A Metamodel Transformation Framework for the Migration of WebML models to MDA

Marco Brambilla[1], Piero Fraternali[1], Massimo Tisi[1]

[1] Politecnico di Milano, Dipartimento di Elettronica e Informazione
P.za L. Da Vinci, 32. I-20133 Milano - Italy
{marco.brambilla, piero.fraternali, massimo.tisi}@polimi.it

Abstract. Traditional methodologies in Model Driven Web Engineering, like WebML, are based on domain specific modeling languages. A Web application is usually designed using several Domain Specific Models (DSM), often based on different formalisms and abstraction levels. In this paper we propose a model-driven procedure for integrating pre-MDA DSMs within the MDA framework. The DSMs, originally expressed in different formalisms, are translated into a unified representation that conforms to a MDA metamodel. The procedure, given the definition of suitable model transformations, is completely automatic. The proposed framework is fully implemented for the WebML metamodel and can be generalized to other Web engineering approaches.

1 Introduction

Trends toward MDA/MDD put at risk of obsolescence the existing Web engineering approaches based on domain specific languages (DSL) and models (DSM). Such pre-existing development methodologies require a complex transformation process to fully benefit from MDA tools. On the other hand, the need of modernizing these languages can be taken as opportunity for homogenizing different models within a unified notation. In this paper, we consider the WebML [9] DSM language for designing Web applications and we propose a transformation framework towards a MDA-compliant architecture. We propose an extension of the currently available WebML metamodels, to include the Derivation modeling, an additional model of the language that allows to specify calculated and derived data within the data models. Upon this metamodel, we propose a multi-step transformation approach based on Higher Order Transformations (HOT). We offer a migration path from a legacy technical space to MDA, based on a set of transformations that can be reused to:

1. transform any DSL defined in the legacy technical space to a corresponding MOF metamodel (metamodel transformation, M2T);
2. map every model defined by means of the legacy DSL to a MDA model (model transformation, M1T);
3. guarantee that the generated model is an instance of the corresponding metamodel. This can be achieved by enforcing the coherence between the

two previous tasks. To this purpose, we use a Higher Order Transformation (HOT) for automatically generating the M1T transformation from the M2T transformation.

Once the framework is in place, the developer can optionally implement a final refinement transformation (M1Tr) to address particular issues of the specific DSL or specific structures that require ad hoc domain knowledge and cannot be automatically inferred.

Thanks to our approach, only one higher order transformation and one metamodel transformation are needed for each technical space. The model transformations for any DSL in that technical space can be automatically generated. Moreover, any change in the DSL does not require to modify the transformations, because the manually written ones (M2T and HOT) depend only on the technical space, while the model transformation (M1T), the only one that depends on the DSL, can be automatically regenerated.

The implementation of our research uses the ATL transformation language, focusing on the migration from the XML/DTD space to Ecore, and is based on the Eclipse Modeling Framework, that provides a mature implementation for several MDA standards.

The paper is organized as follows: Section 2 discusses the related works; Section 3 presents the WebML approach, the metamodel proposed for this language, and some novel evolutions and additions to the metamodel. Section 4 presents the concrete transformation designed for the translation from WebML to MDA; and Section 5 concludes the paper.

2 Related work

The issue of defining a bridge between legacy modeling languages and MDA has been addressed in several works, such as [7], [6] and [15]. The issue is also analogous to the co-adaptation of models compliant to an evolving metamodel ([11], [19], [13]). With respect to these works our framework is the first to use higher order transformations to directly synchronize the model level transformation with the meta level transformation.

More specifically, large efforts have been spent for defining MDA-compliant metamodels of existing Web engineering languages. Some works concentrate on the specific issue of manually mapping WebML to MDA: [16] remodels WebML using MOF and [17] proposes a WebML UML 2.0 profile to facilitate the interoperability between the WebML IDEs (e.g. WebRatio [5]) and UML modelling tools. Our proposal extends these works, as it provides some significant extensions to the WebML metamodel and partially automate the production of both models and metamodels.

Higher order transformations have already been used to perform various tasks in model driven development [7], [10], [12]. To our knowledge this work is the first to address a HOT that translates an ATL transformation at the meta-level to the associated transformation at model level.

Several approaches have focused on the transformation between XML schema languages and metamodels: [20] surveys 13 proposals, classified according to the direction of the transformation (i.e. forward, backward or both) and the concrete formalisms used as source/target of the transformation (i.e. on the XML side either DTD or XML Schema and on the model side either MOF, UML or ER). To the best of our knowledge the only approach conducting a forward transformation from DTD to MOF is [18], which focuses on the same case study, the WebML DSL. With respect to this work, our framework provides, on the basis of the transformations defined at the meta-level, the generation of coherent transformations at the model level, and thus allows the immediate reuse of existing WebML models in MDA.

