From Document Type Definitions to Metamodels: The WebML Case Study

Manuel Wimmer[1,‡] , Andrea Schauerhuber[2,*],
Elisabeth Kapsammer[3], and Gerhard Kramler[1,‡]

[1] Business Informatics Group
Institute of Software Technology and Interactive Systems
University of Technology, Vienna, Austria
wimmer@big.tuwien.ac.at
[2] Women's Postgraduate College for Internet Technologies
Institute of Software Technology and Interactive Systems
University of Technology, Vienna, Austria
schauerhuber@wit.tuwien.ac.at
[3] Information Systems Group
University of Linz, Austria
ek@ifs.uni-linz.ac.at

Abstract. Metamodels are a prerequisite for model-driven engineering (MDE). In the past, DTDs have also been deployed for language definitions. MDE techniques and tools can not be reused for such languages, however. The WebML web modeling language for modeling web applications is one example that does not yet rely on an explicit metamodel. Instead it is implicitly defined within the methodology's accompanying WebRatio tool in terms of a DTD, i.e., a grammar-like textual definition for specifying the structure of XML documents. Code generation then has to rely on XSLT-based model-to-code transformations.

In this paper, we propose a semi-automatic approach for generating metamodels from DTDs. We introduce a set of transformation rules and heuristics for automatically generating MOF-based metamodels from DTD definitions and suggest manual refactorings for the semantic enrichment of the metamodels. We incorporate the transformation rules and heuristics in a *MetaModelGenerator (MMG)* prototype implementation, and provide results on applying our approach to the WebML DTD. The benefits from our work are: (1) visualization of DTD-based languages in terms of MOF-based metamodels enhances understanding of the language, (2) employment of metamodels enables further steps towards model-driven engineering such as model transformations or language extensions through profiles, (3) tool interoperability with other MDE tools is ensured, (4) a proposal for a WebML metamodel has been generated in a semi-automatic way.

‡ This work has been partly funded by the Austrian Federal Ministry of Transport, Innovation and Technology (BMVIT) and FFG under grant FIT-IT-810806.

* This work has been partly funded by the Austrian Federal Ministry for Education, Science, and Culture, and the European Social Fund (ESF) under grant 31.963/46-VII/9/2002.

1 Introduction

XML [37], a Wide Web Consortium (W3C) standard, evolved from HTML [38] in order to cater for several of HTML's limitations, e.g., layout-independence, no extension mechanism, etc. At the turn of the millennium, XML became the silver bullet in software engineering and particularly in web engineering and today is used for different purposes such as data exchange (ebXML [18]), protocol languages (SOAP [41]), configuration languages (e.g., for web frameworks like Struts[1]), multi-delivery (WML [25]), and in many other domain-specific areas (MathML [39], SVG [40], etc.). XML-based languages can be defined by Document Type Definitions (DTD) [37] or alternatively, using XML Schema [42], a language written in XML syntax which is more expressive than DTD [5]. XML documents have to be valid with respect to their DTD or XML Schema, and therefore, DTD as well as XML Schema represent meta-languages for defining XML-based languages (cf. Section 2). A transformation language for XML technologies is the XSLT language [43] which is suitable to transform an XML document into another form.

Recently, model-driven engineering (MDE) [1] has received considerable attention and is well on its way to becoming the new paradigm in software engineering. In MDE, models replace code as the primary artifacts in the software development process. It forces developers to focus on modeling the problem domain and not on programming one possible (platform-specific) solution. Thus, the abstraction from specific programming platforms by modeling at a platform-independent level, and the definition of model transformations, allows generating *several* platform-specific implementations. A prominent example for MDE is the Object Management Group's (OMG) Model Driven Architecture (MDA) [20], which is based on OMG's modeling language, i.e., the Unified Modeling Language (UML) [24], and meta-modeling language, i.e., the Meta Object Facility (MOF) [21]. In particular, MOF - as a meta-modeling language - provides means to define modeling languages in addition to OMG's UML and CWM [19]. Analogous to XML documents, models have to conform to their metamodels (cf. Section 2). While MDA provides the Query/Views/Transformation (QVT) [28] specification as a transformation mechanism, there currently exists no implementation. Still, there are non-standard transformation languages such as the ATLAS Transformation Language (ATL)[2].

The *WebML* web modeling language is one example that relies on XML technologies and whose concepts are defined in terms of a DTD. In contrast to MOF's expressivity, however, DTDs represent a rather restricted mechanism to describe languages. Moreover, the text-based representation of DTDs hampers on the one hand their readability and understandability for humans, and on the other hand the language's extensibility. Additionally, WebML [8] comes with the *WebRatio* modeling tool, which first, internally represents models in XML, and second, uses XSLT for code generation. Since XSLT, however, has not been intended for heavy structural transformations, writing XSLT programs for code generation is hard and error prone.

[1] http://struts.apache.org/

[2] http://www.sciences.univ-nantes.fr/lina/atl/atl

Concerning the problems identified above, a metamodel approach allows expressing transformation rules in a more compact and readable way by using existing model transformation languages such as ATL[3].

In this work, we present a semi-automatic approach to generating MOF-based metamodels from XML-based languages which are defined using DTDs. In this respect the following contributions have been made: First, a MOF-based metamodel of the DTD language has been defined. Second, a set of transformation rules and heuristics for generating metamodels from DTDs has been formulated. Third, necessary user interactions for improving the automatically generated metamodel have been identified. Forth, a prototype for a *MetaModelGenerator* incorporating the transformation rules and heuristics has been implemented and a WebML metamodel has been generated and refactored.

The remainder of this paper is organized as follows. Section 2 presents the architecture of our metamodel generation framework with respect to OMG's four-layer architecture [23] and provides an overview of DTD and MOF concepts. Section 3 discusses a set of rules and heuristics for transforming DTDs to MOF-based metamodels as well as required manual improvements of the automatically generated metamodel. An EMF-based prototype implementation for the *MetaModelGenerator* and the WebML case study are presented in Section 4 and Section 5, respectively. Section 6 gives an overview on related work. Finally, we outline future work and conclusions in Section 7.

2 DTDs and Metamodels

Formal languages require precise definitions in terms of a meta-language in order to be interpretable by computers. In the past, various meta-languages have been employed for defining formal languages. Amongst them are EBNF [36] for describing the syntax of (programming) languages, DTD for defining the elements and attributes of XML documents, and MOF which represents the state-of-the-art for defining modeling languages. In Figure 1, we illustrate these relationships and our transformation framework within the realms of OMG's four-layer architecture.

According to [4], the relation between a model and its metamodel is also related to the relation between a program and the programming language in which it is written, defined by its grammar, or between an XML document and the defining XML schema or DTD. Hence, in OMG's four-layer architecture DTDs can be assigned to the same layer (M2) as metamodels and XML documents can be assigned to the same layer (M1) as models. In particular, the middle part of Figure 1 depicts the relationship between on the one hand languages (*M2*), e.g., specific DTDs such as the WebML DTD, general-purpose metamodels like UML, and domain-specific metamodels, and on the other hand, representations of the real world (*M1*), e.g., XML documents, and (UML) models. The upper part of Figure 1indicates the fact that

[3] In [26] a comparison between a specific model-transformation language and XSLT for transforming UML models to JAVA models proves the benefits of model-transformations based on metamodels over XSLT.

languages themselves also must be formally defined in terms of a meta-language (*M3*). A DTD must conform to the DTD-grammar described in EBNF and metamodels must conform to MOF. *Correspondences* between language elements of the DTD-grammar and MOF can be used for transforming a particular DTD to a MOF-based metamodel. These generic correspondences are implemented as transformation rules in the *MetaModelGenerator* (cf. Figure 1), which takes a DTD as input and produces its corresponding MOF-based metamodel.

To gain a better understanding of the correspondences between DTD and MOF, an overview of the language concepts of DTD (cf. Section 2.1) and MOF (cf. Section 2.2) is given in the following subsections. It has to be emphasized that it is not necessary to explain all language concepts, but only those parts being relevant for the mapping between DTD and MOF.

