ABSTRACT
Metamodels are a prerequisite for model-driven engineering (MDE) in general and consequently for model-driven web engineering in particular. Various modeling languages, just as in the web engineering field, however, are not based on metamodels and standards but instead define proprietary languages rather focused on notational aspects. Thus, MDE techniques and tools can not be deployed for such languages. The WebML web modeling language is one example that does not yet rely on an explicit metamodel. Instead, it is implicitly defined within the accompanying tool in terms of a document type definition (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 metamodel for WebML which is based on the Meta Object Facility (MOF). To establish such a metamodel a semi-automatic approach is provided that allows to generate MOF-based metamodels from DTDs. The metamodel for WebML accomplishes the following aims: First, it represents an initial step towards a transition to employing MDE techniques (e.g., model transformations or language extensions through profiles) within the WebML design methodology. Second, it represents an important step towards a common metamodel for Web modeling. Third, the provision of a MOF-based metamodel ensures interoperability with other MDE tools.

Categories and Subject Descriptors
D.2.10 [Software Engineering]: Design – methodologies, representation.

General Terms
Design, Standardization, Languages

Keywords
Web Modeling Language, Metamodel, DTD, Model Driven Web Engineering

1. INTRODUCTION
In the web engineering research field various modeling approaches have been proposed in the past 10 years, such as WebML [7], UWE [15], W2000 [2], OOHDM [29], OO-H [10], WSDM [9], and OOWS [27], aiming at counteracting a technology-driven and ad hoc development of web applications.

At the same time, model-driven engineering (MDE) [5] has received considerable attention and is well on its way to becoming a promising paradigm in software engineering. In MDE, models replace code as the primary artifacts in the software development process. MDE 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 allow generating several platform-specific implementations.

While some of the above mentioned web modeling approaches already provide tools and techniques for modeling web applications in a platform-independent way, their code generation facilities, if existent, mostly support one specific platform, only, yielding transformations from a platform-independent model directly to code. For these reasons, although first proposals for a transition to the model-driven paradigm in web engineering have already been made, e.g., [18], [17], [32], [30], [19], existing web modeling approaches represent model-driven approaches in the sense of MDE to a limited extent, only.

Thus, the demand arises to bridge existing Web modeling methodologies with MDE. In this respect, metamodels represent an important prerequisite. In contrast to the MDE paradigm, however, most web modeling languages originally have been designed without using meta-modeling techniques, rather focused on notational aspects of the language. With no explicit metamodels available, however, one can not profit from MDE’s advantages such as model transformations and a common format for model exchange (e.g., XMI [22]). The WebML [7] web modeling language is one example that does not yet rely on an explicit metamodel. Instead, it is implicitly defined within the
accompanying tool WebRatio in terms of a DTD [35], i.e., a grammar-like textual definition for specifying a structure for XML documents. In contrast to MOF’s [21] expressivity, however, DTDs represent a rather restricted mechanism for describing 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. WebRatio first, internally represents models in XML [35], and second, uses XSLT [37] for code generation. Since XSLT, however, is not intended for heavy structural transformations, writing XSLT programs for code generation is difficult and error-prone. Concerning these problems, a metamodel-based approach allows expressing transformation rules in a more compact and readable way by using existing model transformation languages such as QVT [28] and ATL [14].

To make WebML MDE-capable, we propose a MOF-based metamodel for WebML. To establish such a metamodel, a semi-automatic approach [33] to generating MOF-based metamodels from DTD-based language definitions has been developed. The contributions of a metamodel for WebML are as follows: (1) Such a metamodel represents an important prerequisite and thus, an initial step towards a transition to employ model-driven engineering techniques (e.g., model transformations or language extensions through profiles) within the WebML design methodology. (2) Additionally, it is also an important step towards a common reference metamodel for Web modeling languages [15]. (3) The provision of a MOF-based metamodel ensures interoperability with other MDE tools. Moreover, our transformation approach enables the visualization of any DTD-based language in terms of MOF-based metamodels and thus, enhances the understandability of those languages.

