN/CEFACT's Core Components, Vienna University of Technology Favoritenstrasse 9-11/188 A-1040 Vienna, Austria {liegl, mayrhofer}@big.tuwien.ac.at, IEEE 2009 World Conference on Services – II
21. Dominguez et. al. A Survey of UML Models to XML Schemas transformation
22. Chen et al. A model Driven XML Transformation Framework for Business Performance Management, IEEE 2009
23. M.Regensburger et al. Model Based Development of Safety-Critical Systems Using

Template Based Code Generation On13th IEEE International Symposium on Pacific Rim Dependable Computing
24. K. Czarnecki and S. Helsen. Feature-based survey of model transformation approaches. IBM Systems Journal, 45(3):621–645, 2006.
25. M.Biehl et al. Integrating Safety Analysis into the Model-based Development Tool chain of Automotive Embedded Systems. ACM 978-1-60558-953-4/10/4, 125-131, April 13-15, 2010, Stockholm, Sweden
26. Object Model Group(OMG). Meta Object Facility (MOF) 2.0 Query/View/Transformation Specification, Version 1.1 – January 2011
27. P.Muller et al. On Executable Meta-Languages applied to Model Transformations IRISA/INRIA: France , EPFL/IC/UP-LGL: INJ, Swizerland, MIPS: Universite de Haute-Alsace, France
28. Dimitrios et al. Eclipse Development Tools for Epsilon. Department of Computer Science University of York Heslington, York, YO10 5DD, UK.
29. I.Malavolta et al. Providing Architectural Languages and Ty through Tools Interoperability though Model Transformation Technologies IEEE TRANSACTIONS ON SOFTWARE ENGINEERING, VOL 36, No. 1, JANUARY/FEBRUARY 2010
30. T.GHERBI et al. MDE between Promises and Challenges, UKSim 2009: 11th International Conference on Computer Modeling and Simulation
31. M.v.Amstel et al. An Exercise in Iterative Domain-Specific Language Design. Eindhoven University of Technology, IWPSE-EVOL '10, September 20-21, 2010 Antwerp, Belgium. Copyright 2010 ACM 978-1-4503-0128-2/09
32. maenad project. [online] Available at :< www.maenad.eu > [Accessed 6 Jun 2011].
33. F.Jouault (Ed.) Model Transformation with ATL, 1st International Workshop, MtATL 2009. Nantes, France, July 8-9, 2009. Preliminary Proceedings
34. J.E. Rivera et al. Orchestrating ATL Model Transformations, GISUM/Alenea Research Group. Universidad de M´alaga (Spain). {rivera,daniruiz,fernando,jmbautista,av}@lcc.uma.es
35. Matthias Biehl. Literature Study on Model Transformations. Embedded Control Systems. Royal Institute of Technology. Sotckholm, Sweden biehl@md.kth.se, July 2010
36. Meta Object Facility (MOF) 2.0 Query/View/Transformation Specification. Version
37. Systemite web site. [Online] Avaliable at:www.systemtie.se [Accessed 1, March 2011]
38. Introduction to Kermeta. [online] Available at:< http://www.kermeta.org/docs/fr.irisa.triskell.kermeta.documentation/build/html.chunked/KerMeta-Manual/ch01.html > [Accessed 28 Sep 2011]
39. J.Zhang et. al. A Model Transformation Classification Method Used in QVT, Proceedings of 2008 International Symposium on IT in Medicine and Education
40. J.Dong et. al. QVT BASED MODEL TRASNFORMATION FOR DESIGN PATTERN EVOLUTIONS
41. S.Kolovos et. al. Update Transformation in the Small with the Epsilon Wizard Language
42. O.Nikiforova et. al. Discussing the differences between Model Driven Architecture and Model Driven Development in the Context of Supporting Tools
43. Eclipse official website : http://www.eclipse.org

Appendix A

The SWxml file is the model file originally available for this study. During the research study we have used it as documentation.

SW.xml


```
<?xml version="1.0"?>
- <SystemWeaver3>
  + <Info>
  + <Meta>
  - <Items>
    - <Item sid="EABT" ancestor="x0000001D04000681" id="x0000001D04000681">
        <Name>requiredBindingTime</Name>
        <VersionText/>
        <VersionNumber>1</VersionNumber>
        <Status>I</Status>
      - <Description>
        - <Binary>
            <![CDATA[]]>
          </Binary>
          <Plain/>
        </Description>
        <Attributes/>
      - <TopNode id="x0000000D1F000682">
          <SubNodes/>
        </TopNode>
      </Item>
    - <Item sid="EABT" ancestor="x0000001D04000684" id="x0000001D04000684">
        <Name>actualBindingTime</Name>
        <VersionText/>
        <VersionNumber>1</VersionNumber>
        <Status>I</Status>
      - <Description>
        - <Binary>
    - <TopNode id="x0000000D1F000685">
        <SubNodes/>
      </TopNode>
  </Item>
- <Item sid="FVFM" ancestor="x0000001D040135F9" id="x0000001D040135F9">
    <Name>FeatureSimpleCase</Name>
    <VersionText/>
    <VersionNumber>1</VersionNumber>
    <Status>I</Status>
  - <Description>
    - <Binary>
        <![CDATA[]]>
      </Binary>
      <Plain/>
    </Description>
  - <Attributes>
      <Attribute sid="CARD" id="x0000000817013961" binary="False">1..2</Attribute>
      <Attribute sid="AICV" id="x0000000817013962" binary="False">True</Attribute>
      <Attribute sid="IDSN" id="x0000000817013963" binary="False">fclass</Attribute>
      <Attribute sid="IDCA" id="x0000000817013964" binary="False">Feature</Attribute>
      <Attribute sid="ABTK" id="x0000000817013965" binary="False">codeGenerationTime</Attribute>
      <Attribute sid="AIDV" id="x0000000817013966" binary="False">False</Attribute>
      <Attribute sid="AIR" id="x0000000817013967" binary="False">False</Attribute>
    </Attributes>
  - <PartGroups>
    + <PartGroup sid="CNTS">
    + <PartGroup sid="VDAS">
    + <PartGroup sid="EOCM">
    + <PartGroup sid="EUAT">
    + <PartGroup sid="EUAV">
```