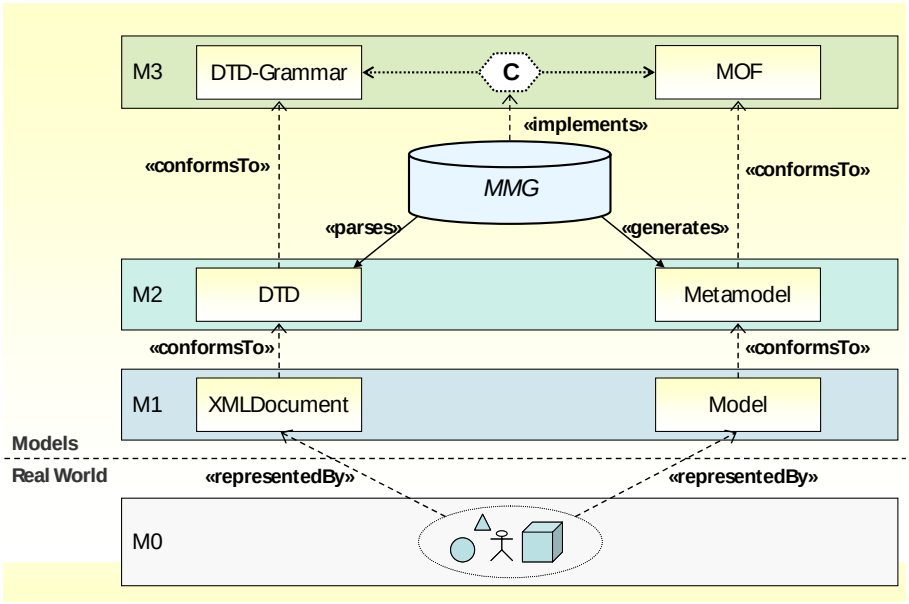

Figure 1: Language Layers and the MetaModelGenerator

2.1 Document Type Definition Language Concepts

The XML *document type declaration* in XML documents contains or points to *markup declarations* that provide a grammar, i.e., the *document type definition* (DTD) for a class of documents [37]. A markup declaration is an *element type declaration*, an *attribute-list declaration*, an *entity declaration*, or *a notation declaration*. While the first two declarations are called *logical structures* and are used for defining structured data, the latter two declarations are called *physical structures* and usually are used for defining unstructured data.

In this work we particularly focus on logical structures, since metamodels themselves represent structured data. Therefore, the logical structures must be transformed completely and correctly into metamodel representations. In the following, we briefly describe element type and attribute-list declarations.

- *Element type declarations* are *first-class citizens* in DTDs. Element types (*XMLElemType*) have a name and are specialized into *XMLAtomicET* (contains no other element types but character data), *XMLEmptyET* (no content is allowed), *XMLAnyET* (the content is not constrained – this declaration is not adequate for language definitions and is therefore missing in Figure 3), *XMLCompositeET-MixedContent* (a mix of character data and child element types), and *XMLCompositeETElemContent* (consists of an *XMLContentParticle*). An *XMLContentParticle* is either an *XMLSequence*, an *XMLChoice*, or an *XMLElementType*). An *XMLChoice* or an *XMLSequence* can be enclosed in parentheses for grouping purposes and suffixed with a '?' (zero or one occurrences), '*' (zero or more occurrences), or '+' (one or more occurrences). For a single element type the cardinality can also be described by one of the three mentioned cardinality symbols. The absence of a particular symbol, however, denotes a cardinality of exactly one.
- *Attribute-list declarations* declare one or multiple *XMLAttributes* (i.e., name-value pairs) for a single element type. Each *XMLAttribute* has a *name*, a *data type*, and a *default declaration*. The most commonly used data types for attributes are: CDATA (String), ID, IDREF (refers to one ID-typed element), IDREFS (refers to multiple ID-typed elements), and Enumeration. There are four possibilities for default declarations: #IMPLIED (zero or one), #REQUIRED (exactly one), #FIXED (the attribute value is constant and immutable), and Literal (the default value is a quoted string).

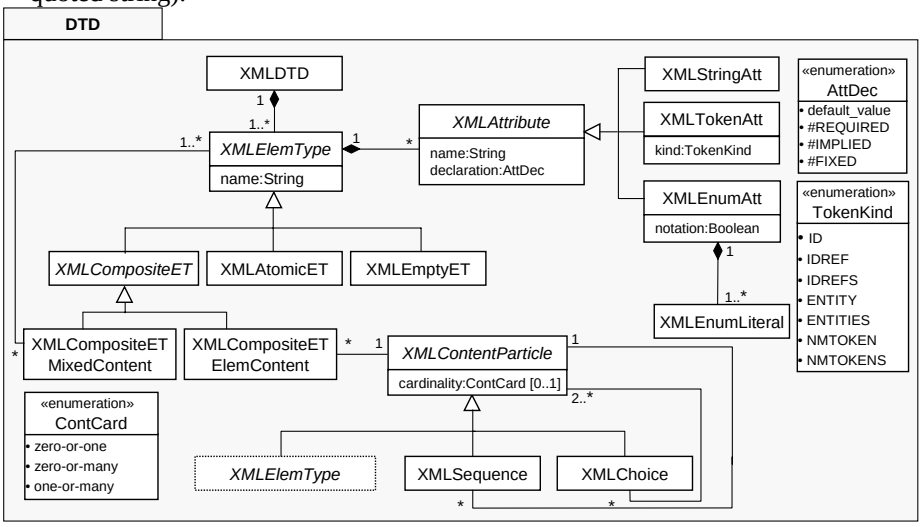

Figure 2: Overview of DTD language concepts

Physical structures are used for embedding not well-formed XML data in a document, e.g., JPEG pictures, as *external unparsed entities*. Because this kind of data is unstructured, it is not possible to transform these specifications directly into meta-

model definitions. Still, some information from physical structures can be used in heuristics in order to improve the quality of the metamodel (cf. Section 4).
To gain a better understanding of the DTD language concepts we build on previous work [13] and reengineer a MOF-based model from the DTD-grammar described in EBNF [36]. In Figure 2 the resulting metamodel is shown as UML class diagram.

2.2 Meta Object Facility Language Concepts

The *Meta Object Facility* (MOF) [21] is the standardized meta-modeling language of the OMG. MOF consists of two parts, namely *essential MOF* (EMOF) and *complete MOF (*CMOF*)*. While the former is a small language based on the principles of object-orientation for defining modeling languages, the latter is a more complex language which provides also concepts for the specification of implementation details of model repositories. Since there is no standardized implementation of MOF available by the time of writing, we are using a slightly modified MOF implementation in Java, which is provided by *Eclipse Modeling Framework* (EMF)[4]. The meta-modeling language in EMF is called *Ecore* and approximately corresponds to EMOF, whose concepts are sufficient in the context of this paper. Figure 3 shows the Ecore language concepts and their relationships as UML class diagram.

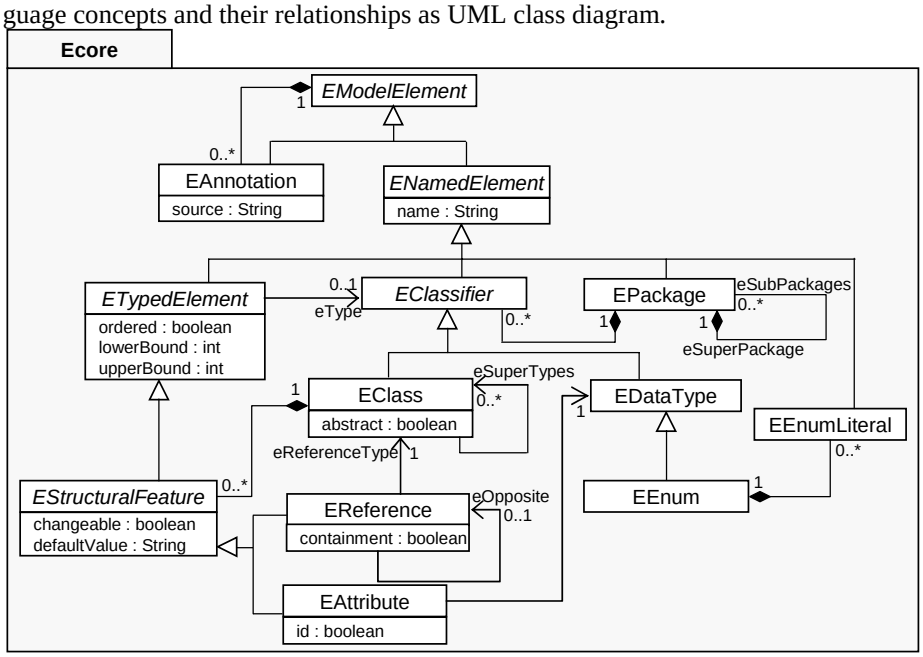

Figure 3: Overview of Ecore language constructs[5]

[4] http://www.eclipse.org/emf

[5] http://download.eclipse.org/tools/emf/2.2.0/javadoc/org/eclipse/emf/ecore/package-summary.html#details

In the following, we give a summary of the relevant Ecore language constructs instead of the corresponding EMOF counterparts in order not to confuse the reader with marginal differences between Ecore and EMOF.

- *EClasses* are the *first-class citizens* in Ecore-based metamodels. An *EClass* has multiple *EReferences* and *EAttributes* for defining its properties, as well as multiple *super classes*.
- *EAttribute* is part of a specific *EClass*. The data type of an attribute is either a simple data type or an enumeration, i.e., *EEnum*. Additionally, an attribute can have a lower and an upper bound multiplicity.
- *EReference* is analogous to *EAttribute* part of a specific *EClass* and can have a lower and an upper bound multiplicity. In addition, *EReferences* refer to *EClasses* and optionally to an opposite *EReference* for expressing bi-directional relationships. Besides, a reference can be declared as *containment reference* (part-of relationship).
- *EPackages* group *EClasses*, *EEnums*, as well as nested *EPackages*. Each element is directly owned by a package and each package can contain multiple model elements.
- *EDataTypes* are available for defining the types of attributes. *String*, *Boolean*, *Integer*, and *Float* are part of Ecore's default data types set.
- *EEnum* is suitable for modeling enumerations of literals and can be used as an attribute's data type. An *EEnum* owns an arbitrary amount of values i.e., *EEnumLiterals*.
- *EAnnotations* are used for describing additional information which cannot be presented directly in Ecore-based metamodels. Each model element can have multiple annotations and each annotation belongs to a specific model element.