The remainder of this paper is organized as follows. Section 2 presents the architecture of our metamodel generation framework, including on the one hand a set of transformation rules, heuristics, and recommended manual refactorings, and on the other hand an implementation within the MetaModelGenerator (MMG), which is based on the Eclipse Modeling Framework (EMF). In Section 3, we discuss the semi-automatically generated WebML metamodel. Section 4 gives an overview of related work. Finally, we outline conclusions and future work in Section 5.

2. FROM DTDs TO METAMODELS

Formal languages require precise definitions in terms of a meta-language in order to be understandable by computers. In the past, various meta-languages have been employed for defining formal languages. Amongst them are EBNF [34] for describing the syntax of (programming) languages, DTD and XML Schema [36] for defining the structure of XML documents in terms of elements and attributes, and MOF, which represents the state-of-the-art for defining modeling languages. In Figure 1, we illustrate these relationships and our transformation framework [33] within the realms of the Object Management Group’s (OMG) four-layer architecture [24].

According to [5], 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, 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 1 indicates the fact that languages themselves may 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 (C) between language elements of the DTD-grammar and MOF can be used for transforming a particular DTD into a MOF-based metamodel. These generic correspondences are implemented as transformation rules and heuristics in the MetaModelGenerator (MMG), which takes a DTD as input and produces a corresponding MOF-based metamodel.
Figure 2 illustrates the DTD-to-MOF framework and implementation details of the MMG, which is based on the Eclipse Modeling Framework (EMF)\(^2\) and on an open source DTD parser\(^3\). 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. Then each element type in the object graph is visited and transformed according to the transformation rules and heuristics described in Section 2.3.1 and Section 2.3.2, respectively. Each transformation rule is implemented as a separate Java method which takes DTD element type objects as input and generates the objects for the corresponding metamodel elements. If a transformation rule uses a heuristic, then the corresponding method calls a helper method which implements the heuristic. 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. This XMI file can be loaded into OMONDO\(^4\) - a graphical editor for Ecore-based metamodels, available as an Eclipse plug-in. In a last step, the metamodel should be refactored by a user according to the semantic enrichment rules explained in Section 2.3.3.

### 2.2 Concepts of DTDs and Metamodels

In the following, we will provide a brief introduction to the main concepts of DTD and MOF. Afterwards, we give an explanation of their correspondences and propose resulting transformation rules and heuristics. Since by the time of writing there is no standardized implementation of MOF 2.0 available, we are using Ecore, a slightly modified EMOF\(^5\) implementation in Java, which is provided by the EMF. Ecore's concepts essentially correspond to EMOF, which is sufficient in the context of this paper. The concepts of DTD and Ecore are given in terms of UML class diagrams (cf. Figure 3 and Figure 4). With respect to Figure 1, these two diagrams belong to M3 and represent the operands on which to define correspondences.

The UML class diagram given in Figure 3 presents the most important DTD concepts and has been designed based on previous work\(^1\) and the DTD-grammar described in EBNF.

**Figure 3: Overview of DTD language concepts**

1. **XMLDTD**: The root element of a DTD.
2. **XMLElement**: An element type declaration.
3. **XMLAttribute**: An attribute declaration.
4. **XMLString**: A character data type.
5. **XMLAtomic**: A data type.
6. **XMLChoice**: A choice of element types.
7. **XMLSequence**: A sequence of element types.
8. **XMLAny**: The universal content model.
9. **XMLEmpty**: An empty element type.
10. **XMLAnonymous**: An anonymous element type.

Element type declarations are first-class citizens in DTDs. Element types (XMLElementType) 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), XMLCompositeETMixedContent (a mix of character data and child element types), and XMLCompositeETElementContent (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 a single 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 4, summarizes the most important concepts of Ecore.

**Figure 4: Overview of Ecore language concepts**

EClasses are the first-class citizens in Ecore-based metamodels. An EClass may have multiple EReferences and EAttributes for defining its properties as well as multiple super classes.