3 WebML method and metamodel

WebML [9] is a DSM language for data-, service-, and process- centric Web applications. In this work we consider the existing definition of the WebML metamodel, we extend it to cover some missing aspects (namely, the derivation model), and we provide a transformation framework toward MDA.

WebML allows specifying the conceptual model of Web applications built on top of a data schema and composed of one or more hypertexts used to publish or manipulate data. The specification consists of several sub-models: the *data model* represents the data schema; the *hypertext model* represents the content of pages, the navigation paths, and the parameters that flow among the components; and the *presentation model* describes the visual aspects of pages.

The data model is the standard Entity-Relationship (E-R) model. Upon the same data model, different hypertext models (*site views*) can be defined (e.g., for different types of users or for different publishing devices). A site view is a graph of *pages*, consisting of connected *units*, representing data publishing components: a unit displays instances of an entity, possibly restricted by a *selector*. Units are related to each other through *links*, representing navigational paths and carrying parameters. WebML allows specifying also update *operations* on the underlying data (e.g., the creation, modification and deletion of instances of entities or relationships) or operations performing arbitrary actions (e.g. sending an e-mail, invoking a remote service, and so on).

Figure 1 shows a simple hypertext, containing two pages. Page *Recent Movies List* contains an index unit defined over the *Movie* entity, which shows the list of movies produced after year 2008 (selector [Year $>$ 2008]) , and a data unit also defined over the *Movie* entity, which displays the details of the movie selected from the index. The link between the two units carries the parameter CurrMovie, used by the selector [OID=CurrMovie] of the data unit. Another link connects *Recent Movies List* page to *Search Movies* page, without carrying any parameter. Page *Search Movies* contains an entry unit for inserting the movie title to be searched, a scroller unit, and a multidata unit displaying a block of search results. Through the scroller unit it is possible to move to the first, previous, next, and last block of results.

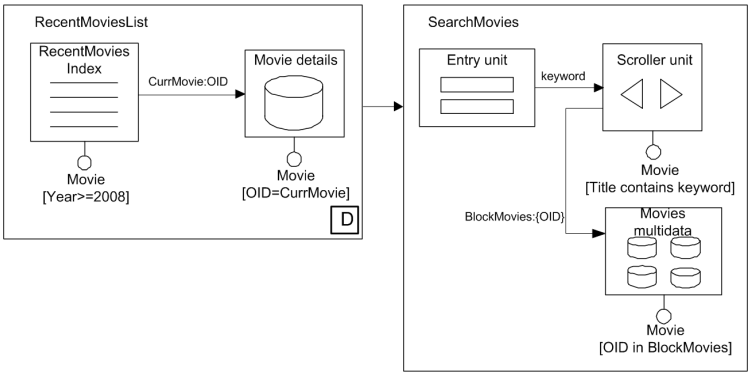

Fig. 1. Example of WebML hypertext model.

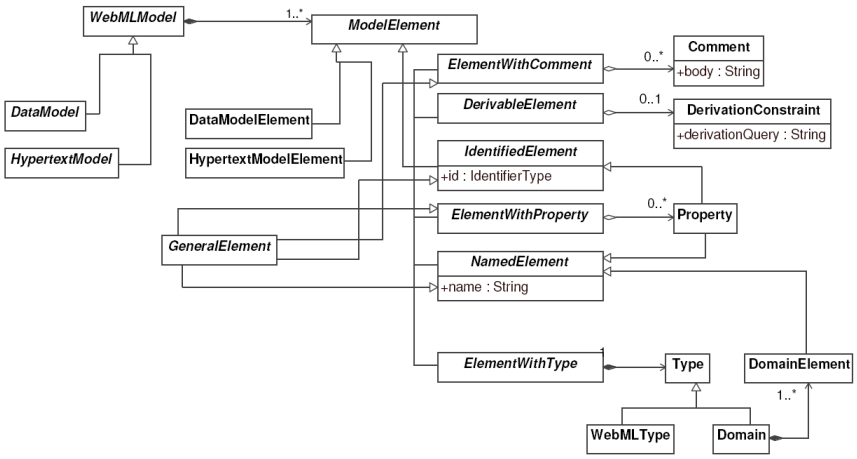

Fig. 2. Overview of the WebML metamodel.

The WebML language is supported by the WebRatio CASE tool [5], a development environment for the visual specification of Web applications and the automatic generation of code for the J2EE platform. The design environment is equipped with a code generator that deploys the specified application and Web Services, by automatically generating all the necessary pieces of code.

