University of Twente
The Netherlands

Generating XML-to-RDF transformations from high level specifications

M. Sc. Telematics Final Project

Muzaffar Mirazimovich Igamberdiev

Graduation committee:
Prof. Dr. Ir. M. Aksit
Dr. Ir. K.G. van den Berg
I. Kurtev, MSc

Software Engineering Group
Departmentof Computer Science
University of Twente
The Netherlands
June 2004

[This page is intentionally blank]

Acknowledgement

This report was produced to complete the graduation phase at the end of my MSc Telematics Program at the University of Twente. It illustrates the final project period at the Software Engineering group at the University of Twente in Enschede in the Netherlands.

I would like to express thanks to my supervisors Klaas van den Berg and Ivan Kurtev for giving me the opportunity to do this project. I would also like to thank them for their help; guiding, support and enthusiasm which made this project the enjoyment.

Special thanks go to my dear parents and sisters, who educated, supported and gave me this opportunity. I am very grateful for their advices, help, support, great patient and love.

I would like to thank individually to my dear teacher Mehmet Aksit and dear friends Malohat Kamilova, Dossay Oryspayev, Arif Colakkadioglu, Zafer Bozlak and Hans Daemen for their support, help and for nice atmosphere.

Abstract

Today's XML technologies of software industry define the syntax of data and do not address the problem of bridging between that syntax and the meaning of data. This problem can be solved by defining data as knowledge structures and by interpreting them as RDF data. To comply with the needs of the problem, a transformation from XML document to RDF document is accomplished by the project. It makes use of model-to-model transformation by MDA. Source and target models are the schemas of XML and RDF respectively. OMG's meta-modeling approach is used to define the relationships between models. The transformation is implemented using XSLT stylesheets; because both input (XML) and output (RDF) documents are represented in XML. The purpose of this project is to automate the generation of these stylesheets. To automate the transformation a high level transformation specification, that defines the mapping between source and target meta-model, is used. This high level transformation is also expressed as an XML document. An XSLT stylesheet is used to generate other XSLT stylesheet that actually performs the transformation. This solution could potentially lend a hand to transform automatically from the syntax (XML) to the meaning (RDF) of data. The approach is illustrated in a case study.

Abbreviations

EBNF – Basic Extended Backus-Naur Form
DTD – Document Type Definition
MDA – Model Driven Architecture
MOF – Meta Object Facility
OMG – Object Management Group
RDF – Resource Description Framework
PIM – Platform Independent Model
PSM – Platform Specific Model
UML – Unified Modeling Language
W3C – World Wide Web Consortium
XML – eXtensible Markup Language
XPath – XML Path Language
XSLT – eXtensible Stylesheet Language Transformation

TABLE OF CONTENTS

1 Introduction

This chapter contains an introduction to the project. First, the background of the project is explained in section 1.1. Then, the problem statement and the objectives of the project are expressed in sections 1.2 and 1.3 respectively. Next, the solution approach is described in section 1.4. Section 1.5 contains the building blocks, concepts for this project, such as XML, RDF, MDA, meta-modeling and XSLT. The final section 1.6 of this chapter includes an overview of this report.

1.1 Background

Many software applications use XML (eXtensible Markup Language) as a data exchange format in different domains. An important characteristic of some applications is that they are focused in how to describe data structure in meaningful way, such as the way that shows up the essential meaning of data rather than syntax that encodes it.

XML defines the structure, syntax of data, but it does not address the meaning of data. RDF (Resource Description Framework) defines meaning of data as Object and Properties in terms of RDF triples, this means that Subject, Object and Predicate. XML is a mark-up language and much more concentrated on syntax of the data, but RDF concentrated on the meaning of data. Most of current XML technologies define document syntax and do not address the problem of bridging between that syntax and the meaning of data. As a result, a number of approaches appeared that provide solutions for this problem and serve different purposes.

1.2 Problem Statement

The main goal of the project is to automate the generation of the transformation stylesheets. To cope with this goal, we explore the source (XML) and target (RDF) models. Both XML and RDF documents are represented in XML format; therefore we can use XSLT for transformations. By feeding XML as an input to XSLT processor we are going to get RDF document as an output (figure 1.1). As shown in figure 1.1, XSLT processor is used to process the transformation specification and transform XML (input) to RDF (output) document. The problem in the aforementioned transformations is that, we need to write XSLT (eXtensible Stylesheet Language Transportation) stylesheets manually to each XML-to-RDF transformations. The need to automate the process expresses the importance of the problem. In this project we focus on XML-to-RDF transformations.

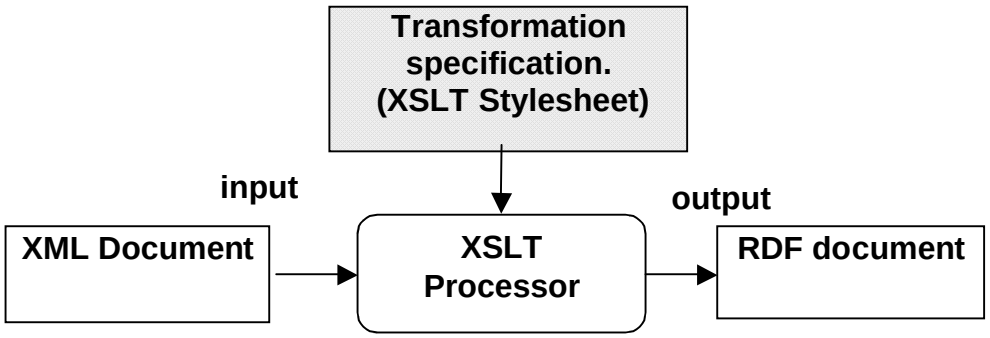


Figure 1.1. Transformation from XML to RDF document

A transformation specification in Figure 1.1 is only valid to a particular XML document. We need the transformation specification; which can transform any XML document to RDF document. In order to do such transformation we should know the structure, grammar of the input and output documents. That's why we have introduced the schemas of input and output document in figure 1.2.

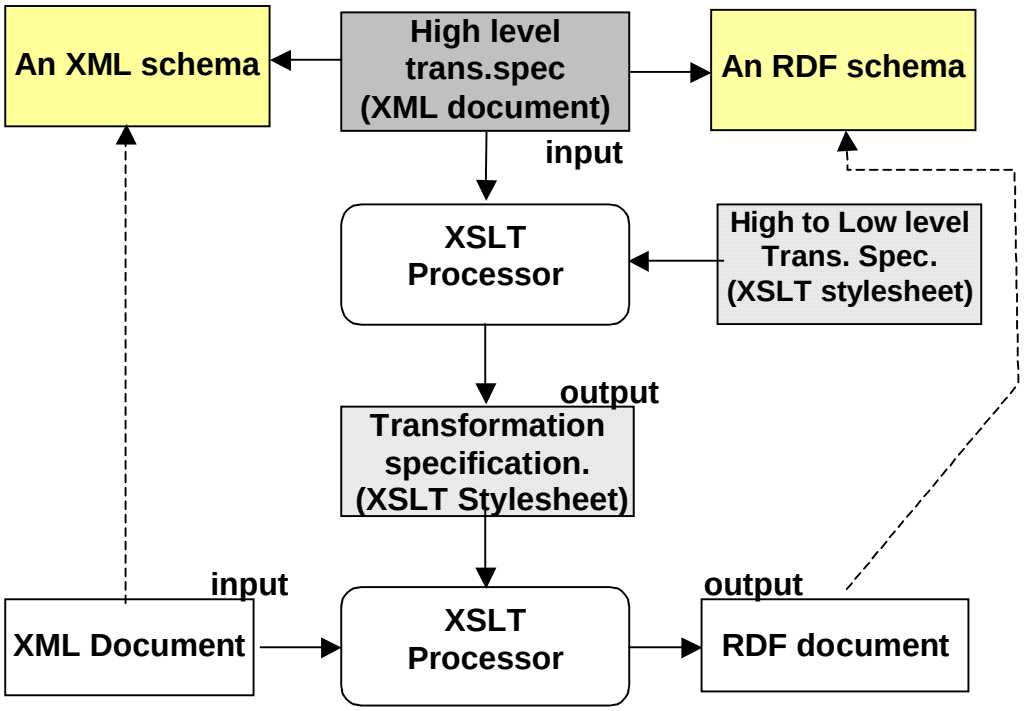


Figure 1.2 XML to RDF transformation from High level transformation specification

The source and target models and their instances and the transformations artifacts can be seen in figure 1.2, which is the main figure of our project.
An XML schema and an RDF schema define the grammar of XML and RDF documents respectively. It means XML and RDF documents are the instance of an XML schema and RDF schema correspondingly. You can see these relationships in figure 1.2, as dashed lines.

To comply with the need of automation, we should auto generate XSLT stylesheets (transformation specification). In order to generate XSLT stylesheets automatically, we come up with a solution, that XSLT stylesheets will be generated by other stylesheets. To make enable the generation of the stylesheets, we propose the mapping between the grammars (schemas) of XML and RDF documents (figure 1.2). In other words, we define a high level transformation specification that addresses aforementioned mapping between XML and RDF schemas. High level transformation specification addresses the mapping of XML schema components to RDF schema components. This kind of transformation logic is expressed in XML format. Based on this transformation logic aforementioned transformation specification generated automatically by 'High to Low Level Transformation Sepcification', which is also XSLT stylesheet.

Transformation from source to target fits in MDA approach. In figure 1.2 we use term '*level*' to refer to the meta-modeling layers. Low level is the M0 level, where the instance of source (XML) and target (RDF) models are defined. With 'High level' term, we refer to M1 layer, where schemas of input (XML) and output (RDF) documents are situated. In this high level (M1 level), the mapping or relationship between these schemas are defined in 'High level transformation specification' in XML document. That's why we have called it '*High level*' transformation specification.

A XSLT stylesheet, which auto generate transformation specification based on the high level transformation specification, called 'High to Low level Transformation Specification', because it actually transforming from the high level (M1) to the low level (M0) transformation specification.

This project aims at exploring how the transformations specified in the high level transformation specification can be implemented in XSLT. A case study based on a concrete source schema and a target RDF schema illustrates the applicability of the approach.

Figure 1.2 is the straightforward one that describes the fundamental effort of this project. It is based on model-to-model transformation notion, which states any input document, an instance of source model will be transformed to the output document, that conform to the target Model. In our case, source and target models are XML and RDF schemas respectively based on particular domain.

1.3 Goals

The goals of this project based on problem statement are:

1. To define a high level transformation specification, that addresses mapping between source (XML schema) and target (RDF schema) models.
2. To perform XML-to-RDF transformation, which is a model-to-model transformation, based on the high level transformation specification.

1.4 Solution Approach

To solve the problem stated above, the following approach is used:

1. Analyze and give concrete examples to source model, its instances and meta-model.
2. Analyze and give concrete examples to target model, its instances and meta-model.
3. Introduce a high level transformation specification between source and target models
4. Generate XML-to-RDF transformation based on the aforementioned transformation specification.

We propose an approach in which a high level specification of a transformation between the source and the target model is used to generate the stylesheet. The generation depends on the concrete XML representation of the instances of the target model. Figure 1.2 represents this approach.

In this approach a transformation specification between an XML schema and an RDF schema is defined as an XML document. This specification is used to be transformed by an XSLT stylesheet (that plays the role of a transformation compiler) to another stylesheet that specifies the transformations from an input XML document to a RDF document (Figure 1.2).

1.5 Concepts

This section consists of the fundamental building blocks of the project. In section 2.1 and 2.2 basic concepts of XML and its validation are introduced respectively. Section 2.3 discusses about RDF. XPath and XSLT are discussed in section 2.4. Section 2.5 gives short overview of MDA. Last section concludes with a summary of this chapter.

1.5.1 XML

XML stands for eXtensible Markup Language. It is W3C recommendation [16]. XML is a markup language to describe data. XML tags are not predefined; user must define his own tags. The freedom of defining the own tag is given to user. The grammar of XML document can be specified in a DTD and XML Schema. XML was created to structure, store and to send information.

There are various applications of XML, ranging from XMLDB, Semantic Web [15] to Web-services format. XML will be widely used in future Web development

1.5.2 Validation of XML documents

To validate an XML document, DTD and XML Schemas can be used. If a document follows the rules defined in DTD or XML schema, it is considered to be a valid document.

1.5.2.1 DTD

DTD stands for Document Type Definition. A DTD can be used to specify the structure of an XML document and to specify what the valid names are of the elements used in the document, their order, number of occurrences allowed and their attributes. However, DTD does not allow one to specify that the content should be of type int, byte, or other simple types. In addition one is not able to specify that values must be within a certain range.

1.5.2.2 XML schema

Another schema language specified by the W3C is XML Schema [18]. XML schema offers facilities for describing the structure and constraining the contents of an XML document. In addition the XML Schema comes with a set of predefined datatypes (e.g., string, byte, long), named simple types, which can be used to constrain the values of attributes and elements. Constraints like numeric ranges and checks on the content itself can also be specified. Custom defined data types (simple and complex) can also be defined.

1.5.3 RDF

The Resource Description Framework (RDF) is a language for representing information about web resources, such as content description, title, author, copyright information, availability schedules, and more. RDF is for describing resources on the web. And it was designed to be read by machines, computers and was not for being displayed to people. RDF uses resources, properties and values to define the meaning of the data. It defines them in terms of RDF-triples Subjects, predicates and objects. RDF is also represented in XML format. RDF became a W3C (World Wide Web Consortium) Recommendation in February 2004. Like XML, the structure or grammar of RDF is defined in RDF Schema.
The examples of use of RDF:

- Describing properties for shopping items, like price and availability
- Describing time schedules for web events
- Describing information about web pages, like created and modified date, title, and author
- Describing content and rating for web pictures
- Describing content for search engines
- Describing electronic libraries

1.5.4 XPath and XSLT

XPath is a language for addressing parts of an XML document. It is a major element in the W3C XSLT standard. Without XPath knowledge you will not be able to create XSLT documents. XPath uses paths to address XML elements.

XPath uses path expressions to identify nodes in an XML document. These path expressions look very much like the expressions you see when you work with a computer file system. It defines a library of standard functions for working with strings, numbers and Boolean expressions. XPath was released as a W3C Recommendation 16th November 1999 as a language for addressing parts of an XML document. XPath was designed to be used by XSLT, XPointer and other XML software.

The World Wide Web Consortium (W3C) started to develop XSL because there was a need for an XML based Stylesheet Language. XSLT is a language for transforming XML documents into other XML documents or other formats. XSLT can also add new elements into the output file, or remove elements. It can rearrange and sort elements, and test and make decisions about which elements to display, and a lot more.

A common way to describe the transformation process is to say that XSLT transforms an XML source tree into an XML result tree. XSLT uses XPath to define the matching patterns for transformations. In the transformation process, XSLT uses XPath to define parts of the source document that match one or more predefined templates. When a match is found, XSLT will transform the matching part of the source document into the result document. The parts of the source document that do not match a template will end up unmodified in the result document. XSLT became a W3C Recommendation in 16. November 1999. There is already XPath 2.0 [6] and XSLT 2.0 Working Draft [7], with many new features. We should see Xpath 2.0, XSLT 2.0 become an official W3C Recommendation sometime this year.

1.5.5 MDA

MDA stands for Model Driven Architecture and is defined by OMG group. The MDA is a new way of writing specifications and developing applications, based on platform-independent models (PIM). A complete MDA specification consists of a definitive platform-independent base UML® model, plus one or more platform-specific models (PSM) and interface definition sets, each describing how the base model is implemented on a different middleware platform. A complete MDA application consists of a definitive PIM, plus one or more PSMs and complete implementations, one on each platform that the application developer decides to support. For the sake of simplicity, we use MDA only as source and target, without referring to PIM and PSM in this project.

1.5.6 Meta- modeling

To describe the meta-modeling we refer to Metadata Architecture of the Meta Object Facility (MOF) [14]. The architecture is based on a four layers model where the layers are related to each other in the following way: the layer above another describes the layer directly below it, while the one directly underneath another is an instance of the one above it. This can be seen in figure 1.3.

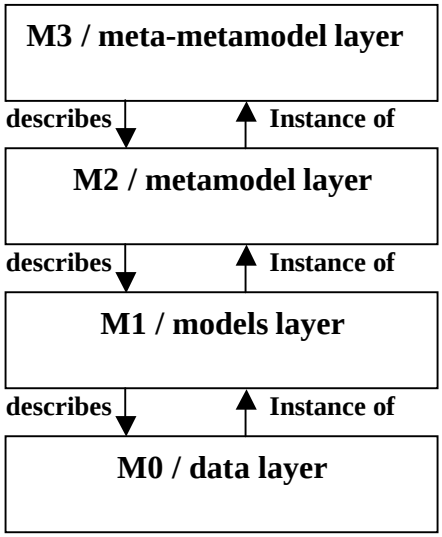


Figure 1.3 The MOF MetaData Architecture

The lowest layer, referred to as M0, contains the information one wishes to describe. The layer describing the information on the data layer is referred to as M1, while the description defining the model located in the M1 is referred to as M2. All layers part of the metadata architecture are described in table1.1.

Description of the architectural layers of the MOF - Table 1.1

Level	Layer	Description
M3	Meta-metamodel (MOF)	Description of the structure of meta-metadata
M2	Metamodel	The description that defines the structure of the metadata in the M1 layer
M1	Model	The metadata that describes the data in the M0 layer
M0	Information	The data one wishes to describe

1.6 Outline

The structure of the report is organized in a kind of way, which first defines the source and target artifacts, than describes the transformations between

them and discusses transformation generation issues. Thus, the report is organized as follows:

In Chapter 2, we define what we have and what we are going to get after the transformation. In other words, it contains the definitions, particular examples and representations for each source and target artifact.

Chapter 3 concentrates on High Level Transformation Specification (artifact 4), which addresses the mapping between source (artifact 2) and target (artifact 9) models, which are XML and RDF schema respectively, and their meta-models. The source (artifact 1) and target (artifact 8) meta-models lend a hand to define the Transformation Grammar (artifact 16).

Chapter 4 discusses about transformation specification issue. It contains the low level transformation specification (artifact 6) and artifact 15, which makes enable auto generation of artifact6. That's why it is called 'high to low level transformation specification', which automates the process of XSLT stylesheets generating.

Chapter 5 consists of the summary of the project, the comparison with related work and the suggestions to future research.

2 Source and Target Artifacts

The chapter consists of definitions and concrete examples for each source and target artifact. The transformation specification artifacts will be discussed in chapter 3 and 4. Section 2.1 gives the specific examples to source and target artifacts. This chapter concludes with summary section 2.2.

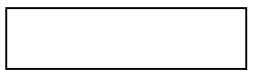
2.1 Artifact definitions and examples

In this section we will define and give examples to source and target artifacts, in order to come up with the mapping between XML and RDF schemas.

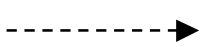
Figure 2.0 is one of the important figure of our project and we refer to this figure further in this report. In this figure, we defined artifacts, to which we are going to refer by their numbers. In order to cover the research topic in detail, we introduced new artifacts as shown in Figure 2.0. During the report we will refer to the figure 2.0. Figure 2.0 is the extended version of figure 1.2 and all artifacts are numbered and several relationships between the artifacts are enumerated. These artifacts will be explained in chapter 3 and 4.