3 Deriving Metamodels from DTDs

In this section, we describe a two-step process for generating metamodels from DTDs. While in the first step, a preliminary metamodel can be automatically generated using a set of transformation rules and heuristics (cf. Section 3.1), in the second phase explicit user interaction is required in order to improve the semantics of the metamodel by certain refactoring actions (cf. Section 3.2).

3.1 Step 1: Transformation Rules and Heuristics

The semi-automatic generation of the metamodels is achieved by two techniques: first, generic transformation rules for transforming DTD concepts into metamodel concepts, which need no user interaction and second heuristics which support the transformation rules and require some user validation. In the following, we describe a complete set of transformation rules for logical structures of DTDs and provide a brief summary in Table 1. Each heuristics is presented in terms of a textual description and a concrete example.

3.1.1 Transformation Rules

In the following the transformation rules are listed in the order in which they are applied to a DTD in the MetaModelGenerator (cf. Section 4). Consequently, first transformation rules for *XML Element Types* are presented followed by transformation rules for *XML Attributes*.

Rule 1 - *DTD::XMLElemType_2_Ecore::EClass*: For each *element type* an *EClass* is created, where the name of the EClass is set to the element type name. For each subclass of the class *XMLElemType* (cf: Figure 2) additional metamodel elements have to be created in the transformation process. Thus, the following case differentiations are necessary in Rule 1:

(1) *XMLEmptyET* – this type requires no further transformation.
(2) *XMLAtomicET* - a separate attribute of data type *String* for #PCDATA has to be created and attached to the corresponding class of the element type.
(3) *XMLCompositeETElemContent* – the contained element types are represented in metamodels by introducing *EReferences* between the owner class (representing the container element type) and the owned class (representing the contained element type). In order to express the *part/whole relationship* the reference is defined as a *containment reference*.
(4) *XMLCompositeETMixedContent* – in addition to the EReferences for the contained element types, an EAttribute for representing the PCDATA is required.
(5) *XMLSequence* and *XMLChoice* - these two subtypes are exceptional cases, because they are not directly reproducible in metamodels structures with the default Ecore concepts. Therefore, we introduce two annotations, *«SEQ»* and *«ALT»*, representing special marks for EClasses to simulate sequences and choices in metamodels in a similar way as in RDFS [44] with *rdf:Seq* and *rdf:Alt* containers. Contained element types of sequences and choices are attached to the sequence and choice classes by containment references. For sequences the order information is annotated on the reference ends.

Rule 1.1 - *DTD::XMLContentParticle.cardinality_2_Ecore::Reference.multiplicity*: Each XMLC*ontentParticle* can have one of the following cardinalities which are represented in metamodels through *multiplicity* (*lower/upper bound*) of the *reference end*:

(1) *Zero-or-one* (?) – reference end with multiplicity *0..1*
(2) *Zero-or-more* (*) – reference end with multiplicity *0..**
(3) *One-or-more* (+) – reference end with multiplicity *1..**
(4) *Default, no symbol* – reference end with multiplicity *1*

Rule 2 - *DTD::XMLAttribute_2_ECore::EAttribute*: For each *XMLAttribute* an *EAttribute* is created. The created *EAttribute* is attached to the *EClass* representing the element type which in turn owns the *XMLAttribute*. The name of the *EAttribute* is the name of the *XMLAttribute*. For the majority, the *data type* of *XMLAttributes* is one out of the following set: {*CDATA, ID, IDREF, IDREFS, Enumeration*}. For each of these possibilities an appropriate transformation is required as listed in the following:

(1) *CDATA* corresponds to the data type *String*.
(2) *ID* likewise corresponds to the data type *String.* Additionally, the *EAttribute* is marked as the *identifier* of the EClass by setting the *id* attribute of the *EAttribute* to *true*. This mapping is correct, although the semantics of identifiers in XML is different from the identifiers in Ecore. In XML, an identifier means unique identification of an element within a document. In contrast, in Ecore, an identifier means unique identification of an object within a set of objects of a specific class. Nevertheless, it is possible to represent an *XMLAttribute* of type ID as an *EAttribute* with id set to *true*, because in XML the restriction is stronger. However, this means, going the reverse direction is not that easy.
(3) *IDREF* is mapped to an *EAttribute* with data type *String*. Furthermore an *EAnnotation* with the literal *«IDREF must be resolved manually»* is assigned to the *EAttribute*, since IDREF doesn't specify any information with respect to the reference element type. In case *Heuristic 1* (cf. Section 3.2) finds the intended referenced class, an *EReference* is produced instead of an *EAttribute*.
(4) *IDREFS* is mapped to an *EAttribute* of type *String* with *EAnnotation «IDREFS must be resolved manually»* in case *Heuristic 1* (cf. Section 3.2) does not find the intended referenced class.
(5) For each *XMLEnumeration* an *EEnum* is generated, except *Heuristic 2* (cf. Section 3.2) detects that the *XMLEnumeration* represents an attribute of type *Boolean*. Then, for each literal of the *XMLEnumeration* an *EEnumLiteral* of the *EEnum* is produced. In DTDs, enumerations are defined for each attribute in isolation, possibly resulting in redundant definitions of the same enumerations. In the corresponding metamodels, however, multiple classes can reuse the same enumeration. Thus, before an *EEnum* is finally created, it is verified whether an identical enumeration already exists. In this case, the *EAttribute's* type is set to the already existing enumeration.

Rule 2.1 - *DTD::XMLAttribute.cardinality_2_Ecore::EAttribute.multiplicity*: Attributes in both, DTDs and in metamodels, have a certain kind of cardinality. In DTDs, the cardinality of an *XMLAttribute* is determined on the one hand, by the differentiation between *single-valued* (e.g., *ID*, *CDATA*, or *IDREF*) and *multi-valued* (e.g., *IDREFS*), and on the other hand, by the *XMLAttribute declaration* (*#REQUIRED*, *#IMPLIED*, *#FIXED*, and *default value*). In the following, we discuss how *XMLAttribute* cardinalities (i.e., combinations of single-valued/multi-valued and *XMLAttribute* declaration values) are transformed into *EAttribute* multiplicities:
(1) In case a *default value* for an attribute declaration is specified, the *multiplicity* of the corresponding *EAttribute* is set to *exactly one* if the *XMLAttribute* is single-valued and *one or more* if in case of multi-valued. The *default value* of the *XMLAttribute* is assigned as the *defaultValueLiteral* of the *EAttribute*.
(2) *FIXED XMLAttributes* are represented in metamodels as *EAttributes* with multiplicity one in case of single-valued, or one-to-more in case of multi-valued. Furthermore, the *default value* of the *XMLAttribute* is assigned to the *defaultValueLiteral* attribute of the *EAttribute* and the *changeable* attribute of the *EAttribute* is set to *false*.

(3) *REQUIRED XMLAttributes* are transformed into *EAttributes* with multiplicity of exactly one in case of singe-valued, and multiplicity of one-or-more in case of multi-valued *XMLAttributs*.
(4) *IMPLIED XMLAttributes* are optional and therefore transformed into an *EAttribute* with multiplicity zero-to-one in case of single-valued, and zero-to-more in case of multi-valued *XMLAttributes*.

Table 1 summarizes the proposed transformation rules. Some transformation rules are supported by heuristics which lead to improved readability and higher quality of the metamodel but require some user validation. These heuristics are explained in the next subsection.

Table 1: Transformation rules between DTD Constructs and Ecore Constructs

	Rule	DTD Concept		Ecore Concept
XML Element Type	*R 1*	*XMLElementType (ET)*		*EClass*
		XMLElementType. name		*EClass.name*
	(1)	XMLEmptyET		no additional elements
	(2)	XMLAtomicET		EAttribute for PCDATA
	(3)	XMLCompositeET ElemContent		Containment References
	(4)	XMLCompositeET MixedContent		Containment References, EAttribute for PCDATA
	(5)	Sequence, Choice		EClasses annotated with *«SEQ»* and *«ALT»*, resp.
	R1.1	*XMLContentParticle.cardinality*		*EReference.multiplicity*
	(1)	Zero-or-one (?)		0..1
	(2)	Zero-or-more (*)		0..*
	(3)	One-or-more (+)		1..*
	(4)	Default, no symbol		1
XML Attribute	*R2*	*XMLAttribute*		*EAttribute*
		XMLAttribute.name		*EAttribute.name*
	(1)	CDATA		String
	(2)	ID		String, Attr. *id* set true
	(3)	IDREF		String or *Heuristic 1*
	(4)	IDREFS		String or *Heuristic 1*
	(5)	XMLEnumeration		EEnum or *Heuristic 2*
		XMLLiteral		EEnumLiteral
	R2.1	*XMLAttribute.cardinality*		*EAttribute.multiplicity*
	(1)	Default value	Single-valued	1 (defaultValue)
			Multi-valued	1..* (defaultValue)
	(2)	#FIXED	Single-valued	1 (dV, unchangeable)
			Multi-valued	1..* (dV, unchangeable)
	(3)	#REQUIRED	Single-valued	1
			Multi-valued	1..*
	(4)	#IMPLIED	Single-valued	0..1
			Multi-valued	0..*

3.1.2 Heuristics

As mentioned before, heuristics are useful in the generation process to achieve a higher-quality metamodel. Note, that the application of heuristics requires user validation, because their modifications to metamodel elements are suggestions, only. Furthermore, the effectiveness of the heuristics is strongly correlated with the quality of the design of the DTDs. For example, the heuristics operate more effective if naming conventions, e.g., for IDREFs, are used (cf. Heuristic 1), or the content of the DTD is split up into several external DTDs which group related elements (cf. Heuristic 3).