Some proposals of WebML metamodels already exist ([8] and [18]). We have extended the metamodel presented in [16] and defined the metamodel summarized in Figure 2: the classes sketched in the figure are further refined for describing all the details of the DSL.

3.1 The derivation syntax and metamodel

Data specification in WebML can be enriched by means of the Derivation metamodel. This metamodel allows to specify entities, attributes, and relationships that are calculated starting from other data. For instance, the price after taxes of an article may be computed as the product of the price before taxes and the VAT.

WebML derivation metamodel permits to derive entities, attributes, and relationships according to calculation rules. WebML provides a specific grammar for this language, that allows to specify the following aspects:

- an *entity* whose instances are defined as a subset of instances of a superentity in a IS-A hierarchy, according to a condition on attributes or relationships;
- an *attribute*, whose value is calculated as an aritmetic formula on other attributes, or as an imported or aggregated attribute starting from of connected entities.
- a *relationship*, that is defined as a subset of another relatioship, or as a concatenation of interconnected relatioships.

Two examples of derived attributes can be seen below:

```
PriceAfterTax = (Price* VAT)
NumberOfItems = Count(Trolley.TrolleyToProduct)
```

The PriceAfterTax is calculated as Price by VAT. The number of items in the trolley is calculated as counting of the instances of the relationship Trolley-ToProduct. An example of derived relatioship is:

```
Artist.Published-Tracks=Artist.ArtistToAlbum.AlbumToTrack.
```

The relationship Published-Tracks is defined as a concatenation of the existing relationships that connect the Artist to its Album and each Album to the respective Tracks.

The full definition of the WebML derivation grammar is stated below as a set of grammar rules in the JavaCC [4] syntax:

```
<Query : ( EntityQuery | RelationshipQuery | AttributeQuery ) >
<EntityQuery : Step <WHERE> Condition ( ";" | <EOF> )>
<RelationshipQuery : ( <SELF> <TO> Step | PathExpression ) ( <WHERE>
    Condition )? ( ";" | <EOF> )>
<AttributeQuery : AttributeValue ( <WHERE> Condition )? ( ";" | <EOF> )>
<Step : <IDENTIFIER> ( <LEFTBRACKET> <AS> <IDENTIFIER> <RIGHTBRACKET> )?>
<PathExpression : ( <SELF> | <IDENTIFIER> ) ( <DOT> Step )*>
<AttributeValue : ( AttributeExpression | <LEFTBRACKET> AttributeValue
    <RIGHTBRACKET> ) ( <OPERATOR> ( AttributeExpression | <LEFTBRACKET>
    AttributeValue <RIGHTBRACKET> ) )*>
<AttributeExpression : ( <STRING> | <NUMBER> | PathExpression |
    <AGGRFUNCTION> <LEFTBRACKET> PathExpression <RIGHTBRACKET> )>
<Member : ( <NOT> )? <IN> PathExpression>
<IsNull : <IS> ( <NOT> )? <NULL>>
<WhereExpression : ( ( <IDENTIFIER> | <SELF> ) <ISA> <IDENTIFIER> |
    AttributeExpression ( Member | IsNull | <COMPARATOR> (
    AttributeExpression | <TRUE> | <FALSE> ) ) | ( <LEFTBRACKET>
    Condition <RIGHTBRACKET> ) )>
<LogicalTerm : WhereExpression ( <AND> WhereExpression )*>
<Condition : LogicalTerm ( <OR> LogicalTerm )*>
```

The derivation syntax has been integrated into the WebML metamodel. The diagram in Figure 3 shows the general metamodel of a WebML derivation query. A new Class of the metamodel, DerivableElement, is the connection point between the Structure metamodel and the Derivation metamodel. The bidirectional references between the Structure model and the Derivation model are managed in the following way: 1) Entities, Roles and Attributes are all DerivableElements (in the Structure metamodel) and they contain a Query (in the Derivation metamodel); 2) the Query (in the Derivation metamodel) in turn contains one or more instances of PathStep, a class that represents a reference to a DerivableElement (in the Strucutre metamodel).

Three kinds of Query are provided and they all contain a ConditionExpression that filters the instances on which the derivation Query is applied, plus other ad hoc constructs: 1) AttributeQuery contains an AttributeExpression to specify the calculation rules for the attribute value; 2) EntityQuery contains a single PathStep, i.e. a reference to the source entity of the derivation; 3) RoleQuery can contain the PathStep to reference a source relationship (analogously to EntityQuery) or a complete ReferencePath that specifies a chain of source relationships in the structural model.

The metamodel of a ConditionExpression and of an AttributeExpression is detailed in Figures 4 and 5.