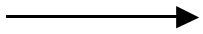
Definition of figure elements of figure 2.0

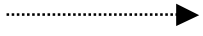
Here, we describe the graphical objects used in the figure:

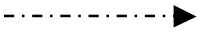

- Artifacts


- Processor


- instance of, in terms of metamodelling


- relationships between artifacts, based on transformation


- instance of, in terms of RDF Data model


- Representation relationships


- self describing of metamodel.

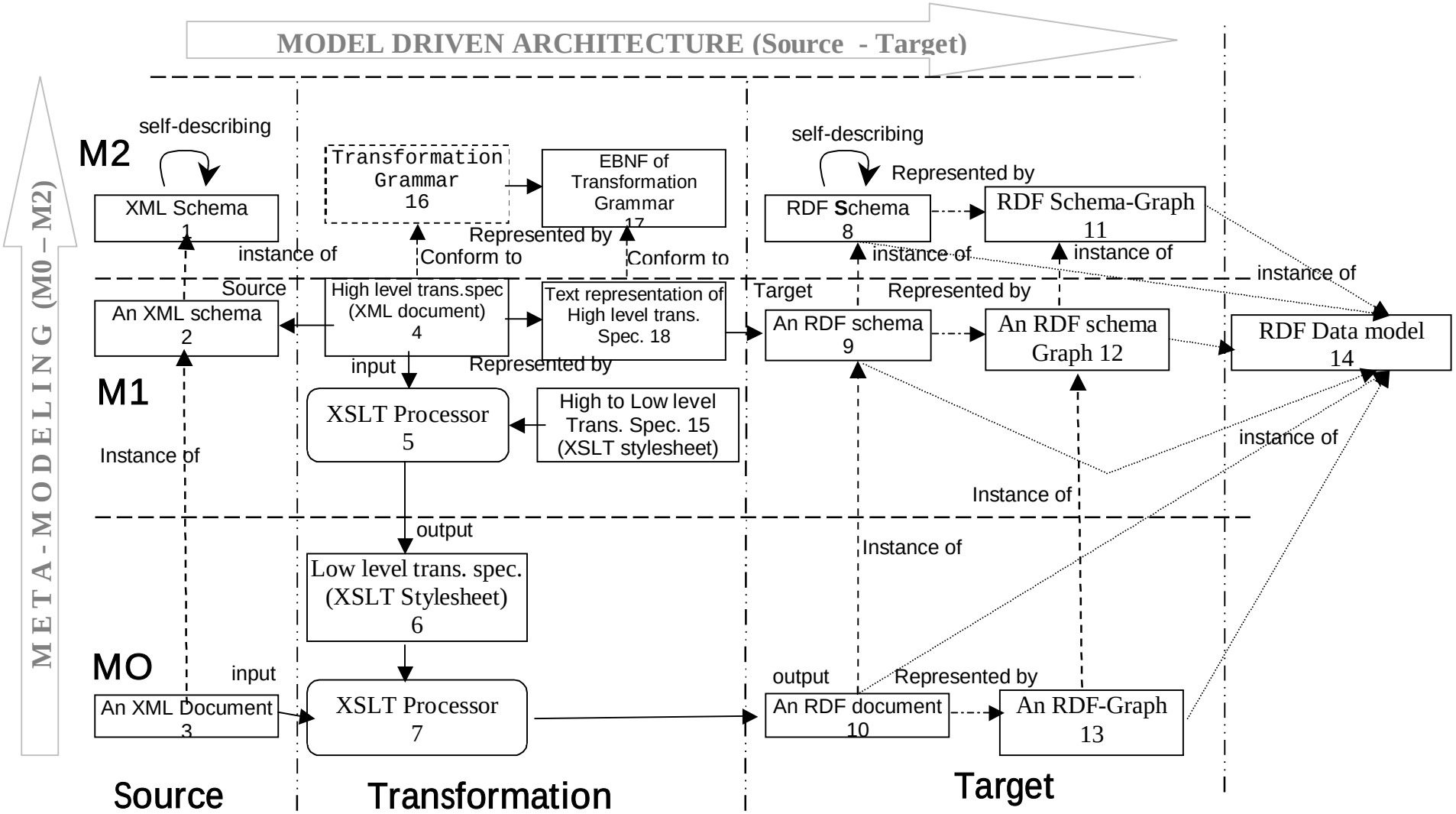


Figure 2.0. Generation of a Transformation specifications

The main figure of our project depicts all artifacts at layers M0, M1 and M2. In M0 layer, input and output and their representations are defined. The schemas of these documents defined in M1 model layer. Meta-models like XML, RDF schemas and Transformation Grammar are specified in layer M3. The relationships in meta-modeling layers are defined with dashed lines in figure 2.0. All these relationships are of 'instance of' type. It means a document is an instance of model, which is the instance of meta-model. The source and the target models refer to XML and RDF schemas.

NOTE:

> During the report we use *schema* term to define a schema on a particular domain. Furthermore, we refer to the grammar of that schema as *Schema*.

We feed an instance of an XML schema (artifact 3) to an XSLT Processor (artifact 7) as an input and get an RDF document as an output (artifact 10) of the transformation. RDF document is also represented in XML, that's why we use XSLT to make transformations. The main goal of the project to enable aforementioned transformation based on Low Level Transformation Specification (artifact 6), which is generated by High to Low Level Transformation Specification (artifact 15) based on High Level Transformation Specification (artifact 4). That means, any XML document that conforms to an XML schema can be transformed to RDF document. For the sake of simplicity, we use MDA (Model Driven Architecture) only as source and target, without referring to PIM and PSM.

NOTE:

> We use a term *Layer* to refer to meta-modeling layers. Here, we have 3 layers that are instances of models (artifacts 3,10 and 13) – M0 layer, model (artifacts 2,9 and 12) – M1 layer and meta-model (artifacts 1,16,17,8 and 11) - M2 layer, which refer to M0, M1 and M2 meta-modeling layers respectively.

As shown in Fig. 2.0, An XSLT stylesheet (artifact 15) generates XSLT transformation (artifact 6), which transform from XML (artifact 3) to RDF (artifact 10). In order to get artifact 6, we need to know mapping or relationships between XML schema and RDF schema, which define the grammar of XML and RDF respectively. That's why we need high level specification (artifact 4) which address aforementioned mapping. Input and output documents are in XML format, that's why it makes sense to use XSLT.

Now, we will introduce source and target artifacts. Source (XML) artifacts are artifacts 1, 2 and 3. And the target (RDF) artifacts are 8,9,10,11,12,13 and 14. The idea behind the project is first define the source and target artifacts, than making the transformations between them. So, we are going to start with first step, defining the source and target artifacts and give some particular examples to these artifacts.

2.1.1 The XML Schema (1)

The XML schema describes the structure of an XML document. The XML Schema language is also referred to as XML Schema Definition (XSD). The purpose of the XML Schema is to define the legal building blocks of an XML document, just like a DTD (Document Type Definition). The XML Schema defines elements, attributes that can appear in a document, which elements are child elements, the order of the child elements, the number of the child elements, whether an element is empty or can include text, data types for elements and attributes, default and fixed values for elements and attributes.

XML Schema is described as the Schema Components Diagram [18] in Appendix F.

2.1.2 An XML schema (2)

The purpose of an XML schema is to define the legal building blocks of an XML document, which belongs to concrete specific domain (in our case, purchaseOrder).
This artifact is a Model, which defines the grammar for artifact 3. And it is a source model, one of the components of model-to-model transformation. At this point, the model described based on particular example, purchase order domain.

```
<xsd:schema xmlns:xsd="http://www.w3.org/2001/XMLSchema">

 <xsd:annotation>
  <xsd:documentation xml:lang="en">
   Purchase order schema for Example.com.
   Copyright 2000 Example.com. All rights reserved.
  </xsd:documentation>
 </xsd:annotation>

 <xsd:element name="purchaseOrder" type="PurchaseOrderType"/>

 <xsd:element name="comment" type="xsd:string"/>

 <xsd:complexType name="PurchaseOrderType">
  <xsd:sequence>
   <xsd:element name="shipTo" type="USAddress"/>
   <xsd:element name="billTo" type="USAddress"/>
   <xsd:element ref="comment" minOccurs="0"/>
   <xsd:element name="items"  type="Items"/>
  </xsd:sequence>
  <xsd:attribute name="orderDate" type="xsd:date"/>
 </xsd:complexType>

 <xsd:complexType name="USAddress">
  <xsd:sequence>
   <xsd:element name="name"   type="xsd:string"/>
   <xsd:element name="street" type="xsd:string"/>
   <xsd:element name="city"   type="xsd:string"/>
   <xsd:element name="state"  type="xsd:string"/>
   <xsd:element name="zip"    type="xsd:decimal"/>
  </xsd:sequence>
  <xsd:attribute name="country" type="xsd:NMTOKEN"
     fixed="US"/>
 </xsd:complexType>

 <xsd:complexType name="Items">
  <xsd:sequence>
```

```
   <xsd:element name="item" minOccurs="0" maxOccurs="unbounded">
    <xsd:complexType>
     <xsd:sequence>
      <xsd:element name="productName" type="xsd:string"/>
      <xsd:element name="quantity">
       <xsd:simpleType>
        <xsd:restriction base="xsd:positiveInteger">
         <xsd:maxExclusive value="100"/>
        </xsd:restriction>
       </xsd:simpleType>
      </xsd:element>
      <xsd:element name="USPrice"  type="xsd:decimal"/>
      <xsd:element ref="comment"   minOccurs="0"/>
      <xsd:element name="shipDate" type="xsd:date" minOccurs="0"/>
     </xsd:sequence>
     <xsd:attribute name="partNum" type="SKU" use="required"/>
    </xsd:complexType>
   </xsd:element>
  </xsd:sequence>
 </xsd:complexType>
</xsd:schema>
```

2.1.3 An XML Document (3)

XML was designed to describe data and focus on defining the syntax of data. An XML document uses a Document Type Definition (DTD) or an XML schema (purchaseOrder XML schema) to describe the data.
This artifact is the instance of source model (artifact 2) and conforms to the model. And it is the input file, which will be given as input to the artifact 7.

```
<?xml version="1.0"?>
<purchaseOrder orderDate="1999-10-20">
    <shipTo country="US">
        <name>Alice Smith</name>
        <street>123 Maple Street</street>
        <city>Mill Valley</city>
        <state>CA</state>
        <zip>90952</zip>
    </shipTo>
    <billTo country="US">
        <name>Robert Smith</name>
        <street>8 Oak Avenue</street>
        <city>Old Town</city>
        <state>PA</state>
        <zip>95819</zip>
    </billTo>
    <comment>Hurry, my lawn is going wild!</comment>
    <items>
        <item partNum="872-AA">
            <productName>Lawnmower</productName>
            <quantity>1</quantity>
            <USPrice>148.95</USPrice>
            <comment>Confirm this is electric</comment>
        </item>
        <item partNum="926-AA">
            <productName>Baby Monitor</productName>
            <quantity>1</quantity>
            <USPrice>39.98</USPrice>
            <shipDate>1999-05-21</shipDate>
        </item>
    </items>
</purchaseOrder>
```

2.1.4 XSLT Processors (5 and 7)

These are the simple XSLT processors, which are used to transform input to output. We have used two XSLT processors. The first one is msxsl.exe, which is command line XSLT processor by Microsoft. And the second one is the built-in XSLT processor of XMLSPY2004 release 4.

2.1.4.1 XSLT Processor (5)

This processor will be fed from two sources (artifact 4 and 15). Artifact 4 contributes by high language specification in order to transform from source to target. And Artifact 15 will contribute with transformation stylesheets.

2.1.4.2 XSLT Processor (7)

It is the same processor as in (5), but only input and stylsheet are different. This processor will be fed from artifacts 3 and 6. It will perform transformation based on XML document (artifact 3) using XSLT stylesheet (artifact 6)

2.1.5 RDF Schema (8)

RDF Schema is a vocabulary description language, as described in RDF Schema specification [2]. It defines classes and properties that may be used to describe classes of resources and properties. It provides mechanisms for declaring these properties and for defining the relationships between these properties and other resources. It specifies mechanisms that may be used to name and describe properties and the classes of resource they describe.

It is the meta-model of RDF-schema (artifact 9). All grammar rules for artifact 9 are defined here. It defines the target model structure. You can find it in appendix B.

2.1.6 An RDF schema (9)

An RDF schema provides information about the interpretation of the statements given in an RDF data model. An RDF-schema provide mechanisms for declaring properties and relationship between them in a specific domain (purchaseOrder domain).

This is also one of the components of model-to-model transformation, which defines the target model. Output document (artifact 10) should conform to this RDF schema. A part of an example RDF schema is given below:

```
<?xml version="1.0"?>

   <rdf:RDF xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
            xmlns:xsd="http://www.w3.org/2001/XMLSchema#"
             xmlns:rdfs="http://www.w3.org/2000/01/rdf-schema#">

   <rdfs:Class rdf:ID="Address">
     <rdfs:comment>Address Class</rdfs:comment>
```

```
     <rdfs:subClassOf  rdf:resource="http://www.w3.org/1999/02/22-      rdf-
     syntax-ns#Resource"/>
</rdfs:Class>

<rdfs:Class rdf:ID="ShipTo">
  <rdfs:comment>Student Class</rdfs:comment>
  <rdfs:subClassOf rdf:resource="#Address"/>
</rdfs:Class>

<rdfs:Class rdf:ID="BillTo">
  <rdfs:comment>Teacher Class</rdfs:comment>
  <rdfs:subClassOf rdf:resource="#Address"/>
</rdfs:Class>

<rdfs:Class rdf:ID="PurchaseOrder">
  <rdfs:comment>PurchaseOrder Class</rdfs:comment>
     <rdfs:subClassOf rdf:resource="http://www.w3.org/1999/02/22-rdf-syntax-
     ns#Resource"/>
</rdfs:Class>

<rdfs:Class rdf:ID="Item">
  <rdfs:comment>Items Class</rdfs:comment>
     <rdfs:subClassOf rdf:resource="http://www.w3.org/1999/02/22-rdf-syntax-
     ns#Resource"/>
   <rdfs:domain rdf:resource="#items"/>
</rdfs:Class>

<rdf:Property rdf:ID="orderDate">
   <rdfs:comment>orderDate description</rdfs:comment>
   <rdfs:domain rdf:resource="#PurchaseOrder"/>
</rdf:Property>

<rdf:Property rdf:ID="shipTo">
   <rdfs:comment>shipTo description</rdfs:comment>
   <rdfs:domain rdf:resource="#PurchaseOrder"/>
   <rdfs:range rdf:resource="#ShipTo"/>
</rdf:Property>

<rdf:Property rdf:ID="billTo">
   <rdfs:comment>billTo description</rdfs:comment>
   <rdfs:domain rdf:resource="#PurchaseOrder"/>
   <rdfs:range rdf:resource="#BillTo"/>
</rdf:Property>

<rdf:Property rdf:ID="items">
     <rdfs:comment>Items of product that have to be sended</rdfs:comment>
   <rdfs:domain rdf:resource="#PurchaseOrder"/>
   <rdfs:range rdf:resource="http://www.w3.org/1999/02/22-rdf-
     syntax-ns#Seq"/>
</rdf:Property>

<rdf:Property rdf:ID="country">
   <rdfs:comment>Country of BillTo or ShipTo</rdfs:comment>
   <rdfs:domain rdf:resource="#Address"/>
   <rdfs:range rdf:resource="xsd:string"/>
</rdf:Property>
     . . . . .
     . . . . .
     . . . . .

</rdf:RDF>
```

2.1.7 An RDF Document (10)

An RDF/XML Document is an RDF Document written in the recommended XML transfer syntax for RDF. The RDF/XML syntax itself provides considerable flexibility in the syntactic expression of the data model [2]. An RDF document is an instance of an RDF schema (purchaseOrder schema).

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

```
<rdf:RDF xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#">

 <PurchaseOrder>
 <orderdate>1999-10-20</orderdate>
 <shipTo>
 <ShipTo rdf:nodeID="shipToinf">
       <country> US </country>
       <name> Alice Smith </name>
       <street> 123 Maple Street</street>
       <city> Mill Valley </city>
       <state> CA </state>
       <zip> 90952 </zip>
 </ShipTo>
 </shipTo>
 <billTo>
 <BillTo rdf:nodeID="billToinf">
       <country> US </country>
       <name> Robert Smith </name>
       <street> 8 Oak Avenue </street>
       <city> Old Town </city>
       <state> PA </state>
       <zip> 95819 </zip>
 </BillTo>
 </billTo>
 <comment> Hurry, my lawn is going wild! </comment>
 <items>
 <rdf:Seq>
       <rdf:li>
       <Item rdf:nodeID="iteminf">
              <partNum> 872-AA </partNum>
              <productName> Lawnmower </productName>
              <quantity> 1 </quantity>
              <USPrice> 148.95 </USPrice>
              <comment> Confirm this is electric </comment>
         </Item>
       </rdf:li>
       <rdf:li>
       <Item rdf:nodeID="iteminf">
              <partNum> 926-AA </partNum>
              <productName> Baby Monitor </productName>
              <quantity> 1 </quantity>
              <USPrice> 39.98 </USPrice>
              <shipDate> 1999-05-21 </shipDate>
            </Item>

        </rdf:li>

 </rdf:Seq>
 </items>
 </PurchaseOrder>
 </rdf:RDF>
```

2.1.8 RDF-Schema Graph (11)

RDF Schema Graph - is a set of triples that belongs to a triple set in the RDF-Data model. It represents classes and properties that may be used to describe classes, properties and other resources. This graph represents the relationships between RDF Core Classes and Properties. You can see the details in figure 2.1

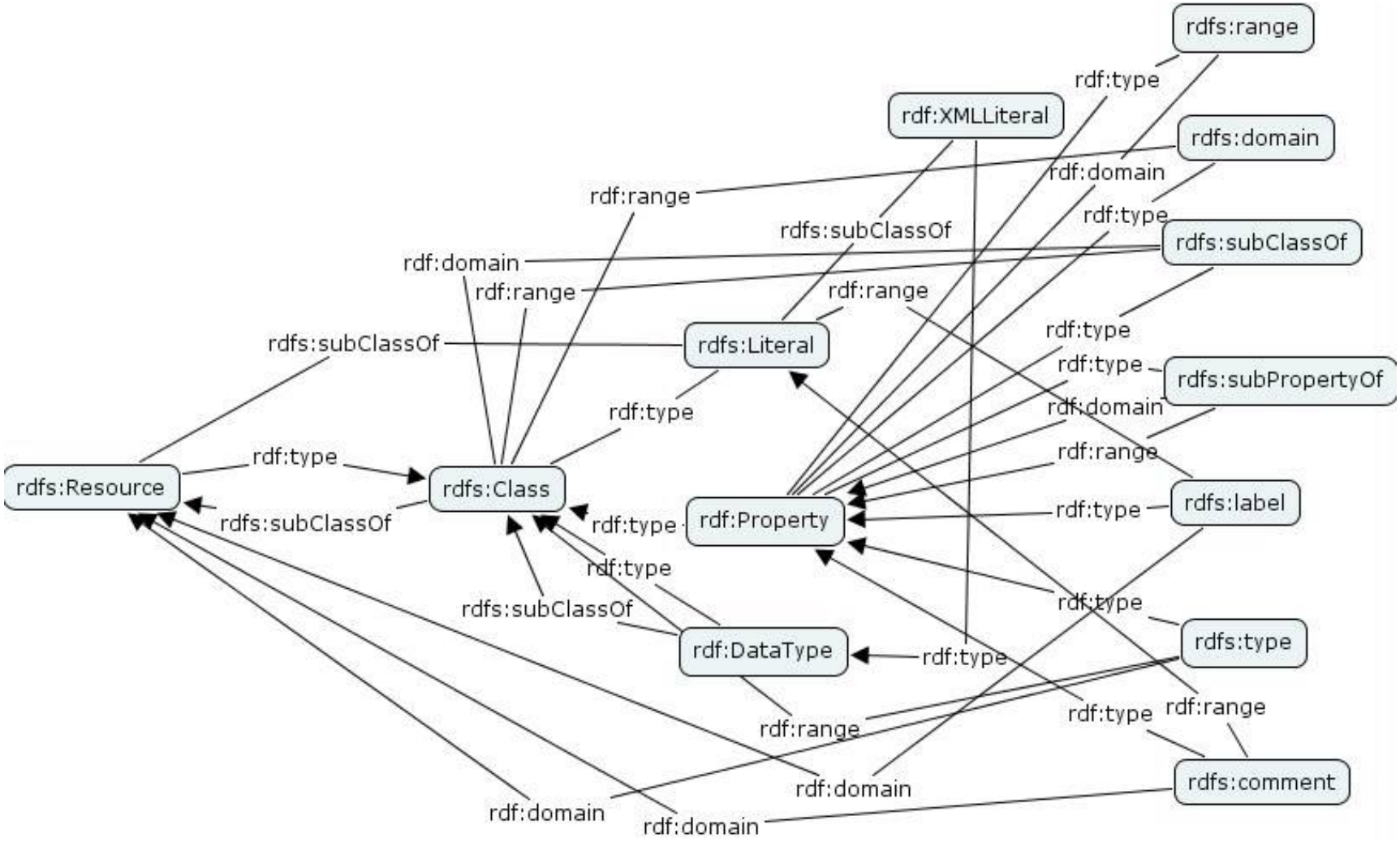


Figure 2.1 RDF –Schema Graph

2.1.9 An RDF schema Graph (12)

An RDF schema Graph is a set of triples that conform to a triple set in the RDF-Data model. An RDF schema is based in a certain domain, in our case purchaseOrder domain. In our case, for example, a triple can consist of *purchaseOrder Subject*, *orderDate predicate* and *Object* (figure 2.2), that represent value of OrderDate propersty(predicate).

