A UML 2.0 Profile for WebML Modeling

Nathalie Moreno
Dept. Lenguajes y Ciencias de la Computacion.
University of Malaga
Malaga, Spain
vergara@lcc.uma.es

Piero Fraternalli
Dipartimento di Elettronica e Informazione
Politecnico di Milano
Milano, Italy
piero.fraternali@polimi.it

Antonio Vallecillo
Dept. Lenguages y Ciencias de la Computacion.
University of Malaga
Malaga, Spain
av@lcc.uma.es

ABSTRACT

In recent years, we have witnessed how the Web Engineering community considers the use of standard UML notation, techniques and supporting tools for modeling Web systems, including the adaptation of their own modeling languages, representation diagrams and development processes to UML. This interest for being MOF and UML-compliant arises from the increasing need to be able to interoperate with other notations and tools, and to exchange data and models, thus facilitating and improving reuse. WebML, like any other Domain Specific Language (DSL), allows to express in a precise and natural way the concepts and mechanisms of its domain of reference. However, it cannot fully interoperate with other notations, nor can it be integrated with other tools. As a solution to these requirements, in this paper we describe a UML 2.0 profile for WebML which allows WebML models to be used in conjunction with other notations and modeling tools.

Categories and Subject Descriptors

D.2.10 [**Software Engineering**]: Design—*Methodologies, representation*

General Terms

Design, Standardization, Languages

Keywords

WebML, UML, UML Profiles, Metamodels

1. INTRODUCTION

UML is a general purpose visual modeling language for specifying, constructing and documenting the artefacts of systems that can be used with all major application domains and implementation platforms. It has been widely adopted by both industry and academia as the standard language for describing software systems. This is reflected by the fact that it is currently supported by hundreds of model-driven commercial tools, which have been successfully used in numerous development projects. However, the fact that UML is a general purpose notation may limit its suitability for modeling some particular domains (e.g. Web applications), for which specialized languages and tools may be more appropriate (e.g. WebML/WebRatio [3], UWE/ArgoUWE [8], OO-H/VisualWade [5], etc).

Up to UML version 2.0 [10], both the lack of precision in the UML definition and the semantic gap between specific Web application concepts and the UML constructs hindered its full application in this context. However, with the advent of UML 2.0 the situation has changed, since not only its semantics have been defined more precisely , but it also incorporates a whole new set of diagrams and concepts which are more appropriate for modeling the structure and behavior of Web applications. In addition, UML 2.0 provides enhanced profiling capabilities, which allow the precise definition of the Meta Object Facility (MOF) based domain-specific languages. These languages allow the semantics of the basic UML elements to be extended and refined, and we will show how they can be successfully used to model WebML applications.

In general, OMG defines three possible approaches for defining domain-specific languages. The first one is based on the definition of a new language, an alternative to UML, using the mechanisms provided by OMG for defining object-based visual languages (i.e., the same mechanisms that have been used for defining UML and its metamodel). In this way, the syntax and semantics of the elements of the new language are defined to fit the domain's specific characteristics. The problem is that standard UML tools will not be able to deal with such a new language. The second and third alternatives are based on the particularization of UML (either a heavyweight or a light-weight particularization respectively), by specializing some of its elements, imposing new restrictions on them but respecting the UML metamodel, and without modifying the original semantics of the UML elements (i.e., the properties of the UML classes, associations, attributes, etc., will remain the same, but new constraints will be added to their original definitions and relationships). Syntactic sugar can also be defined in a profile, in terms of icons and symbols for the newly defined elements.

The first approach is the one followed by languages such as CWM (Common Warehouse Metamodel), since the semantics of some of the language constructs do not match the

ICWE'06 Workshops, July 10-14, 2006, Palo Alto, CA
ACM 1-59593-435-9/06/07

semantics of the corresponding UML model elements. In order to support the other alternatives, UML provides a set of extension mechanisms (stereotypes, tag definitions, and constraints) for specializing its elements, allowing customized extensions of UML for particular application domains. These customizations are sets of UML extensions, grouped into UML profiles.

Probably, the main advantage of UML profiles is not the extension of the UML metamodel (which is already too large and cumbersome to be used in full), but that they allow "restricting" the set of UML elements that need to be used in a given domain, particularizing the semantics of those elements in order to capture the semantics of the domain-specific elements they represent. It is important to notice that such a particularization can only be done by refinement, and without breaking the original semantics of UML elements.