In the following the heuristics of our framework are described. These heuristics are used to exploit the semantically rich language constructs of Ecore, namely (1) *typed references*, (2) *data types*, and (3) *packages* as a grouping mechanism.

Heuristic 1 (IDREF(S) Resolution): A DTD does not restrict which element types can be referenced from an attribute of data type IDREF or IDREFS. Thus, it is possible to reference any element having an ID attribute in an XML document from any IDREF or IDREFS attribute. Due to this peculiarity of DTDs, it is not possible to determine which element type may be referenced based on the information given in the DTD. To solve this ambiguity, general knowledge of the problem domain is required. Still, sometimes it is possible to find the referenced element types relying on naming conventions of element types and attributes (cf. Figure 4). Considering an element type *Relationship* which has an attribute named *entity* of type IDREF and a second element type Entity, one can assume that the second element is referenced by the attribute *entity*.

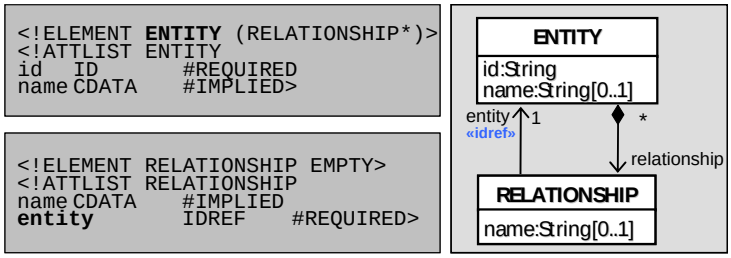

Figure 4: Example for Heuristic 1

Heuristic 1 is designed to find such name-matches in DTDs. This is done for each IDREF(S) attribute by iterating over the set of generated EClasses and verifying if the attribute name matches (*ignore lower/upper case*) one of the class names. If a match is found, the iteration is terminated and an EReference from the class which owns the IDREF(S) attribute to the identified class is generated. The multiplicity of the reference end is set to the multiplicity of the attribute. Note, that the user still must validate the generated references in order to detect random name-matches, which means that a referenced class is not the intended referenced element. Thus, the generated reference end is annotated with *«IDREDF(S)»*. In Figure 4, the element type *Relationship* on the left hand side refers to the element type *Entity* by the attribute *entity* of type of IDREF. In the metamodel, this is represented as a reference between the class

Relationship and *Entity* with role name *entity* and multiplicity 1 (because the XML attribute has been defined as #REQUIRED).

Heuristic 2 (Boolean Identification): In DTDs, there is no direct way to specify an XML attribute of type *Boolean*. Instead, an element's attribute can be of type *Enumeration* with two literals *true* and *false*. In this case *Rule 2* produces an *EEnumeration* with two literals, namely *true* and *false*. But for this special case an attribute of type *Boolean* is much richer in terms of semantics and represent a more compact specification than the corresponding enumeration. *Heuristic 2* recognizes such optimization possibilities and generates an attribute of type *Boolean* for the following sets of enumeration literals: {*true, false*}, {*1, 0*}, {*on, off*}, and {*yes, no*}. In addition the user can configure the *MMG* with additional synonyms for *true* and *false*. An example use case for *Heuristic 2* is illustrated in Figure 5.

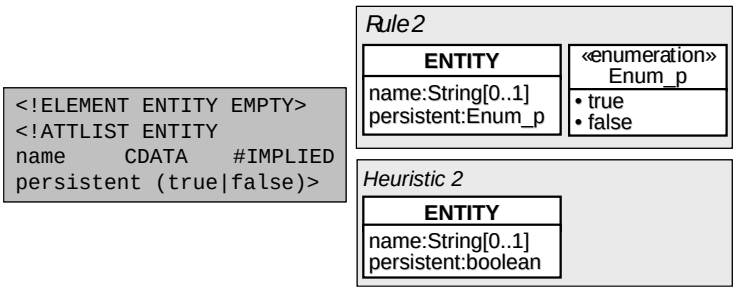

Figure 5: Example for Heuristic 2

The MMG receives as input for transformation a DTD consisting of the element *Entity* with an attribute *persistent*. The attribute in turn is of type *Enumeration* with the literals *true* and *false*. On the upper right hand side of Figure 5 the corresponding model elements are shown resulting from a transformation according to *Rule 2*. The lower right hand side instead shows the corresponding model elements resulting from a transformation according to *Heuristic 2*.

Heuristic 3 (Grouping Mechanism): In DTDs, there is no mechanism for grouping related element declarations. In metamodels, on the contrary, packages are the intended grouping mechanism. A package groups *packageable elements* which are primary classes and nested packages. This feature allows hierarchically structured metamodels, which are more readable and better understandable than flattened metamodels. In DTDs, the grouping mechanism can be simulated by defining *external DTDs* and referencing these in a so called *root-DTD*. The root-DTD is equivalent to the *root-package* in a metamodel and the external DTDs are equivalent to *subpackages* of the root package. If the user designs DTDs in this manner, it is possible to configure the *MMG* to generate a package for each external DTD and one root package for the root-DTD. Figure 6 shows an example use case for Heuristic 3. On the left hand side two DTDs are shown, whereas the DTD called *WebML* imports the element declarations of the DTD called *Structure*. On the right hand side the corresponding package structure and classes are shown for the generated metamodel.

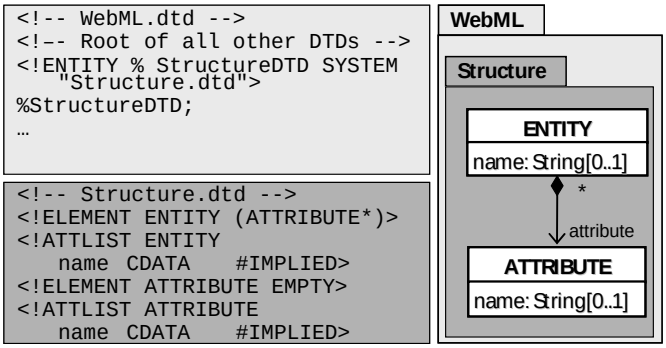

Figure 6: Example for Heuristic 3

3.2 Step 2: Semantic enrichment of generated metamodels

The last step towards a MOF-based metamodel requires user interaction for semantic enrichment, as well as validation of the automatically produced metamodel. As already mentioned, such user interactions are required because DTDs are poorer in semantics than metamodels, which is due to the limited set of constructs of the DTD language. The most relevant semantic enrichment tasks concern the following problems of DTDs: (1) DTDs have no concepts to express *inheritance.* (2) DTDs have a limited set of data types that can not be extended (e.g., to support *Integer* and *Boolean* data types). (3) The resolution of IDREF(S) requires domain knowledge. (4) DTDs have no constructs to describe bi-directional associations. In the following, these peculiarities are described in more detail. In particular, we provide the required user interactions to reduce the semantic ambiguities.

- In DTDs there is no means to *inherit* attributes and associations from the same kind of *super* elements. This means that attributes and references are defined for each element in isolation even if there is a super entity in the problem domain which owns these shared properties. Consequently, the automatically generated metamodels lack this information. In metamodels, however, inheritance is a central object-oriented concept. Thus, the user has to manually refactor the generated metamodels in order to achieve inheritance relationships and reduce redundant definitions of attributes and references, leading to an improved structure and higher readability. For this task we suggest to introduce new classes - normally marked as abstract classes – which collect common properties of available *subclasses*.
- In DTDs the only built-in data type is *String* which is expressed by defining an attribute of type CDATA. The proposed heuristics are able to reengineer attributes of type *Boolean*, but there is no way to reengineer attributes of type *Integer*. Attributes which are intended to be *Integers* are normally defined as CDATA in DTDs, because there is no other possibility. In contrast, the *Integer* data type is very important in metamodels, e.g., for an attribute named *lowerBound*, which

represents a multiplicity constraint. In the generated metamodels, the user must thus check all attributes if any of them should be of type *Integer*. This task requires domain knowledge and is not automatically contrivable.