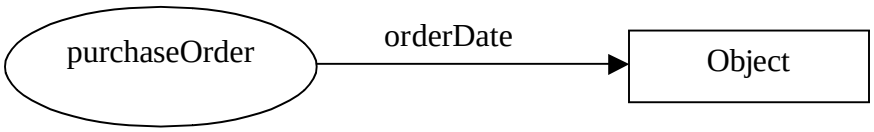


Figure 2.2 Triples in term of PurchaseOrder domain

In figure2.3 and figure 2.4 we depicted An RDF schema Graph for certain domain, i.e. for purchaseOrder domain. In figure 2.3, we defined Classes of this domain skipping the properties of these classes. Property definitions of these classes appear on figure 2.4.

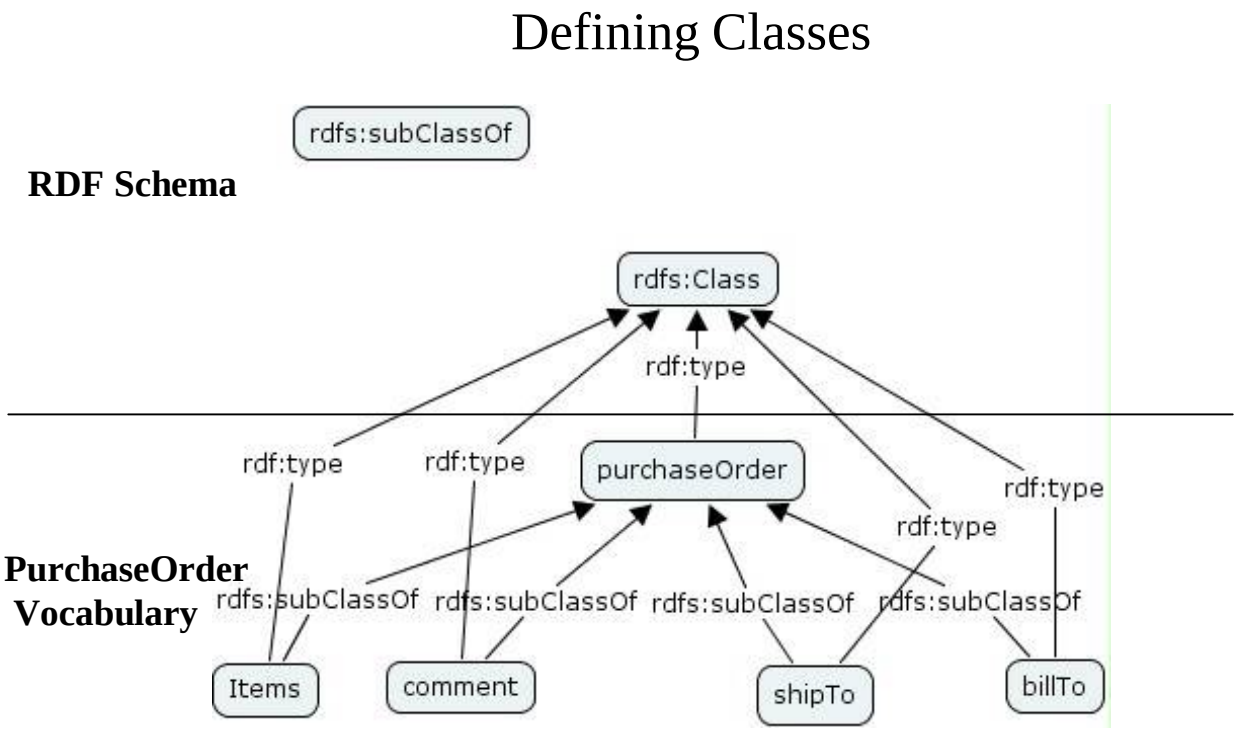


Figure 2.3 An RDF schema Graph in term of Classes (for purchaseOrder domain)

We can redraw figure 2.3 and figure 2.4 together. To avoid the complexity of figure, we've just separated the definition of Classes and Properties into different figures.

Another thing here to notice is that we have depicted two layers, in term of metamodeling in both figures. The above layer that represents RDF Schema defines core classes (rdfs:Class) and core properties (rdf:Property) respectively in figure 2.3 and figure 2.4.

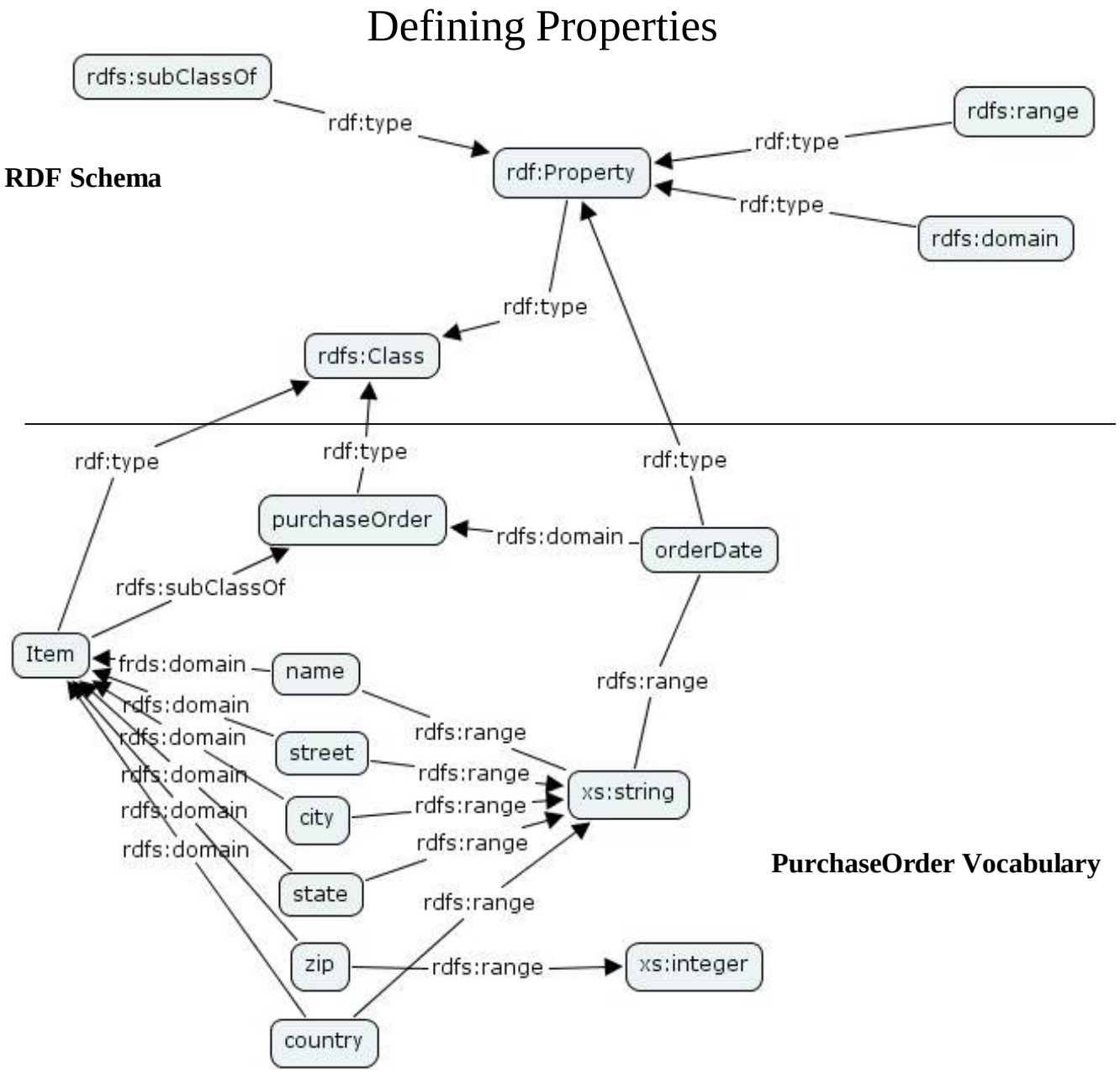


Figure 2.4 An RDF Schema Graph in term of Properties (for purchaseOrder domain)

The lower layer that represens An RDF schema for purchaseOrder domain define concrete classes and properties that belong to that domain. Here, we have depicted two layers to show the interaction between RDF Schema and An RDF schema. All classes and properties in an RDF schema are the instances of rdfs:Class and rdf:Property core classes of RDF Schema respectively.

2.1.10 An RDF Graph (13)

An RDF Graph - is a set of triples that belongs to a triple set in a RDF-Data model. A set of such triples is called an RDF graph. This can be illustrated by a node and directed-arc diagram, in which each triple is represented as a node-arc-node link (hence the term "graph"). In our case, for example, a triple, which consist of *purchaseOrder Subject*, *orderDate predicate* and *"1999-10-20" Object* (figure 2.5). Difference from An RDF schema Graph, here we have concrete value for the Object.

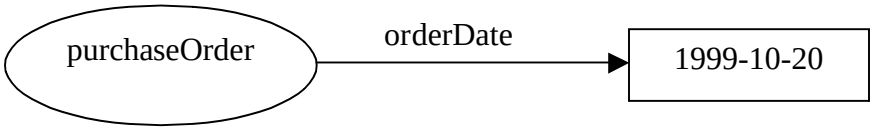


Figure 2.5. RDF Triples on PurchaseOrder domain

We have represented the simple statements in purchaseOrder example in RDF-Graph as in figure 2.6.

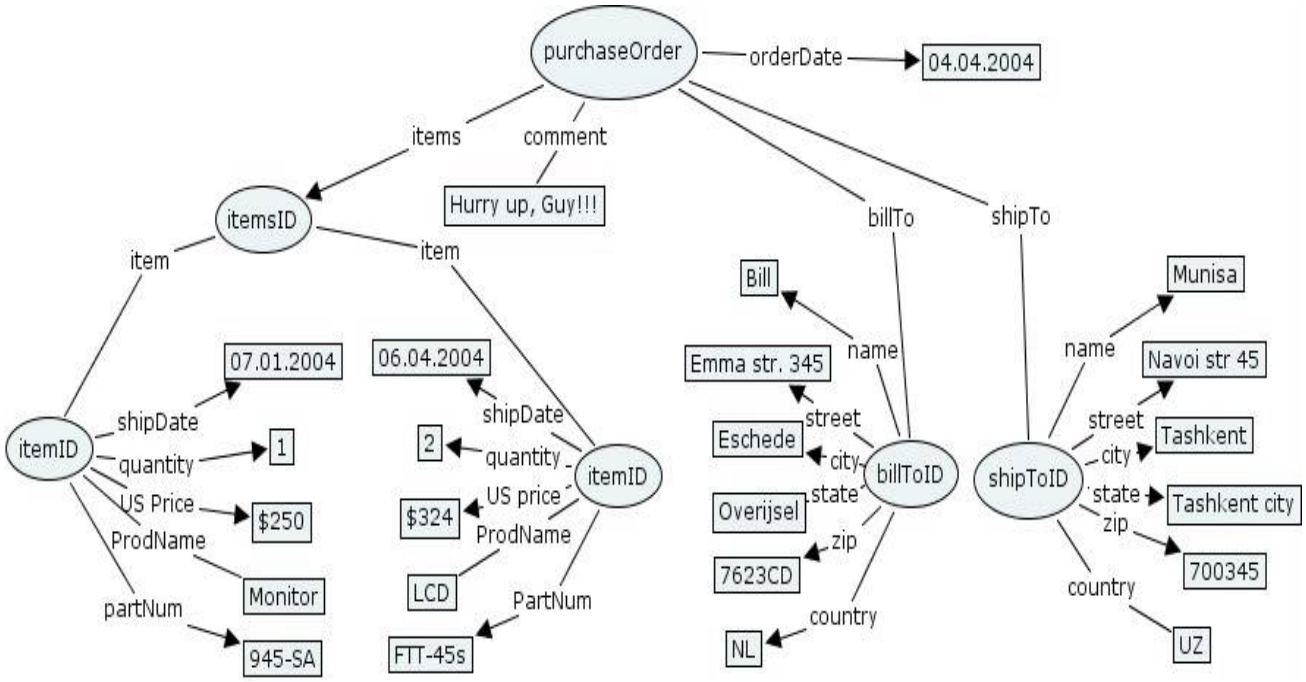


Figure 2.6 An RDF Graph Describing purchaseOrder

2.1.11 The RDF Data Model (14)

The RDF-Data model - as specified in a RDF Model and Syntax Specification [11], defines a simple model for describing interrelationships among resources in terms of named properties and values. It can be a set of statements, a table or any formal specification for describing interrelationships among resources.

The RDF data model is defined formally as follows:

1. There is a set called *Resources*.
2. There is a set called *Literals*.
3. There is a subset of *Resources* called *Properties*.
4. There is a set called *Statements*, each element of which is a triple of the form

{pred, sub, obj}

Where pred is a property (member of *Properties*), sub is a resource (member of *Resources*), and obj is either a resource or a literal (member of *Literals*).

These are the rules for the essential elements of RDF, as mentioned in [9]:

Subjects – can be RDF URI reference or blank node;
Predicate – can be RDI URI Reference;
Object – can be RDF URI reference, literal and blank node.

2.2 Summary

This chapter have discussed about the definitions and particular examples to each source and target artifacts. These examples are based on 'PurchaseOrder' example domain. Based on these artifacts we will come up with high level specification in next chapter.

3 High Level Transformation Specifications

This chapter converse about the transformation specification, based on the examples in the previous chapter. Section 3.1 gives the mapping or relationship between source and target artifacts in M0, M1 and M2 meta-modeling layers. Then the one of the core artifacts of the project – High Level Transformation Specifications are discussed in section 3.2. This chapter concludes with summary section 3.3.

3.1 Mapping between XML and RDF artifacts.

Based on the aforementioned examples, we describe the mapping between source and target model in this section. To find out the mapping we use comparison tables (tables 3.1, 3.2 and 3.3) for each layer, and compared the documents. These tables are the intermediate means to understand the mapping. First we compare layer M0, XML and RDF documents. Later XML and RDF schemas are compared in layer M1; at last we give the mapping between XML and RDF Schemas at layer M2. This mappings tables have several kind of mappings like, one-one, one-many, zero-one, many-one, many-zero and zero-many. The left part of the table corresponds, maps to the right one. Based on left and right columns you can identify the aforesaid mappings.

Mapping between XML and RDF Document based on purchaseOrder example – layer M0 (table 3.1)	
XML Document	RDF Document
<?xml version="1.0" encoding="iso-8859-1" ?>	<?xml version="1.0" encoding="iso-8859-1" ?>
	<rdf:RDF xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#" xmlns:prop="http://www.blabla.com/prop">
<purchaseOrder orderDate="1999-10-20">	<rdf:Description rdf:about="http://www.mysite.com/purchaseOrder"> <prop:orderdate>1999-10-20</prop:orderdate>
<shipTo country="US">	<prop:shipTo> <rdf:Description rdf:nodeID="shipToinf"> <prop:country> US </prop:country>
<name>Alice Smith</name>	<prop:name> Alice Smith </prop:name>
<street>123 Maple Street</street>	<prop:street> 123 Maple Street</prop:street>
<city>Mill Valley</city>	<prop:city> Mill Valley </prop:city>
<city>Mill Valley</city>	<prop:state> CA </prop:state>
<zip>90952</zip>	<prop:zip> 90952 </prop:zip>
</shipTo>	</rdf:Description> </prop:shipTo>
<billTo country="US">	<prop:billTo> <rdf:Description rdf:nodeID="billToinf"> <prop:country> US </prop:country>
<name>Robert Smith</name>	<prop:name> Robert Smith </prop:name>
<street>8 Oak Avenue</street>	<prop:street> 8 Oak Avenue </prop:street>
<city>Old Town</city>	<prop:city> Old Town </prop:city>
<state>PA</state>	<prop:state> PA </prop:state>
<zip>95819</zip>	<prop:zip> 95819 </prop:zip>
</billTo>	</rdf:Description> </prop:billTo>
<comment>Hurry, my lawn is going	<prop:comment> Hurry, my lawn is going wild!

wild!</comment>	</prop:comment>
<items>	<prop:items>
<item partNum="872-AA">	<prop:item> <rdf:Description rdf:nodeID="iteminf"> <prop:partNum> 872-AA </prop:partNum>
<productName>Lawnmower</productName>	<prop:productName> Lawnmower </prop:productName>
<quantity>1</quantity>	<prop:quantity> 1 </prop:quantity>
<USPrice>148.95</USPrice>	<prop:USPrice> 148.95 </prop:USPrice>
<comment>Confirm this is electric</comment>	<prop:comment> Confirm this is electric </prop:comment>
</item>	</rdf:Description> </prop:item>
<item partNum="926-AA">	<prop:item> <rdf:Description rdf:nodeID="iteminf"> <prop:partNum> 926-AA </prop:partNum>
<productName>Baby Monitor</productName>	<prop:productName> Baby Monitor </prop:productName>
<quantity>1</quantity>	<prop:quantity> 1 </prop:quantity>
<USPrice>39.98</USPrice>	<prop:USPrice> 39.98 </prop:USPrice>
<shipDate>1999-05-21</shipDate>	<prop:shipDate> 1999-05-21 </prop:shipDate>
</item>	</rdf:Description> </prop:item>
</items>	</rdf:Description> </prop:items>
</purchaseOrder>	</rdf:Description> </rdf:RDF>

Table 3.1 represents a mapping in the lowest layer (M0), which is the relationship between XML and RDF documents. The code at left column is XML document code and the right one is for RDF document. As you noticed, each line of XML document corresponds to one or more lines of RDF document. Our main goal is to find the mapping in layer M1. But, for the sake of easiness, we started to find out the mapping between input (XML document) and output (RDF document) documents at layer M0. In this kind of step by step way, you can come up easily with the mapping or relationships between artifacts at layer M1.