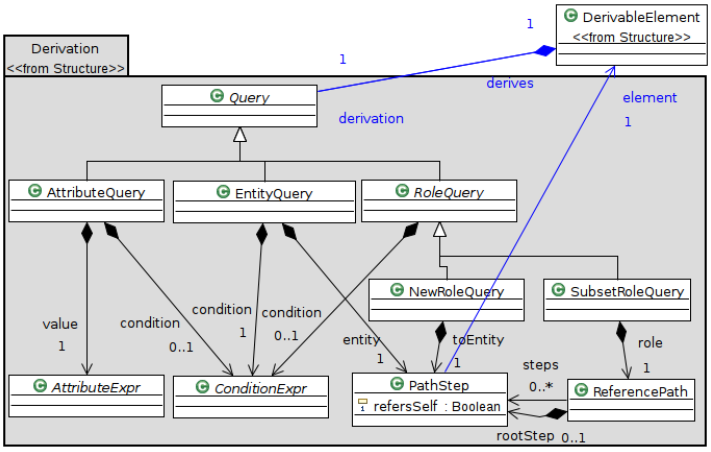

Fig. 3. Metamodel of a derivation query

In Figure 4, the several different kinds of ConditionExpression contain one or more ReferencePath to reference operands in the Structure metamodel and AttributeAtoms to reference atomic values in AttributeExpressions.

In Figure 5, AttributeExpression can contain as operands one or more elements of the Constant hierarchy and one or more ReferencePaths to attribute values in the Structure metamodel.

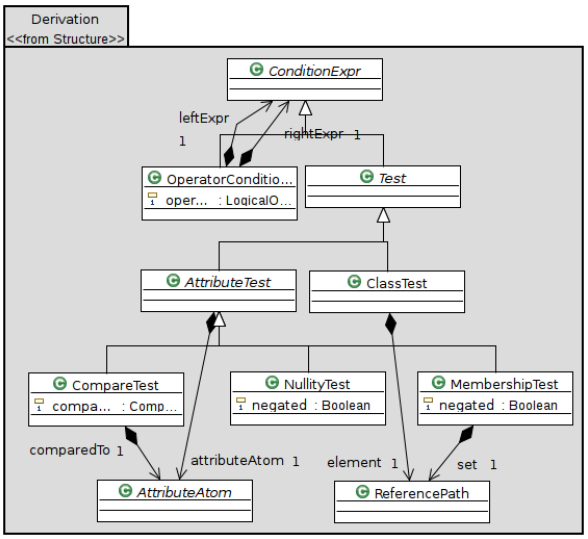

Fig. 4. Metamodel of an OQL condition expression

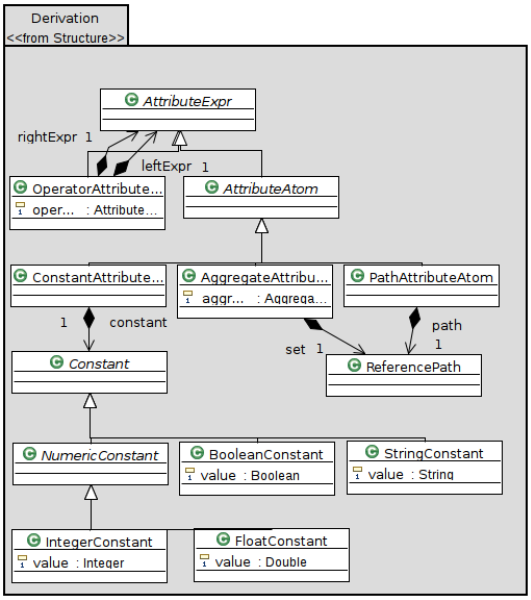

Fig. 5. Metamodel of an OQL attribute expression

4 Transformation of WebML models to MDA

The migration of any DSM from a legacy space to MDA is a task that involves three related mappings:

1. the legacy DSM language must be mapped to an ad-hoc MOF metamodel;
2. the DSMs must be transformed into models in the MDA technical space;
3. the *conforms to* relationship that connects models to their language must be mapped from the legacy technical space to the MDA technical space.

These correspondences are not independent. Normally, mapping (1) and (2) are designed independently, in such a way to implicitly satisfy mapping (3). Maintaining coherence manually is a time-consuming and error-prone activity. The framework proposed in this paper allows the automatic synchronization of the transformations. In particular, we formally define two of these correspondences for the WebML case, namely (1) and (3), and we automatically generate the other, i.e. (2), by means of HOTs (higher order transformations).