- Currently, our framework's resolution mechanism for IDREF(S) is limited to name-matches. Some IDREF(S) therefore, are automatically resolved according to *Heuristic 1*. Due to the possibility of random name-matches, the user has to validate, if the resolution of the IDREF(S) is correct or another class should be referenced. Therefore, the user has to verify reference ends with annotation *«IDREDF(S)»* if the resolution is correct.
- The framework currently marks all IDREF(S) attributes that could not be resolved by naming conventions. Thus, the user has to refactor all attributes which are marked with the annotation *«IDREF(S) must be resolved manually»*. Knowledge of the problem domain is required to create the corresponding references to the intended referred classes. The multiplicity of the reference end has to be set to the multiplicity of the attribute. The role name of the reference end is typically set to the attribute's name.
- It is not possible to describe *bi-directional associations* in DTDs. Using IDREF and IDREFS attribute types, only *one-way references* can be expressed. In contrast, metamodels use bi-directional associations as a central modeling technique. In particular, in Ecore two uni-directional references are connectable through the *eOpposite* attribute of class *EReference* to represent bi-directional associations. DTDs lack this information, however, which requires the user to manually connect two uni-directional references resulting from IDREF(S) attributes and mark them as bi-directional associations.

4 MetaModelGenerator (MMG)

The aforementioned transformation rules and heuristics have been implemented in a Java program called *MetaModelGenerator* (MMG). The MMG is based on the Eclipse Modeling Framework (EMF) and on an open source DTD parser[6].
In Figure 7, the DTD-to-MOF framework and implementation details of the *MMG* are graphically depicted and more precisely described in the following:

1. In a first step a specific DTD serves as input to the DTD parser, which parses the DTD and builds a Java object graph of DTD element types in memory.
2. Then each element in the object graph is visited and transformed regarding to the proposed transformation rules and heuristics in Section 3.
    - Each transformation rule is implemented as a separate Java method which takes DTD element objects as input and generates the object for the corresponding metamodel elements.
    - If a transformation rule uses a heuristic, then the corresponding method calls a helper method which implements the heuristic.

[6] http://www.wutka.com/dtdparser.html

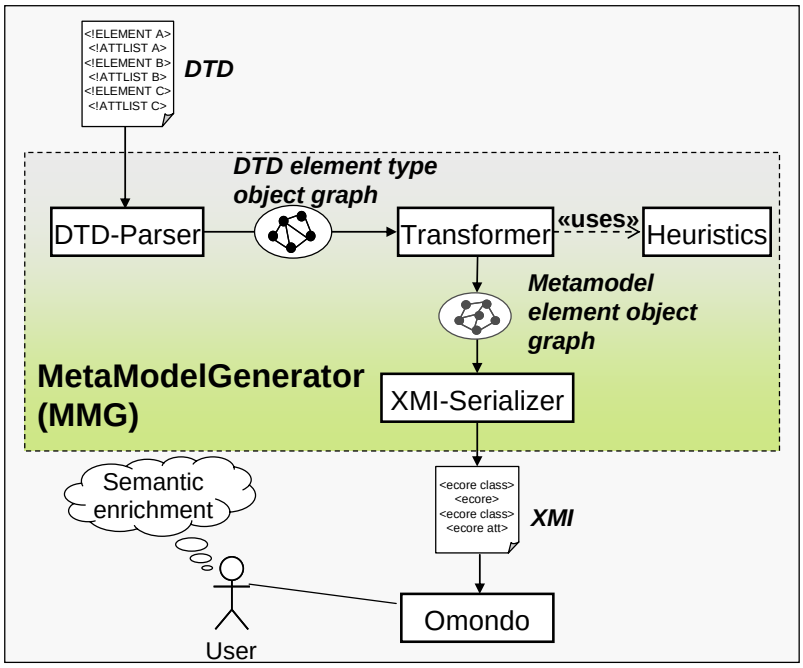

Figure 7: Architecture and Mode of Operation of the *MMG*

3. As soon as the complete element object graph of the metamodel has been generated, the default *XMI Serializer* of EMF is activated in order to serialize the metamodel as an XMI file.
4. This XMI file can be loaded into OMONDO[7] - a graphical editor for Ecore-based metamodels, available as an Eclipse plug-in.
5. In a last step, the metamodel should be refactored by the user according to the semantic enrichment rules explained in Section 3.

5 The WebML Case Study

In the following, we present the results of applying our approach to WebML's DTD-based language definition. As a prerequisite, we introduce necessary concepts of the web modeling language first.

Web applications typically consist of three layers known as the *content* layer, the *hypertext* layer, and the *presentation* layer. WebML provides two kinds of models: While the *data model* corresponds to the content layer, the *hypertext model* corresponds to the hypertext layer, but also provides modeling means for presentation layer concepts. The WebML language concepts are described in terms of notational elements and accompanying textual definitions [8]. In addition, WebML comes with

[7] http://www.omondo.de/

a modeling tool WebRatio, which incorporates the WebML language definition in terms of a DTD. Thus, WebML models created are internally represented in XML and XSL is used for code generation.

For demonstrating the appropriateness of our approach, we only present results on the WebML data model DTD part (cf. Listing 1) for two reasons. First, the *MMG* produced a hypertext metamodel draft consisting of more than 50 meta-classes, and therefore, would an unnecessary large example. And second, the WebML data model is based on the Entity-Relationship (ER) Model [10], which is very close to UML class diagrams and hence also easy to understand for non-hypertext modeling experts.

5.1 WebML Data Model

Modeling the content layer of a web application enables the specification of the data used by the application. Since WebML's data model is based on the ER model, it supports ER modeling concepts: An *entity type* represents a description of common features, i.e., *attributes*, of a set of objects. Note, that unlike UML class diagrams, ER diagrams model structural features, only. Attributes can have a data type, e.g., String, Integer, Float, Date, Time, Boolean, and Enumeration. One or a combination of an entity type's attributes form the *primary key*[8], which uniquely identifies *entity instances*. Entity types that are associated with each other are connected by *relationships*.

5.2 The MMG Output

Listing 1 presents the data model DTD[9] which consists of six concepts, namely WebMLTypes, DOMAIN, DOMAINVALUE, ENTITY, ATTRIBUTE, and RELATIONSHIP.

[8] WebML does not require the user to specify primary keys, instead a unique identifier (OID) is added automatically for each entity by WebRatio.

[9] WebML's DTD has been published as part of the WebRatio's User Guide. Please note that few tool related concepts, which are not relevant at modeling level, have been omitted.

```
<!--Structure Document Type Definition-->
<!ENTITY % WebMLTypes
"(String|Text|Password|Number|Integer|Float|Date|Time|TimeStamp|Boolean|
URL|BLOB|OID)">
<!ELEMENT Structure (DOMAIN*, ENTITY*)>
<!ATTLIST Structure
       id     ID     #REQUIRED>
<!ELEMENT DOMAIN (DOMAINVALUE*)>
<!ATTLIST DOMAIN
       id     ID     #REQUIRED
       name   CDATA  #IMPLIED
<!ELEMENT DOMAINVALUE EMPTY>
<!ATTLIST DOMAINVALUE
       value  CDATA  #REQUIRED>
<!ELEMENT ENTITY (ATTRIBUTE*, RELATIONSHIP*)>
<!ATTLIST ENTITY
       id             ID     #REQUIRED
       name           CDATA  #IMPLIED
       superEntity    IDREF  #IMPLIED
       value          CDATA  #IMPLIED
       duration(      persistent|volatile) 'persistent'>
<!ELEMENT ATTRIBUTE EMPTY>
<!ATTLIST ATTRIBUTE
       id             ID             #REQUIRED
       name           CDATA          #IMPLIED
       type           %WebMLTypes;   #IMPLIED
       userType       IDREF          #IMPLIED
       value          CDATA          #IMPLIED>
<!ELEMENT RELATIONSHIP EMPTY>
<!ATTLIST RELATIONSHIP
       id             ID     #REQUIRED
       name           CDATA  #IMPLIED
       roleName       CDATA  #IMPLIED
       to             IDREF  #REQUIRED
       inverse        IDREF  #REQUIRED
       minCard        CDATA  #REQUIRED
       maxCard        CDATA  #REQUIRED
       value          CDATA  #IMPLIED>
```

Listing 1: WebML (data model) language definition - Structure.dtd

The DTD has served as input to the *MMG*, which has been employed to produce a preliminary version of the WebML metamodel shown in Figure 8 and Figure 9, respectively. According to our transformation rules

1. one meta-class has been generated for each DTD element declaration,
2. the elements' attributes have been added to their corresponding elements,
3. annotations have been made in order to draw the user's attention to attributes that have not automatically been resolved,
4. associations between meta-classes have been added,
5. the multiplicity has been set for attributes and associations,
6. and an enumeration has been produced for the entity *WebMLTypes*.