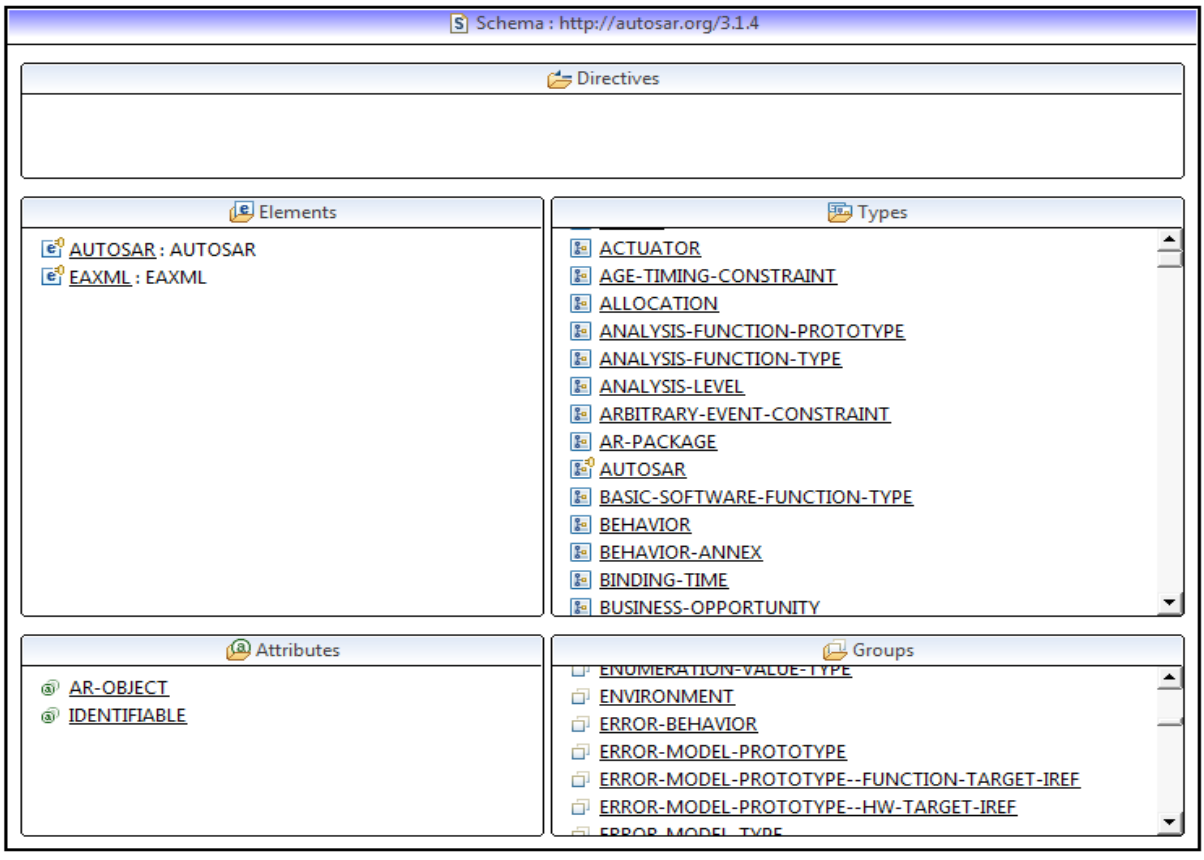
```
            </Node>
          - <Node id="x0000000D1C01397F" item_id="x0000001D0401397C" part_id="x000000051501397E">
                <Path>FeatureSimpleCase:userAttributeElementType</Path>
                <Attributes/>
                <SubNodes/>
            </Node>
          - <Node id="x0000000D1C013A14" item_id="x0000001D04013987" part_id="x0000000515013A13">
                <Path>FeatureSimpleCase:userAttributeValue</Path>
                <Attributes/>
                <SubNodes/>
            </Node>
        </SubNodes>
    </TopNode>
</Item>
- <Item sid="IECM" ancestor="x0000001D0401382E" id="x0000001D0401382E">
    <Name>comment</Name>
    <VersionText/>
    <VersionNumber>1</VersionNumber>
    <Status>I</Status>
  - <Description>
      - <Binary>
            <![CDATA[]]>
        </Binary>
        <Plain/>
    </Description>
  - <Attributes>
        <Attribute sid="ABDY" id="x0000000817013832" binary="False">this is a model of the right window of a car</Attribute>
    </Attributes>
  - <TopNode id="x0000000D1F01382F">
        <SubNodes/>
    </TopNode>
</Item>
- <Item sid="GCTR" ancestor="x0000001D04013969" id="x0000001D04013969">
    <Name>traceabilitySpecifications</Name>
    <VersionText/>
    <VersionNumber>1</VersionNumber>
    <Status>I</Status>
  - <Description>
      - <Binary>
            <![CDATA[]]>
        </Binary>
        <Plain/>
    </Description>
  - <Attributes>
        <Attribute sid="AACA" id="x000000081701393B" binary="False">yes</Attribute>
        <Attribute sid="AACC" id="x000000081701393C" binary="False">yes</Attribute>
        <Attribute sid="AACD" id="x000000081701393D" binary="False">yes</Attribute>
        <Attribute sid="AACN" id="x000000081701393E" binary="False">yes</Attribute>
        <Attribute sid="AAM" id="x000000081701393F" binary="False">yes</Attribute>
        <Attribute sid="AARD" id="x0000000817013940" binary="False">yes</Attribute>
        <Attribute sid="AARF" id="x0000000817013941" binary="False">yes</Attribute>
        <Attribute sid="AARG" id="x0000000817013942" binary="False">yes</Attribute>
        <Attribute sid="AARM" id="x0000000817013943" binary="False">yes</Attribute>
    </Attributes>
  - <TopNode id="x0000000D1F01393A">
        <SubNodes/>
    </TopNode>
</Item>
- <Item sid="GCTR" ancestor="x0000001D04013969" id="x0000001D04013969">
    <Name>traceabilitySpecifications</Name>
    <VersionText/>
    <VersionNumber>1</VersionNumber>
    <Status>I</Status>
  - <Description>
      - <Binary>
            <![CDATA[]]>
        </Binary>
```

```
            </Description>
            <Attributes/>
          - <TopNode id="x0000000D1F01396A">
                <SubNodes/>
            </TopNode>
        </Item>
      - <Item sid="DTCD" ancestor="x0000001D0401396E" id="x0000001D0401396E">
            <Name>featureParameter</Name>
            <VersionText/>
            <VersionNumber>1</VersionNumber>
            <Status>I</Status>
          - <Description>
              - <Binary>
                    <![CDATA[]]>
                </Binary>
                <Plain/>
            </Description>
            <Attributes/>
          - <TopNode id="x0000000D1F01396F">
                <SubNodes/>
            </TopNode>
        </Item>
      - <Item sid="FVFM" ancestor="x0000001D04013974" id="x0000001D04013974">
            <Name>childNode</Name>
            <VersionText/>
            <VersionNumber>1</VersionNumber>
            <Status>I</Status>
          - <Description>
              - <Binary>
                    <![CDATA[]]>
            </Binary>
            <Plain/>
        </Description>
      - <Attributes>
            <Attribute sid="AIR" id="x0000000817013976" binary="False">False</Attribute>
        </Attributes>
      - <TopNode id="x0000000D1F013975">
            <SubNodes/>
        </TopNode>
    </Item>
  - <Item sid="EAUE" ancestor="x0000001D0401397C" id="x0000001D0401397C">
        <Name>userAttributeElementType</Name>
        <VersionText/>
        <VersionNumber>1</VersionNumber>
        <Status>I</Status>
      - <Description>
          - <Binary>
                <![CDATA[]]>
            </Binary>
            <Plain/>
        </Description>
      - <Attributes>
            <Attribute sid="IDSN" id="x00000008170139A5" binary="False">userattrbeleTyp</Attribute>
            <Attribute sid="IDCA" id="x00000008170139A6" binary="False">analysis</Attribute>
        </Attributes>
```

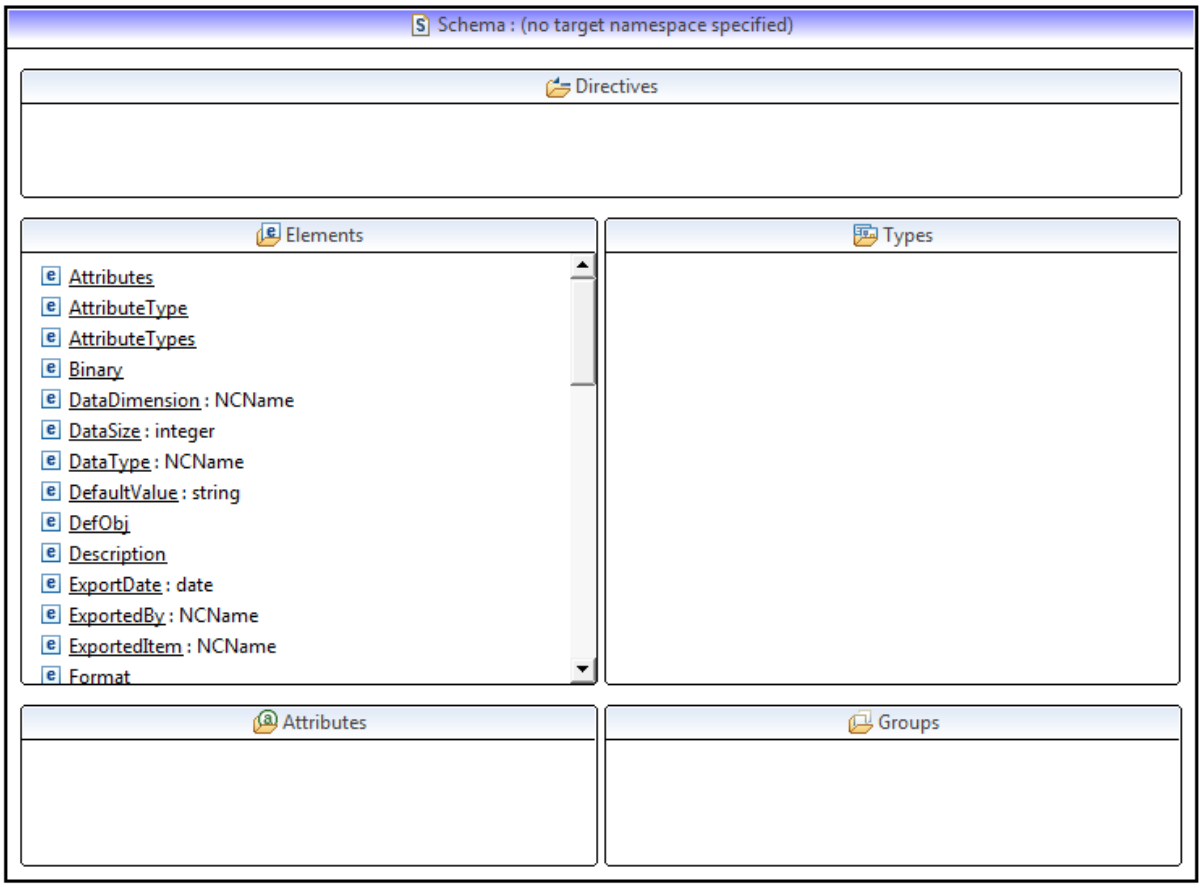
```
        - <Attributes>
            <Attribute sid="IDSN" id="x00000008170139A5" binary="False">userattrbeleTyp</Attribute>
            <Attribute sid="IDCA" id="x00000008170139A6" binary="False">analysis</Attribute>
          </Attributes>
        - <TopNode id="x0000000D1F01397D">
            <SubNodes/>
          </TopNode>
      </Item>
    - <Item sid="EAAV" ancestor="x0000001D04013987" id="x0000001D04013987">
        <Name>userAttributeValue</Name>
        <VersionText/>
        <VersionNumber>1</VersionNumber>
        <Status>I</Status>
        - <Description>
          - <Binary>
              <![CDATA[]]>
            </Binary>
            <Plain/>
          </Description>
        - <Attributes>
            <Attribute sid="IDSN" id="x0000000817013A11" binary="False">uaV</Attribute>
            <Attribute sid="IDCA" id="x0000000817013A12" binary="False">feature</Attribute>
          </Attributes>
        - <TopNode id="x0000000D1F013988">
            <SubNodes/>
          </TopNode>
      </Item>
  </Items>
</SystemWeaver3>
```