An 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, an EReference refers to an EClass and optionally to an

---

\(^1\) Based on http://download.eclipse.org/tools/emf/2.2.0/javadoc/org/eclipse/emf/ecore/package-summary.html#details

\(^2\) http://www.eclipse.org/emf

\(^3\) http://www.wutka.com/dtdparser.html

\(^4\) http://www.omondo.de/

\(^5\) MOF consists of two parts, namely essential MOF (EMOF) and complete MOF (CMOF).
opposite EReference for expressing bi-directional relationships. Besides, a reference can be declared as a containment reference.

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 serve for defining the types of attributes. String, Boolean, Integer, and Float are part of Ecore’s default data types set.

EEnum allows to model 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.

### 2.3 DTD – Metamodel Correspondences

In the following we give a brief overview of our transformation framework consisting of a set of transformation rules (cf. Section 2.3.1), heuristics (cf. Section 2.3.2), and manual refactorings (cf. Section 2.3.3) and refer the interested reader to [33] for a more elaborate discussion.

#### 2.3.1 Transformation Rules

We designed transformation rules and sub rules, first, for transforming element types of DTDs and second, for transforming their attributes. Some of them are supported by heuristics (cf. Section 2.3.2) which lead to improved readability and higher quality of the metamodel but require some user validation (cf. Section 2.3.3). Table 1 summarizes the proposed transformation rules.

**Rule 1 – DTD::XMLElemType _2_ Ecore::EClass.** For each XMLElemType an EClass is created and the name of the EClass is set to the element type name. Depending on the particular subclass of XMLElemType additional metamodel elements have to be created in the transformation process (cf. Table 1).

**Rule 1.1 – DTD::XMLContentParticle.cardinality _2_ Ecore::EReference.multiplicity.** Each XMLContentParticle may have a certain cardinality, which is represented in metamodels through multiplicity (lower/upper bound) of the reference end.

**Rule 2 – DTD::XMLAttribute _2_ Ecore::EAttribute.** For each XMLAttribute an EAttribute is created, which is attached to the EClass representing the XMLElemType, which in turn owns the XMLAttribute. The name of the EAttribute is set to the name of the XMLAttribute. The data type of XMLAttribute is one of the following: {CDATA, ID, IDREF, IDREFS, Enumeration} with each requiring an appropriate transformation (cf. Table 1).

**Rule 2.1 – DTD::XMLAttribute.cardinality _2_ Ecore::EAttribute.multiplicity.** Attributes in both, DTDs and 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, and IDREF) and multi-valued (e.g., IDREFS) and on the other hand by the XMLAttribute declaration (#REQUIRED, #IMPLIED, #FIXED, and default value). Table 1 illustrates how XMLAttribute cardinalities are transformed into EAttribute multiplicities.

<table>
<thead>
<tr>
<th>Table 1: Transformation rules between DTD and Ecore</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Rule</strong></td>
</tr>
<tr>
<td>R1</td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td>(1)</td>
</tr>
<tr>
<td>(2)</td>
</tr>
<tr>
<td>(3)</td>
</tr>
<tr>
<td>(4)</td>
</tr>
<tr>
<td>(5)</td>
</tr>
<tr>
<td>R2</td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td>(1)</td>
</tr>
<tr>
<td>(2)</td>
</tr>
<tr>
<td>(3)</td>
</tr>
<tr>
<td>(4)</td>
</tr>
<tr>
<td>(5)</td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td>R2.1</td>
</tr>
<tr>
<td>(1)</td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td>(2)</td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td>(3)</td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td>(4)</td>
</tr>
<tr>
<td></td>
</tr>
</tbody>
</table>

#### 2.3.2 Heuristics

The effectiveness of the proposed heuristics is strongly correlated with the quality of the DTDs’ design. For example, the heuristics operate more effectively if naming conventions, e.g., for IDREFs, are used or the content of the DTD is split up into several external DTDs, which group related element types. The proposed heuristics are deployed to exploit the following semantically rich language constructs of Ecore, namely (1) typed references, (2) data types, and (3) packages as a grouping mechanism. The heuristics of our framework are described in the following and summarized in Table 2.