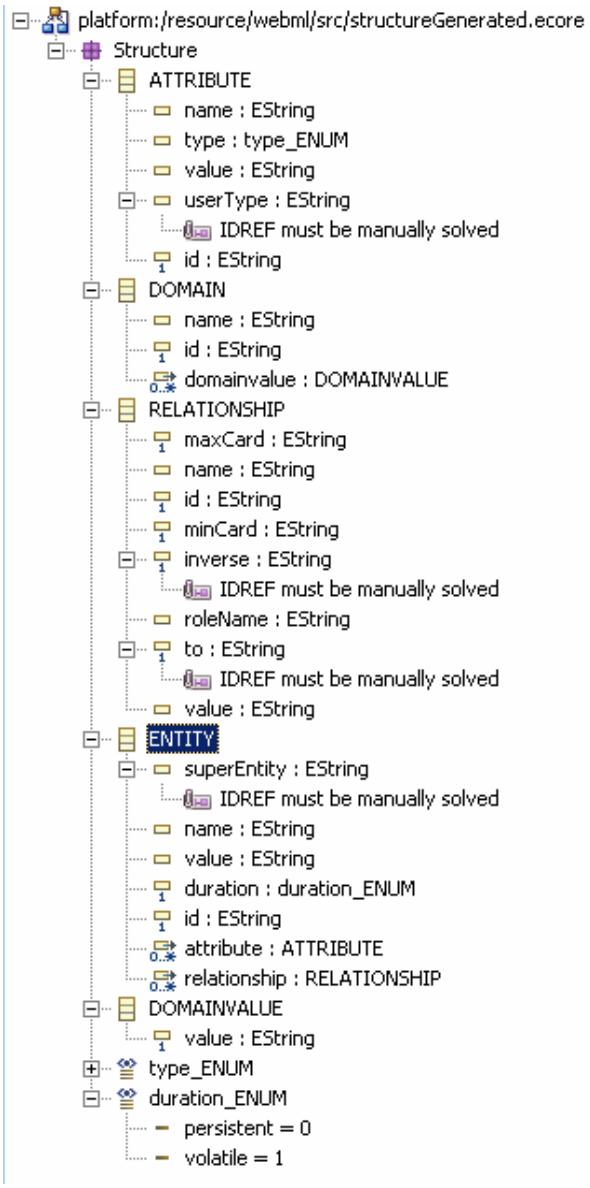

Figure 8: Automatically generated WebML Metamodel (EMF Tree-View)

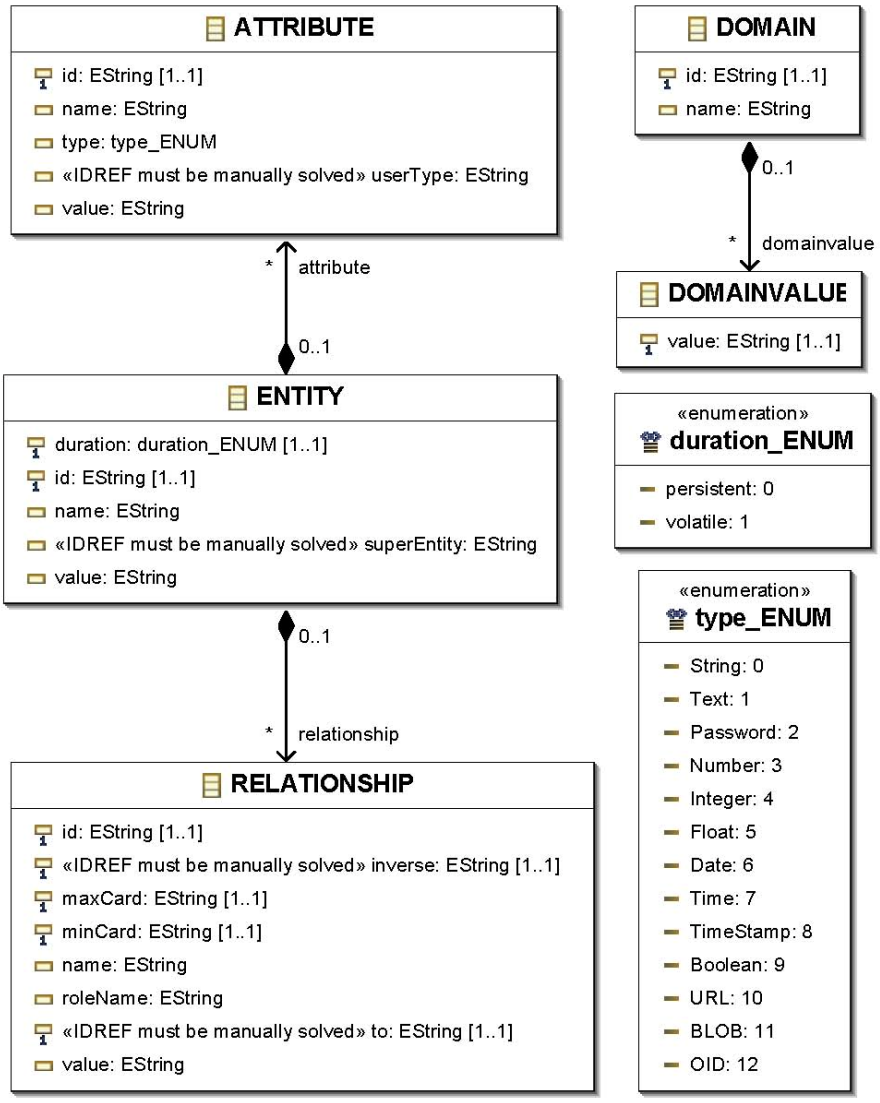

Figure 9: Automatically generated WebML Metamodel (Omondo View)

5.3 The WebML Metamodel – Necessary Manual Adaptations

Through considering additional information given in [8], we performed several ameliorations to the metamodel depicted in Figure 9.

1. The list of WebML types described in [8] includes the type *Enumeration*. In both, the DTD and the generated *type_ENUM* meta-class, however,

such a concept could not be found. In fact, the *Enumeration* concept has been realized by the meta-classes *DOMAIN* and *DOMAINVALUE*, and by *DOMAIN* being referenced by the *userType* attribute of meta-class *ATTRIBUTE*.

2. The class *RELATIONSHIP* owns two attributes with annotation «*IDREF must be resolved manually*». The first attribute called *inverse* represents the opposite relationship which allows specifying bi-directional relationships. Therefore, the attribute is refactored as a *recursive* reference from and to *RELATIONSHIP* with the reference end named *inverse*. The second attribute called *to* represents the referenced class. Therefore, the attribute is refactored as a reference pointing to class *ENTITY*, because an instance of relationship points exactly to one instance of entity.
3. The class *RELATIONSHIP* owns two further attributes which describe the cardinalities (*minCard, maxCard*) of a relationship end. Cardinalities are normally of type (non-negative) *Integer*, hence we have changed the data type from S*tring* to *Integer*.
4. The class *ENTITY* possesses an attribute called *superEntity* with annotation «*IDREF must be resolved manually*». This attribute references the only parent entity, thus this concept represents *single inheritance*. We changed the metamodel in this respect and modeled a recursive reference from and to class *ENTITY*.
5. The most serious user intervention is necessary for describing the possible data types for the attributes (cf. class *ATTRIBUTE*). Two possible kinds of data types are available in the automatically generated metamodel: (1) primitive type (cf. attribute *type* of class *ATTRIBUTE*) or (2) user defined enumeration (cf. attribute *usertype* of class *ATTRIBUTE*). To more precisely describe this fact, we have introduced a new class called *TYPE* which serves as the *super class* of *DOMAIN* and of the also newly introduced class *BUILT-IN_TYPE*. The class *ATTRIBUTE* receives a reference to the class *TYPE* instead of expressing the two possible kinds of data types by the attributes *type* and *usertype*.

We apply the above refactoring tasks to the automatically generated metamodel and present the resulting final WebML metamodel in Figure 10.

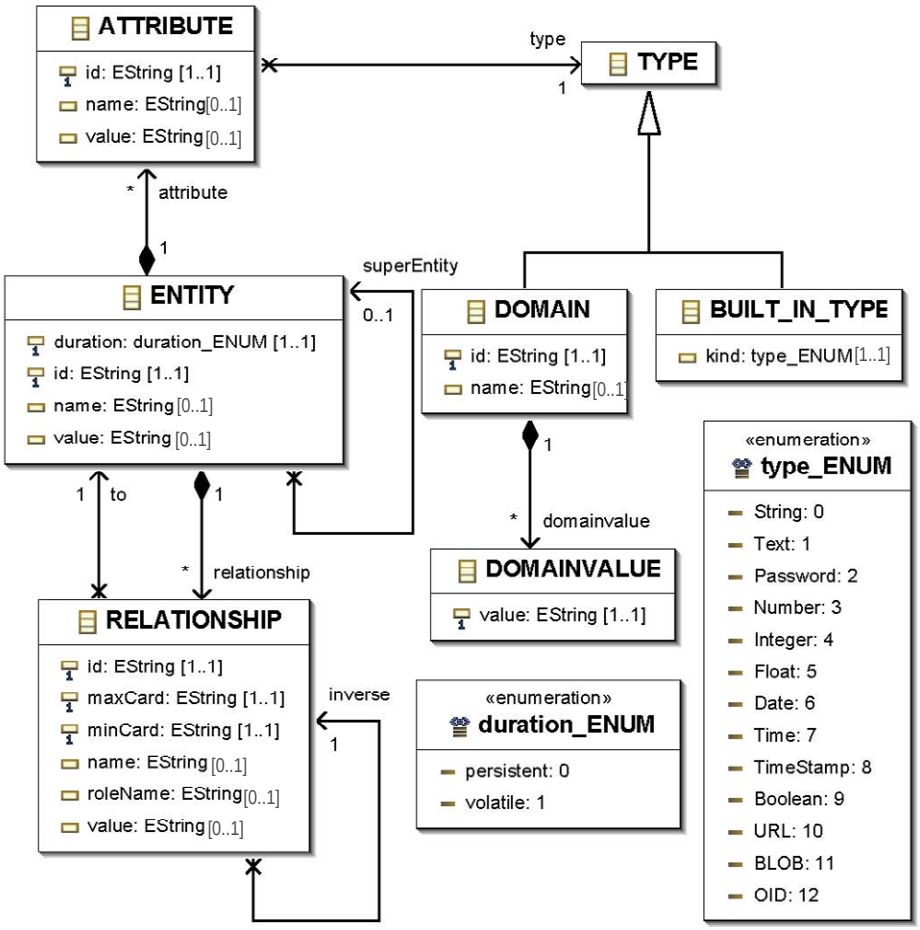

Figure 10: Final WebML Metamodel (Omondo View)

6 Related Work

There have already been several approaches to transformation from the *model technical space* to the *XML technical space* and vice versa (cf. Figure 11). In [32], an elaborate overview of existing approaches is given. In particular, a categorization into *forward engineering* approaches (i.e., transformation from the XML technical space to the model technical space) and *reverse engineering* approaches (i.e., transformation from the model technical space to the XML technical space) has been made.