Appendix B – XML Schema meta-models

EAXML developed by KTH



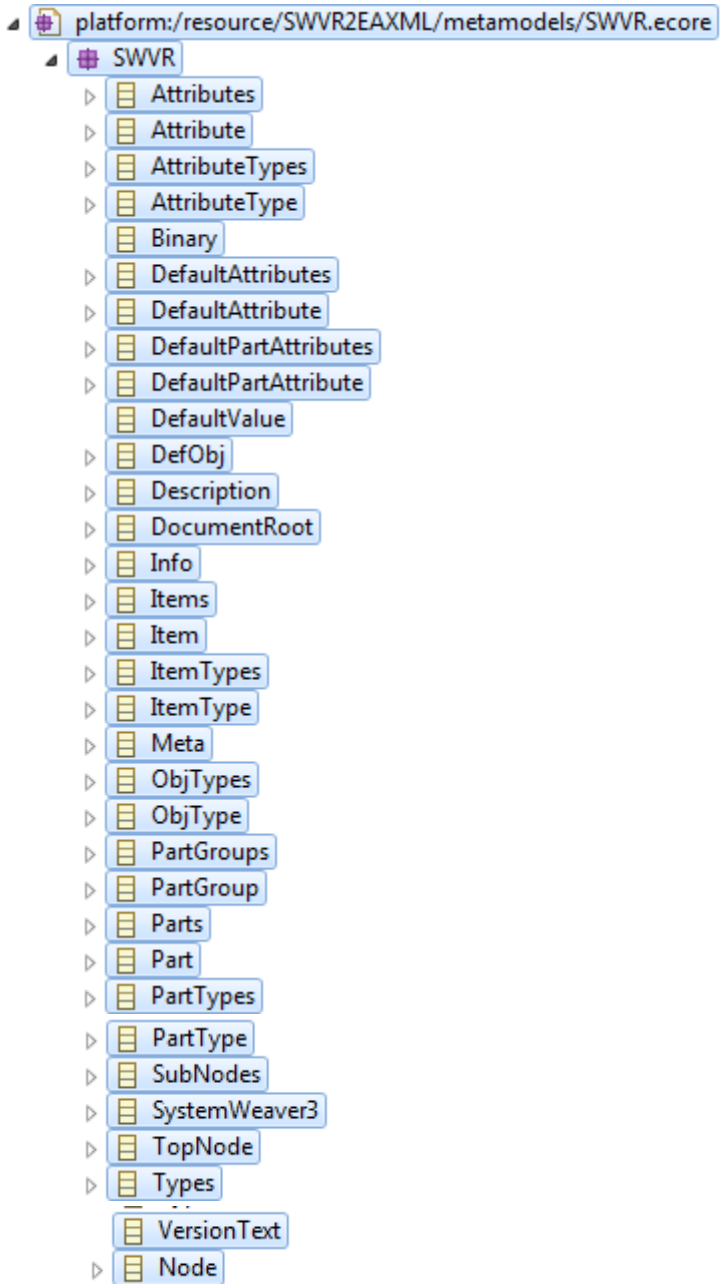
SW in XML Schema format



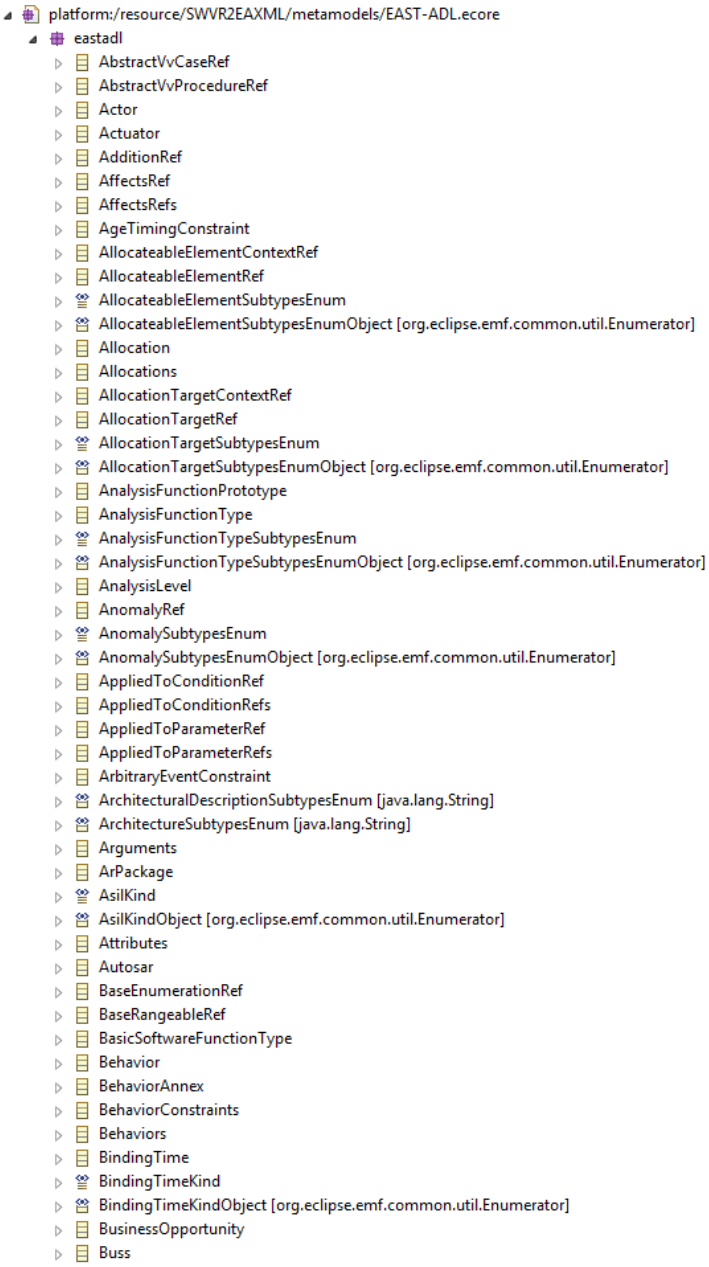
Appendix C – The EMF meta-models

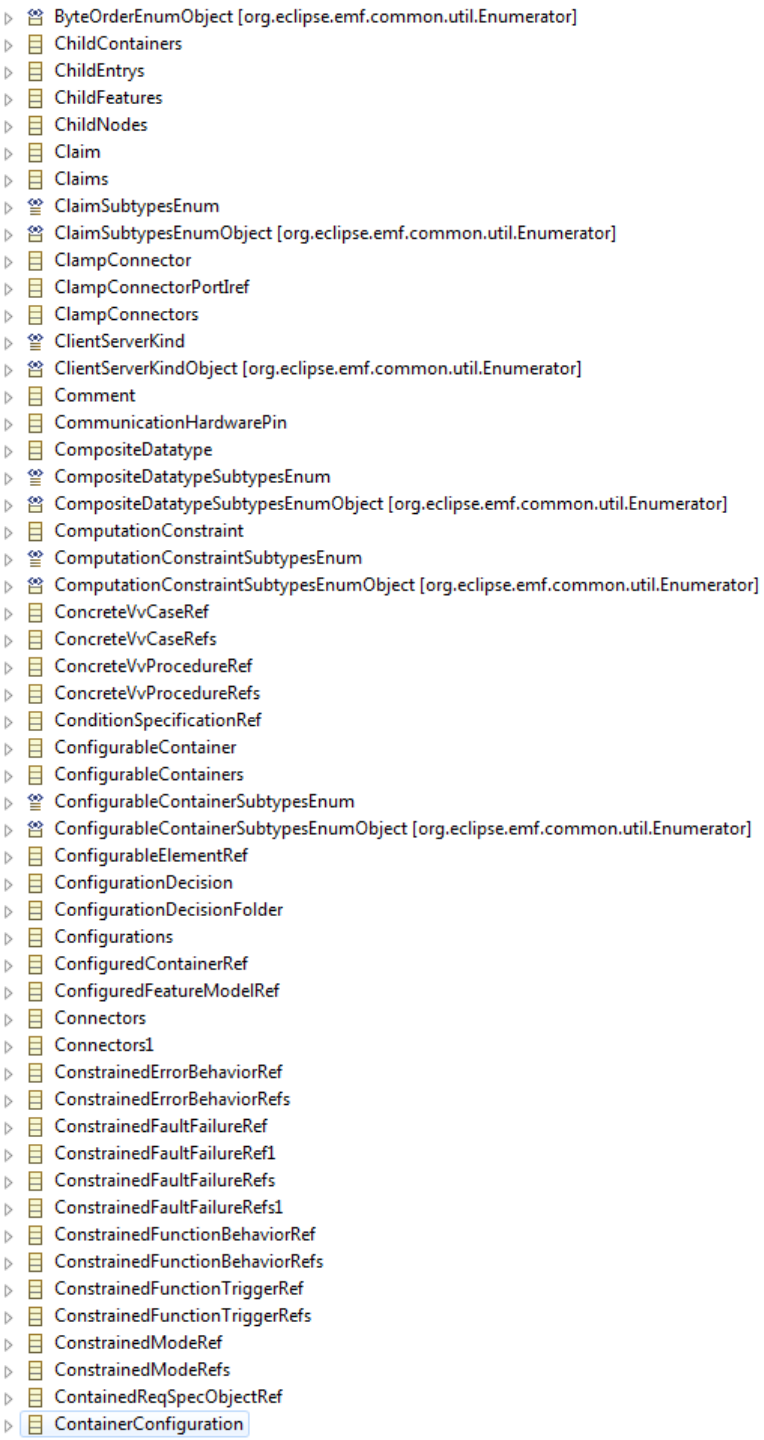
These two meta-models were first imported to EMF project and are used to implement the EMF meta-model.