Table 3.1 has just represented the mapping on specific 'purchaseOrder' document example. In order to find out the general rules and relationships between XML and RDF documents we have introduced new table, which defines the mapping between an XML schema and an RDF schema at layer M1.

Mapping between an XML and an RDF schema based on purchaseOrder example – layer M1 (table 3.2)	
An XML schema	An RDF schema
<xsd:schema xmlns:xsd="http://www.w3.org/2001/XMLSchema">	<?xml version="1.0"?> <rdf:RDF xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#" xmlns:xsd="http://www.w3.org/2001/XMLSchema#" xmlns:rdfs="http://www.w3.org/2000/01/rdf-schema#">
<xsd:element name="purchaseOrder" type="PurchaseOrderType"/>	<rdfs:Class rdf:ID="PurchaseOrder">

<xsd:element name="comment" type="xsd:string"/>	<rdf:Property rdf:ID="comment"> <rdfs:range rdf:resource = "http://www.w3.org/1999/02/22-rdf-syntax-ns#Literal"/>
<xsd:complexType name="PurchaseOrderType">	<rdfs:Class rdf:ID="PurchaseOrder"> <rdfs:subClassOf rdf:resource="http://www.w3.org/1999/02/22-rdf-syntax-ns#Resource"/>
<xsd:sequence>	
<xsd:element name="shipTo" type="USAddress"/>	<rdfs:Class rdf:ID="ShipTo"> <rdfs:subClassOf rdf:resource="#Address"/>
<xsd:element name="billTo" type="USAddress"/>	<rdfs:Class rdf:ID="BillTo"> <rdfs:subClassOf rdf:resource="#Address"/>
<xsd:element ref="comment" minOccurs="0"/>	<rdf:Property rdf:ID="comment"> <rdfs:domain rdf:resource="#PurchaseOrder"/>
<xsd:element name="items" type="Items"/>	<rdf:Property rdf:ID="items"> <rdfs:domain rdf:resource="#PurchaseOrder"/> <rdfs:range rdf:resource = "http://www.w3.org/1999/02/22-rdf-syntax-ns#Seq"/>
</xsd:sequence>	
<xsd:attribute name="orderDate" type="xsd:date"/>	<rdf:Property rdf:ID="orderDate"> <rdfs:domain rdf:resource="#PurchaseOrder"/>
</xsd:complexType>	
<xsd:complexType name="USAddress"> <xsd:sequence>	<rdfs:Class rdf:ID="Address"> <rdfs:subClassOf rdf:resource = "http://www.w3.org/1999/02/22-rdf-syntax-ns#Resource"/>
<xsd:element name="name" type="xsd:string"/>	<rdf:Property rdf:ID="name"> <rdfs:domain rdf:resource="#Address"/> <rdfs:range rdf:resource="xsd:string"/>
<xsd:element name="street" type="xsd:string"/>	<rdf:Property rdf:ID="street"> <rdfs:domain rdf:resource="#Address"/> <rdfs:range rdf:resource="xsd:string"/>
<xsd:element name="city" type="xsd:string"/>	<rdf:Property rdf:ID="city"> <rdfs:domain rdf:resource="#Address"/> <rdfs:range rdf:resource="xsd:string"/>
<xsd:element name="state" type="xsd:string"/>	<rdf:Property rdf:ID="state"> <rdfs:domain rdf:resource="#Address"/> <rdfs:range rdf:resource="xsd:string"/>
<xsd:element name="zip" type="xsd:decimal"/>	<rdf:Property rdf:ID="zip"> <rdfs:domain rdf:resource="#Address"/> <rdfs:range rdf:resource="xsd:string"/>
</xsd:sequence>	
<xsd:attribute name="country" type="xsd:NMTOKEN" fixed="US"/>	<rdf:Property rdf:ID="country"> <rdfs:domain rdf:resource="#Address"/> <rdfs:range rdf:resource="xsd:string"/>
</xsd:complexType>	
<xsd:complexType name="Items"> <xsd:sequence>	<rdf:Property rdf:ID="items"> <rdfs:domain rdf:resource="#PurchaseOrder"/> <rdfs:range rdf:resource = "http://www.w3.org/1999/02/22-rdf-syntax-ns#Seq"/>
<xsd:element name="item" minOccurs="0" maxOccurs="unbounded">	<rdfs:Class rdf:ID="Item"> <rdfs:subClassOf rdf:resource = "http://www.w3.org/1999/02/22-rdf-syntax-ns#Resource"/> <rdfs:domain rdf:resource="#items"/>
<xsd:complexType> <xsd:sequence>	
<xsd:element name="productName" type="xsd:string"/>	<rdf:Property rdf:ID="productName"> <rdfs:domain rdf:resource="#Item"/> <rdfs:range rdf:resource="xsd:string"/>
<xsd:element name="quantity"> <xsd:simpleType>	<rdf:Property rdf:ID="quantity"> <rdfs:domain rdf:resource="#Item"/> <rdfs:range rdf:resource="xsd:integer"/>
<xsd:simpleType>	
<xsd:restriction base="xsd:positiveInteger">	
<xsd:maxExclusive value="100"/>	
</xsd:restriction> </xsd:simpleType>	

</xsd:element>	
<xsd:element name="USPrice" type="xsd:decimal"/>	<rdf:Property rdf:ID="usprice"> <rdfs:domain rdf:resource="#Item"/> <rdfs:range rdf:resource="xsd:string"/>
<xsd:element ref="comment" minOccurs="0"/>	<rdf:Property rdf:ID="comment"> <rdfs:domain rdf:resource="#item"/> <rdfs:domain rdf:resource="#PurchaseOrder"/> <rdfs:range rdf:resource = "http://www.w3.org/1999/02/22-rdf-syntax-ns#Literal"/>
<xsd:element name="shipDate" type="xsd:date" minOccurs="0"/>	<rdf:Property rdf:ID="shipDate"> <rdfs:domain rdf:resource="#Item"/> <rdfs:range rdf:resource="xsd:date"/>
</xsd:sequence>	
<xsd:attribute name="partNum" type="SKU" use="required"/>	<rdf:Property rdf:ID="partNum"> <rdfs:domain rdf:resource="#Item"/> <rdfs:range rdf:resource="xsd:string"/>
</xsd:complexType>	
</xsd:element> </xsd:sequence>	
</xsd:complexType>1	</rdf:RDF>

As it noticed from table 3.2, there are different kinds of mappings than in table 3.1. Here, we can come across with one-to-one, one-to-many and many-to-one relationships between artifacts. They show us how the XML schema components can mapped to RDF schema components.

Again, this schema mapping is also based on 'purchaseOrder' example. To find out the more general mapping, that can be valid in any example, we have initiated a new table. Based on the table 3.2 we've defined another mapping table for XML Schema and RDF Schema. However, it does not have one-to-one mapping for some elements, the table can be used for any domain.

XML Schema – RDF Schema mapping – layer M2 (table 3.3)	
XML Schema	RDF Schema
Element	rdfs:Class, rdf:Property
Attribute	rdfs:Class, rdf:Property
complexType	rdfs:Class
Sequence	
datatypes(string, int, date)	rdfs:Literal or xsd:string, xsd:int and xsd:date rdfs:Datatypes
Documentation	rdfs:comment
Annotation	rdfs:comment
Schema	Rdf:RDF
simpleType	Rdf:Class
restriction	
Pattern	
	rdfs:resource
	rdfs:domain
	rdfs:subClassOf
	rdfs:range

	Rdf:type
	rdfs:subpropertyOf
	rdfs:label

And of course, there are several mapping alternatives to choose in table 3.3. It is free to implementer to select the appropriate one, according to his objectives. In the next section we make some assumption to select the one of these alternatives and do further work based on selected one. This assumption will be made in high level transformation specification (artifact 4).

In this example, table1, table2 and table3 represent the mapping between XML and RDF artifacts in layer M0, M1 and M2 respectively. You can notice that it is much more concrete, 0ne-to-one mapping in layer M0, but when you go towards layer M2 the mapping become more general and there might not be one-to-one mapping or even we can expect no mapping alternative to some elements or classes of artifacts. We have tried to find out the most appropriate mapping between source (XML) and target (RDF).

There are some mapping alternatives. For instance, you can map the instances of XML Schema Element to Class and Property instances of RDF Schema (table 3.3). To come up with high level transformation specification we select one of the alternatives like in aforementioned example. This kind of selection assumptions are discussed in next section.

Now, based on these mapping tables we are going to write high level transformation specification that will be used to automatically generate stylesheets.

3.2 High Level Transformation Specification

In this section we mainly will focus on high level transformation specifications. It consist of the artifacts 4, 16, 17 and 18. And we will define and give some examples to aforementioned artifacts. Transformation generating issues will be discussed in chapter 4. The way we follow to come up with high level transformation specification (artifact 4) is starting with Transformation Grammar (artifact 16) and EBNF (artifact 17) of it and text representation (artifact 18) of artifact4. Aforesaid 3 tables lend a hand to define these four artifacts.

3.2.1 Transformation Grammar (16)

This section describes the concrete syntax of the transformation grammar and informally explains the semantics of the syntactical constructs. This artifact is the adaptation of transformation language by authors of [10]. This Grammar is represented in XML Schema and you can find it in Appendix C.

Artifact 16 is XML Schema file which defines the grammar of high level transformation specification, which is one of the core artifacts of the project. Here, we introduce the main logic of Transformation Grammar. The aforementioned 3 tables affect the Transformation Grammar to define the rules of artifact4. It defines the grammar between XML schema meta-model and target RDF meta-model.

In the frame of the project, the transformation grammar based on a set of *Transformation Rules*. As discussed in [10], there are two kinds of transformation rules: *Class* and *Attribute Rules* (figure 3.1). A transformation can contain one or more Class and Attribute Rules. Every rule has a name, a source and a target.

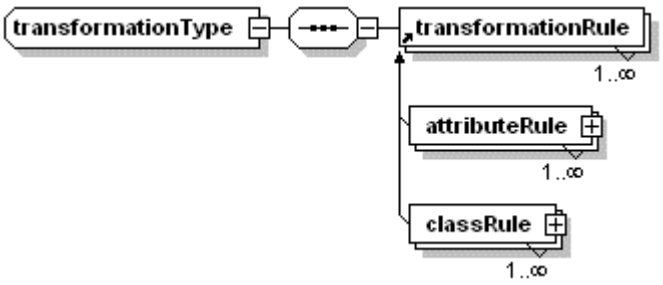


Figure 3.1 Transformation – root element of Transformation Grammar.

Source of the rule specifies a pattern. A pattern enumerates components from the source schema and may specify some constraints. For every instance of the pattern within an instance XML document some actions are taken. Actions are based on the information in the *target rule*. Rule target is specific to the rule type. The target of a class rule enumerates classes from the target model. These classes are instantiated for every occurrence of the rule pattern. The target of an attribute rule enumerates attributes and specifies how their values are obtained. These values are determined for every instance instantiated by a given class rule.

A Class Rule

ClassRule can contain source, action (instantiation), attributeRule and inherited rules and supersededRules (figure 3.2). A class rule specifies a correspondence between some components from the source XML schema and a set of classes from RDF target model. The source of the Class rule is discussed above. We have simplified the transformation grammar in [10], and consider action only as an instantiation. Every instantiation uses a class. The instance created from that class might be assigned with an *object identifier* and later referred to through this identifier. Referring is possible because the transformation establishes a connection between the node for which the rule is applied and the objects created by the rule.

The target of a class rule specifies a list of instantiations that will be executed when the rule is applied to an occurrence of the rule source pattern. Every instantiation uses a class. The object created from that class might be assigned with an object identifier and later referred to through this identifier. Referring is possible because the transformation engine establishes a connection between the node for which the rule is applied and the objects created by the rule.

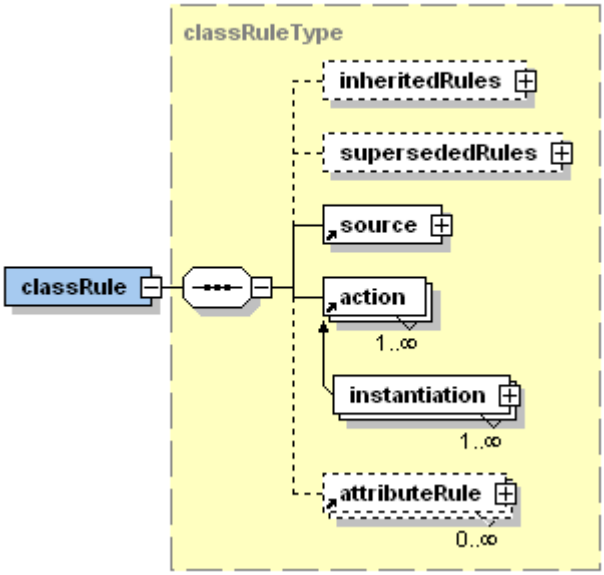


Figure 3.1 ClassRule of Transformation Grammar

Class Rule is can also consist of attributeRule, which will be discussed in the next paragraph.

An Attribute Rule

The objects created by class rules receive values for their attributes from two possible sources. The first one is an expression specified in an attribute assignment within a given instantiation. If the expression is missing then the values are provided by attribute rules. Attribute rules locate nodes in the source tree from which the attribute values can be determined. You can see the Attribute Rule grammar in the following code:

```
<!-- attributeRule -->
<xs:element              name="attributeRule"              type="attributeRuleType"
substitutionGroup="transformationRule"/>
        <!-- attributeRuleType -->
        <xs:complexType name="attributeRuleType">
         <xs:complexContent>
```

```
                <xs:extension base="transformationRuleType">
                 <xs:sequence>
                   <xs:element ref="source"/>
                        <xs:element     name="target"     type="targetAttributeType"
maxOccurs="unbounded"/>
                 </xs:sequence>
                 <xs:attribute ref="ownerClassRule" use="required"/>
                 <xs:attribute ref="name" use="required"/>
                 </xs:extension>
         </xs:complexContent>
        </xs:complexType>
```

Values may be obtained from the source tree nodes (e.g. values of attributes and text nodes, element names, etc.) or may be other objects associated with the source nodes and created by some class rule. In both cases some operations may be additionally applied to form the final value. Every class rule is associated with a set of attribute rules and plays the role of *owner* for these attribute rules. Every attribute rule has at least one attribute as a target and defines how its values are obtained from the source tree. An attribute may be a target of more than one attribute rule.

The attribute value may be a result of the evaluation of an expression. If the expression is skipped then the object instantiated by a class rule for the attribute source node is used as a value. If no object is associated with the source node then an error is generated.

In this project, we make a simplification and we use single form Attribute Rules. The single form specifies only one possible source for the attribute value. The form with alternatives specifies multiple alternative sources. They may have different paths and expressions. We only consider AttributeRules in single Form.

As a result of the adaptation of transformation language [10], we come up with a Transformation Grammar with the following simplification, adaptations:

1. Rules without inheritance. We do not consider inheritance among the rules.
2. Rules without superseding. The superseding is not considered in our transformation grammar.
3. We consider only one component in the source. There can be one or more components within the source, but for the sake of simplicity we consider only one.
4. We consider the attributes within the one source. In other words, we only take care about single Form attributes in our Transformation grammar.

3.2.2 EBNF representation of Transformation Grammar (17)

The syntax is specified in a grammar following EBNF notation. The following grammar uses the Basic Extended Backus-Naur Form (EBNF).

Types of Rules

[1] TransformationRule::= ClassRule | AttributeRule

Class Rule

[2] ClassRule ::= RuleName 'ClassRule'
InheritedRule ? '{'
RuleSource
ClassRuleTarget
AttributeRules ? '}'

[3] InheritedRule ::= 'inherits' RuleName

[4] ClassRuleTarget ::= 'target' '[' Action (',' Action)* ']'

[5] Action ::= Instantiation

Instantiation Action

[6] Instantiation ::= (VariableName ':')? TypeName ('=' Expression)? AttributeList?

[7] AttributeList ::= '{'AttributeAssignment (',' AttributeAssignment)* '}'

[8] AttributeAssignment ::= AttributeName ('=' Expression)?

Attribute Rules Section

[9] AttributeRules ::= 'Attribute {' InPlaceAttributeRule+ '}'

[10] InPlaceAttributeRule ::= RuleName (SingleForm | AlternativesForm)

Attribute Rule

[11] AttributeRule ::= RuleName 'AttributeRule' OwnerRule
'{'(SingleForm | AlternativesForm) '}'

[12] OwnerRule ::= 'owner=' RuleName

Attribute Rule in Single Form

[13] SingleForm ::= RuleSource AttributeRuleTarget

[14] AttributeRuleTarget::= ‘target[‘ TargetAttribute (‘,’ TargetAttribute)* ‘]’

[15] TargetAttribute ::= (VariableName ‘.’)? AttributeName ‘=’ Expression

Attribute Rule in Form with Alternatives

[16] AlternativesForm ::= AttributeRuleTarget (‘alt {‘ RuleSource? AttributeRuleTarget ‘}’) +

Rule Source

[17] RuleSource::= ‘source[‘ SourceElement (‘,’ SourceElement)* Predicate? ‘]’

[18] SourceElement ::= (Identifier | Variable)

[19] Variable ::= VariableName “:” Identifier (‘=’ Expression)?

[20] Predicate ::= ‘predicate {’ Expression ‘}’

From this EBNF notation, we do not consider Attribute Rule in Form with Alternatives. Moreover, we do not consider inherentaance and superseding among the rules.

3.2.3 Text representation of High Level Trans. Specification (18)

After defining the Transformation Grammar, we define the simple high level transformation specification based on it. For the sake of simplicity, first we come with text representation of artifact4 based on Purchase Order example.