In the shorter term, we certainly see an increasing interest in applying MOF, UML and UML profiles in the area of modeling Web applications. Several authors have proposed interesting approaches in terms of metamodels and UML extensions as cited in [4, 2, 9, 6]. This interest for being MOF and UML-compliant arises from the increasing need to being able to interoperate with other notations and tools, and to exchange data and models, thus facilitating and improving reuse. WebML, like any other Domain Specific Language (DSL), allows us to express in a precise and natural way the concepts and mechanisms of its domain of reference. However, it cannot fully interoperate with other notations, nor can it be integrated with other tools. This is where the requirements for interoperability come into play, so for example WebML models can be used in conjunction with other notations, or users can edit WebML models with their favorite modeling tools.

As a solution to WebML interoperability requirements, in this paper we describe a UML 2.0 profile for WebML which consists of two main parts. First, it defines the WebML metamodel as well as the semantics, properties and related elements of each metaclass. Second, WebML concepts are mapped to UML elements. This mapping contains information about every WebML concept, the UML base element that represents each concept, and the stereotype that extends the metaclass so that the specific domain terminology can be used.

The remainder of the paper is organized as follows. Section 2 briefly describes the WebML metamodel. On the basis of this, Section 3 proposes a UML 2.0 profile for WebML illustrating how to use it by means of an example. Section 4 disscusses some criteria and principles required for building a UML-based DSL representation and points to open problems requiring further investigation in dealing with the use of UML 2.0 for modeling Web applications. Finally, some conclusions and future work are outlined in Section 5.

2. METAMODELING WEBML

A metamodel is the "perfect" way to model a dynamically evolving notation and maintain it in a homogeneous and comprehensive way [2]. In this sense, the combination of MOF and the Object Constraint Language (OCL) can represent a good starting point in the redefinition of WebML and therefore it can also be a good starting point for its implementation in UML 2.0 as we will see in next section.

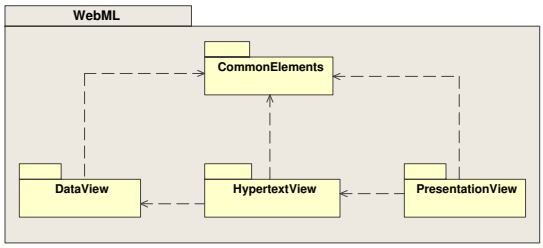

Figure 1: The WebML metamodel packages

Mapping a DSL like WebML to MOF involves representing each element of the domain — its syntax and semantics — as a MOF artefact and after that performing a refactoring process in order to introduce further MOF-elements such as compositions/aggregations or abstract classes that preserve the original semantics of WebML and allow to group concepts with similar attributes. When there is a formal description of the language grammar available this task is relatively simple for the syntax (although the metamodel generated must be augmented with several OCL constraints to cover the semantical features of the language). In the WebML case, a BNF grammar has been defined but only for the its derivation language (WebML-OQL). Another starting point considered for deriving the metamodel has been the set of Document Type Definition (DTDs) that WebRatio use for storing WebML projects [1]. However, special care must be taken with them since they introduce many auxiliary concepts, some of which do not really correspond with WebML terms or are no longer used as a consequence of the language evolution.

Starting from this knowledge, we have represented the whole WebML concept space by means of four metamodel packages, as illustrated in Figure 1: *CommonElemets*, *DataView*, *HypertextView* and *PresentationView*. Complying with a package requires to comply with its abstract syntax, well formedness rules, semantics and notation.

- The *CommonElements* package comprises core concepts used when metamodeling WebML elements, such as data types or common features (e.g. name, comment, identifier, properties, etc.). The other packages have dependencies on the *CommonElements* package because of association and generalization relationships.
- The *DataView* package addresses WebML data model concepts, such as entities, attributes, relationships, ISA hierarchies, etc., which are reused by the *HypertextView* package.
- The *HypertextView* package establishes the overall structure of the WebML hypertexts, in terms of site views, areas, pages, content units, etc., and shows how these artefacts can be assembled and interconnected to constitute a WebML hypertext model.

- Finally, the *PresentationView* package is concerned with how WebML represents pages on the screen. Each WebML page has associated one or more style sheets specifying a different way of presenting its instances on the screen, where style sheets are XML documents obeying the WebML presentation DTDs mentioned in [1].

In the next subsections, the contents of each package are described, until reaching the core WebML concepts: all the entities, relations and constraints which are instantiated within WebML models. We take always advantage of the metamodel to review the essential ideas of WebML.

2.1 The CommonElements package

As shown in Figure 2, the abstract *ModelElement* metaclass is the central element of the abstraction. It represents any WebML modeling element from both the data, hypertext and presentation models. Its abstract sub-metaclasses state the fact that an element of WebML may has a name, an identifier, a comment, a property, a type or a derivation constraint associated to its definition.