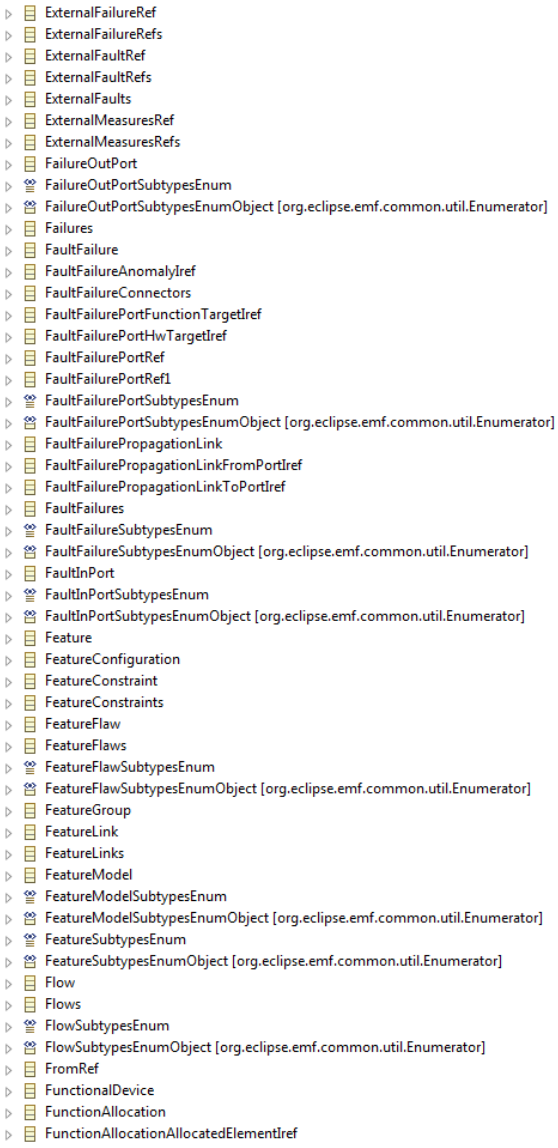
SW Meta-model



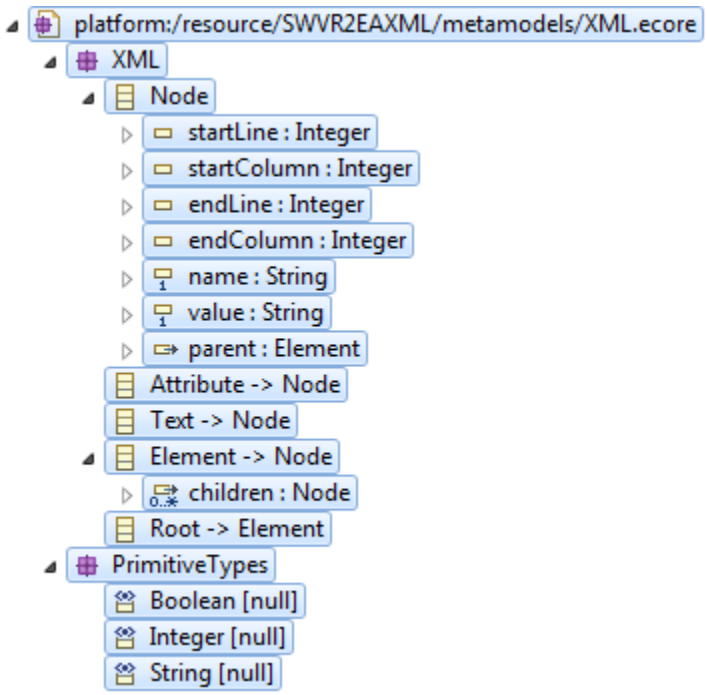
EAXML Meta-model


platform:/resource/SWVR2EAXML/metamodels/EAST-ADL.ecore
eastadl
AbstractVvCaseRef
AbstractVvProcedureRef
Actor
Actuator
AdditionRef
AffectsRef
AffectsRefs
AgeTimingConstraint
AllocateableElementContextRef
AllocateableElementRef
AllocateableElementSubtypesEnum
AllocateableElementSubtypesEnumObject [org.eclipse.emf.common.util.Enumerator]
Allocation
Allocations
AllocationTargetContextRef
AllocationTargetRef
AllocationTargetSubtypesEnum
AllocationTargetSubtypesEnumObject [org.eclipse.emf.common.util.Enumerator]
AnalysisFunctionPrototype
AnalysisFunctionType
AnalysisFunctionTypeSubtypesEnum
AnalysisFunctionTypeSubtypesEnumObject [org.eclipse.emf.common.util.Enumerator]
AnalysisLevel
AnomalyRef
AnomalySubtypesEnum
AnomalySubtypesEnumObject [org.eclipse.emf.common.util.Enumerator]
AppliedToConditionRef
AppliedToConditionRefs
AppliedToParameterRef
AppliedToParameterRefs
ArbitraryEventConstraint
ArchitecturalDescriptionSubtypesEnum [java.lang.String]
ArchitectureSubtypesEnum [java.lang.String]
Arguments
ArPackage
AsilKind
AsilKindObject [org.eclipse.emf.common.util.Enumerator]
Attributes
Autosar
BaseEnumerationRef
BaseRangeableRef
BasicSoftwareFunctionType
Behavior
BehaviorAnnex
BehaviorConstraints
Behaviors
BindingTime
BindingTimeKind
BindingTimeKindObject [org.eclipse.emf.common.util.Enumerator]
BusinessOpportunity
Buss


ByteOrderEnumObject [org.eclipse.emf.common.util.Enumerator]
ChildContainers
ChildEntrys
ChildFeatures
ChildNodes
Claim
Claims
ClaimSubtypesEnum
ClaimSubtypesEnumObject [org.eclipse.emf.common.util.Enumerator]
ClampConnector
ClampConnectorPortIref
ClampConnectors
ClientServerKind
ClientServerKindObject [org.eclipse.emf.common.util.Enumerator]
Comment
CommunicationHardwarePin
CompositeDatatype
CompositeDatatypeSubtypesEnum
CompositeDatatypeSubtypesEnumObject [org.eclipse.emf.common.util.Enumerator]
ComputationConstraint
ComputationConstraintSubtypesEnum
ComputationConstraintSubtypesEnumObject [org.eclipse.emf.common.util.Enumerator]
ConcreteVvCaseRef
ConcreteVvCaseRefs
ConcreteVvProcedureRef
ConcreteVvProcedureRefs
ConditionSpecificationRef
ConfigurableContainer
ConfigurableContainers
ConfigurableContainerSubtypesEnum
ConfigurableContainerSubtypesEnumObject [org.eclipse.emf.common.util.Enumerator]
ConfigurableElementRef
ConfigurationDecision
ConfigurationDecisionFolder
Configurations
ConfiguredContainerRef
ConfiguredFeatureModelRef
Connectors
Connectors1
ConstrainedErrorBehaviorRef
ConstrainedErrorBehaviorRefs
ConstrainedFaultFailureRef
ConstrainedFaultFailureRef1
ConstrainedFaultFailureRefs
ConstrainedFaultFailureRefs1
ConstrainedFunctionBehaviorRef
ConstrainedFunctionBehaviorRefs
ConstrainedFunctionTriggerRef
ConstrainedFunctionTriggerRefs
ConstrainedModeRef
ConstrainedModeRefs
ContainedReqSpecObjectRef
ContainerConfiguration


ExternalFailureRef
ExternalFailureRefs
ExternalFaultRef
ExternalFaultRefs
ExternalFaults
ExternalMeasuresRef
ExternalMeasuresRefs
FailureOutPort
FailureOutPortSubtypesEnum
FailureOutPortSubtypesEnumObject [org.eclipse.emf.common.util.Enumerator]
Failures
FaultFailure
FaultFailureAnomalyIref
FaultFailureConnectors
FaultFailurePortFunctionTargetIref
FaultFailurePortHwTargetIref
FaultFailurePortRef
FaultFailurePortRef1
FaultFailurePortSubtypesEnum
FaultFailurePortSubtypesEnumObject [org.eclipse.emf.common.util.Enumerator]
FaultFailurePropagationLink
FaultFailurePropagationLinkFromPortIref
FaultFailurePropagationLinkToPortIref
FaultFailures
FaultFailureSubtypesEnum
FaultFailureSubtypesEnumObject [org.eclipse.emf.common.util.Enumerator]
FaultInPort
FaultInPortSubtypesEnum
FaultInPortSubtypesEnumObject [org.eclipse.emf.common.util.Enumerator]
Feature
FeatureConfiguration
FeatureConstraint
FeatureConstraints
FeatureFlaw
FeatureFlaws
FeatureFlawSubtypesEnum
FeatureFlawSubtypesEnumObject [org.eclipse.emf.common.util.Enumerator]
FeatureGroup
FeatureLink
FeatureLinks
FeatureModel
FeatureModelSubtypesEnum
FeatureModelSubtypesEnumObject [org.eclipse.emf.common.util.Enumerator]
FeatureSubtypesEnum
FeatureSubtypesEnumObject [org.eclipse.emf.common.util.Enumerator]
Flow
Flows
FlowSubtypesEnum
FlowSubtypesEnumObject [org.eclipse.emf.common.util.Enumerator]
FromRef
FunctionalDevice
FunctionAllocation
FunctionAllocationAllocatedElementIref