We have written down Transformation specification for ComplexType *USAddress*, which consists of Class and Attribute Rules. These rules define mapping between source (RDF schema) and target (RDF schema) model. For each instance of schema *component*[1] shown in *source*, one or more instances of target schema will be created.
A class rule specifies a correspondence between some components from the source XML schema and a set of components from the target model. Every class rule has a name, a source, a target and is associated with a number of attribute rules. It has the following syntax:

```
ClassRule{
      Source [variable: schema component]
      Target [variable:target_component{attr1, attr2, attr3….}]
}
```

[1] *A component* - is anything which may be defined or declared: an element, an attribute, a simple type, a complex type, a group, an attribute group, or an identity constraint [12]

Attribute rules specify how the values for the attributes of the instances of target model are determined. Every attribute rule has a name and specifies the target attribute.

```
AttributeRule{
        Source [variable: schema component]
        Target [variable: target_component_name]
}
```

All attribute rules should refer, associate to some Class rule. In other words, the attributes defined in Class rule are described in Attribute rules. Here we define the rules for Complex Type USAddress from PurchaseOrder example.

```
USAddress Class Rule
{
Source [USAddressType: Complex Type]
Target [rdfAddress: Address{name, street, city, state, zip}]
}

Attribute Rules
{
name
    source [name: Element='/name']
    target [name]
    street
source [street: Element='/street']
    target [street]
    city
source [city: Element='/city']
    target [city]
    state
source [state: Element='/state']
    target [state]
    zip
source [zip: Element='/zip']
    target [zip]
    country
source [country: Attribute='/country']
    target [country]
}
```

For the Schema component naming, we have used XML Schema Component names from [18], Schema Component Diagram (Appendix E). Based on this diagram we have named XML Schema components in the following way:

Simple Type
Complex Type
Element
Attribute

USAddress ClassRule‘s source addressing the complex type USAddressType. Target addresses the RDF class with attributes name, street, city, state, zip. The meaning of this ClassRule is for the each instance of USAdressType, a RDF class will be instantiated with aforementioned properties (attributes).

Attribute Rules define the each attribute that is listed in ClassRule. Each of them should refer to the class rule. Later we will introduce *OwnerClass* attribute that make available this reference. Here, Source of the attribute again address the source XML document with the path expression to select the node that match this attribute. In a target part you can define the name of the attribute that will be instantiated in a target RDF class.

Following this transformation grammar we can define the Class and Attribute Rules to the other schema components of PurchaseOrder example.

3.2.4 High Level Transformation Specification (4)

This artifact is based on a meta-model, artifact 16. This meta-model defines the rules, grammar of high level transformation Specification.

Based on the transformation grammar, we come up with high level transformation specification for PurchaseOrder example. Full code of artifact 4 you can find it in appendix D. We defined the USAddressType ClassRules in the following way:

```
<classRule name="billToUSAddress">
  <source>
    <element name="billTo">
    <predicate>@country='US'</predicate>
    </element>
  </source>
  <instantiation className="rdfAddress" objectIdentifier="rdfAddress">
  <propertyAssignments>
    <property name="name"/>
    <property name="street"/>
    <property name="city"/>
    <property name="state"/>
    <property name="zip"/>
    <property name="country"/>
  </propertyAssignments>
  </instantiation>
</classRule>
```

As it is seen from the code, ClassRule has a name, which is defined in attribute ‘name’. It has two child elements: source and instantiation (target), which conform to the grammar defined in artifact 16. Source describe the element ‘billTo’ with predicate ‘@country=US’ to select from the source. The predicate is optional as described in artifact16. It helps to select the instances of schema components. Another possibility instead of source element, there can be an attribute.

Instantiation element has attributes ClassName to name the Class and objectIdentifier to identify the instances of Class in target document. PropertyAssignments element lists the property that belongs to the current class. Now, we come up with the Attribute rules that relates to that ClassRule.

```
<attributeRule name="name" ownerClassRule="billToUSAddress">
<source>
<element name="name" path="name"/>
</source>
<target name="name" objectIdentifier="rdfName"/>
</attributeRule>
<attributeRule name="street" ownerClassRule="billToUSAddress">
<source>
<element name="street" path="street"/>
</source>
<target name="street" objectIdentifier="rdfStreet"/>
</attributeRule>
<attributeRule name="city" ownerClassRule="billToUSAddress">
<source>
<element name="city" path="city"/>
</source>
<target name="city" objectIdentifier="rdfCity"/>
</attributeRule>
<attributeRule name="state" ownerClassRule="billToUSAddress">
<source>
<element name="state" path="state"/>
</source>
<target name="state" objectIdentifier="rdfState"/>
</attributeRule>
<attributeRule name="zip" ownerClassRule="billToUSAddress">
<source>
<element name="zip" path="zip"/>
</source>
<target name="zip" objectIdentifier="rdfZip"/>
</attributeRule>
<attributeRule name="country" ownerClassRule="billToUSAddress">
<source>
<attribute name="country" path="@country"/>
</source>
<target name="country" objectIdentifier="rdfCountry"/>
</attributeRule>
```

Each Attribute rule has a name, which identified in attribute ‘name’. And every attribute has an attribute ‘ownerClassRule’ which refer to the Class Rule to which it belongs. Again the attribute rule also has source and target child elements. In source the element or attribute to select from the source is defined with name (attribute ‘name’) and the path that is XPath expression to select from the source. Target element again addresses the target document. Like in ClassRule, it has ‘name’ attribute to name the attribute, and ‘objectIdentifier’ to identify the attribute in target document.

Following this kind of way, you can get the high level transformation specification not only to PurchaseOrder example, but also to any XML document. You can find the full description of high level transformation specification based on PurchaseOrder example in Appendix D.

3.3 Summary

In this chapter we considered the mappings or relationships in three meta-modeling layers, M0, M1 and M2. According to the examples mentioned in chapter 2, section 3.1 has introduced the mapping tables to represent the mapping between the artifacts at M0, M1 and M2 meta-modeling layers. Section 3.2 consists of the high level transformation specification and PurchaseOrder examples. These mappings' and high level transformation specification's outcome will be used in the next chapter.

4 Generating XML-to-RDF transformations

This chapter is focused on the generation of XML-to-RDF transformations issues based on high level transformation specification discussed in chapter 3. It uses model-to-model transformation notion. Section 4.1 discusses about the general algorithm of auto generation of XSLT stylesheet (artifact 6). XPath\XSLT 1.0 and XPath\XSLT 2.0 features are discussed in order to select the appropriate one to our project in section 4.2. Future work issues of generation are discussed in section 4.3. And this chapter comes to an end by summary section 4.4.

4.1 The general algorithm of auto generation of XSLT stylesheet

In this section we provide the example to artifacts 6 based on PurchaseOrder example. We manually write down the artifact6, to transform from XML document (artifact 3) to RDF document (artifact 10). Then we try to auto generate artifact 6 by artifact 15 based on high level transformation specification (artifact 4). After getting the result only for purchaseOrder example, we will try to generalize the auto generation to any instance of XML schema in section 4.2 and 4.3.

4.1.1 An example transformation on purchaseOrder domain (6)

In order to get a low level transformation (artifact 6), we have tried to make a transformation (xslt-stylesheet) on specific domain, than later generalize this XSLT-stylesheet using the high level transformation specification (artifact 4). To achieve this goal, first of all, we've taken a simple example of source and target.

We have chosen purchaseOrder example. It is about purchase ordering of a simple company, which has certain goods to sell. The clients order the goods, and the purchase order is saved. The purchaseOrder information like billTo, shipto addresses, ordered items, order date and comments are stored there.

After defining the aforementioned example in XML and RDF documents, we can see what is the input and output document. According to this data we 'literally' come up with XSLT stylesheet, which take the XML document as input and give RDF document as output. We can see some part of the code below:

```
<xsl:template match="/">
   <rdf:RDF xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#">
     <xsl:for-each select="//node()[@country='US']">
          <xsl:element name="{concat('prop:',name(.))}">
           <rdf:Description nodeID="{concat(name(.),'inf')}">
            <rdf:type resource="http://www.zonozz.org/library/USAddress"/>
```

```
                    <xsl:element name="{concat('prop:',name(name))}">
                            <xsl:value-of select="/name"/>
                    </xsl:element>
                    <xsl:element name="{concat('prop:',name(street))}">
                            <xsl:value-of select="/street"/>
                    </xsl:element>
                    <xsl:element name="{concat('prop:',name(city))}">
                            <xsl:value-of select="/city"/>
                    </xsl:element>
                    <xsl:element name="{concat('prop:',name(state))}">
                            <xsl:value-of select="/state"/>
                    </xsl:element>
                    <xsl:element name="{concat('prop:',name(zip))}">
                            <xsl:value-of select="/zip"/>
                    </xsl:element>
                    <xsl:element name="{concat('prop:',name(@country))}">
                                    <xsl:value-of select="@country"/>
                            </xsl:element>
                    </rdf:Description>
            </xsl:element>
    </xsl:for-each>
    . . . . . .
    . . . . . .
```

The full code for artifact 6 is described on Appendix E.
Here, from code you can see that we are selecting the all nodes that has attribute country which is equal to 'US'. By this way we are selecting the instances of USAddressType. And here we use concat() string function to give a value to the 'name' attribute of the elements. And then the path information will be gained from artifact 4 later in artifact 15. All XSLT stylesheet tags will be generated by artifact 15 and the name and path information will be taken from artifact4.

4.1.2 High to Low Level Transformation Specification (15)

This artifact is the main artifact of the project. Based on artifact 4, it generates artifact 6, which actually does the real transformation.

In order to come up with artifact 15, we introduce an algorithm to make auto generation of artifact 6. This algorithm is general and can be applied to any example. Later we introduce PurchaseOrder example based on this algorithm. In order to get the algorithm, the outputs from previous sections and chapters are used. The algorithm consists of the following 3 main steps:

1. Define the XSLT stylesheet code that generates XSLT stylesheet code part of artifact 6
2. Define the XSLT stylesheet code that selects the transformation logic from artifact4 and output it in artifact 6.
3. Integration of two aforementioned steps to generate artifact 6 based on artifact 4

We follow the aforementioned algorithm steps to get artifact 15.

1. Define the XSLT stylesheet code that generates XSLT stylesheet code part of artifact 6

In this step we just create all code related to the other XSLT stylesheet, which is artifact 6.
Here, we should be care about the XSLT stylesheet tags. We are creating the XSLT stylesheet tags with using another XSLT stylesheet tags (code). The tags that we will get in a result and the actual tags that create that tags can be confused easily. To do not confuse the tags, we have several ways to go, i.e. several ways to code the stylesheets. These ways are:

1. Using 'fake' namespaces to code result XSLT stylesheet. For instance:

   ```
   <xsl:stylesheet version="1.0" xmlns:xsl=http://www.w3.org/1999/XSL/Transform
   xmlns:fake=http://www.w3.org/fakeXSLT>
   ```

   In this case, after transformation we should process the output to rename 'fake' name space to 'xsl' namespace in order to enable the transformation.

2. Using > and < entity references to code the result XSLT styleseet code. We tried this version in small example. It works well, but the readability of the code is not good. You can experience this in the following code:

```
<?xml version="1.0" encoding="UTF-8"?>
<xsl:transform version="1.0" xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
xmlns:prop="http://www.blabla.com/prop">
       <xsl:output method="xml" version="4.0" encoding="iso-8859-1"
indent="yes"/>
       <xsl:template match="/">
              <rdf:RDF xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#">
                     <xsl:variable name="nodename" select="name(/node())"/>
                     <rdf:Description
rdf:about="http://www.mysite.com/{$nodename}">
                           <prop:orderdate>
                                  <xsl:value-of
select="purchaseOrder/@orderDate"/>
                           </prop:orderdate>
                           <xsl:apply-templates select="/node()/*"/>
                     </rdf:Description>
              </rdf:RDF>
       </xsl:template>
       <xsl:template match="shipTo | billTo">

       &lt;prop:<xsl:value-of select="name(.)"/>&gt;

       <xsl:variable name="nodename" select="name(.)"/>
              <rdf:Description rdf:nodeID="{$nodename}inf">
                     <xsl:for-each select="./@* | ./*">

       &lt;prop:<xsl:value-of select="name(.)"/>&gt;
              <xsl:value-of select="."/>
       &lt;/prop:<xsl:value-of select="name(.)"/>&gt;

       </xsl:for-each>
```

```
        </rdf:Description>

        &lt;/prop:<xsl:value-of select="name(.)"/>&gt;

        </xsl:template>
        <xsl:template match="comment">
        &lt;prop:<xsl:value-of select="name(.)"/>&gt;
        <xsl:value-of select="."/>
        &lt;/prop:<xsl:value-of select="name(.)"/>&gt;
        </xsl:template>
        <xsl:template match="items">
        &lt;prop:<xsl:value-of select="name(.)"/>&gt;
        <xsl:variable name="nodename" select="name(.)"/>
                <rdf:Description rdf:nodeID="{$nodename}inf">
                        <xsl:for-each select="./item">
        &lt;prop:<xsl:value-of select="name(.)"/>&gt;
        <xsl:variable name="nodename2" select="name(.)"/>
        <rdf:Description rdf:nodeID="{$nodename2}inf">
        <xsl:for-each select="./@* | ./*">
                &lt;prop:<xsl:value-of select="name(.)"/>&gt;
                     <xsl:value-of select="."/>
                &lt;/prop:<xsl:value-of select="name(.)"/>&gt;
        </xsl:for-each>
                                </rdf:Description>
        &lt;/prop:<xsl:value-of select="name(.)"/>&gt;
 </xsl:for-each>
</rdf:Description>
  &lt;/prop:<xsl:value-of select="name(.)"/>&gt;
</xsl:template>
</xsl:transform>
```

As you can see from the code, the readability of code is not so good. However, the result XSLT code is generated correctly; we did not select this approach.

3. Coding the result XSLT stylesheet with <![CDATA[...]]> unparsed data notation. It will be difficult to code with commentary part like CDATA. Everything inside the CDATA section is ignored by the XLST processor. We need again additional processing of code to transform this intermediate code.
4. Creating new elements and attributes to code the resulting XSLT stylesheet (artifact 6). We have chosen this approach, because it is readable and you do not need additional text processing. We are creating elements and attribute for artifact 6, using XSLT's elements <xsl:element../> and <xsl:attribute... />.

   For instance, to creat the <xsl:output...> element code, we use the following code:

```
    <xsl:element name="xsl:output">
            <xsl:attribute name="method">xml</xsl:attribute>
            <xsl:attribute name="version">4.0</xsl:attribute>
            <xsl:attribute name="encoding">iso-8859-1</xsl:attribute>
            <xsl:attribute name="indent">yes</xsl:attribute>
    </xsl:element>
```

As a result of the transformation we will get the following XSLT stylesheet code:

```
<xsl:output method="xml" version="4.0" encoding="iso-8859-1" indent="yes"/>
```

So, we use the same logic during the whole generation issues.
Here are, we giving the list of the tasks to in this step, such as tasks to generate XSLT stylesheet part of code in artifact 6:

- instantiating the xsl:output element that will create also XML file as result of transformation, because XSLT file is XML file [`<xsl:output method="xml"....`]
- creating the root element of output XSLT stylesheet (artifact6) [<xsl:transform version="1.0"]
- xsl:template element that matches the target document [`<xsl:template match="/">`]
- RDF namespace is created for the output RDF document (artifact 10) [`<rdf:RDF xmlns:rdf="ht…`]
- Creating loops to select from nodeset [`<xsl:for-each select="//node()[@country='US']">`]
- Creating the elements and attributes [`<xsl:element name="{concat('prop:',name(name))}">`]

In the list above, rectangle brackets [], demonstrate the example of transformation from artifact 6.

2. Define the XSLT stylesheet code that selects the transformation logic from artifact4 and output it in artifact 6.

In the second step of the algorithm we use the same approach to create the elements and attributes of resulting XSLT stylesheet (artifact6). The goal of this step is to select the 'transformation logic' from artifact 4. This 'transformation logic' is Class, Attribute rule names, object identifiers, paths to select, relationship between Class and attribute rules. It will process Class and Attribute Rules based on the followings steps:

1. Process each ClassRule in artifact 4
2. Test whether the node element or attribute in source part of ClassRule
3. Check, test whether this node (element or attribute) has a predicate
4. When it has a predicate, place it as condition.
5. If it has not any predicate select without condition based on the node (element/attribute) name
6. Generate the code (in artifact 6) that will create RDF property with the name of the selected node in target RDF document (artifact 10)
7. For each property(attribute) in PropertyAssigments tag (refer to artifact 4), instantiate element with the name and values of that property (attribute).
8. Select the attributeRules that belongs to the selected ClassRule
9. Select the 'property path' that matches with the property name from the Class Rule (artifact 4)

10. Select the value according to the aforementioned property path

These steps are general ones. And it can be adapted to a particular example. We tried this algorithm in purchseOrder example.

We come up with XSLT stylesheet code part (step 1) and transformation logic selection part (step 2). Now, we should put together the results of two steps. It is done in third step.

3. Integration of two aforementioned steps to generate artifact 6 based on artifact 4

We already know about the steps to get the transformation logic and XSLT stylsheet code. To automate the generation of artifact 6, we are going to integrate aforesaid two steps. If we pay attention to the steps, the first one is much more involved in XSLT coding stuff and second one is much more focused on XPath and relationship issues between Class and Attribute Rules.

The idea of this integration step is to properly nest XPath and relationship issues (step 2) into XSLT code (step 1). It means, XPath expressions of XSLT code of step one, will be replaced by selection (XPath expressions) and relationship issues (XPath expressions' conditions). So, we can grasp this integration logic from symbolic figure 4.1.

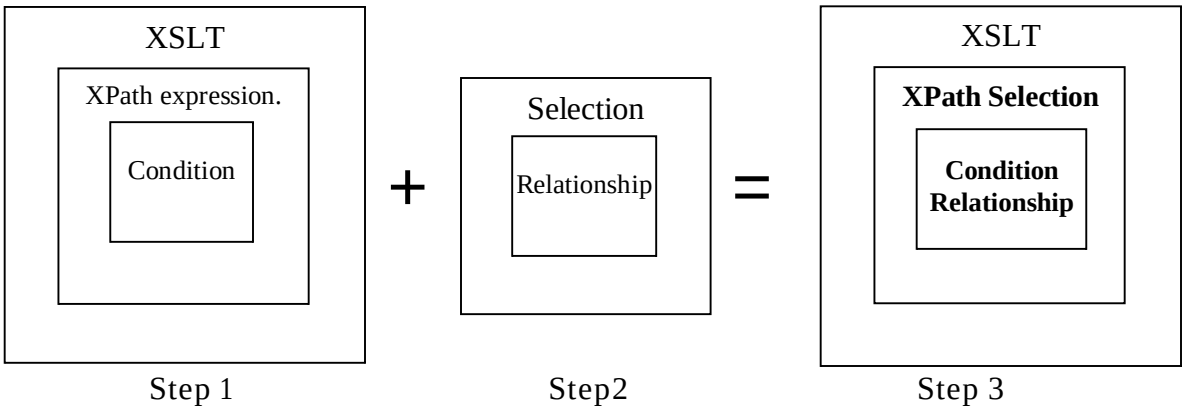


Figure 4.1 integration step of step 1 and step 2.

Based on the figure 4.1, we can see the integration of Step 1 and Step 2. Step 1 defines only XSLT code part of artifact 6. Step 2 consists of selection and relationship issues between Class and Attribute rules. XPath expressions can be integrated by 'selection' and XPath expression conditions can be put together with 'relationship issues'. According to that

XPath expressions will be selected based on selection issues from artifact 4. And the relationship issues on artifact 4 will take place in condition part of XPath expressions.

Apart from that, testing issues also should be considered in this integration. It can be done based on conditional processing of XSLT using <xsl:if test=””../> and <xsl:choose>.