**Heuristic 1 - IDREF(S) Resolution.** A DTD does not restrict which element types can be referenced from an attribute of 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 neither possible to determine if certain element types may be referenced,
only, nor which element type(s) may be referenced based on the information given in the DTD. Sometimes, however, it is possible to find the referenced element types relying on naming conventions of element types and attributes. Note, that the user still must validate the generated references in order to detect random name-matches, which means that a referenced class does not correspond to the intended referenced element.

**Heuristic 2 - Boolean Identification.** DTDs do not allow to specify XML attributes of type Boolean explicitly. Instead, an element’s attribute can be of type Enumeration with two literals, e.g., true and false. For this special case, however, an attribute of type Boolean is semantically richer and more compact. *Heuristic 2* recognizes such optimization possibilities and generates an attribute of type Boolean.

**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. 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 from within a so called root DTD. A root DTD is equivalent to a root package in a metamodel and external DTDs are equivalent to subpackages of the root package.

<table>
<thead>
<tr>
<th>Table 2: Heuristics</th>
<th>Heuristic</th>
<th>DTD Concept</th>
<th>Ecore Concept</th>
</tr>
</thead>
</table>
| **H1**              | if (XMLTokenAtt.kind == IDREF) && (XMLAtt.name == EAttribute.name) | 1) EReference from EClass 2 annotate with «IDREF(S)>> | EAttribute with type IDREF to EC2 |}
| **H2**              | if XMLEnumAtt is one of \{true, false\}, \{1, 0\}, \{on, off\}, \{yes, no\} | EAttribute.type is Boolean | EAttribute.type is Boolean |
| **H3**              | if DTD imports external DTDs | EPackages of the external DTDs are nested within the root DTD EPackage | EPackages of the external DTDs are nested within the root DTD EPackage |

### 2.3.3 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. Such user interactions are strongly recommended because DTDs are poorer in semantics than MOF-based metamodels, which is due to a limited set of concepts. The most important semantic enrichment tasks require domain knowledge and concern the following problems of DTDs:

1. DTDs provide no explicit concepts to express *inheritance*. Thus, the user has to manually refactor the generated metamodels in order to achieve inheritance relationships, e.g., by introducing new (abstract) classes and reduce redundant definitions of attributes and references, leading to an improved structure and higher readability.

2. DTDs have a limited set of datatypes that can not be extended (e.g., to support *Integer* or *Boolean* data types). Thus, the user has to check all attributes of the generated metamodel, if any of them should be of type *Integer* or another special type.

3. Some IDREF(S) may be automatically resolved according to *Heuristic 1*. Due to the possibility of random name matches, however, the user has to validate if the resolution of the IDREF(S) is correct or if another class should be referenced. Furthermore, 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 classes.

4. It is not possible to describe *bidirectional associations* in DTDs using the inherent mechanisms (i.e., IDREF(S)). In contrast, metamodels use bi-directional associations as a central modeling concept. 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 which requires the user to manually connect two uni-directional references resulting from IDREF(S) attributes and mark them as bi-directional associations.

### 3. A METAMODEL FOR WebML

In the following we present an Ecore-based metamodel for WebML. We first give an overview on the package structure (cf. Section 3.1) and then describe some of the packages in more detail (cf. Section 3.2 - 3.5). Concluding this section, we point out problematic parts of the WebML DTD, with respect to an unambiguous language definition due to DTD’s weaker semantic expressiveness, and discuss the solutions to those problems within the WebML metamodel (cf. Section 3.6).

#### 3.1 Overview

The WebML language definition consists of several DTDs with WebML.dtd being the root DTD that imports the others. In the following we focus on the main language concepts that have been introduced in [7] and that are defined within Structure.dtd and Navigation.dtd. Other tool-related DTDs that specify the mapping to a relational database and the graphical illustration of WebML elements within the editor are not regarded in this paper.

Figure 5 presents a high-level view of the semi-automatically generated WebML metamodel, i.e., its packages and their interrelationships.