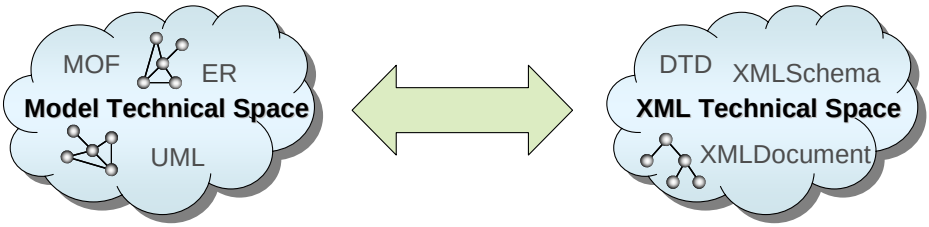

Figure 11 Bridging Technical Spaces

In Table 2, we further investigate related approaches with respect to what standards/technologies are used in both, the model technical space and the XML technical space, i.e., which standards/technologies are mapped onto each other, and with respect to tool support, i.e., if the specified mappings have been implemented in a transformation tool.

Table 2: An Overview of existing Transformation Approaches

	Transformation Direction	**Model Technical Space**	**XML Technical Space**	**Tool Support**
Bird et al. [5]	Forward	ORM	XML Schema	No
Bernauer et al. [6]	Reverse	UML (Profile)	XML Schema	No
Booch et al. [7][10]	Reverse	UML	DTD	?
Bordbar et al.	Reverse	UML	XML Schema	Yes
Conrad et al. [9]	Forward	UML	DTD	No
Hucka [15]	Forward	UML	XML Schema	No?
Mani et al. [16]	Reverse	ER	XML Schema	No
Mello et al. [17]	Reverse	ORM/NIAM and ER	DTD	yes
Provost [27]	Forward	UML	XML Schema	No
Rational [29]	Reverse	UML (Profile)	DTD	Yes
	Forward	UML (Profile)	XML Schema	Yes
Routledge et al. [30], [31]	Forward	UML (Profile)	XML Schema	no
Salim et al. [32]	Reverse	UML	XML Schema	no
Skogan [34]	Forward	UML	DTD	yes
Our Approach	Reverse	MOF (Ecore)	DTD	yes

Basically, approaches related to our work provide mappings between the XML technical space, relying on DTDs or XML Schema, and the model technical space, relying on UML(Profiles) but also on ORM and ER. In [29], a script for Rational Rose is proposed that allows migration from DTDs to XML Schema by mapping to a UML Profile as an intermediary step (i.e., representing both a forward engineering and a reverse engineering approach).

To the best of our knowledge, there is no approach to mapping between concepts of DTD and concepts of MOF. Our approach of transforming DTDs to MOF-based

[10] Please not, that we adopted this reference from [32]. Unfortunately, the file has been removed from the stated URL. Our evaluation in Table 2 is therefore based on information provided in [32].

metamodels belongs to the reverse engineering category, since the transformation is directed from the XML technical space to the model space.
According to [1], “the relation between a model and its metamodel is also related to the relation between a program and the programming language in which it is written, defined by its grammar, or between an XML document and the defining XML schema or DTD.” Hence, in OMG’s four-layer architecture DTDs belong to the same layer (M2) as metamodels and XML documents belong to the same layer (M1) as models.

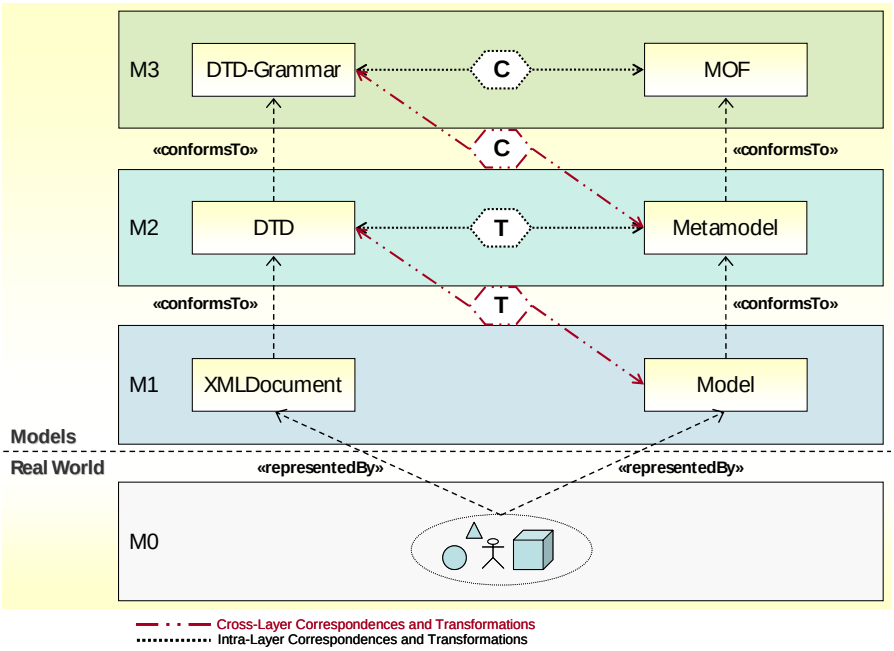

Figure 12 Intra-Layer and Cross-Layer Correspondences and Transformations

By providing mappings and transformations between concepts of DTD and concepts of MOF, our work differs from the existing approaches in that we provide *intra-level* correspondences and transformations with respect to OMG’s four-layer architecture, while existing approaches usually define *cross-layer* correspondences and transformations (cf. Figure 12). In contrast to other approaches, we are mapping DTDs to domain *languages* instead of mapping them to domain *models*.
With intra-layer mappings, one is able to derive intra-level mappings at lower levels of the architecture. Deriving mappings at M2 from mappings at M3 allows performing transformations at M1, i.e., transformations of XML documents in UML models. Cross-layer transformation approaches, however, are limited to transforming XML documents into *object models*, which have to conform to the UML model. Therefore, while in our approach we are still able to rely on *linguistic instantiations* between layers, cross-layer transformation approaches have to rely on *ontological instantiations* at M1 [1].

7 Conclusions and Future Work

In this work we have presented an approach to mapping and transforming concepts between the model technical space and the XML technical space. In particular, we have introduced a set of generic transformation rules, heuristics, and required user interactions for the semi-automatic generation of MOF-based metamodels from DTDs. Our transformation framework, the *MetaModelGenerator*, has been implemented on top of the Eclipse Modeling Framework and its applicability has been shown in the WebML case study. Future work concerns three disjoint extensions to our framework.

Refinement of the DTD-to-MOF transformation approach. The simplification of a transition from WebML's DTD-based language definition to a MOF-based language definition by automatically generating a first draft of a metamodel represents the major motivation for designing our transformation framework. Within the WebML case study, however, not all transformation rules were tested (e.g., the transformation of *XMLChoice* and *XMLSequence*). Thus, the appropriateness of our transformation framework needs further testing within additional case studies. Based on the results, we will refine the transformation rules in order to provide a generic transformation tool. We are also looking for further heuristics for resolving IDREF(S). In this respect, we are using *algorithms* for exploring similarities between strings as well as o*ntology-based* techniques like finding synonyms using WordNet[11]. In particular, with this approach it is possible to find *inheritance relationships*.

Model-driven transformation approach. We plan to use the DTD metamodel we proposed in Section 2 for describing the DTD-to-MOF transformation rules and heuristics as ATL transformations. We will define a weaving model between the concepts of DTD and MOF in the AMW [11] using a special bridging language [14]. The semantics of this bridging language is operationally specified in an adjacent code generator [14], which in our case produces ATL code for generating MOF-based metamodels. In this respect, we do not only generate metamodels from DTDs in order to *enable MDE*, but just in doing so, we *apply model-driven engineering techniques*. We are going to compare our current Java-based approach with this future model-driven approach in order to learn their commonalities and differences. In particular, we expect that the model-driven approach leads to improved readability of the transformation "program" which in turn is also more flexible for future extensions.