These 3 steps were the main steps of the algorithm. We can use this algorithm to a specific example domain. We applied this algorithm for PurchaseOrder example. You can find this example in appendix F. There you can find the notes in form of commentaries to the each steps mentioned here.

After defining auto generation issues (artifact 15), we will discuss about the tools to make enable this auto generation in next section.

4 .2 Comparison of XPath/XSLT 1.0 and XPath/XSLT 2.0

To generate aforementioned artifacts we need to use XPath/XSLT 1.0 or XPath/ XSLT 2.0. In order to decide, which one to use, we define our demands, what we need.

We need to select a node or nodeset, which is the instance of schema component, from input document based on schema. It means we should be able to select the elements that are the instances of some schema component. This feature is missing in XSLT\XPATH 1.0. We decided to compare and reason about which version to use. We selected the instances of schema components based on attributes in XSLT 1.0.

XSLT 1.0 was published in November 1999, and version 2.0 represents a significant increase in the capability of the language. XSLT 2.0 has been developed in parallel with XPath 2.0
This brings an enhanced data model with a type system based on sequences of nodes or atomic values support for all the built-in types defined in XML Schema, and a wide range of new functions and operators.

We have investigated the new features of XPath\XSLT 2.0. In a nutshell, these are some new features of XPath, XSLT 2.0:

XPath 2.0 features. Table 4.1

XPath 2.0 features	Description
More general path expressions	XPath 2.0 has generalized its path expressions. Now, a primary expression (literal, function call, variable reference, or

	parenthesized expression) can appear at any step in the path, rather than just the first step
If . . .then . . .else	Expressions can now include bits of flow control within them, using if ... then ... else
For loops	You can now generate sequences with for loops

XPath new futures described in table 4.1 are used in XSLT 2.0. XSLT 2.0 supports XPath2.0. You can see the more features of XSLT 2.0 in table 4.2.

XSLT 2.0 features. Table 4.2	
XSLT 2.0 features	Description
48 data type, and optional use of schemas	XSLT 1.0 dealt with four types of data - strings, numbers, Booleans, and noesets. XSLT 2.0 has 48 atomic (built-in) data types, plus lists and unions constructed from them. With schema support, there is a new xsl:import-schema declaration. Every data type name that is not a built-in name must be defined in an imported schema. XPath expressions will be able to validate values against in-scope schemas, and will be able to use constructors and casts to imported data types.
Strong type checking	If . . .then . . .else
Works with XPath 2.0	XSLT 2.0 work together with XPath 2.0
Grouping	Processing groups of related elements is a problem that comes up repeatedly. The new xsl:for-each-group instruction makes life much easier to process groups of related elements.
Regular expression matching	String handling in XSLT has always been a tedious process. Both XSLT and XPath have added new facilities for dealing with regular expressions. The most complex of these is xsl:analyze-string, which takes an input string and a regular expression. It

	partitions the input into a set of substrings matching the expression. These can be processed by xsl:matching-substring and xsl:non-matching-substring instructions, which can construct any content required for each of them.
Multiple input and outputs	XSLT 1.0 allowed for a primary input document, auxiliary input via the document() function, and a single output. Version 2.0 provides for multiple inputs in several ways. There will be an input() function that provides access to a sequence of input nodes, and a collection() function that allows specification of a URI that defines a node sequence. Both of these provide access to multiple documents or document fragments. In addition, the unparsed-text() function will read arbitrary external resources (e.g., files) and return each one as a string.
User-defined functions	User-defined stylesheet functions allow creation of functions that can be called from within XPath expressions.

According the comparison XPath, XSLT 1.0 falls short in schema based selection. XPath, XSLT 2.0. have new features that can be used in our selection. XPath 1.0 does not support selection based on schema component. For instance, with XPath 1.0 you can not select an instance of Complex type, simple type, notation, identity constrain.. etc. XSLT and XPath 1.0 capabilities are not enough for that, therefore we started to investigate XPath and XSLT 2.0 to enable aforementioned selection.

First of all, we have studied XPath 2.0 [6] and [4], then XSLT 2.0 working draft[7].

In XSLT 2.0, there is a schema support, there is a new xsl:import-schema declaration. Every data type name that is not a built-in name must be defined in an imported schema. XPath expressions will be able to validate values against in-scope schemas, and will be able to use constructors and casts to imported data types.

In order process schema-aware XSLT stylesheets we need a XSLT processor that supports XSLT 2.0. According to [7], there are two kinds of XSLT processors: basic and schema aware XSLT processors. The latter is appropriate for our goal. “Stylesheets and Schemas” section of [7] describes facilities that are available only with a schema-aware XSLT processor.

Information from a schema can be used both statically (when the stylesheet is compiled), and dynamically (during evaluation of the stylesheet to transform a source document). There are places within a stylesheet, and within XPath expressions and patterns in a stylesheet, where it is possible to refer to named type definitions in a schema, or to element and attribute declarations. For example, it is possible to declare the types expected for the parameters of a function.

Additional schema components (type definitions, element declarations, and attribute declarations) may be added to the in-scope schema components by means of the xsl:import-schema declaration in a stylesheet.

Based on aforementioned information we rewrite USAddress transformation stylesheet in XSLT 2.0.

```
<?xml version="1.0" encoding="UTF-8"?>

<xsl:transform version="2.0" xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
           xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
           xmlns:prop="http://www.blabla.com/prop">

<xsl:output method="xml" version="4.0" encoding="iso-8859-1" indent="yes"/>
<xsl:import-schema namespace="http://www.w3.org/1999/xhtml"
                   schema-location="po.xsd" />
<xsl:template match="element(*, USAddress)" priority="2">

<rdf:RDF xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#">

  <xsl:for-each select="purchaseOrder/*[@country='US']">
      &lt;prop:<xsl:value-of select="name()"/>&gt;
      <xsl:variable name="nodename" select="name()"/>
      <rdf:Description rdf:nodeID="{$nodename}inf"
type="http://www.blabla.com/prop/USAddressCT">

  <xsl:for-each select="node()">
    &lt;prop:<xsl:value-of select="name(.)"/>&gt;
    <xsl:value-of select="."/>
    &lt;/prop:<xsl:value-of select="name(.)"/>&gt;
  </xsl:for-each>

  </rdf:Description>

  </xsl:for-each>
   </rdf:RDF>
</xsl:template>

<xsl:template match="document-node()" priority="1">
<xsl:message terminate="yes">source document is NOT VALID</xsl:message>
</xsl:template>
```

```
</xsl:transform>
```

In order to run the XSLT 2.0 code, we need a schema aware XSLT processor. These kinds of processors are not in the market so far. We find the XPath/XSLT 2.0 suitable to our project, but because of the missing XSLT processor, we keep going on our project in XPath/XSLT 1.0

4.3 Future work

In this project we make generation of XSLT stylesheet in Xpath/XSLT 1.0. We get some results, and the following issues can be improved in future work:

- Selection of the instances of the schema components. This issue can be solved with XPath/XSLT 2.o. It supports schema based selection. Because of not having schema-based XSLT processor, we make our specifications and transformation in XPath/XSLT 1.0. But this problem can be solved in near future after release of the schema-aware XSLT processors;
- More improved specification generation based on new features of XPath/XSLT 2.0. For instance: Transformation Specifications can be feed to XSLT processor from many input documents;
- Making the improvement on the current artifacts, that makes them available to support complex XML documents, which are the instances of XML schema.

4.4 Summary

This chapter have discussed about the generation of the transformation specification. Section 4.1 has conversed about general generation algorithm and some examples. The comparison of XPath/XSLT 1.0 and XPath/XSLT 2.0 has taken place in section 4.2. The future work is discussed in section 4.3.

5 Conclusion

This chapter contains a summary of the project in section 5.1 and overview and comparison of related work in section 5.2. And it finishes with section 5.3, which describes suggestions for future research.

5.1 Summary

The main problem of the project is to automate the XSLT stylesheet generation. To solve this problem, first of all we have defined the instances of models, models themselves, meta-model, some XML and Graph representations of them. Then come up with high level transformation specification. Finally, we automate the generation of XSLT stylesheets based on this high level transformation specification. All aforementioned models are described in terms of 18 artifacts as shown in figure 2.0. And later on, we refer to these artifacts during the whole report.

We discussed about the background of the problem, the problem statement, solution approach and the main building blocks, notions of our project in chapter 1.

In chapter 2, we have introduced definitions and concrete examples for each source and target artifacts.

In chapter 3, we define all source and target models and related documents. And we have given an example for each artifact and have described the relationships, mappings between artifacts. After having a concrete example for input and output document we come up with transformation specification to transform input (XML) to output document (RDF). Then we explained the main logic of high level transformation specification.

In chapter 4, we have described the implementation of the transformation. We make enable to automatically generate XSLT stylesheet based on high level transformation specification.

This solution could potentially lend a hand to automatically transform from the syntax (XML document) to the meaning (RDF document) of data. The strength of this approach is that it can be applied to any XML document based on its schema. And the weakness of this approach is that its auto generation based on XPath/XSLT 1.0, which has some limitations to generate the XSLT stylesheets. By releasing of schema-aware XSLT processors this approach can be improved with promising features of XPath/XSLT 2.0.

5.2 Comparison to other work

In this section, we discuss three other approaches and their useful features which can be adopted in our project.

5.2.1 XML processing based on model transformations

This thesis [17], concentrates on XML processing. The author tries to automate XML processing by XML data binding. XML data binding allows converting, translating or transforming an XML document to objects and the other way around.

He designed and implemented unmarshaler in Java to transform any XML document (that follows an XML schema) to Objects, which are the instance of classes. Here XML document and its XML Schema are given, but the structure of Classes is not static one, it changes, which should be dealt in particular manner. By dealing with this issue, he addressed the shortcoming of XML data binding.

He used model-to-model transformation based on MDA. By treating the XML schema as a source model and the application classes as a target model a transformation between these two models could be specified (see Figure 5.1).

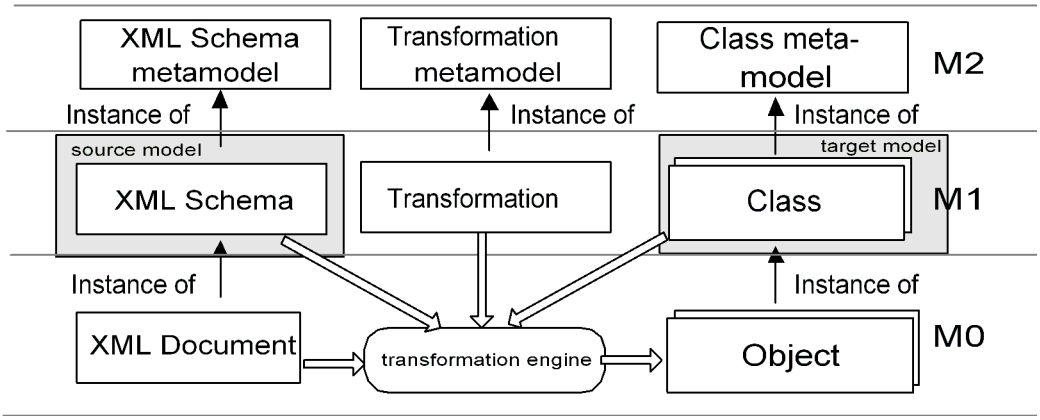


Figure 5.1 MDA based approach to XML processing

How does it relate to our Project?

In our case, we have RDF schema and RDF document instead of Classes and Object respectively. And we are using XSLT for transformations not Java. Transformation Engine of the related work relates to our several artifacts (see Fig.2.0). We are not implementing Transformation engine, just generating XSLT stylesheet (artifact 6) by another XSLT (artifact 15) using high level transformation specification (artifact 4). The important

difference is that, the RDF schema (artifact 9), which is given and not changing like Classes in the related work. We can use the same high level transformation specification used in this approach, but with adaptation to RDF domain.

5.2.2 Interpreting XML via an RDF Schema.

This paper [8], presents a procedure that can be used to turn XML documents into knowledge structures, by interpreting them as RDF data via an RDF-Schema specification. This allows semantic annotation of XML documents via external RDF-Schema specifications. This procedure could potentially multiply the availability of semantically annotated data.

How does it relate to our Project?

This approach can be used to come up with RDF document alternatives based on RDF schema. These alternatives will be used to evaluate the result of our transformation. Moreover, it can contribute to high level transformation specification to address the instantiations of RDF class and attributes. RDF schema grammar will be used to semantically annotate our XML document.

5.2.3 Model Driven Architecture based XML Processing

In this work [10], high level transformation language, that address mapping between XML schema and Classes, is described. It can be used in model-to-model transformation. Here the authors describe XML schema and Classes as source and target models respectively. The syntax of the transformation language and informally explains the semantics of the syntactical constructs. The syntax is specified in a grammar following EBNF notation. A given transformation contains declarative rules that encode how the syntax constructs defined in the source schema represent components in the target model. Rules also have procedural interpretation that allows their execution over instance XML documents.

How does it relate to our Project?

In our project, we need high level transformation specification (artifact 4), that will be used by an XSLT stylesheet (artifact 15) in order to generate another XSLT stylesheet (artifact 4), which actually perform transformation from input XML document to output RDF Document. We already have all artifacts, except 4 and 15.

And when we compare related work1 (5.2.1) and our project, the ‘real’ Transformation Engine in related work1 includes Transformation Compiler and Language Processor and implemented in Java programming

language. But in our case, 4 artifacts represent the Transformation Engine, which are two XSLT Processors (artifact5 and artifact 7), Low Level Transformation Specification (artifact 6) and High to Low Level Transformation Specification (artifact 15) and we use XSLT stylesheets for transformation.

We can use the same transformation specification, as artifact 4, from the related work. Than we need some adaptation to RDF and come up with High to Low Level Transformation Specification (artifact 15) .

5.3 Suggestions for Future Research

The results of this project could lend a hand on the following issues:

- The solution approach, presented in this project, can be used to make a transformation tool from XML to RDF.
- High level transformation specification can be generalized and can be used to transform to different target models, apart from RDF.

REFERENCES:

[1] Beckett, D. and McBride, B. RDF/XML Syntax Specification (Revised) W3C Recommendation, http://www.w3.org/TR/2004/REC-rdf-syntax-grammar-20040210/, 10 February 2004.

[2] Brickley, D, Guha, R. and McBride, B. RDF Vocabulary Description Language 1.0: RDF Schema W3C Recommendation. http://www.w3.org/TR/2004/REC-rdf-schema-20040210/, 10 February 2004.

[3] Brown, A., Fuchs, M., Robie, J. and Wadler, P. XML Schema: Formal Description W3C Working Draft. http://www.w3.org/TR/xmlschema-formal/, 25 September 2001.

[4] Draper, D. Fankhauser, P., Fernández, M., Malhotra, A., Rose, K., Rys, M., Siméon, J. and Wadler, P. XQuery 1.0 and XPath 2.0 Formal Semantics, W3C Working Draft 20. Work in progress. http://www.w3.org/TR/xquery-semantics/, February 2004.

[5] Fallside, D. XML schema part 0: Primer. W3C Recommendation. http://www.w3.org/TR/xmlschema-0/, May 2001.

[6] Kay, M. XML Path Language (XPath) 2.0, W3C Working Draft 12. Work in progress. http://www.w3.org/TR/xpath20/, November 2003.

[7] Kay, M. XSL Transformations (XSLT) Version 2.0, W3C Working Draft. Work in progress. http://www.w3.org/TR/2003/WD-xslt20-20031112/, 12 November 2003.

[8] Klein, M. Interpreting XML via an RDF schema, chapter in Knowledge Annotation for the Semantic Web IOS Press, Amsterdam, 2003.

[9] Klyne, G., Carroll, J. and McBride, B.
Resource Description Framework (RDF): Concepts and Abstract Syntax W3C Recommendation. http://www.w3.org/TR/2004/REC-rdf-concepts-20040210/, 10 February 2004.

[10] Kurtev, I. and van den Berg, K. Model Driven Architecture based XML Processing. In Proceeding of the 2003 ACM Symposium on Document Engineering.

[11] Lassila, O. and Swick, R. Resource Description Framework (RDF) Model and Syntax Specification W3C Recommendation. http://www.w3.org/TR/1999/REC-rdf-syntax-19990222/, 22 February 1999.

[12] Manola, F., Miller, E and McBride, B. RDF Primer W3C Recommendation. http://www.w3.org/TR/2004/REC-rdf-primer-20040210/,10 February 2004.

[13] Miller, J. and Mukerji, J. MDA Guide Version 1.0.1. http://www.omg.org/mda/mda_files/MDA_Guide_Version1-0.pdf, 12 June 2003.

[14] Object Management Group. (2002). *MetaObjectFacility(MOF) Specification*. http://www.omg.org/docs/formal/02-04-03.pdf.

[15] Object Management Group. Semantic Web activity http://www.w3.org/2001/sw/, May 2001.

[16] Quin, L. Extensible Markup Language (XML) http://www.w3.org/XML/.

[17] Rosheuvel, A. XML processing based on model transformations. MSc thesis at the Software Engineering Group. University of Twente, Enschede. December 2003.

[18] Thompson. H., Beech, D., Maloney, M. and Mendelsohn, N. XML Schema Part 1: Structures W3C Recommendation. http://www.w3.org/TR/xmlschema-1/, 2 May 2001.

Appendix A – Glossary

In this glossary the main terms used in this project are defined. You can find them in alphabetical order.

in-scope schema components
: The schema components that may be referenced by name in a stylesheet are referred to as the in-scope schema components. This set is the same throughout a stylesheet

model
: Defines a language to describe an Information Domain

meta model
: Defines a language to specifying model

MOF
: The Meta-Object Facility (MOF) technology provides a model repository that can be used to specify and manipulate models, thus encouraging consistency in manipulating models in all phases of the use of MDA [13]

UML
: The Unified Modeling Language (UML) is a standard modeling language for visualizing, specifying, and documenting software systems. Models used with MDA can be expressed using the UML language. UML 2 will integrate a set of concepts for completely specifying the behavior of objects, the UML action semantics [13]

MDA
: Model Driven Architecture (MDA) - an approach to IT system specification that separates the specification of functionality from the specification of the implementation of that functionality on a specific technology platform [13]

RDF
: The Resource Description Framework (RDF) is a general-purpose language for representing information in the Web. It provides a way to express simple statements about resources, using named properties and values

schema component

Schema component is the generic term for the building blocks that comprise the abstract data model of the schema. Type definitions and element and attribute declarations are referred to collectively as schema components

XML Schema

An XML Schema is a set of schema components

XSLT Processor

There are two types of XSLT processors: A basic and schema-aware XSLT processor. A basic XSLT processor is an XSLT processor that implements all the mandatory requirements of XSLT 2.0 specification with the exception of certain explicitly-identified constructs related to schema processing. A schema-aware XSLT processor is an XSLT processor that implements all the mandatory requirements of XSLT 2.0, including those features that a basic XSLT processor signals as an error.