---

7 The complete metamodel is available at http://big.tuwien.ac.at/projects/webml/. For an in-depth description of each modeling concept we refer the reader to [7].
HypertextOrganization, and AccessControl. Concepts from the Content package are used for modeling the content level of a web application. The other packages contain modeling concepts for the hypertext level. Some concepts from the HypertextOrganization package, e.g., Page, can also be found at the presentation level. The integration of a Presentation package, however, is subject to future work, since WebML provides design support for the presentation level within the WebRatio tool, only, and these mechanisms have not been defined in [7] as being part of the language.

3.2 Content Package

The Content package (cf. Figure 6) contains modeling concepts that allow to model the content layer of a web application, which regards the specification of the data used by the application.

3.3 Hypertext Package

The Hypertext package (cf. Figure 8) summarizes ContentUnits, used, for example, to display information from the content layer in a certain way, which may be connected by Links. The hypertext layer represents a view on the content layer of a Web application, only, and thus, the Hypertext Package reuses concepts from the Content Package, namely, Entity, Relationship, and Attribute. In order to handle the large amount of different kinds of ContentUnits and to reduce redundant feature definitions we introduced a generalization hierarchy, which includes the additional abstract classes ContentUnit, DisplayUnit, and SortableUnit. The abstract class LinkableElement has been introduced in order to cope with language concepts of other packages, e.g., ContentManagement::ContentManagementUnit, that can also be connected by links (cf. Section 3.6.4).

3.4 ContentManagement Package

The ContentManagement package contains modeling concepts that allow the modification of data from the content layer. Similar to the generalization hierarchy in the Hypertext package, we also introduce additional abstract classes in the ContentManagement package (cf. Figure 7), i.e., OperationUnit, ContentManagementUnit, EntityManagementUnit, and RelationshipManagementUnit.
Since the specific ContentManagementUnits are able to create, modify, and delete Entities as well as establish or delete Relationships between Entities from the content layer, the ContentManagement package reuses concepts from the Content Package, namely Entity and Relationship.

### 3.5 HypertextOrganization Package

The Page, Area, and SiteView modeling concepts are used to organize and structure information, e.g., Hypertext::ContentUnits, as well as operations on data from the content level, e.g., ContentManagement::OperationUnits. They are grouped within the HypertextOrganization package (cf. Figure 9).

**Figure 9: HypertextOrganization Package**

The HypertextOrganization package builds on the Hypertext package and the ContentManagement package. The abstract classes introduced in the Hypertext and ContentManagement packages allow to more precisely define what kind of units can be part of a Page, an Area, and a SiteView (cf. Section 3.6.5).

### 3.6 WebML DTD vs. WebML Metamodel

As already mentioned, DTDs lack expressivity when compared to metamodels. While metamodels provide a mechanism to constrain the instance layer, e.g., with OCL [23], such constraints have to be implemented within the respective modeling tool in case of a DTD-based language. In the following, we provide concrete examples of such limitations, which we identified in the DTD-based language. In metamodels, however, such constraints can be ensured by xor-constraints expressed in OCL at the instance layer. In metamodels, however, such constraints can be ensured by xor-constraints expressed in OCL at the instance layer. In metamodels, however, such constraints can be ensured by xor-constraints expressed in OCL at the instance layer. In metamodels, however, such constraints can be ensured by xor-constraints expressed in OCL at the instance layer.

#### 3.6.1 Awkward Cardinalities

As already explained in Section 2.2, DTDs offer a restricted mechanism to specify cardinalities, i.e., there are no language concepts for defining cardinalities having a lower bound greater than one and for defining cardinalities having an upper bound other than ‘1’ or ‘*’. For example, the definition of the AlternativePage modeling concept requires the AlternativePage to have at least two sub-pages. This is expressed in the WebML DTD as follows:

```xml
<!ELEMENT AlternativePage (Page, Page+)>  
```

Yet, this definition might be misleading. One possible interpretation is that the first XMLContentParticle represents a special page, e.g., a default page. Another possible, i.e., the correct, interpretation, however, is that the first and the second XMLContentParticle together represent one set of Pages, i.e., one containment reference, but with special restrictions on their cardinalities, i.e., 2..*. In metamodels, this constraint can be expressed unambiguously, which is shown by the AlternativePage::page reference in Figure 9.