Deriving model transformations from metamodel mappings. We also plan to perform transformations at *M1* level, i.e., transform *XML documents*, which conform to a DTD, into *models*, which again conform to a corresponding metamodel. Therefore, the *MetaModelGenerator* should be capable of producing a *Model Generator* (MG) for a given DTD.

[11] http://wordnet.princeton.edu/

Acknowledgements

We thank Maristella Matera for supporting us with her very valuable insight into the WebML language, and www.webratio.org for allowing us to publish parts of WebML's DTD language definition.

References

[1] C. Atkinson, T. Kühne: Model-Driven Development: A Metamodeling Foundation. *IEEE Software* 20 (5), 36-41, 2003.

[2] L. Baresi, S. Colazzo, L. Mainetti, and S. Morasca. W2000: A Modeling Notation for Complex Web Applications. In E. Mendes and N. Mosley (eds.) Web Engineering: Theory and Practice of Metrics and Measurement for Web Development, Springer-Verlag, ISBN: 3-540-28196-7, 2006.

[3] B. Bordbar, A. Staikopoulos. Automated Generation of Metamodels for Web service Languages. Second European Workshop on Model Driven Architecture (MDA), Canterbury, UK, September 2004

[4] J. Bézivin. On the Unification Power of Models. *Journal on Software and Systems Modeling*, 4(2):171-188, May 2005.

[5] L. Bird, A. Goodchild, and T. Halpin. Object Role Modeling and XML-Schema. In Proceedings of the *19th International Conference on Conceptual Modeling (ER'2000)*, Salt Lake City, USA, October 2000, Springer, pp. 309-322.

[6] M. Bernauer, G. Kappel, G. Kramler. *Representing XML Schema in UML - A UML Profile for XML Schema.* Technical Report, Available at: http://www.big.tuwien.ac.at/research/publications/2003/1303.pdf, November 2003.

[7] G. Booch, M. Christerson, M. Fuchs, and J. Koistinen. *UML for XML Schema Mapping Specification*. Rational Software and CommerceOne, 1999. Available at: http://www.rational.com/media/uml/resources/media/uml_xmlschema33.pdf

[8] S. Ceri, P. Fraternali, A. Bongio, M. Brambilla, S. Comai, M. Matera. *Designing Data-Intensive Web Applications*. Morgan-Kaufmann, 2003.

[9] R. Conrad, D. Scheffner, and J.C. Freytag. XML Conceptual Modeling using UML. In Proceedings of the *19th International Conference on Conceptual Modeling (ER'2000)*, Salt Lake City, USA, October 2000, Springer, pp. 558-571.

[10] P. P. Chen. The Entity-Relationship Model – Toward a Unified View of Data. *ACM TODS*, 1 (1), March 1976.

[11] Didonet Del Fabro M., Bézivin J., Jouault F., Breton E., Gueltas G.: AMW: a generic model weaver. Proc. of the 1ères Journées sur l'Ingénierie Dirigée par les Modèles, (2005)

[12] I. Garrigós, S. Casteleyn, J. Gómez. A Structured Approach to Personalize Websites using the OO-H Personalization Framework in Web Technologies Research and Development. In *Proc. of the 7th Asia-Pacific Web Conference (APWeb 2005)*, Springer-Verlag, ISBN 3-540-25207-X, Shangai, China, March-April 2005.

[13] G. Kappel, E. Kapsammer, W. Retschitzegger. Integrating XML and Relational Database Systems. *World Wide Web Journal (WWWJ)*, Kluwer Academic Publishers, Vol. 7(4), December 2004, pp. 343-384.

[14] G. Kappel et al.: Lifting Metamodels to Ontologies: A Step to the Semantic Integration of Modeling Languages, Submitted for publication, 2006

[15] M. Hucka. SCHUCS: An UML-Based Approach for Describing Data Representations Intended for XML Encoding. In Proceedings of the *2nd International Conference on The Unified Modeling Language (UML'99)*, October 1999.
[16] M. Mani, D. Lee, and R.R. Muntz. Semantic Data Modeling using XML Schemas. In Proceedings. *20th International Conference on Conceptual Modeling (ER 2001)*, Yokohama, Japan, November 2001, Springer, pp. 149-163.
[17] R.d.S. Mello and C.A. Heuser. "A Rule-Based Conversion of a DTD to a Conceptual Schema". In Proceedings. 20th International Conference on Conceptual Modeling (ER 2001), Yokohama, Japan, November 2001, Springer, pp. 133-148.
[18] OASIS. *ebXML Technical Architecture Specification v1.0.4.* http://www.ebxml.org/, February, 2001
[19] Object Management Group (OMG). *Common Warehouse Metamodel (CWM) Specification, v1.1*. http://www.omg.org/docs/formal/03-03-02.pdf, March 2003.
[20] Object Management Group (OMG). *MDA Guide Version 1.0.1*. http://www.omg.org/docs/omg/03-06-01.pdf, June 2003.
[21] Object Management Group (OMG). *Meta Object Facility (MOF) 2.0 Core Specification Version 2.0*. http://www.omg.org/docs/ptc/04-10-15.pdf, October 2004.
[22] Object Management Group (OMG), *MOF 2.0/XMI Mapping Specification, v2.1*. http://www.omg.org/docs/formal/05-09-01.pdf, September 2005.
[23] Object Management Group (OMG). *UML Specification: Infrastructure Version 2.0.* http://www.omg.org/docs/ptc/04-10-14.pdf, October 2004.
[24] Object Management Group (OMG). *UML Specification: Superstructure Version 2.0*. http://www.omg.org/docs/formal/05-07-04.pdf, August 2005.
[25] Open Mobile Alliance (OMA). *Wireless Markup Language Version 2.0*. http://www.openmobilealliance.org/tech/affiliates/wap/wap-238-wml-20010911-a.pdf, September 2001.
[26] M. Peltier, R. Ziserman, J. Bézivin. On levels of model transformation. In: *In XML Europe 2000. -, Paris, France, pages 1-17*, 2000.
[27] W. Provost. *UML For W3C XML Schema Design*. http://www.xml.com/lpt/a/2002/08/07/wxs_uml.html, August 2002.
[28] QVT-Merge Group. *Revised submission for MOF 2.1 Query/View/Transformation*. http://www.omg.org/docs/ad/05-07-01.pdf, 2005.
[29] Rational Software Corporation. *Migrating from XML DTD to XML Schema using UML*. Rational Software White Paper, 2000. Available at: http://www.rational.com/media/whitepapers/TP189draft.pdf
[30] N. Routledge, L. Bird, and A. Goodchild. UML and XML Schema. In Proceedings of the *13th Australasian Database Conference (ADC 2002)*, Melbourne, Australia, ACS.
[31] N. Routledge, A. Goodchild, and L. Bird. *XML Schema Profile Definition*. Honours thesis extract, DSTC, Queensland, Australia, 2002. Available at: http://titanium.dstc.edu.au/papers/xml-schema-profile.pdf
[32] F.D. Salim, R. Price, S. Krishnaswamy, M. Indrawan. UML documentation support for XML schema. In proceedings of the *Australian Software Engineering Conference (ASWEC'04)*, 2004.
[33] D. Schwabe, R. Guimarães, G. Rossi. Cohesive Design of Personalized Web Applications. IEEE Internet Computing 6 (2), March-April, 2002.
[34] D. Skogan. UML as a Schema Language for XML based Data Interchange. In Proceedings of the *2nd International Conference on The Unified Modeling Language (UML'99)*, October 1999.
[35] Sun Corporation. *Java™ Architecture for XML Binding (JAXB)*. Available at: http://java.sun.com/xml/jaxb/index.html

[36] N. Wirth. What can we do about the unnecessary diversity of notation for syntactic definitions? *CACM*, 20 (11), November 1977, pp. 822-823.
[37] World Wide Web Consortium (W3C). *Extensible Markup Language (XML) 1.1 Specification*. http://www.w3.org/TR/xml11/, April 2004.
[38] World Wide Web Consortium (W3C). *HTML 4.01 Specification*. http://www.w3.org/TR/html4/, December 1999.
[39] World Wide Web Consortium (W3C). *Mathematical Markup Language (MathML) Version 2.0*. http://www.w3.org/TR/MathML/, October 2003.
[40] World Wide Web Consortium (W3C). *Scalable Vector Graphics (SVG) 1.1 Specification*. http://www.w3.org/TR/SVG/, January 2003.
[41] World Wide Web Consortium (W3C). *SOAP Version 1.2 Part 1: Messaging Framework*. http://www.w3.org/TR/soap12-part1, June 2003.
[42] World Wide Web Consortium (W3C). *XML Schema Part 0: Primer Second Edition.* http://www.w3.org/TR/XML Schema-0/, October 2004.
[43] World Wide Web Consortium (W3C). *XSL Transformations (XSLT) Version 1.0.* http://www.w3.org/TR/xslt, November 1999.
[44] World Wide Web Consortium (W3C). *RDF Vocabulary Description Language 1.0: RDF Schema Version 1.0.* http://www.w3.org/TR/rdf-schema/, February 2004.