As shown in Figure 6, the translation framework consists of three phases, followed by a final optional step:

1. The *metamodel generation* phase addresses mapping (1), by performing the automatic translation of the WebML DSL to a MDA metamodel. The translation involves a first step of injection and a second step of metamodel transformation (M2T). These steps require the availability of the metametamodel (MMM) of the involved technical spaces, i.e. the legacy metametamodel and Ecore. The legacy MMM needs to be expressed as an MDA metamodel (i.e., conforming to Ecore).
   The DSL injector parses the concrete syntax in which the DSL is expressed and derives a representation of the DSL abstract syntax as an instance of the legacy MMM. Subsequently the transformation M2T is defined as a set of transformation rules that map the syntactical constructs of the legacy MMM to Ecore. The application of the M2T transformation translates any metamodel in the legacy technical space into a correspondent Ecore metamodel. Notice that the M2T transformation relies only on a mapping between the two MMMs, i.e. the two technical spaces. Once this transformation has been specified, it can be reused for the migration *of any DSL* between the addressed technical spaces.
2. The *model generation* phase addresses mapping (2), by automatically translating the legacy DSMs into models compliant with the new metamodel. This phase is analogous to the previous one, but applied at a lower level in the MDA stack: it again involves an injection step followed by a transformation step. The injection step performs the parsing of the concrete syntax of the DSM, and generates an instance of the metamodel associated with the DSM syntax. Subsequently, the model transformation step (M1T) computes the final MDA model as an instance of the metamodel produced by the metamodel generation.

3. The *higher order transformation* phase addresses mapping (3), guaranteeing the coherence between the *conforms to* relationship of the two technical spaces. This task is performed in two sub-tasks: 1) a *promotion transformation* obtains the DSL metamodel by promoting the model M1 resulting from the model generation phase to metamodel (M2); 2) a HOT derives the M1T transformation by translating the M2T transformation. The HOT has to be defined manually and encapsulates the translation of the *conforms to* relationship to the new technical space.
4. Finally, the *refinement model transformation* can be optionally applied to adapt the resulting model to some manually introduced variations of the DSL metamodel. This phase typically affects marginal aspects of the automatically generated models, and will not be treated in detail in the rest of the paper.

The three main transformations involve the definition of a mapping on the higher level of abstraction:

- the M2T transformation is defined by a *M3 Mapping* between the elements of the DSL syntax (i.e. the legacy metametamodel) and the elements of Ecore;
- the M1T transformation is defined by a *M2 Mapping* associating each construct of the DSL to a corresponding pattern of elements in the output metamodel;
- the HOT is defined by a *Conformance Mapping* that associates the ATL rules of M2T with ATL rules of M1T. The mapping grants that the translated rule maintains the *conforms to* semantics: given an ATL rule R2 that translates the pattern P2a to P2b, the Conformance Mapping associates R2 to the transformation R1 that translates the pattern P1a to P1b, such that P1a *conforms to* P2a and P1b *conforms to* P2b.

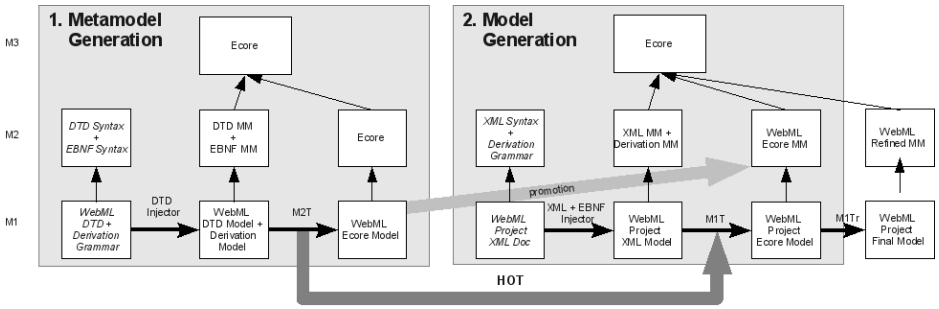

Fig. 6. Diagram of the general framework

We now describe the transformations that compose the framework for transforming WebML to a MDA-compliant representation, according the three aforementioned phases: Metamodel Generation, Model Generation, and Higher Order

Transformation. The framework implementation is orchestrated by means of an Ant script that uses the tasks provided by the AM3 project [1]. The complete sources of the prototype framework can be downloaded from the project website [3].

4.1 The Legacy Technical Space: WebML DTD/XML and Derivation grammar

The WebML legacy technical space comprises two models: the WebML main metamodel, based on XML files that conform to given Document Type Definitions (DTD), plus the derivation model, expressed according to the grammar presented in Section 3.1. For the main WebML hypertext model:

- the DSL syntax is defined by the DTD grammar (level M3),
- the DSL is specified by a DTD document (level M2),
- the DSM is an XML document (level M1),
- the *conforms to* relationship corresponds to the *validity* relationship in the DTD/XML technical space, i.e. to the relationship that connects a valid XML document to its associated DTD.