#### 3.6.2 Missing role concept

In DTDs, it is not possible to express that an element type can be deployed in different contexts, i.e., a role concept such as in UML is missing. As an example, the MultiChoiceIndexUnit may have two Selectors, with one being used in the role of a preselector. In the WebML DTD, this is expressed as follows:

```xml
<!ELEMENT MultiChoiceIndexUnit (Preselector?,Selector?,...)>  
<!ELEMENT Preselector (Selector|SelectorCondition)>  
<!ELEMENT Selector (SelectorCondition)>  
```

Since the Preselector element type declaration is identical to the Selector element type declaration, one can conclude that the Preselector element type represents the same concept as the Selector but used in a special context. In contrast, in metamodels this context information can be incorporated by reference names. Therefore, the WebML metamodel only contains the Selector class, which is referenced as a preselector by the MultiChoiceIndexUnit (cf. Figure 8). A similar example can be found in the ContentManagement package, where a Selector can act as sourceSelector or targetSelector for RelationshipManagementUnits (cf. Figure 7).

#### 3.6.3 Missing XOR constraints

DTDs do not provide a mechanism to express xor-constraints for attributes, which is frequently required for IDREF(S) attributes. The only way to define such constraints in DTDs is setting the cardinality of the attributes as #IMPLIED which means zero-or-one. However, this declaration does not ensure the intended constraint (i.e., the interrelationship between the attributes), because all attributes or none of the attributes could still occur at the same time at the instance layer. Consider the following example from the WebML DTD: An Area can have either a defaultArea or a defaultPage, but not both at the same time.

```xml
<!ELEMENT Area (...)>  
<ATTLIST Area  
    defaultPage IDREF #IMPLIED  
    defaultArea IDREF #IMPLIED  
>`

The attribute list declaration is not able to ensure this constraint at the instance layer. In metamodels, however, such a constraint can be ensured by xor-constraints expressed in OCL between the attributes as well as between the references resulting from IDREF(S) resolutions. Within the corresponding metamodel (cf. Figure 9) an xor-constraint between the references defaultPage and defaultArea has to be introduced to ensure that only one of the two references occurs at the instance layer.

#### 3.6.4 Unknown Referenced Element Types

As already mentioned, it is not possible to identify which element type(s) may be referenced from an IDREF-typed attribute based on the information given in the DTD. This peculiarity of DTDs is particularly problematic, if several element types can be referenced. These types potentially have a common supertype, which, however, cannot be specified in the DTD. For example, the IDREF-typed attribute to of the Link element type declaration

---

8 Please note, that for readability purposes the OCL xor-constraints are illustrated in UML syntax.
does not restrict the referenced elements to those that the designer originally intended to reference.

```xml
<!ELEMENT Link (…)>
<!ATTLIST Link
  to IDREF #REQUIRED
type (normal|automatic|transport) 'normal' "">
```

In WebML, three disjoint Link types are available, i.e., normal Link, automatic Link, and transport Link. Besides the Link concept, there are also the OKLink and KOLink modeling concepts from the ContentManagement package, which are specifically used to define links from ContentManagementUnits. Furthermore, besides ContentUnits and OperationUnits, there are other linkable elements in the HypertextOrganization package, namely Page and Area. Consequently, there are multiple sourceElement–link–targetElement tuples of which some are allowed in WebML, only (cf. Table 3).

<table>
<thead>
<tr>
<th>From/To</th>
<th>Content Unit</th>
<th>Operation Unit</th>
<th>Page</th>
<th>Area</th>
</tr>
</thead>
<tbody>
<tr>
<td>Content Unit</td>
<td>normal</td>
<td>automatic</td>
<td>KO</td>
<td>KO</td>
</tr>
<tr>
<td></td>
<td>transport</td>
<td>transport</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Operation Unit</td>
<td>transport</td>
<td>OK</td>
<td>KO</td>
<td>KO</td>
</tr>
<tr>
<td></td>
<td>transport</td>
<td>OK</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Page</td>
<td>normal</td>
<td>transport</td>
<td>KO</td>
<td>KO</td>
</tr>
<tr>
<td></td>
<td>normal</td>
<td>normal</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

These sourceElement–link–targetElement tuples, however, are not restricted to the allowed ones in the WebML DTD. Instead these constraints are ensured implicitly within the tool support. Aiming at a precise definition of sourceElement–link–targetElement tuples in the WebML metamodel, we introduce the LinkableElement concept (cf. Figure 8), which acts as a super class for all possible sources and targets. In addition, we have to define appropriate OCL constraints to restrict the sourceElement–link–targetElement tuples to those that are allowed in WebML (cf. Table 3) and that are not yet captured by the metamodel.

### 3.6.5 Missing inheritance mechanism

DTDs provide no concepts for specifying inheritance relationships. In the WebML DTD, Pages contain different kinds of ContentUnits.