XML Meta-Model



Appendix D - ATL Transformation Files

XML2SW.atl

```
module SWXML2SW; -- Module Template
create OUT : SWVR from IN : XML;


--------------------------------------------------------------------------------
-- HELPERS ---------------------------------------------------------------------
--------------------------------------------------------------------------------

-- Returns a sequence of all XML sub elements for XML element, recursivelly
-- from this element to last element (children)
-- CONTEXT: XML!Element
-- OUT:     Sequence of XML elements
helper context XML!Element def : allSubElements : Sequence(XML!Element) =
   let subElems : Sequence(XML!Element) =
      XML!Element.allInstances()->select(c |              -- 1. get all elements with this parent
         c.parent = self
      )->collect(elems | elems.allSubElements)->flatten()  in -- 2. for every element get subelements
      subElems->union(                                 -- 3. when all subelements are apprehended
                                                -- for every element, add topmost elements (child of this parent).
         self.children->select(ch | ch.oclIsTypeOf(XML!Element)
      ))->flatten();

-- Returns a sequence of all XML sub attributes for XML element, recursivelly
-- from this attribute to last attribute (children)
-- CONTEXT: XML!Element
-- OUT:     Sequence of XML attributes
helper context XML!Element def : allSubAttributes : Sequence(XML!Attribute) =
   let subAttrs : Sequence(XML!Attribute) =
      XML!Element.allInstances()->select(c |
         c.parent = self
      )->collect(attrs | attrs.allSubAttributes)->flatten()  in
      subAttrs->union(
         self.children->select(at | at.oclIsTypeOf(XML!Attribute)
      ))->flatten();

-- Returns the element's set of children using their name
-- CONTEXT: XML!Element
-- RETURN: Set(XML!Element)
helper context XML!Element def: getChildrenByName(name : String) : Set(XML!Element) =
   self.children->select(e | e.oclIsTypeOf(XML!Element) and e.name = name);

-- This getName() helper returns the name of XML Element  "Name"
-- It first gets the element's children named "Name"
-- Then it parses to the XML Element child called  "Text"
-- At the end it returns the  associated value
-- CONTEXT: XML!Element
-- RETURN: String
helper context XML!Element def : getName() : String =
   self.children->select(c | c.oclIsTypeOf(XML!Element) and c.name = 'Name')->first().children
        ->select(e | e.oclIsKindOf(XML!Text))->first().value;

-- This getInfo() helper returns the name of XML Element  "DataSize"
-- It first gets the element's children named "DataSize"
-- Then it parses to the XML Element child called  "Text"
-- At the end it returns the  associated value
-- CONTEXT: XML!Element
-- RETURN: String
helper context XML!Element def : getSize() : String =
```

```
-- RETURN: String
helper context XML!Element def : getDimension() : String =
   self.children->select(c | c.oclIsTypeOf(XML!Element) and c.name = 'DataDimension')->first().children
         ->select(e | e.oclIsKindOf(XML!Text))->first().value;

-- This getInfo() helper returns the name of XML Element  "DataType"
-- It first gets the element's children named "DataType"
-- Then it parses to the XML Element child called  "Text"
-- At the end it returns the  associated value
-- CONTEXT: XML!Element
-- RETURN: String
helper context XML!Element def : getType() : String =
   self.children->select(c | c.oclIsTypeOf(XML!Element) and c.name = 'DataType')->first().children
         ->select(e | e.oclIsKindOf(XML!Text))->first().value;

-- This getInfo() helper returns the name of XML Element  "Range"
-- It first gets the element's children named "Data"
-- Then it parses to the XML Element child called  "Text"
-- At the end it returns the  associated value
-- CONTEXT: XML!Element
-- RETURN: String
helper context XML!Element def : getRange() : String =
   self.children->select(c | c.oclIsTypeOf(XML!Element) and c.name = 'Range')->first().children
      --->select(d | d.oclIsTypeOf(XML!Element) and d.name = 'text')->first().children
      ->select(e | e.oclIsKindOf(XML!Text))->first().value;

-- This getInfo() helper returns the name of XML Element  "Format"
-- It first gets the element's children named "Format"
-- Then it parses to the XML Element child called  "Text"
-- At the end it returns the  associated value
-- CONTEXT: XML!Element
-- RETURN: String
helper context XML!Element def : getFormat() : String =
   self.children->select(c | c.oclIsTypeOf(XML!Element) and c.name = 'Format')->first().children
      ->select(e | e.oclIsKindOf(XML!Text))->first().value;


-- This getRestritcted() helper returns the name of XML Element  "IsRestricted"
-- It first gets the element's children named "IsRestricted"
-- Then it parses to the XML Element child called  "Text"
-- At the end it returns the  associated value
-- CONTEXT: XML!Element
-- RETURN: String
helper context XML!Element def : getRestriction() : String =
   self.children->select(c | c.oclIsTypeOf(XML!Element) and c.name = 'IsRestricted')->first().children
      --->select(d | d.oclIsTypeOf(XML!Element) and d.name = 'text')->first().children
      ->select(e | e.oclIsKindOf(XML!Text))->first().value;

-- This getPla() helper returns the name of XML Element  "Plain"
-- It first gets the element's children named "Description"
-- It then parses to the next child Element called 'Plain''
-- Then it parses to the XML Element child called  "Text"
-- At the end it returns the  associated value
-- CONTEXT: XML!Element
-- RETURN: String
helper context XML!Element def : getPla() : String =
   self.children->select(c | c.oclIsKindOf(XML!Element) and c.name = 'Plain')->first().children
      --self.children ->select(d | d.oclIsKindOf(XML!Element) and d.name = 'Plain')->first().children
```

```
    ->select(e | e.oclIsKindOf(XML!Text))->first().value;

helper context XML!Element def : getExportedBy() : String =
  self.children->select(c | c.oclIsTypeOf(XML!Element) and c.name = 'ExportedBy')->first().children
    ->select(e | e.oclIsKindOf(XML!Text))->first().value;

helper context XML!Element def: getDefaultValue():String =
    self.children -> select (c|c.oclIsTypeOf(XML!Element) and c.name = 'DefaultValue') -> first().children
      ->select(e | e.oclIsKindOf(XML!Text))->first().value;

helper context XML!Element def: getPath():String =
    self.children -> select (c|c.oclIsKindOf(XML!Element) and c.name = 'SubNodes') -> first().children
      ->select(e | e.oclIsKindOf(XML!Text))->first().value;

helper context XML!Element def: getSimpleStringDataValue() : String =
  let eltC : Sequence(XML!Text) =
    self.children->select(d | d.oclIsKindOf(XML!Text))->asSequence()
  in
    if eltC->notEmpty()
    then
      eltC->first().value
    else
      ''
    endif;
-------------------------------------------------------------------------------------------------------------------------------------------


-------------------------------------------------------------------------------

-- rule to generate the items node of the systemweaver  user model in xmi
rule itesms
{
  from
    s:XML!Element (
      s.name = 'Items'
    )
  to
    itms: SWVR!Items(
        item <-  Sequence{s.getChildrenByName('Item')->collect(e | thisModule.resolveTemp(e, 'itm'))}
      )
}

--rule to generate an item of systemweaver
rule item
{
  from
    s:XML!Element(
    s.name = 'Item'
    )
  to
    itm: SWVR!Item
    (

      versionText    <- vtext, --Sequence{s.getChildrenByName('VersionText')->collect(e | thisModule.resolveTemp(e, 'vtext'))},
      name          <- s.getName(),
      versionNumber  <- s.getIntegerAttrVal('versionNumber'),
      status        <- s.getAttrVal('status'),
      ancestor      <- s.getAttrVal('ancestor'),
      id            <- s.getAttrVal('id'),
```

```
    ),
    attrbs:SWVR!Attributes(

  ),
    vtext:SWVR!VersionText(),
    desc:SWVR!Description
    (
      binary  <- bin
      --plain   <- s.getPla()
    ),
    bin:SWVR!Binary()
}

rule PartGroups
{
  from
  s:XML!Element
  (
    s.name = 'PartGroups'
  )
  to partGrous:SWVR!PartGroups
  (
    partGroup <- Sequence{s.getChildrenByName('PartGroup')->collect(e | thisModule.resolveTemp(e, 'pgrp'))}
  )
}

---rule to generate the PartGroup of the systemWeaver
rule PartGroup{
  from
    s:XML!Element
    (
      s.name = 'PartGroup'
    )
  to
    pgrp:SWVR!PartGroup
    (
      parts  <- Sequence{s.getChildrenByName('Parts')->collect(e | thisModule.resolveTemp(e, 'parts'))},
      name   <- s.getName(),
      sid    <- s.getAttrVal('sid')


    )
}

-----rule to generate the Parts node in systemweaver user model from the SystemWeaver XML
rule Parts
{
  from
  s:XML!Element
  (
    s.name = 'Parts'
  )
  to
  parts:SWVR!Parts
  (
    part <- Sequence{s.getChildrenByName('Part')->collect(e | thisModule.resolveTemp(e, 'pa'))}-> first()
```

```
rule Part
{
   from
    s:XML!Element(
      s.name = 'Part'
    )

    to
    pa:SWVR!Part
    (
      name        <- s.getName(),
      no          <- s.getAttrVal('no'),
      defObj      <- dob,
      attributes  <- atrbs1, --Sequence{s.getChildrenByName('Attributes')->collect(e | thisModule.resolveTemp(e, 'attrbs1'))},
      ancestor    <- s.getAttrVal('ancestor'),
      id          <- s.getAttrVal('id')
    ),
    dob: SWVR!DefObj
    (
      id   <- s.getAttrVal('id')
    ),
    atrbs1:SWVR!Attributes
    (
      attribute <- Sequence{s.getChildrenByName('Attribute')->collect(e|thisModule.resolveTemp(e, 'attrbute1'))}
    )
}
rule Attribute
{
   from i:XML!Element (i.name = 'Attribute')
   to attribute1:SWVR!Attribute(
       id      <- i.getAttrVal('id'),
       binary  <- 'Binary',
       sid     <- i.getAttrVal('sid')
   )

}

-- rule to generate the topnode of SystemWeaver from the SystemWeaver XML
rule topNode
{
   from
      s:XML!Element
      (
         s.name = 'TopNode'
      )
   to
       top:SWVR!TopNode(
          subNodes <- subnode,
          id <- s.getAttrVal('id')
       ),
    subnode:SWVR!SubNodes(
      node <- Sequence{s.getChildrenByName('Node') -> collect(e|thisModule.resolveTemp(e, 'nod'))}
    )

}

rule Nodes
{
```