For the derivation model:

- the DSL syntax is defined by the JavaCC syntax for describing an EBNF (level M3),
- the DSL is specified by a the JavaCC rules presented in Section 3.1 (level M2),
- the DSM is a phrase generated by this grammar (e.g. a correct derivation query) (level M1),
- the *conforms to* relationship corresponds to the relationship that connects a correct phrase to its associated grammar (e.g. a correct derivation query to the derivation grammar rules).

Moving from the DTD/XML technical space to the MDA technical space requires the following mappings:

- *M3 mapping:* to map the DSL grammar (DTD+EBNF) to Ecore, associating to each DTD construct (e.g. ELEMENT) and each EBNF construct (e.g. NONTERMINAL) a correspondent Ecore translation (e.g. an EClass).
- *M2 mapping:* to map a specific DSL definition to a correspondent Ecore metamodel, associating each DTD definition (e.g. a specific ELEMENT) and each EBNF definition (e.g. a specific NONTERMINAL) to a correspondent Ecore element (e.g. a specific EClass).
- *Conformance mapping:* to map the *validity* relationship to the *conforms to* relationship, so that if a document is valid with respect to its associated DTD and EBNF then its correspondent model conforms to its metamodel.

The difference in expressive power between the DTD syntax, the EBNF syntax and Ecore, makes the bridging between these formalisms a non-deterministic

activity. Compared to Ecore, the DTD syntax is ambiguous in several points and a bridging algorithm between the two technical spaces can only rely on default policies and heuristics to choose the optimal translation. Examples of lack of expressiveness are the general CDATA attribute type, that can be mapped on different Ecore types such as EString, EInteger, or EFloat; and the IDREF attribute type that can be mapped as an EReference without specifying the associated eType. A comparison on the expressive power and features of DTDs and Ecore can be found in [18].

The discussion about the optimal policies for the M3 mapping are outside the scope of this paper. The most convenient heuristics can be different, depending on the considered DSL, and can be refined over time. The framework we provide assures that, upon changes on the M3 mapping (or on the DSL itself), the M2 mapping is automatically synchronized, thus mantaining the coherence between M3 and M2.

4.2 Metamodel Generation

The Metamodel Generation phase transforms a DSL specified by means of a DTD + EBNF into an Ecore metamodel.

The Ecore metamodel is obtained merging: 1) the DTD metamodel, 2) the EBNF metamodel, 3) cross-reference links between the metamodels. The DTD metamodel is a refined version of the one provided in [14]. The first class objects of this metamodel are Element, Attribute, Sequence, etc. The EBNF metamodel has been developed ad-hoc and its first class objects are Rule, NonTerminal, Disjunction, etc.. A set of cross-reference links specify which constructs of the two input technical spaces (DTD and EBNF) could be referenced by the other technical space. In the WebML case to generate the associations PathStep-DerivableElement and DerivableElement-Query explained in Section 3, a cross-reference link has to be added between the Element EClass of the DTD metamodel and the NonTerminal EClass of the EBNF metamodel.

DSL Injection. The DSL Injection step consists in parsing the concrete syntax of the DTD specification and of the EBNF specification to derive an instance of the defined metamodel. Developing an injector for the legacy metametamodel is generally a simple task, because a parser of the concrete syntax of the metametamodel is usually available in the legacy technical space and can easily be extended with semantic actions to build the correspondent metamodel elements. In our work, a prototype DtdEbnfInjector class has been developed using the DTD Parser provided in [2].

M2T. The M2T transformation defines the M3 mapping, between the DSL MM and Ecore. M2T is implemented as an ATL transformation defining the translation policies between, e.g., the elements of a DTD and the classes of a metamodel. M2T is defined without any domain specific knowledge: it is a

general transformation that can be reused to translate every DSL defined by means of a DTD, an EBNF and eventual cross-references in an Ecore Metamodel.

The M2T rules translate DTD Elements, Attributes and Children to EClass, EAttribute and EReference elements, based on several heuristic choices. For instance:

- the Children of a DTD Element are always translated as containment references in Ecore;
- all the DTD Attributes are translated to simple EAttributes, without considering their type. This heuristic is sufficient for simple cases, especially when the use of IDREFs is limited and can be dealt with in the M1Tr transformation.

4.3 Model Generation

The Model Generation phase transforms a WebML project specified as an XML document into an instance of the WebML metamodel generated in the Metamodel Generation phase. The core is an XML injection step, followed by the generated transformation M1T. The metamodels involved in this phase are the XML metamodel and the WebML metamodel. The former is a standard metamodel provided by the EMF project whose first class objects are Tag, Node, Attribute, etc.