Appendix B – RDF Schema (8)

RDF Schema defines the grammar of an RDF schema.
It is W3C Recommendation. In[2], RDF Schema defined as RDF/XML:

```
<rdf:RDF xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
xmlns:rdfs="http://www.w3.org/2000/01/rdf-schema#"
xmlns:owl="http://www.w3.org/2002/07/owl#">
 <owl:Ontology rdf:about="http://www.w3.org/2000/01/rdf-schema#"/>
 <rdfs:Class rdf:about="http://www.w3.org/2000/01/rdf-schema#Resource">
  <rdfs:isDefinedBy rdf:resource="http://www.w3.org/2000/01/rdf-schema#"/>
  <rdfs:label>Resource</rdfs:label>
  <rdfs:comment>The class resource, everything.</rdfs:comment>
 </rdfs:Class>
 <rdf:Property rdf:about="http://www.w3.org/1999/02/22-rdf-syntax-ns#type">
  <rdfs:isDefinedBy rdf:resource="http://www.w3.org/1999/02/22-rdf-syntax-ns#"/>
  <rdfs:label>type</rdfs:label>
  <rdfs:comment>The subject is an instance of a class.</rdfs:comment>
  <rdfs:range rdf:resource="http://www.w3.org/2000/01/rdf-schema#Class"/>
  <rdfs:domain rdf:resource="http://www.w3.org/2000/01/rdf-schema#Resource"/>
 </rdf:Property>
 <rdfs:Class rdf:about="http://www.w3.org/2000/01/rdf-schema#Class">
  <rdfs:isDefinedBy rdf:resource="http://www.w3.org/2000/01/rdf-schema#"/>
  <rdfs:label>Class</rdfs:label>
  <rdfs:comment>The class of classes.</rdfs:comment>
  <rdfs:subClassOf rdf:resource="http://www.w3.org/2000/01/rdf-schema#Resource"/>
 </rdfs:Class>
 <rdf:Property rdf:about="http://www.w3.org/2000/01/rdf-schema#subClassOf">
  <rdfs:isDefinedBy rdf:resource="http://www.w3.org/2000/01/rdf-schema#"/>
  <rdfs:label>subClassOf</rdfs:label>
  <rdfs:comment>The subject is a subclass of a class.</rdfs:comment>
  <rdfs:range rdf:resource="http://www.w3.org/2000/01/rdf-schema#Class"/>
  <rdfs:domain rdf:resource="http://www.w3.org/2000/01/rdf-schema#Class"/>
 </rdf:Property>
 <rdf:Property rdf:about="http://www.w3.org/2000/01/rdf-schema#subPropertyOf">
  <rdfs:isDefinedBy rdf:resource="http://www.w3.org/2000/01/rdf-schema#"/>
  <rdfs:label>subPropertyOf</rdfs:label>
  <rdfs:comment>The subject is a subproperty of a property.</rdfs:comment>
  <rdfs:range rdf:resource="http://www.w3.org/1999/02/22-rdf-syntax-ns#Property"/>
  <rdfs:domain rdf:resource="http://www.w3.org/1999/02/22-rdf-syntax-ns#Property"/>
 </rdf:Property>
 <rdfs:Class rdf:about="http://www.w3.org/1999/02/22-rdf-syntax-ns#Property">
  <rdfs:isDefinedBy rdf:resource="http://www.w3.org/1999/02/22-rdf-syntax-ns#"/>
  <rdfs:label>Property</rdfs:label>
  <rdfs:comment>The class of RDF properties.</rdfs:comment>
  <rdfs:subClassOf rdf:resource="http://www.w3.org/2000/01/rdf-schema#Resource"/>
 </rdfs:Class>
 <rdf:Property rdf:about="http://www.w3.org/2000/01/rdf-schema#comment">
  <rdfs:isDefinedBy rdf:resource="http://www.w3.org/2000/01/rdf-schema#"/>
  <rdfs:label>comment</rdfs:label>
  <rdfs:comment>A description of the subject resource.</rdfs:comment>
  <rdfs:domain rdf:resource="http://www.w3.org/2000/01/rdf-schema#Resource"/>
  <rdfs:range rdf:resource="http://www.w3.org/2000/01/rdf-schema#Literal"/>
 </rdf:Property>
 <rdf:Property rdf:about="http://www.w3.org/2000/01/rdf-schema#label">
  <rdfs:isDefinedBy rdf:resource="http://www.w3.org/2000/01/rdf-schema#"/>
  <rdfs:label>label</rdfs:label>
  <rdfs:comment>A human-readable name for the subject.</rdfs:comment>
  <rdfs:domain rdf:resource="http://www.w3.org/2000/01/rdf-schema#Resource"/>
  <rdfs:range rdf:resource="http://www.w3.org/2000/01/rdf-schema#Literal"/>
 </rdf:Property>
 <rdf:Property rdf:about="http://www.w3.org/2000/01/rdf-schema#domain">
  <rdfs:isDefinedBy rdf:resource="http://www.w3.org/2000/01/rdf-schema#"/>
  <rdfs:label>domain</rdfs:label>
  <rdfs:comment>A domain of the subject property.</rdfs:comment>
  <rdfs:range rdf:resource="http://www.w3.org/2000/01/rdf-schema#Class"/>
  <rdfs:domain rdf:resource="http://www.w3.org/1999/02/22-rdf-syntax-ns#Property"/>
```

```
 </rdf:Property>
 <rdf:Property rdf:about="http://www.w3.org/2000/01/rdf-schema#range">
  <rdfs:isDefinedBy rdf:resource="http://www.w3.org/2000/01/rdf-schema#"/>
  <rdfs:label>range</rdfs:label>
  <rdfs:comment>A range of the subject property.</rdfs:comment>
  <rdfs:range rdf:resource="http://www.w3.org/2000/01/rdf-schema#Class"/>
  <rdfs:domain rdf:resource="http://www.w3.org/1999/02/22-rdf-syntax-ns#Property"/>
 </rdf:Property>
 <rdf:Property rdf:about="http://www.w3.org/2000/01/rdf-schema#seeAlso">
  <rdfs:isDefinedBy rdf:resource="http://www.w3.org/2000/01/rdf-schema#"/>
  <rdfs:label>seeAlso</rdfs:label>
  <rdfs:comment>Further information about the subject resource.</rdfs:comment>
  <rdfs:range rdf:resource="http://www.w3.org/2000/01/rdf-schema#Resource"/>
  <rdfs:domain rdf:resource="http://www.w3.org/2000/01/rdf-schema#Resource"/>
 </rdf:Property>
 <rdf:Property rdf:about="http://www.w3.org/2000/01/rdf-schema#isDefinedBy">
  <rdfs:isDefinedBy rdf:resource="http://www.w3.org/2000/01/rdf-schema#"/>
  <rdfs:subPropertyOf rdf:resource="http://www.w3.org/2000/01/rdf-schema#seeAlso"/>
  <rdfs:label>isDefinedBy</rdfs:label>
  <rdfs:comment>The defininition of the subject resource.</rdfs:comment>
  <rdfs:range rdf:resource="http://www.w3.org/2000/01/rdf-schema#Resource"/>
  <rdfs:domain rdf:resource="http://www.w3.org/2000/01/rdf-schema#Resource"/>
 </rdf:Property>
 <rdfs:Class rdf:about="http://www.w3.org/2000/01/rdf-schema#Literal">
  <rdfs:isDefinedBy rdf:resource="http://www.w3.org/2000/01/rdf-schema#"/>
  <rdfs:label>Literal</rdfs:label>
  <rdfs:comment>The class of literal values, eg. textual strings and
integers.</rdfs:comment>
  <rdfs:subClassOf rdf:resource="http://www.w3.org/2000/01/rdf-schema#Resource"/>
 </rdfs:Class>
 <rdfs:Class rdf:about="http://www.w3.org/1999/02/22-rdf-syntax-ns#Statement">
  <rdfs:isDefinedBy rdf:resource="http://www.w3.org/1999/02/22-rdf-syntax-ns#"/>
  <rdfs:label>Statement</rdfs:label>
  <rdfs:subClassOf rdf:resource="http://www.w3.org/2000/01/rdf-schema#Resource"/>
  <rdfs:comment>The class of RDF statements.</rdfs:comment>
 </rdfs:Class>
 <rdf:Property rdf:about="http://www.w3.org/1999/02/22-rdf-syntax-ns#subject">
  <rdfs:isDefinedBy rdf:resource="http://www.w3.org/1999/02/22-rdf-syntax-ns#"/>
  <rdfs:label>subject</rdfs:label>
  <rdfs:comment>The subject of the subject RDF statement.</rdfs:comment>
  <rdfs:domain rdf:resource="http://www.w3.org/1999/02/22-rdf-syntax-ns#Statement"/>
  <rdfs:range rdf:resource="http://www.w3.org/2000/01/rdf-schema#Resource"/>
 </rdf:Property>
 <rdf:Property rdf:about="http://www.w3.org/1999/02/22-rdf-syntax-ns#predicate">
  <rdfs:isDefinedBy rdf:resource="http://www.w3.org/1999/02/22-rdf-syntax-ns#"/>
  <rdfs:label>predicate</rdfs:label>
  <rdfs:comment>The predicate of the subject RDF statement.</rdfs:comment>
  <rdfs:domain rdf:resource="http://www.w3.org/1999/02/22-rdf-syntax-ns#Statement"/>
  <rdfs:range rdf:resource="http://www.w3.org/2000/01/rdf-schema#Resource"/>
 </rdf:Property>
 <rdf:Property rdf:about="http://www.w3.org/1999/02/22-rdf-syntax-ns#object">
  <rdfs:isDefinedBy rdf:resource="http://www.w3.org/1999/02/22-rdf-syntax-ns#"/>
  <rdfs:label>object</rdfs:label>
  <rdfs:comment>The object of the subject RDF statement.</rdfs:comment>
  <rdfs:domain rdf:resource="http://www.w3.org/1999/02/22-rdf-syntax-ns#Statement"/>
  <rdfs:range rdf:resource="http://www.w3.org/2000/01/rdf-schema#Resource"/>
 </rdf:Property>
 <rdfs:Class rdf:about="http://www.w3.org/2000/01/rdf-schema#Container">
  <rdfs:isDefinedBy rdf:resource="http://www.w3.org/2000/01/rdf-schema#"/>
  <rdfs:label>Container</rdfs:label>
  <rdfs:subClassOf rdf:resource="http://www.w3.org/2000/01/rdf-schema#Resource"/>
  <rdfs:comment>The class of RDF containers.</rdfs:comment>
 </rdfs:Class>
 <rdfs:Class rdf:about="http://www.w3.org/1999/02/22-rdf-syntax-ns#Bag">
  <rdfs:isDefinedBy rdf:resource="http://www.w3.org/1999/02/22-rdf-syntax-ns#"/>
  <rdfs:label>Bag</rdfs:label>
  <rdfs:comment>The class of unordered containers.</rdfs:comment>
  <rdfs:subClassOf rdf:resource="http://www.w3.org/2000/01/rdf-schema#Container"/>
 </rdfs:Class>
 <rdfs:Class rdf:about="http://www.w3.org/1999/02/22-rdf-syntax-ns#Seq">
  <rdfs:isDefinedBy rdf:resource="http://www.w3.org/1999/02/22-rdf-syntax-ns#"/>
```

```
  <rdfs:label>Seq</rdfs:label>
  <rdfs:comment>The class of ordered containers.</rdfs:comment>
  <rdfs:subClassOf rdf:resource="http://www.w3.org/2000/01/rdf-schema#Container"/>
 </rdfs:Class>
 <rdfs:Class rdf:about="http://www.w3.org/1999/02/22-rdf-syntax-ns#Alt">
  <rdfs:isDefinedBy rdf:resource="http://www.w3.org/1999/02/22-rdf-syntax-ns#"/>
  <rdfs:label>Alt</rdfs:label>
  <rdfs:comment>The class of containers of alternatives.</rdfs:comment>
  <rdfs:subClassOf rdf:resource="http://www.w3.org/2000/01/rdf-schema#Container"/>
 </rdfs:Class>
 <rdfs:Class rdf:about="http://www.w3.org/2000/01/rdf-
schema#ContainerMembershipProperty">
  <rdfs:isDefinedBy rdf:resource="http://www.w3.org/2000/01/rdf-schema#"/>
  <rdfs:label>ContainerMembershipProperty</rdfs:label>
  <rdfs:comment>The class of container membership properties, rdf:_1, rdf:_2, ...,
                   all of which are sub-properties of 'member'.</rdfs:comment>
  <rdfs:subClassOf rdf:resource="http://www.w3.org/1999/02/22-rdf-syntax-
ns#Property"/>
 </rdfs:Class>
 <rdf:Property rdf:about="http://www.w3.org/2000/01/rdf-schema#member">
  <rdfs:isDefinedBy rdf:resource="http://www.w3.org/2000/01/rdf-schema#"/>
  <rdfs:label>member</rdfs:label>
  <rdfs:comment>A member of the subject resource.</rdfs:comment>
  <rdfs:domain rdf:resource="http://www.w3.org/2000/01/rdf-schema#Resource"/>
  <rdfs:range rdf:resource="http://www.w3.org/2000/01/rdf-schema#Resource"/>
 </rdf:Property>
 <rdf:Property rdf:about="http://www.w3.org/1999/02/22-rdf-syntax-ns#value">
  <rdfs:isDefinedBy rdf:resource="http://www.w3.org/1999/02/22-rdf-syntax-ns#"/>
  <rdfs:label>value</rdfs:label>
  <rdfs:comment>Idiomatic property used for structured values.</rdfs:comment>
  <rdfs:domain rdf:resource="http://www.w3.org/2000/01/rdf-schema#Resource"/>
  <rdfs:range rdf:resource="http://www.w3.org/2000/01/rdf-schema#Resource"/>
 </rdf:Property>
 <!-- the following are new additions, Nov 2002 -->
 <rdfs:Class rdf:about="http://www.w3.org/1999/02/22-rdf-syntax-ns#List">
  <rdfs:isDefinedBy rdf:resource="http://www.w3.org/1999/02/22-rdf-syntax-ns#"/>
  <rdfs:label>List</rdfs:label>
  <rdfs:comment>The class of RDF Lists.</rdfs:comment>
  <rdfs:subClassOf rdf:resource="http://www.w3.org/2000/01/rdf-schema#Resource"/>
 </rdfs:Class>
 <rdf:List rdf:about="http://www.w3.org/1999/02/22-rdf-syntax-ns#nil">
  <rdfs:isDefinedBy rdf:resource="http://www.w3.org/1999/02/22-rdf-syntax-ns#"/>
  <rdfs:label>nil</rdfs:label>
  <rdfs:comment>The empty list, with no items in it. If the rest of a list is nil
then the list has no more items in it.</rdfs:comment>
 </rdf:List>
 <rdf:Property rdf:about="http://www.w3.org/1999/02/22-rdf-syntax-ns#first">
  <rdfs:isDefinedBy rdf:resource="http://www.w3.org/1999/02/22-rdf-syntax-ns#"/>
  <rdfs:label>first</rdfs:label>
  <rdfs:comment>The first item in the subject RDF list.</rdfs:comment>
  <rdfs:domain rdf:resource="http://www.w3.org/1999/02/22-rdf-syntax-ns#List"/>
  <rdfs:range rdf:resource="http://www.w3.org/2000/01/rdf-schema#Resource"/>
 </rdf:Property>
 <rdf:Property rdf:about="http://www.w3.org/1999/02/22-rdf-syntax-ns#rest">
  <rdfs:isDefinedBy rdf:resource="http://www.w3.org/1999/02/22-rdf-syntax-ns#"/>
  <rdfs:label>rest</rdfs:label>
  <rdfs:comment>The rest of the subject RDF list after the first
item.</rdfs:comment>
  <rdfs:domain rdf:resource="http://www.w3.org/1999/02/22-rdf-syntax-ns#List"/>
  <rdfs:range rdf:resource="http://www.w3.org/1999/02/22-rdf-syntax-ns#List"/>
 </rdf:Property>
 <rdfs:Class rdf:about="http://www.w3.org/2000/01/rdf-schema#Datatype">
  <rdfs:isDefinedBy rdf:resource="http://www.w3.org/2000/01/rdf-schema#"/>
  <rdfs:label>Datatype</rdfs:label>
  <rdfs:comment>The class of RDF datatypes.</rdfs:comment>
  <rdfs:subClassOf rdf:resource="http://www.w3.org/2000/01/rdf-schema#Class"/>
 </rdfs:Class>
 <rdfs:Datatype rdf:about="http://www.w3.org/1999/02/22-rdf-syntax-ns#XMLLiteral">
  <rdfs:subClassOf rdf:resource="http://www.w3.org/2000/01/rdf-schema#Literal"/>
  <rdfs:isDefinedBy rdf:resource="http://www.w3.org/1999/02/22-rdf-syntax-ns#"/>
  <rdfs:label>XMLLiteral</rdfs:label>
```

```
      <rdfs:comment>The class of XML literal values.</rdfs:comment>
     </rdfs:Datatype>
     <rdf:Description rdf:about="http://www.w3.org/2000/01/rdf-schema#">
      <rdfs:seeAlso rdf:resource="http://www.w3.org/2000/01/rdf-schema-more"/>
     </rdf:Description>
    </rdf:RDF>
```

Appendix C – Transformation Grammar (16)