SW2EAXML.atl

```
--path =/SWVR2EAXML/modeldiagrams/SWVR.ecore;
--path =/SWVR2EAXML/metamodels/EAST-ADL.ecore;

module SWVR2EAXML; -- Module Template
create  OUT:eastadl from IN:SWVR ;


helper context SWVR!PartGroups def: getStringAttrValue(btName:String) : String =
  let eltC : Sequence(SWVR!PartGroup) =
    self.partGroup->select(a | a.oclIsTypeOf(SWVR!PartGroup) and a.name = btName)->asSequence()
  in
    if eltC->notEmpty()
    then
      eltC->first().name
    else
      ''
    endif;

helper context SWVR!PartGroups def:getPartGroupNames():Set(String) =
  self.partGroup -> collect(e|e.name).asSet();


helper def : isRequiredBT() : SWVR!PartGroup =
    SWVR!PartGroup.name  = thisModule.requiredBT;

helper def:getRequiredBT:SWVR!PartGroups =
  SWVR!PartGroup.allInstances() ->
    select(e|e.name = 'requiredBindingTime')-> first();

helper def:getNames():SWVR!PartGroups =
  SWVR!PartGroup.allInstances() ->
    collect(e| e.name);

helper context SWVR!PartGroups def: getPartGroupNames(name:String):String =
  let nam : String = SWVR!PartGroup.allInstances() -> select(e|e.name = name) in
          if nam ->isEmpty() then
              OclUndefined
            else
              name
          endif;

helper context SWVR!PartGroups def:getPartGroupId(id:String):String =
  let id :String=SWVR!PartGroup.allInstances() ->  select(e|e.sid = id) in
  if id -> isEmpty() then
    OclUndefined
  else
    id
  endif;

helper context SWVR!Parts def:getPartNames(name:String):String=
  let nam :String = SWVR!Part.allInstances() -> select(e|e.name) in
  if nam -> isEmpty() then
    ''
  else
    nam
  endif;

helper context SWVR!partGroups def:getOnedRship(name:String) :String=
```

```
rule Items2childFeature
{
  from i:SWVR!Items
  to  o: eastadl!Elements(
    --group <- 'this is the group',
    feature <- i.item
  )
}

rule Item2Feature
{
  from
  i: SWVR!Item
       (i.partGroups.oclIsTypeOf(SWVR!PartGroups))
  to
  featu: eastadl!Feature
  (
    category                <-  'Feature',
    uaTypeRefs               <- utyp,
    uaValues                <-  uval1,
    ownedComments                <- let cmnt: String =  i.partGroups.partGroup -> select(nam | nam.name = 'OwnedComment')  in
                                    if cmnt ->isEmpty() then
                                      OclUndefined
                                    else
                                      thisModule.ownedcomment(i.partGroups.partGroup)
                                    endif,
    name                    <- i.name,
    traceableSpecificationRefs   <- ttraceableSpecificationRefs,
    ownedRelationships          <- let rln: String =  i.partGroups.partGroup -> select(ors | ors.name = 'ownedRelationShip')  in
                                    if rln ->isEmpty() then
                                      OclUndefined
                                    else
                                      thisModule.pgrou2orelationship(i.partGroups.partGroup)
                                    endif,

    actualBindingTime          <- let pgname : String =  i.partGroups.partGroup -> select(e|e.name = 'actualBindingTime') in
                          if pgname ->isEmpty() then
                            OclUndefined
                          else
                            thisModule.itemtoBT(i.partGroups.partGroup)   --i.partGroups.partGroup-> collect(e| thisModule.itemtoBT(e)) --
                          endif,
    cardinality                <- '1..2',
    childNodes                 <- let pgname : String =  i.partGroups.partGroup-> select(e|e.name = 'childNode') in
                                      if pgname ->isEmpty() then
                                        OclUndefined
                                      else
                                        thisModule.ChildNode(i.partGroups.partGroup)
                                      endif,
    featureParameter           <-  let pgname : String =  i.partGroups.partGroup -> select(e|e.name = 'featureParameter') in
                                      if pgname ->isEmpty() then
                                        OclUndefined
                                      else
                                        thisModule. featureParam(i.partGroups.partGroup)
                                      endif,
    requiredBindingTime         <-  let pgname : String =  i.partGroups.partGroup -> select(e|e.name = 'requiredBindingTime') in
                                      if pgname ->isEmpty() then
```

```
),
utyp:eastadl!UaTypeRefs(
   --group <- 'group',
   uaTypeRef <- utref
),
utref:eastadl!UaTypeRef
(
      value <- 'The correspondence in SystemWeaver is not found',
      dest <-  #USERAttributeElementType
),
uval1:eastadl!UaValues
 (
        --group <- 'group',
        userAttributeValue  <-  let uaval : String = i.partGroups.partGroup -> select(e|e.name = 'userAttributeValue') in
                                    if uaval ->isEmpty() then
                                        OclUndefined
                                    else
                                        thisModule. partGroup2userattrb(i.partGroups.partGroup)
                                    endif
 ),

ttraceableSpecificationRefs:eastadl!TraceableSpecificationRefs(
   name  <- i.partGroups.getPartGroupNames('traceableSpecifications'),
   --group <- 'group',
   traceableSpecificationRef <-tsr
),
tsr:eastadl!TraceableSpecificationRef (
   value <- i.status,
   dest  <- #EAString
)
}
lazy rule chNode
{
  from i:SWVR!PartGroup
  to chNodes:eastadl!ChildNodes
  (
      featureGroup <- thisModule.cNodd(i.parts)

  )
}

lazy rule cNodd
{
  from i:SWVR!Parts
  to   o:eastadl!FeatureGroup
  (
    name <- thisModule.prt(i.part) --i.part -> collect(e|thisModule.ChildNode(e))
  )
}
lazy rule ownedcomment
{
  from i:SWVR!PartGroup
  to    o:eastadl!OwnedComments
  (
      -- comment <- thisModule.comment(i.parts.part)
--      rational < -
  )
}
```

```
--    category    <-
--    uaTypeRefs <-
--    uaValues <-
      name <-  'actualBindingTime',
--    ownedComments
--    kind <-
      uuid <- 'x0000001D040135F9'
   )
}

lazy rule itemtoRT
{
   from i:SWVR!Part
   to    abt:eastadl!BindingTime
   (
         name <-  'requiredBindingTime', -- i.getPartGroupNames('actualBindingTime')
         uuid <-  'x0000001D040135F6'
   )
}

lazy rule partGroup2userattrb
{
   from i:SWVR!Part
   to   uav: eastadl!UserAttributeValue
   (
      --shortName <- i.name,
      --category  <- 'category',
      --uaTypeRefs <- lazy uatyperefs
     --   uaValues <- thisModule.UaValues(),
      name <- 'userAtributeValue', --i.name,
      --ownedComments <- thisModule.ownedcomment(i),
      key  <- 'key',
      value <- 'value',
      uuid <-'x0000001D040135F5'
   )
}

lazy rule pgrou2orelationship
{
   from i:SWVR!PartGroup
   to     o: eastadl!OwnedRelationships(
         --group <- 'Feature',
         deriveRequirement <- dr,
         extend <- ex,
         featureLink <- fl,
         include <- inc,
         multiLevelReference <- mrf,
         realization <- rln,
         refine <- rfn,
         requirementsLink <- rl,
         satisfy <- sat,
         verify <- ver

   ),
   dr:eastadl    ! DeriveRequirement (),
   ex:eastadl    ! Extend(),
   fl:eastadl    ! FeatureLink(
      --shortName <- i.partGroups.getPartGroupNames('')
   )
```

```
}

lazy rule featureParam
{
   from i:SWVR!PartGroup (i.parts.oclIsTypeOf(SWVR!Parts))
   to   o:eastadl!EaDatatypePrototype
   (
        shortName <-i.getPartGroupNames('featureParameter'),
        category  <- 'category',
        name  <- i.parts.part.name, --'featureParameter',
        uuid <-i.id
   )
}

lazy rule ChildNode{

   from i:SWVR!PartGroup(i.parts.part.oclIsTypeOf(SWVR!Part))
   to     o :eastadl!ChildNodes
   (
        --group         <- 'group',
     --feature       <- thisModule.Item2Feature(),
     featureGroup   <- fg, -- i -> collect(e|thisModule.fgroup(e)), -- thisModule.fgroup(),
     vehicleFeature <- i-> collect(e|thisModule.prt(e))

   ),
   fg:eastadl!FeatureGroup
   (
   --cardinality   <- i.cardinality,
      childFeatures <- cf,
      name <- 'childNodes' ,
      uuid <- 'x0000001D040135F5'
   ),
   cf:eastadl!ChildFeatures(

      --group  <-
      --feature  <-
    vehicleFeature <-  i -> collect(e|thisModule.prt(e)) --thisModule.prt(i)
   )



   }
lazy rule chilGroup
{
   from i:SWVR!PartGroup(i.parts.oclIsTypeOf(SWVR!Parts))
   to   o:eastadl!FeatureGroup
   (
        name <- i.parts.part.name

   )
}

lazy rule prt
{
   from i:SWVR!Part
   to     o: eastadl!VehicleFeature
   (
      shortName <- i.name,
      category <- 'category',
      --uaTypeRefs <- uaTypeRefs
      --uaValues <- uval,
       name  <- i.name,
--     let nam: String = SWVR!PartGroups -> select(e| e.partGroup.name = 'childNode') in
--           if nam -> isEmpty() then
--             OclUndefined
```

```
--              i.name
--          endif, -- -- i.partGroups.getPartGroupNames('childNode'),
      --ownedComments  <-
      --ownedRelationShips
      --traceableSpecificationRefs
      --actualBindingTime
      cardinality  <- '1..2',
      --childNodes  <- thisModule.ChildNode(i),
      --featureParameter
      --requiredBindingTime
      --deviationAttributeSet
      isCustomerVisible <- 'true',
      --isDesignVariabilityRationale <-
      isRemoved <-  'true',
      uuid <-  'x0000001D040135F5'
   )
}

lazy rule fgroup
{
   from i:SWVR!partGroup
   to   o: eastadl!FeatureGroup
   (
      --cardinality   <- i.cardinality,
      childFeatures <- cf,
      name <- i.name ,
      uuid <- 'x0000001D040135F5'
   ),
   cf:eastadl!ChildFeatures(

      --group  <-
      --feature  <-
    vehicleFeature <-  i -> collect(e|thisModule.prt(e)) --thisModule.prt(i)
)
}

lazy rule Item2Feature
{
   from
   i: SWVR!Item
         (i.partGroups.oclIsTypeOf(SWVR!PartGroups))
   to
   featu: eastadl!Feature
   (
      category                <- 'Feature',
      uaTypeRefs                <- utyp,
      uaValues                <- uval1,
      ownedComments                <- desc,
      name                  <- i.name,
      traceableSpecificationRefs    <- TraceableSpecificationRefs,
      ownedRelationships          <- ownedrship,
      actualBindingTime          <- let pgname : String = SWVR!PartGroup.allInstances() -> select(e|e.name = 'actualBindingTime') in
                              if pgname ->isEmpty() then
                                 OclUndefined
                               else
                                 thisModule.itemtoBT(i.partGroups.partGroup)
                              endif,
      cardinality                <- '1..2'
```