DSM Injection. The injection of the DSM is easily performed extending the standard XMLInjector, an injector provided by the AM3 project, to convert an XML document to an instance of the XML metamodel. The Java injector is extended to launch a JavaCC parser when it encounters a derivation element.

M1T. M1T is the transformation that maps an XML+EBNF model to an Ecore model, instance of the generated DSM metamodel, i.e. the WebML metamodel. Being an ATL transformation that has the DSL metamodel as the output metamodel, M1T can not be independent of the DSL metamodel. For this reason, traditional transformation-based approaches to the migration of DSMs to MDA require one to develop a different M1T transformation for each DSL. The generative approach that we propose overcomes this problem: in our framework M1T is still a DSL-specific transformation, but it is generated by a DSL-agnostic HOT.

M1Tr. Being generated from M2T, the M1T transformation is a DSL-specific transformation that does not use any DSL-specific knowledge. Since M2T and HOT are DSL-agnostic transformations, every possible DSL-specific transformation that is needed to the DSL bridging has to be specified in a subsequent step. This step that lies outside from the generative framework is represented by the optional M1Tr (i.e. M1T refinement) transformation.

M1Tr translates the generated model to an instance of a manually defined DSL metamodel that usually will have only a limited set of differences with the

generated one. M1Tr is a DSL-specific transformation that can be used to solve different issues, such as the different expressive power of the metametamodels, the structural limitations of the DSL-agnostic transformations, the implementation limits of the prototype (especially with respect to the M2T syntax).

In the WebML case, M1Tr can adapt the generated models to the official WebML metamodel, manually designed from scratch in [16] and extended in this work with the derivation model.

4.4 Higher Order Transformation

The Higher Order Transformation phase is responsible for the translation of the M2T transformation, that generates the new metamodel, to the M1T transformation, that generates the new models. The only metamodel involved in this phase is the ATL metamodel that provides a representation of an ATL transformation as an instance model.

The provided prototype HOT, in its current state, is able to transform only a very limited set of the ATL features. This translates in a set of constraints that our prototype framework imposes on the syntax of M2T:

- only matched (declarative) rules are allowed
- the source pattern of each rule must comprise only one source pattern element (as in ATL 2004)
- the target pattern of each rule must comprise only one simple or iterative target pattern element
- local variable sections or imperative block sections are not allowed
- OCL declarative expressions are restricted to the basic path expressions for accessing element features and to some basic operations on collections, such as union.

While these restrictions did not hamper the translation of a DTD into Ecore, in general more advanced constructs might be needed for complex technical spaces or heuristics. In this case, the HOT transformation will need to be extended or, in the worst case scenario, to be implemented with a general purpose language (e.g., Java).

ATL Injection/Extraction. The execution of the HOT has to be preceded and followed respectively by an injection and an extraction of the ATL transformations. M2T is injected as an instance of the ATL metamodel and, after the HOT has been executed, M1T is extracted to its textual form. The injection and extraction of the ATL transformations is implemented using the Injector and Extractor provided by the AM3 project.

HOT. The following code is a simplified rule extracted from our HOT:

```
rule RestrictedElement {
  from
    matched : ATL!MatchedRule (
```

```
    matched.inPattern.elements.first().type.name = 'DTDMM!RestrictedElement' )
using {
  matchedElements : Sequence(OclAny) = DTDMM!Element.allInstances()->
    select(e | e.oclType().toString() =
      'DTDMM!'+matched.inPattern.elements.first().type.name); }
to
  atl : distinct ATL!MatchedRule foreach (e in matchedElements) (
    name <- e.name, inPattern <- inPat, outPattern <- outPat, isRefining <- false,
    isAbstract <- false ),
  inPat : distinct ATL!InPattern foreach (e in matchedElements) (
    elements <- elementin, filter <- oc ),
  elementin : distinct ATL!SimpleInPatternElement foreach (e in matchedElements) (
    id <- 'tag0', varName <- 'tag',  type <- intype ),
  intype : distinct ATL!OclModelElement foreach (e in matchedElements)  (
    name <- 'XML!Tag' ),
  oc : distinct ATL!OperatorCallExp foreach (e in matchedElements)  (
    operationName <- '=', source <- noac, arguments <- s ),
  noac : distinct ATL!NavigationOrAttributeCallExp foreach (e in matchedElements)  (
    name <- 'name', source <-fv ),
  fv : distinct ATL!VariableExp foreach (e in matchedElements)  (
    name <- 'tag', referredVariable <- elementin ),
  s : distinct ATL!StringExp foreach (e in matchedElements)  (
    stringSymbol <- e.name ),
  -- OutPattern
  [...]
```

The rules of the HOT match in their source pattern the ATL elements of the input transformation. Then the rules can make use of the DSL metamodels to derive information on the structure of the DSL. This sample rule translates the matched rules of M2T that are applied to a RestrictedElement in the WebML DTD. When one of these rules is matched, the *using* part of the rule queries the WebML metamodel for all the instances of RestrictedElement. These instances were certainly matched by the rule in M2T and their result is saved in the matchedElements variable. Finally a set of output ATL rules is generated, by iterating on the matchedElements variable. The small excerpt alone generates a minimal scheleton of a rule without considering any of the structural features of the output pattern.