This transformation grammar (artifact 16) describes the rules and grammar for a high level transformation specification (artifact4).

```
<?xml version="1.0" encoding="UTF-8"?>
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema">
  <!--tranformationRules: This is the root element -->
  <xs:element name="transformation" type="transformationType"/>
  <!-- tranformationType -->
  <xs:complexType name="transformationType">
    <xs:sequence>
      <xs:element ref="transformationRule" maxOccurs="unbounded"/>
    </xs:sequence>
  </xs:complexType>
  <!-- transformationRule -->
  <xs:element name="transformationRule" type="transformationRuleType" abstract="true"/>
  <!-- transformationRuleType -->
  <xs:complexType name="transformationRuleType" abstract="true"/>
  <!-- classRuleType -->
  <xs:complexType name="classRuleType">
    <xs:complexContent>
      <xs:extension base="transformationRuleType">
        <xs:sequence>
          <xs:element name="inheritedRules" type="inheritedRulesType" minOccurs="0"/>
          <xs:element name="supersededRules" type="supersededRulesType" minOccurs="0"/>
          <xs:element ref="source"/>
          <xs:element ref="action" maxOccurs="unbounded"/>
          <xs:element ref="attributeRule" minOccurs="0" maxOccurs="unbounded"/>
        </xs:sequence>
        <xs:attribute ref="name" use="required"/>
        <xs:attribute name="abstract" type="xs:boolean" default="false"/>
      </xs:extension>
    </xs:complexContent>
  </xs:complexType>
  <!-- source -->
  <xs:element name="source" type="ruleSourceType"/>
  <!-- name -->
  <xs:attribute name="name" type="xs:string"/>
  <!-- classRuleType -->
  <xs:element name="classRule" type="classRuleType"
substitutionGroup="transformationRule"/>
  <!-- parentRulesType -->
  <xs:complexType name="inheritedRulesType">
    <xs:sequence maxOccurs="unbounded">
      <xs:element name="classRule" type="inheritedRuleType"/>
    </xs:sequence>
  </xs:complexType>
  <!-- parentRuleType -->
  <xs:complexType name="inheritedRuleType">
    <xs:attribute ref="name" use="required"/>
  </xs:complexType>
  <!-- supersededRulesType -->
  <xs:complexType name="supersededRulesType">
    <xs:sequence maxOccurs="unbounded">
      <xs:element name="supersededRule" type="supersededRuleType"/>
    </xs:sequence>
  </xs:complexType>
  <!-- supersededRuleType -->
  <xs:complexType name="supersededRuleType">
```

```
    <xs:attribute ref="name" use="required"/>
  </xs:complexType>
  <!-- ruleSourceType -->
  <xs:complexType name="ruleSourceType">
    <xs:choice>
      <xs:element ref="attribute"/>
      <xs:element ref="element"/>
    </xs:choice>
  </xs:complexType>
  <!-- component -->
  <xs:element name="attribute" type="componentType"/>
  <xs:element name="element" type="componentType"/>
  <!-- componentType-->
  <xs:complexType name="componentType">
    <xs:sequence>
      <xs:element name="predicate" type="xs:string" minOccurs="0"/>
    </xs:sequence>
    <xs:attribute ref="name" use="required"/>
    <xs:attribute name="path" type="xs:string" use="optional"/>
  </xs:complexType>
  <!-- action -->
  <xs:element name="action" type="actionType" abstract="true"/>
  <!-- actionType -->
  <xs:complexType name="actionType"/>
  <!--instantiationType -->
  <xs:element name="instantiation" type="instantiationType" substitutionGroup="action"/>
  <!-- instantiationType -->
  <xs:complexType name="instantiationType">
    <xs:complexContent>
      <xs:extension base="actionType">
        <xs:sequence>
          <xs:element ref="propertyAssignments" minOccurs="0"/>
        </xs:sequence>
        <xs:attribute name="objectIdentifier" type="xs:string" use="optional"/>
        <xs:attribute name="className" type="xs:string" use="required"/>
      </xs:extension>
    </xs:complexContent>
  </xs:complexType>
  <!-- propertyAssignments -->
  <xs:element name="propertyAssignments" type="propertiesType"/>
  <!-- propertiesType -->
  <xs:complexType name="propertiesType">
    <xs:sequence>
      <xs:element ref="property" maxOccurs="unbounded"/>
    </xs:sequence>
  </xs:complexType>
  <!-- property -->
  <xs:element name="property" type="propertyType"/>
  <!-- propertyType -->
  <xs:complexType name="propertyType">
    <xs:attribute ref="name" use="required"/>
    <xs:attribute name="expression" type="xs:string" use="optional"/>
  </xs:complexType>
  <!-- attributeRule -->
  <xs:element name="attributeRule" type="attributeRuleType"
substitutionGroup="transformationRule"/>
  <!-- attributeRuleType -->
  <xs:complexType name="attributeRuleType">
    <xs:complexContent>
      <xs:extension base="transformationRuleType">
        <xs:sequence>
          <xs:element ref="source"/>
          <xs:element name="target" type="targetAttributeType" maxOccurs="unbounded"/>
        </xs:sequence>
        <xs:attribute ref="ownerClassRule" use="required"/>
        <xs:attribute ref="name" use="required"/>
      </xs:extension>
    </xs:complexContent>
  </xs:complexType>
  <!-- ownerClassRule -->
  <xs:attribute name="ownerClassRule" type="xs:string"/>
```

```
  <!-- targetAttributeType-->
  <xs:complexType name="targetAttributeType">
    <xs:complexContent>
      <xs:extension base="propertyType">
        <xs:attribute name="objectIdentifier" type="xs:string" use="optional"/>
      </xs:extension>
    </xs:complexContent>
  </xs:complexType>
  <!-- expression -->
  <xs:element name="expression" type="xs:string"/>
</xs:schema>
```

Appendix D – High level Transformation Specification (4)

This high level transformation specification defines Class and Attribute rules for PurchaseOrder example.

```
<?xml version="1.0" encoding="UTF-8"?>
<transformation xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:noNamespaceSchemaLocation="C:\LOCAL\MMMM\WORK\DOC21\IMPL\artifact16.xsd">
 <classRule name="billToUSAddress">
  <source>
   <element name="billTo">
    <predicate>@country='US'</predicate>
   </element>
  </source>
  <instantiation className="rdfAddress" objectIdentifier="AddressID">
   <propertyAssignments>
    <property name="name"/>
    <property name="street"/>
    <property name="city"/>
    <property name="state"/>
    <property name="zip"/>
    <property name="country"/>
   </propertyAssignments>
  </instantiation>
 </classRule>
 <classRule name="shipToUSAddress">
  <source>
   <element name="shipTo">
    <predicate>@country='US'</predicate>
   </element>
  </source>
  <instantiation className="rdfAddress" objectIdentifier="rdfAddress">
   <propertyAssignments>
    <property name="name"/>
    <property name="street"/>
    <property name="city"/>
    <property name="state"/>
    <property name="zip"/>
    <property name="country"/>
   </propertyAssignments>
  </instantiation>
 </classRule>
 <classRule name="Product_item">
  <source>
   <element name="item"/>
  </source>
  <instantiation className="rdfItem" objectIdentifier="ItemID">
   <propertyAssignments>
    <property name="productName"/>
    <property name="quantity"/>
    <property name="USPrice"/>
    <property name="comment"/>
    <property name="partNum"/>
```

```
  </propertyAssignments>
 </instantiation>
</classRule>
<attributeRule name="name" ownerClassRule="billToUSAddress">
 <source>
  <element name="name" path="name"/>
 </source>
 <target name="name" objectIdentifier="rdfName"/>
</attributeRule>
<attributeRule name="street" ownerClassRule="billToUSAddress">
 <source>
  <element name="street" path="street"/>
 </source>
 <target name="street" objectIdentifier="rdfStreet"/>
</attributeRule>
<attributeRule ownerClassRule="Product_item" name="productName">
 <source>
  <element name="productName" path="productName"/>
 </source>
 <target name="ProductName" objectIdentifier="itemName"/>
</attributeRule>
<attributeRule ownerClassRule="Product_item" name="productQuantity">
 <source>
  <element name="quantity" path="quantity"/>
 </source>
 <target name="quantity" objectIdentifier="itemquantity"/>
</attributeRule>
<attributeRule ownerClassRule="Product_item" name="productPrice">
 <source>
  <element name="USPrice" path="USPrice"/>
 </source>
 <target name="USPrice" objectIdentifier="itemPrice"/>
</attributeRule>
<attributeRule ownerClassRule="Product_item" name="comment">
 <source>
  <element name="comment" path="comment"/>
 </source>
 <target name="ProducComment" objectIdentifier="itemComment"/>
</attributeRule>
<attributeRule ownerClassRule="Product_item" name="partNum">
 <source>
  <attribute name="partNum" path="@partNum"/>
 </source>
 <target name="ProductpartNum" objectIdentifier="itempartNum"/>
</attributeRule>
<attributeRule name="city" ownerClassRule="billToUSAddress">
 <source>
  <element name="city" path="city"/>
 </source>
 <target name="city" objectIdentifier="rdfCity"/>
</attributeRule>
<attributeRule name="state" ownerClassRule="billToUSAddress">
 <source>
  <element name="state" path="state"/>
 </source>
 <target name="state" objectIdentifier="rdfState"/>
</attributeRule>
<attributeRule name="zip" ownerClassRule="billToUSAddress">
 <source>
  <element name="zip" path="zip"/>
 </source>
 <target name="zip" objectIdentifier="rdfZip"/>
</attributeRule>
<attributeRule name="country" ownerClassRule="billToUSAddress">
 <source>
  <attribute name="country" path="@country"/>
 </source>
 <target name="country" objectIdentifier="rdfCountry"/>
</attributeRule>
<attributeRule name="name" ownerClassRule="shipToUSAddress">
 <source>
```

```
   <element name="name" path="name"/>
  </source>
  <target name="name" objectIdentifier="rdfName"/>
 </attributeRule>
 <attributeRule name="street" ownerClassRule="shipToUSAddress">
  <source>
   <element name="street" path="street"/>
  </source>
  <target name="street" objectIdentifier="rdfStreet"/>
 </attributeRule>
 <attributeRule name="city" ownerClassRule="shipToUSAddress">
  <source>
   <element name="city" path="city"/>
  </source>
  <target name="city" objectIdentifier="rdfCity"/>
 </attributeRule>
 <attributeRule name="state" ownerClassRule="shipToUSAddress">
  <source>
   <element name="state" path="state"/>
  </source>
  <target name="state" objectIdentifier="rdfState"/>
 </attributeRule>
 <attributeRule name="zip" ownerClassRule="shipToUSAddress">
  <source>
   <element name="zip" path="zip"/>
  </source>
  <target name="zip" objectIdentifier="rdfZip"/>
 </attributeRule>
 <attributeRule name="country" ownerClassRule="shipToUSAddress">
  <source>
   <attribute name="country" path="@country"/>
  </source>
  <target name="country" objectIdentifier="rdfCountry"/>
 </attributeRule>
</transformation>
```

Appendix E – Low level Transformation specification (6)

This artifact was written manually to transform from XML document to RDF document. This code will help to get the artifact 15, which is making generation of XSLT stylesheets automatically.

```
<?xml version="1.0" encoding="UTF-8"?>
<xsl:transform xmlns:xsl="http://www.w3.org/1999/XSL/Transform" version="1.0">
  <xsl:output method="xml" version="4.0" encoding="iso-8859-1" indent="yes"/>
  <xsl:template match="/">
    <rdf:RDF xmlns:prop="http://www.blabla.com/properties"
xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#">
      <xsl:for-each select="//billTo[@country='US']">
        <xsl:element name="{concat('prop:',name(.))}">
          <rdf:Description nodeID="{concat(name(.),'inf')}">
            <rdf:type resource="http://www.zonozz.org/billToUSAddress"/>
            <xsl:element name="prop:name">
              <xsl:value-of select="name"/>
            </xsl:element>
            <xsl:element name="prop:street">
              <xsl:value-of select="street"/>
            </xsl:element>
            <xsl:element name="prop:city">
              <xsl:value-of select="city"/>
            </xsl:element>
            <xsl:element name="prop:state">
              <xsl:value-of select="state"/>
            </xsl:element>
            <xsl:element name="prop:zip">
              <xsl:value-of select="zip"/>
            </xsl:element>
            <xsl:element name="prop:country">
              <xsl:value-of select="@country"/>
            </xsl:element>
          </rdf:Description>
        </xsl:element>
      </xsl:for-each>
      <xsl:for-each select="//shipTo[@country='US']">
        <xsl:element name="{concat('prop:',name(.))}">
          <rdf:Description nodeID="{concat(name(.),'inf')}">
            <rdf:type resource="http://www.zonozz.org/shipToUSAddress"/>
            <xsl:element name="prop:name">
              <xsl:value-of select="name"/>
            </xsl:element>
            <xsl:element name="prop:street">
              <xsl:value-of select="street"/>
            </xsl:element>
            <xsl:element name="prop:city">
              <xsl:value-of select="city"/>
            </xsl:element>
            <xsl:element name="prop:state">
              <xsl:value-of select="state"/>
            </xsl:element>
            <xsl:element name="prop:zip">
              <xsl:value-of select="zip"/>
            </xsl:element>
            <xsl:element name="prop:country">
              <xsl:value-of select="@country"/>
            </xsl:element>
          </rdf:Description>
        </xsl:element>
      </xsl:for-each>
    </rdf:RDF>
  </xsl:template>
</xsl:transform>
```

Appendix F – High to Low Level Transfromation Specification (15)

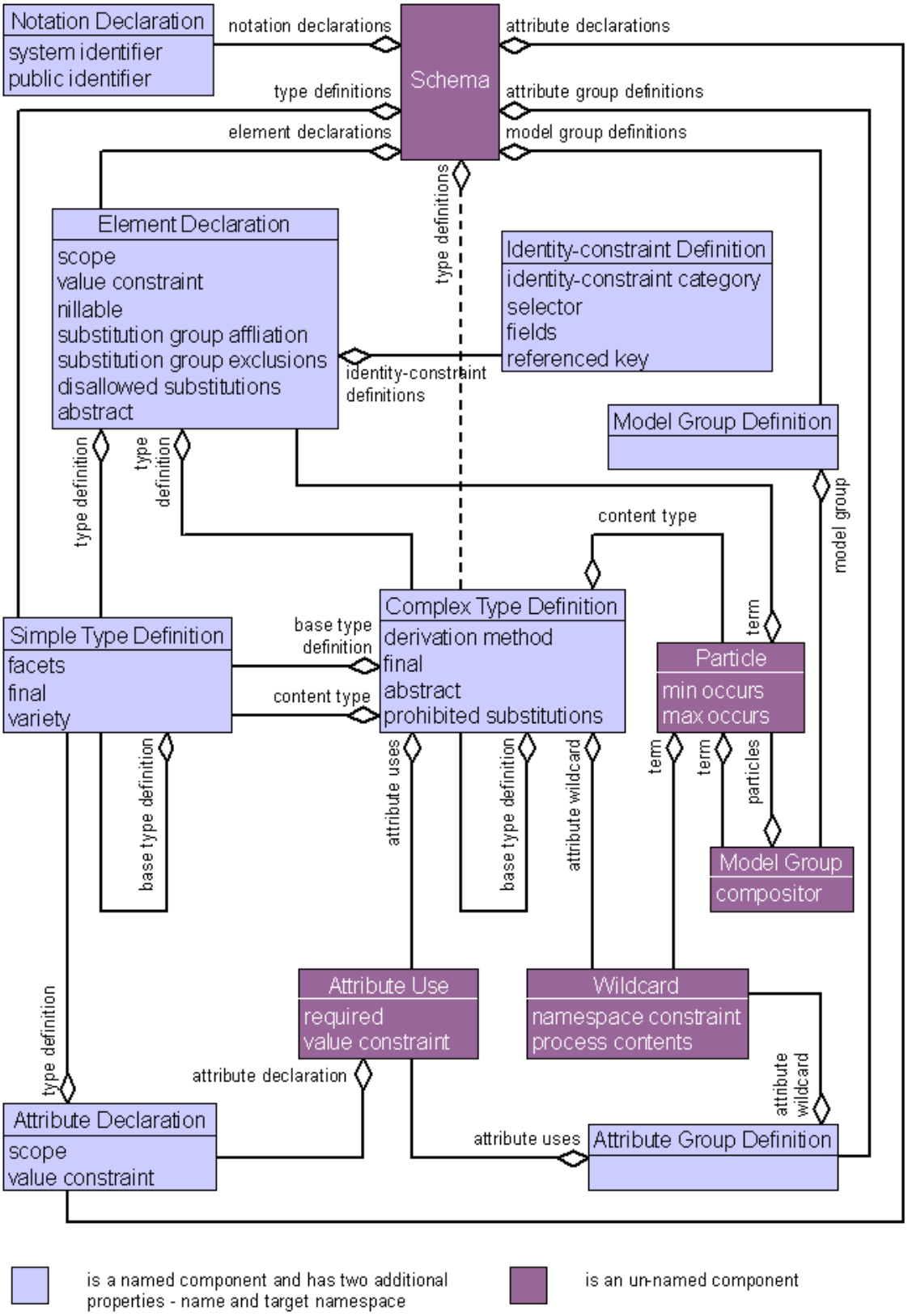
This artifact is the result of the step by step applying the algorithm defined in chapter 4. It is applied to PurchaseOrder domain.

```
<?xml version="1.0" encoding="UTF-8"?>
<xsl:stylesheet version="1.0" xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
xmlns:prop="http://www.blabla.com/properties">
 <xsl:output method="xml" version="1.0" encoding="UTF-8" indent="yes"/>
 <xsl:template match="/">
  <xsl:element name="xsl:transform">
   <xsl:attribute name="version">1.0</xsl:attribute>
   <xsl:element name="xsl:output">
    <xsl:attribute name="method">xml</xsl:attribute>
    <xsl:attribute name="version">4.0</xsl:attribute>
    <xsl:attribute name="encoding">iso-8859-1</xsl:attribute>
    <xsl:attribute name="indent">yes</xsl:attribute>
   </xsl:element>
   <xsl:element name="xsl:template">
    <xsl:attribute name="match">/</xsl:attribute>
    <rdf:RDF xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#">
     <xsl:for-each select="transformation/classRule">
      <xsl:element name="xsl:for-each">
       <xsl:if test="name(source/element)='element'">
        <xsl:variable name="source" select="source/element/predicate"/>
        <xsl:choose>
         <xsl:when test="$source!=''">
          <xsl:variable name="predicate" select="concat('[',$source,']')"/>
          <xsl:variable name="name" select="source/element/@name"/>
          <xsl:attribute name="select">
                       <xsl:value-of select="concat('//',$name,$predicate)"/>
              </xsl:attribute>
         </xsl:when>
         <xsl:otherwise>
          <xsl:variable name="predicate" select="$source"/>
          <xsl:variable name="name" select="source/element/@name"/>
          <xsl:attribute name="select">
                      <xsl:value-of select="concat('//',$name,$predicate)"/>
                </xsl:attribute>
         </xsl:otherwise>
        </xsl:choose>
       </xsl:if>
       <xsl:if test="name(source/attribute)='attribute'">
        <xsl:variable name="source" select="source/attribute/predicate"/>
        <xsl:choose>
         <xsl:when test="$source!=''">
          <xsl:variable name="predicate" select="concat('[',$source,']')"/>
          <xsl:variable name="name" select="source/attribute/@name"/>
          <xsl:attribute name="select">
                       <xsl:value-of select="concat('//@',$name,$predicate)"/>
          </xsl:attribute>
         </xsl:when>
         <xsl:otherwise>
          <xsl:variable name="predicate" select="$source"/>
          <xsl:variable name="name" select="source/attribute/@name"/>
          <xsl:attribute name="select">
                       <xsl:value-of select="concat('//@',$name,$predicate)"/>
          </xsl:attribute>
         </xsl:otherwise>
        </xsl:choose>
       </xsl:if>
       <xsl:element name="xsl:element">
        <xsl:attribute name="name">{concat('prop:',name(.))}</xsl:attribute>
```

```
        <xsl:element name="rdf:Description">
         <xsl:attribute name="nodeID">{concat(name(.),'inf')}</xsl:attribute>
         <xsl:variable name="class_name" select="@name"/>
         <xsl:element name="rdf:type">
          <xsl:attribute name="resource">
                        <xsl:value-of select="concat('http://www.zonozz.org/',@name)"/>
          </xsl:attribute>
         </xsl:element>
         <xsl:for-each select="instantiation/propertyAssignments/property">
          <xsl:element name="xsl:element">
           <xsl:variable name="property_name" select="@name"/>
           <xsl:variable name="path"
select="//attributeRule[@ownerClassRule=$class_name]"/>
           <xsl:variable name="property_path"
select="$path/source/element/@path[../@name=$property_name] |
$path/source/attribute/@path[../@name=$property_name]"/>
           <xsl:attribute name="name">
                         <xsl:value-of select="concat('prop:',$property_name)"/>
           </xsl:attribute>
           <xsl:element name="xsl:value-of">
            <xsl:attribute name="select">
                             <xsl:value-of select="$property_path"/>
            </xsl:attribute>
           </xsl:element>
          </xsl:element>
         </xsl:for-each>
        </xsl:element>
       </xsl:element>
      </xsl:element>
     </xsl:for-each>
    </rdf:RDF>
   </xsl:element>
  </xsl:element>
 </xsl:template>
</xsl:stylesheet>
```

Appendix F – XML Schema Components Diagram (1)

This diagram consists of name and un-named components. In order to get detailed view of The XML Schema we can look to [18]. It is A Schema for Schemas (normative). All structure of XML-Schema is described by XML-Schema.



XML Schema component model