EAXML2SW.atl

```
module EAXML2SWVR; -- Module Template
create OUT:SWVR  from IN:eastadl ;



--------------------------------HELPER---------------------------------
--helper context eastadl!Feature def:getAllItemNames():String
-- let names: String -> eastadl!Feature or eastadl!requiredBindingTime or eastadl!actualBindingTime

--helper context eastadl!Feature def:getNames(name:String):String =
-- let nam :String=eastadl!BindingTime.allInstances() ->  select(e|e.name = nam) in
-- if nam -> isEmpty() then
--    OclUndefined
-- else
--    nam
-- endif;


--------------------------------RULES---------------------------------
rule Items
{
   from i:eastadl!Elements
   to o:SWVR!Items(
      item <- i.feature,
      item <- let nam : String = eastadl!BindingTime.allInstances() -> select(e|e.name = 'requiredBindingTime') in
            if nam ->isEmpty() then
                OclUndefined
            else
               thisModule.RequiredATItem()
            endif,
      item <- let cnod: String = eastadl!BindingTime.allInstances() -> select(e|e.name ='actualBindingTime')in
            if cnod -> isEmpty() then
               OclUndefined
            else
               thisModule.ActualBTItem()
            endif,
      item <-  let ondC: String=eastadl!FeatureGroup.allInstances() -> select(e|e.name  =  'childNodes') in
                  if ondC -> isEmpty() then
                     OclUndefined
                  else
                     thisModule.ChildNodes2Item()
                  endif
--item <-  let userattb: String=eastadl!UserAttributeValue.allInstances() -> select(e| e.name = '') in
--                ,
--    item <- let reship:Sequence(eastadl!OwnedRelationships)= i.feature.ownedRelationships -> select(e| e= 'ownedRelationships') in
--             if reship = i.feature.oclIsTypeOf(eastadl!OwnedRelationships) then
--                'OwnedRelationships'
--             else
--                OclUndefined
--             endif
      )

}

rule feature2swvr
{
   from i:eastadl!Feature
   to it:SWVR!Item
   (
        name              <- i.name,
        ancestor          <- i.uuid,
        id                <- i.uuid,
         status             <- i.shortName,
         description          <- desc, --Sequence{i.ownedComments} -> collect(e|thisModule.resolveTemp(e, 'desc')), --description,
         attributes         <- thisModule.Attribute(),
         partGroups          <- prtGroups,
         topNode            <-  topn,
```

```
        versionNumber       <- '1',
        sid             <- 'ASGD'
    ),
    prtGroups:SWVR!PartGroups(
        partGroup <- let nam : String = eastadl!BindingTime.allInstances() -> select(e|e.name = 'requiredBindingTime') in
                     if nam ->isEmpty() then
                       OclUndefined
                     else
                        thisModule.RequiredAT(i.requiredBindingTime)
                     endif,
        partGroup <- let abt: String = eastadl!BindingTime.allInstances() -> select(e|e.name ='actualBindingTime')in
                     if abt -> isEmpty() then
                       OclUndefined
                     else
                        thisModule.ActualBT(i.actualBindingTime)
                     endif,
        partGroup <-  let chnod: String = eastadl!FeatureGroup.allInstances()-> select(e|e.name  =  'childNodes') in
                      if chnod -> isEmpty() then
                         OclUndefined
                      else
                        thisModule.ChildNodes2(i.childNodes)
                      endif
--      partGroup <- let ondC: Sequence(eastadl!OwnedComments) = eastadl!Feature.allInstances() -> select(e|e.childNodes = 'OwnedComments') in
--                   if ondC -> isEmpty() then
--                     OclUndefined
--                   else
--                     thisModule.ChildNode2Item()
--                   endif
    ),
    desc:SWVR!Description(
        binary <- bin,
        plain  <- 'plain'
    ),
    bin:SWVR!Binary
    (),
    topn:SWVR!TopNode(
      subNodes <- sbnd,
      id <- i.uuid
    ),
    sbnd:SWVR!SubNodes
    (
      node <- nd
    ),
    nd:SWVR!Node
    (
        path            <- 'path',
        attributes      <- thisModule.Attribute(),
        subNodes        <- sbnd,
        id              <- i.uuid,
        itemId          <- i.uuid,
        partId          <- i.uuid
    )
}

lazy rule ChildNodes2
{
    from i:eastadl!ChildNodes --(i.featureGroup.oclIsTypeOf(eastadl!FeatureGroup))
    to   o:SWVR!PartGroup
    (
      name <- 'childNode',
      parts <- prts, -- i.featureGroup, -- -> select(e|thisModule.resolveTemp(e, 'chiildNd')),
      sid <- 'SIID'
      ),
      prts:SWVR!Parts
      (
         part <-prt
      ),
```

```
    prt:SWVR!Part
    (
      name <-thisModule.ChileNodes2group( i.featureGroup)
    )
}

lazy rule childParts
{
    from i:eastadl!VehicleFeature
    to  o:SWVR!Parts
    (
      --name <- i.name --part <- thisModule.ChildNodes2group(i)
    )
}
lazy rule RequiredAT
{
    from i: eastadl!BindingTime
    to pgrp2:SWVR!PartGroup
    (
      name <- let nam : String = eastadl!BindingTime.allInstances() -> select(e|e.name = 'requiredBindingTime') in
            if nam ->isEmpty() then
                OclUndefined
              else
                i.name
            endif,
      sid  <- 'RAPT',
      parts<-  prts
    ),
    prts:SWVR!Parts
    (part <- prt),
    prt:SWVR!Part
    (
          name <- i.name,
          no <- '',
          defObj <- dob,
          attributes <- attrb,
          ancestor <- i.uuid,
          id <-  i.uuid
    ),
    dob:SWVR!DefObj
    (
      id  <-   i.uuid
    ),
    attrb:SWVR!Attributes(
      attribute <- atrb
    ),
    atrb:SWVR!Attribute
    (
          binary    <- 'binary',
          id       <-  i.uuid,
          sid      <-  'SIDD'
    )
}

lazy rule ActualBT
{
    from i: eastadl!BindingTime
    to pgrp2:SWVR!PartGroup
    (
      name <- let nam : String = eastadl!BindingTime.allInstances() -> select(e|e.name = 'actualBindingTime') in
                if nam ->isEmpty() then
                OclUndefined
              else
                i.name
            endif,
      sid  <- 'RAPT',
```

```
    parts<-  prts
  ),
  prts:SWVR!Parts
  (part <- prt),
  prt:SWVR!Part
  (
       name <- i.name,
       no <- '',
       defObj <- dob,
       attributes <- attrb,
       ancestor <- i.uuid,
       id <-  i.uuid
  ),
  dob:SWVR!DefObj
  (
    id  <-   i.uuid
  ),
  attrb:SWVR!Attributes(
    attribute <- atrb
  ),
  atrb:SWVR!Attribute
  (
       binary    <- 'binary',
       id       <-  i.uuid,
       sid      <-  'SIDD'
  )
}

lazy rule RequiredATItem
{
  from i: eastadl!BindingTime
  to pgrp2:SWVR!Item
  (
    name <- let nam : String = eastadl!BindingTime.allInstances() -> select(e|e.name = 'requiredBindingTime') in
         if nam ->isEmpty() then
              OclUndefined
           else
             'requiredBindingTime'
         endif,
    sid  <- 'SID2',
    id <-  'x696758504',
    versionText   <- '1',
    versionNumber <- '1',
    status <- '0',
    description <-desc,
    ancestor   <-  'x696758504',
    topNode <- topn

  ),
  desc:SWVR!Description(
       binary <- bin,
       plain  <- 'plain'
  ),
  bin:SWVR!Binary
  () ,
    topn:SWVR!TopNode(
    subNodes <- sbnd,
      id <-  'x696758504'
    ),
    sbnd:SWVR!SubNodes
    (
      node <- nd
    ),
    nd:SWVR!Node
    (
         path           <- 'path',
```

```
        attributes    <- thisModule.Attribute(),
        subNodes      <- sbnd,
        id            <-  'x696758504',
        itemId        <-  'x696758504',
        partId        <-  'x696758504'

    )

}

lazy rule ActualBTItem
{
    from i: eastadl!BindingTime
    to pgrp2:SWVR!Item
    (
        name <- let nam : String = eastadl!BindingTime.allInstances() -> select(f|f.name = 'actualBindingTime') in
            if nam ->isEmpty() then
                OclUndefined
            else
                'actualBindingTime'
            endif,
        sid                 <- 'SID2',
        id                  <- 'x696758504',
        versionText         <- '1',
        versionNumber       <- '1',
        status              <- '0',
        description         <- desc,
        ancestor            <- 'x696758504',
        topNode             <- topn

    ),
    desc:SWVR!Description(
        plain <- 'plain'
    ),
        topn:SWVR!TopNode(
        subNodes <- sbnd,
        id <-  'x696758504'
    ),
    sbnd:SWVR!SubNodes
    (
        node <- nd
    ),
    nd:SWVR!Node
    (
        path          <- 'path',
        attributes    <- thisModule.Attribute(),
        subNodes      <- sbnd,
        id            <-  'x696758504',
        itemId        <-  'x696758504',
        partId        <-  'x696758504'
    )
}

 lazy rule ChildNodes2Item
{
    from s:eastadl!ChildNodes
    to childNd:SWVR!Item(
        name        <- 'childNode',
        attributes  <- thisModule.Attribute()
    )
}
```

```
{
   from s:eastadl!FeatureGroup
   to childNd:SWVR!Part(

         name      <- s.name,
         no          <- '',
         defObj     <- dob,
         attributes <- attrb,
         ancestor   <- s.uuid,
         id             <-  s.uuid
   ),
   dob:SWVR!DefObj
   (
      id  <-    'x696758504'
   ),
   attrb:SWVR!Attributes(
      attribute <- atrb
   ),
   atrb:SWVR!Attribute
   (
         binary     <-  'binary',
         id          <-  'x696758504',
         sid         <-  'SIDD'
   )

--       parts <- prts
--    ),
--    prts:SWVR!Parts(
--       part <- prt
--    ),
-- prt:SWVR!Part
-- (

-- childNd:SWVR!PartGroup(
--       name <- 'childNode',
--       sid  <- 'SID2',
--       parts <- prts
-- ),
--
-- prts:SWVR!Parts(part <- prt),

}

lazy rule OwnedComment2Item
{
   from inp:eastadl!OwnedComments
   to  target:SWVR!Item(
      name <- 'OwnedComment'
   )
}

 lazy rule Attribute
{
   from inp:eastadl!ChildNodes
   to out: SWVR!Attributes(
      attribute <- atrb
   ),
   dob:SWVR!DefObj
   (
      id  <-  'x88'-- inp.uuid
   ),
   attrb:SWVR!Attributes(
      attribute <- atrb
   ),
```

```
atrb:SWVR!Attribute
(

        binary    <- 'binary',
        id        <- 'x88',-- inp.uuid,
        sid       <-  'SIDD'
  )
}
```