The previous excerpt gives an idea of how an approach based on such a high level of abstraction has a necessary drawback in terms of development cost. In particular the complexity of the HOT grows with the expressive power of the language in which M2T is specified. Our approach to face this issue is limiting the ATL features supported by the HOT to a defined set.

5 Conclusions

In this paper we have discussed a MDA framework to move a set of related legacy DSMs in the field of Web engineering to a MDA-compliant architecture. The transformation framework covers the translation of both the metamodel and the model levels, and grants automatic coherence of the two.

Our experience over the WebML language showed the feasability of a comprehensive transformation framework for a multi-language modeling approach for Web applications. We showed how it is possible to make homogeneous different metamodels, represented by different syntaxes, and how this can be translated to a unified MDA-compliant metamodel.

Although this work has been developed in the context of the WebML methodology, the approach is more general and can be applied to any other Web engineering DSM, simply by providing the appropriate transformations. The whole Model-driven Web Engineering field can therefore benefit of this contribution, that poses the basis of a general framework for model transformations toward MDA.

References

1. Am3 - http://www.eclipse.org/gmt/am3/.
2. Dtdparser - http://www.wutka.com/dtdparser.html.
3. Framework implementation - http://home.dei.polimi.it/mbrambil/legacytomda.
4. Javacc - https://javacc.dev.java.net/.
5. Webratio - http://www.webratio.com/.
6. Anas Abouzahra, Jean Bézivin, Marcos Didonet Del Fabro, and Frédéric Jouault. A practical approach to bridging domain specific languages with uml profiles. In *Best Practices for Model Driven Software Development at OOPSLA'05*, San Diego.
7. J. Bezivin, G. Hillairet, F. Jouault, I. Kurtev, and W. Piers. Bridging the ms/dsl tools and the eclipse modeling framework. *International Workshop on Software Factories at OOPSLA*, 2005.
8. R. G. Cattell, D. K. Barry, M. Berler, J. Eastman, D. Jordan, C. Russell, O. Schadow, T. Stanienda, and F. Velez. *The Object Data Standard: ODMG 3.0*. Morgan Kaufmann, 1 edition, 2000.
9. S. Ceri, P. Fraternali, A. Bongio, M. Brambilla, S. Comai, and M. Matera. *Designing Data-Intensive Web Applications*. Morgan Kaufmann, December 2002.
10. A. Cicchetti, D. Di Ruscio, and A. Pierantonio. A metamodel independent approach to difference representation. *TOOLS Europe*, 2007.
11. M. D. Del Fabro and P. Valduriez. Semi-automatic model integration using matching transformations and weaving models. *ACM symposium on Applied Computing*, 2007.
12. Bas Graaf and Arie van Deursen. Using mde for generic comparison of views. In *4th MoDeVVa Workshop at MODELS 2007*.
13. Boris Gruschko. Towards synchronizing models with evolving metamodels. *Int. Workshop on Model-Driven Software Evolution at ECSMR*, 2007.
14. Pierrick Guyard. Dtd metamodel - www.eclipse.org/gmt/am3/zoos/atlanticzoo.
15. H. Kern and S. Kuhne. Model interchange between aris and eclipse emf. *7th OOPSLA Workshop on Domain-Specific Modeling at OOPSLA*, 2007.
16. N. Moreno, P. Fraternali, and A. Vallecillo. Webml modelling in uml. *Software, IET*, 1:67–80, 2007.
17. N. Moreno, P. Fraternalli, and A. Vallecillo. A uml 2.0 profile for webml modeling. In *International Conference on Web Engineering 2006*, Palo Alto, California. ACM.
18. A. Schauerhuber, M. Wimmer, E. Kapsammer, W. Schwinger, and W. Retschitzegger. Bridging webml to model-driven engineering: from document type definitions to meta object facility. *Software, IET*, 1:81–97, 2007.
19. Guido Wachsmuth. Metamodel adaptation and model co-adaptation. In *ECOOP 2007: Object-Oriented Programming*, pages 600–624. Springer, 2007.
20. M. Wimmer, A. Schauerhuber, E. Kapsammer, and G. Kramler. From document type definitions to metamodels: The webml case study. *Report for Vienna University of Technology, March*, 2006.