```xml
<!ELEMENT Page (ContentUnits,…)>
<!ELEMENT ContentUnits ANY>
```

The problem of the missing inheritance mechanism in DTDs often results in the definition of Any element types for allowing the containment of certain element types. Still, the Any element type does not restrict which element types are allowed, i.e., only ContentUnits, and which are not allowed at the instance layer. Again, these constraints must be ensured by the tool.

In the metamodel, we therefore introduce an abstract class ContentUnit (cf. Figure 8), which ensures that Pages from the HypertextOrganization package contain subclasses of ContentUnit, only. A similar example can be found in the ContentManagement package (cf Figure 7), where the OperationUnit is introduced as an abstract class, which ensures that Areas and Stieviews from the HypertextOrganization package contain subclasses of OperationUnit, only.

### 4. RELATED WORK

With respect to our approach of defining a MOF-based metamodel for WebML we distinguish between two kinds of related work: first, related work concerning our primary goal to design a metamodel for WebML, i.e., metamodels of other web modeling languages, and second, related work concerning our methodology in designing a metamodel for WebML, i.e., transformation of DTDs to MOF-based metamodels.

**Metamodels in Web Engineering Methodologies.** To the best of our knowledge, three web modeling approaches [2], [15], [19] are currently defined on top of a metamodel.

W2000 [2], a successor of HDM [11], originally has been defined as an extension to UML. In [3], the metamodel approach (i.e., the provision of a metamodel based on MOF 1.4 [20]) has been motivated and adopted as a necessity for providing tool support for an evolving language definition.

The metamodel of UWE [15] has been designed as a conservative extension to the UML 1.4 metamodel [26], and thus is implicitly based on MOF 1.4. It is intended as a step towards a future common metamodel for the Web application domain, which will support the concepts of all Web design methodologies. Similar to [2], a language definition already existed as UML Profile.

Muller et al. [19] present a model-driven design and development approach with the Netsilon tool. The tool is based on a metamodel specified with MOF 1.4 and the Xion action language. The decision for a metamodel-based approach has been motivated by the fact that in the web application domain the semantic distance between existing modeling elements (e.g., of UML) and newly defined modeling elements is becoming too large.

Our work is complementary to [2], [15], in that we propose a metamodel for another prominent web design methodology, i.e., WebML, and thus make a further step towards a common metamodel for the web application domain [15]. But even more important to us is that, by proposing a metamodel for WebML we enable the transition to model-driven engineering techniques within the WebML design methodology. Our approach to design the metamodel is different from others, in that we generated the WebML metamodel semi-automatically, instead of manually deriving it from an existing language definition. Besides, the resulting WebML metamodel is based on Ecore and thus, basically corresponds to MOF 2.0, while the metamodels of [2], [15], [19] are based on MOF 1.4.

**Transforming DTDs to metamodels.** There already exist several approaches for transformation from the model technical space to the XML technical space and vice versa. In [33], we present an elaborate overview of existing approaches. 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. Only some of them provide tool-based transformation support. To the best of our knowledge, there is no approach mapping between concepts of DTD and concepts of MOF. In doing so, our work differs from the existing approaches in that we support intra-layer correspondences (M3) and transformations (M2) (cf. Figure 1), while existing approaches usually define cross-layer
correspondences (from M3 to M2) and transformations (from M2 to M1). With intra-layer mappings, one is able to derive intra-layer mappings at lower layers of the architecture. Deriving mappings at M2 from mappings at M3 allows performing transformations at M1, i.e., transformations of XML documents to UML models (cf. future work in Section 5). Cross-layer transformation approaches, however, are limited to transforming XML documents into object models, which have to conform to a 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].