WebML data types, they are are represented by the *Type* metaclass. In this way, WebML enumeration types and their corresponding values have been realized by metaclasses *Domain* and *DomainElement*, while WebML primitive types like *Integer* or *String* have been meta-modeled by the *WebMLType* metaclass.

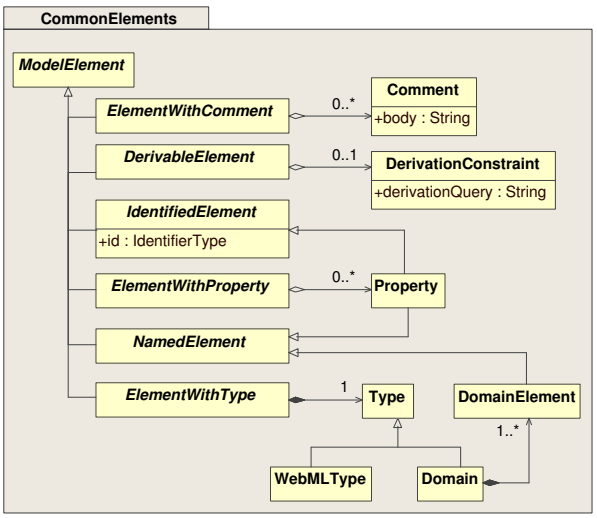

Figure 2: The CommonElements package

2.2 The DataView package

All WebML specifications contain a *DataModel* describing the data structure. Content can be modelled using an Entity-Relationship (E-R) model (or, equivalently, a simplified UML class diagram) comprising *DataModelElements* such as *Entities* and *Relationships*.

Entities are described by means of typed *Attributes* where each entity has at least one attribute namely the OID. The type of an *Attribute* may be either a *WebMLType* or a *Domain* defined by the user.

Entities can be organized in generalization hierarchies, which express the derivation of a specific concept from a more general one. In particular, single inheritance and binary *Relationships* are allowed in the *DataModel*. Each binary relationship is characterized by two *RelationshipRoles* that can be annotated with minimum and maximum cardinality constraints.

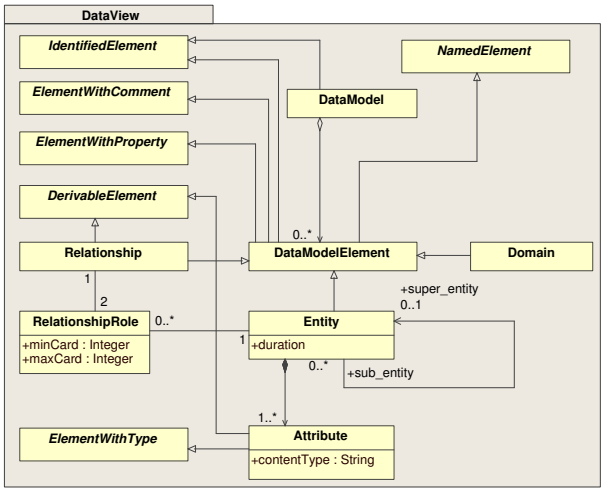

Figure 3: The DataView package

Finally, all *DataModelElements* must be distinguishable by means of a unique identifier (the OID). The MOF WebML metamodel defines this property using a single special purpose metaclass, called *IdentifiedElement*.

Summarizing the previous requirements, Figure 3 represents the ER data schema metamodeled from the WebML viewpoint. We would like to point out that although there are other proposals for metamodeling the ER model, they do not reflect how WebML and WebRatio work. For example, if we metamodeled *RelationshipRoles* as metaatributes of the *Relationship* metaclass, we would miss part of the semantics of the WebML approach.

2.3 The HypertextView package

The *HypertextView* package is further structured in the *Core*, *SiteView*, *ServiceView*, *AreaView*, *PageView*, *OperationUnitView*, *ContentUnitView*, *TransactionView*, *GLParameter*, *ParameterView* and *LinkView* sub-packages (see Figure 4). The former contains the basic core modeling elements for organizing the hypertext structure of Web applications. The latter depends on the *Core* package and it contains further sub-packages for exploiting specific hypertext elements.

Due to space limitations, we provide only a brief overview of the *HypertextView* package. The interested reader can download the complete MagicDraw model from [1].