SW2XML.atl

```
module SWVR2XML; -- Module Template
create OUT:XML from IN:SWVR ;

-------------------------------------------------------------------------------

-- Rule 'RuleBase'
-- Create XML Root element (r2ml:RuleBase) from the R2ML RuleBase element
rule SystemWeaver {
   from i : SWVR!Items (
      i.oclIsTypeOf(SWVR!Items)
      )
   to o : XML!Root (
         name <- 'SystemWeaver3.0',
         children <- Sequence { itm --xm, xmln,xml
                     }
      ),

   itm : XML!Element(
         name <- 'Items',
         children <- Sequence{ i.item}
   )
}

rule Item
{
   from i:SWVR!Item
   to o:XML!Element
   (
      name <- 'Item',
      children <- Sequence {atrb,atrb2,ances,  itmn,versT, versN, sta, i.attributes, i.description, i.partGroups, i.topNode}
   ),
   atrb:XML!Attribute
   (
      name <- 'id',
      value <- i.id
   ),
      atrb2:XML!Attribute(
      name <- 'sid',
      value <- i.sid
   ),
      ances:XML!Attribute(
      name <- 'ancestor',
      value <- i.id
   ),
      itmn:XML!Element(
      name <- 'Name',
      children <- tex
   ),
   tex:XML!Text(
      name <- '#Text',
      value <- i.name
   ),
      versT:XML!Element(
      name <- 'VersionText',
      children <- vtext
   ),
  vtext:XML!Text(
      name <-    '#Text',
      value <- ''
  ),
  versN:XML!Element(
      name <- 'VersionNumber',
      children <- vn
  ),
```

```
  value <- '1'
 ),
 sta:XML!Element
 (
     name <- 'Status',
     children <- stat
 ),
  stat:XML!Text(
     name <-    '#Text',
     value <- 'I'
     )
}
rule Attrb
{
   from i:SWVR!Attributes
   to     o: XML!Element(
            name <- 'Attributes',
            children <-  i.attribute
   )

}

rule Attribute{
   from i:SWVR!Attribute
   to    o: XML!Element(
      name     <- 'Attribute',
      children <- Sequence{sid, id, binary}
      ),
      sid:XML!Attribute(
            name <- 'sid',
            value <- i.sid
      ),
      id:XML!Attribute(
         name <- 'id',
         value <- i.id
      ),
      binary :XML!Attribute
      (
            name <- 'binary',
            value <- i.binary
      )

}


rule Descrip
{
   from i:SWVR!Description
   to    o:XML!Element(
      name <- 'Description',
      children <-  plain
   ),
   plain:XML!Element
   (
      name <- 'plain',
      value <- i.plain
   )
}


rule PartGroups
{
   from i:SWVR!PartGroups
   to o:XML!Element(
```

```
    )
}

rule partGroup
{
    from i: SWVR!PartGroup
    to   o: XML!Element(
        name  <- 'PartGroup',
        children <-Sequence{nam, sid, i.parts}
    ),
    nam:XML!Element(
        name <- 'Name',
        children <- nm
    ),
    nm:XML!Text
    (
        name <- '#Text',
        value <- i.name
    ),
    sid:XML!Attribute(
        name <- 'sid',
        value <- i.sid
    )

}

rule parts
{
    from i:SWVR!Parts
    to   o: XML!Element
    (
        name     <-  'Parts',
        children <-  i.part
    )
}

rule part
{
    from i:SWVR!Part
    to     o:XML!Element
    (
          name <- 'Part',
          children <- Sequence{nam, no, i.defObj, i.attributes, ances, id}
    ),
    nam:XML!Element
    (
        name <- 'Name',
        children <- txt
    ),
    txt:XML!Text(
        name <- '#Text',
        value <- i.name
    ),
    no:XML!Element(
        name <- 'No',
        value <- '1'
    ),
    ances :XML!Attribute
    (
        name <- 'ancestor',
        value <- i.ancestor
    ),
    id:XML!Attribute
    (
```

```
    name    <- 'PartGroups',
    children <-  i.partGroup
```

```
    name <- 'id',
    value <- i.id
```

```
  )
}

rule DObj
{
  from i:SWVR!DefObj
  to     o: XML!Element(
         name <- 'DefObj',
         children <- idd
  ),
  idd :XML!Attribute
  (
    name <- 'id',
    value <- i.id
  )

}

rule topNode
{
  from i:SWVR!TopNode
  to     o: XML!Element
  (
         name   <- 'TopNode',
         children <- Sequence{id, i.subNodes}
  ),
  id:XML!Attribute
  (
    name <- 'id',
    value <-  i.id
  )
}
```