5. CONCLUSION AND FUTURE WORK
In this work we have proposed a MOF-based metamodel for WebML which has been generated semi-automatically from an existing DTD-based language definition. Our approach for the generation of MOF-based metamodels from DTDs relies on a set of generic transformation rules, heuristics, and user interactions to manually improve the automatically generated metamodels. Since there is no implementation of MOF 2.0, we have built our transformation framework, the MetaModelGenerator, on the EMF. Thus, the WebML metamodel now is available as an Ecore-based metamodel. With the provision of such a metamodel, the WebML design methodology is now ready to move on to a model-driven web development approach. At the same time, another step towards a common web modeling metamodel [15] has been made.

Concerning future work, we particularly strive for first, the refinement of the proposed metamodel and second, its extension with concepts from the aspect-oriented software development (AOSD) paradigm for providing modeling the customization aspect of ubiquitous web applications.

A common metamodel for Web modeling. In a first step, we plan to incorporate recent concepts of WebML (i.e., concepts which are partly supported in WebRatio, but not defined in the WebML DTD) for modeling context-aware [6] and service-enabled web applications [16]. According to [15], we plan to investigate other existing web modeling approaches in order to integrate their concepts within a common Web modeling metamodel. Instead of integrating the concepts of different methodologies in a common metamodel, another interesting approach would be to integrate them at an even higher level, i.e., in a domain ontology, while making use of our ongoing research approach ModelCVS [12].

aspectUWA - Modeling Customization in Ubiquitous Web Applications. Modeling of customization in ubiquitous web applications (UWA) is a complex task, affecting all levels of a UWA, i.e., the content, the hypertext, and the presentation. Hence, customization represents a crosscutting concern. The aspect of customization, however, can not be properly captured by current Web modeling approaches. In fact, it is often intermingled with the core Web application. We propose to use aspect-orientation as driving paradigm for capturing customization of UWAs at the modeling level [31]. In particular, we plan the extension of an existing Web modeling language (e.g., a refined version of the WebML metamodel) with concepts from the aspect-orientation paradigm. In [4], adaptivity has already been identified as a crosscutting concern in Web applications, i.e., UWE has been extended with aspect-oriented techniques allowing customization at the hypertext level, only. Besides these two main directions, further work concerns three disjoint extensions to our transformation framework. First, the transformation framework needs further testing within other case studies and refinements of transformation rules and heuristics. Second, a comparison of our currently Java-based MetaModelGenerator with a model-driven transformation approach represents another interesting future research direction. In particular, the proposed DTD metamodel can be reused for describing the DTD-to-MOF transformation rules and heuristics as ATL transformations. In this respect, one does not only generate metamodels from DTDs in order to enable MDE, but just in doing so, applies MDE techniques. And third, one could perform transformations at M1 level, i.e., transformations of XML documents, which conform to a DTD, into models, which again conform to a corresponding metamodel, by deriving transformation rules from the existing mappings at higher layers. Therefore, the MetaModelGenerator should be capable of producing a ModelGenerator (MG) for a given DTD. With this approach, we would be able to transform existing WebML models, represented as XML documents, into models that conform to our MOF-based WebML metamodel. Thus, existing WebML projects can be migrated to the model technical space.

6. ACKNOWLEDGMENTS
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.

7. REFERENCES


[34] N. Wirth. What can we do about the unnecessary diversity of notation for syntactic definitions? CACM, 20 (11), November 1977, pp. 822-823.