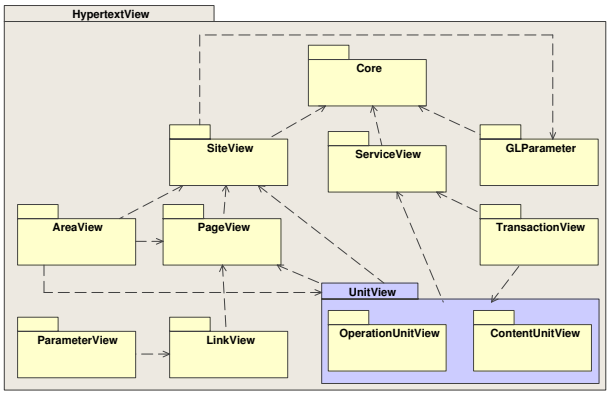

Figure 4: Package Structure of the Hypertext WebML Metamodel

The *SiteView* package allows the definition of different hypertext models targeted to different types of users or to different access devices upon the same *DataModel*. The *SiteView* metaclass of this package represents a collection of *Pages*, *Areas*, *OperationUnits*, *GLParameters* and *Transactions* allowing users to perform a set of activities. *Pages* can be clustered into *Areas*, dealing with a homogeneous subject (e.g., the Amazon Web Store includes a book area, a music area, and so on). A first kind of navigation, which does not depend on page content, can be expressed over *SiteViews*, *Areas* and *Pages*: if a *Page* or *Area* is marked as "landmark", it is assumed to be reachable (through suitable navigation commands) from all the other areas and pages in the same module; if a *Page* or *Area* is marked as "default", it is assumed to be displayed by default when the enclosing module is accessed; if a *Page* is marked as "home", it is displayed by default when accessing the application. These features are metamodeled as metarelationships between the participant metaclasses.

Pages comprise *ContentUnits*, representing atomic components for content publishing: the content displayed in a *Unit* typically comes from an *Entity* of the *DataModel*, and can be determined by means of a *Selector*, which is a logical *Condition* filtering the entity instances to be published. Instances selected for displaying can be sorted according to *OrderingClause*.

In general, *Units* are connected to each other through *Links*, which carry *Parameters* from one *Unit* to another and allow the user to navigate the hypertext. *Links* express the "wiring" of the application. Five kinds of links are defined: *NonContextualLinks* allow only navigation; *Transport* links allow only parameter passing and are not rendered as navigation devices; *Automatic* links are normal links automatically "navigated" by the system on page load; *OK* and *KO* links are output links of operations, followed respectively after execution success or failure.

Apart from *ContentUnits*, WebML comprises *OperationUnits*, defined as components for executing arbitrary business logic. *OperationUnits*, unlike *ContentUnits*, do not publish content and thus are positioned outside pages. Units (*ContentUnits* and *OperationUnits*) may have input and output *Parameters* (e.g., the OID of the object to be displayed or modified, the username and password for authenticating the user, etc.). *Parameter* passing is expressed as a side effect of navigation: *Units* are connected by *Links*, which have a threefold purpose: enabling user navigation, supporting the passage of parameters, and triggering the execution of *Units*.

Additionally, well-formed rules complete the semantics of the metamodel and its elements. For example:

- A page cannot be in a *SiteView* and in an *Area* at the same time.
- If a page, area or siteview is marked as "landmark", "home" or "default", it must be contained in its relevant parent element.
- Only a *Siteview* can be marked as "home".

3. REPRESENTING WEBML IN UML

Early attempts at encoding WebML in UML 1.X did not succeed in producing a completely equivalent profile, due to the lack of appropriate structuring concepts and mechanisms. However, the advent of UML 2.0 is appropriate time to reconsider the design choices of WebML and assess the applicability of UML 2.0 to a DSL that has been successfully proven in the Web Engineering field. Based on the previously introduced metamodel and new UML 2.0 facilities, we have evaluated it as a platform for redesigning WebML seeking two main goals:

1. To provide UML modelers with a UML profile that can help them structure their Web application specifications according to a mature Web Engineering proposal. In this way, UML modelers can reuse their knowledge of standard UML, any tools that support UML (at design level) as well as WebRatio for generating a complete implementation from UML models.
2. To provide WebML modelers with a UML profile for expressing their specifications in a standard way, decoupling the abstract syntax and semantics of the modeling element from its particular representation, hence allowing its use within any standard UML tool environment.

3.1 Running example: The ACME Music Inc. System

Before discussing the WebML representation in UML and how the main concepts of WebML are mapped to UML 2.0 concepts, let us introduce a simplified running example to illustrate of how to use the profile: the product catalogue and content management system of "ACME Music Inc.". It is a simple online shopping cart application for the purchase of music CDs where the customer uses a web browser to interact with the application's interface. When the user starts the application, the home page of this site displays a list of available music CDs and their associated information: title, id number, name of the performing artist, price, number of tracks, reviews, etc. Then, the customer can select CDs for

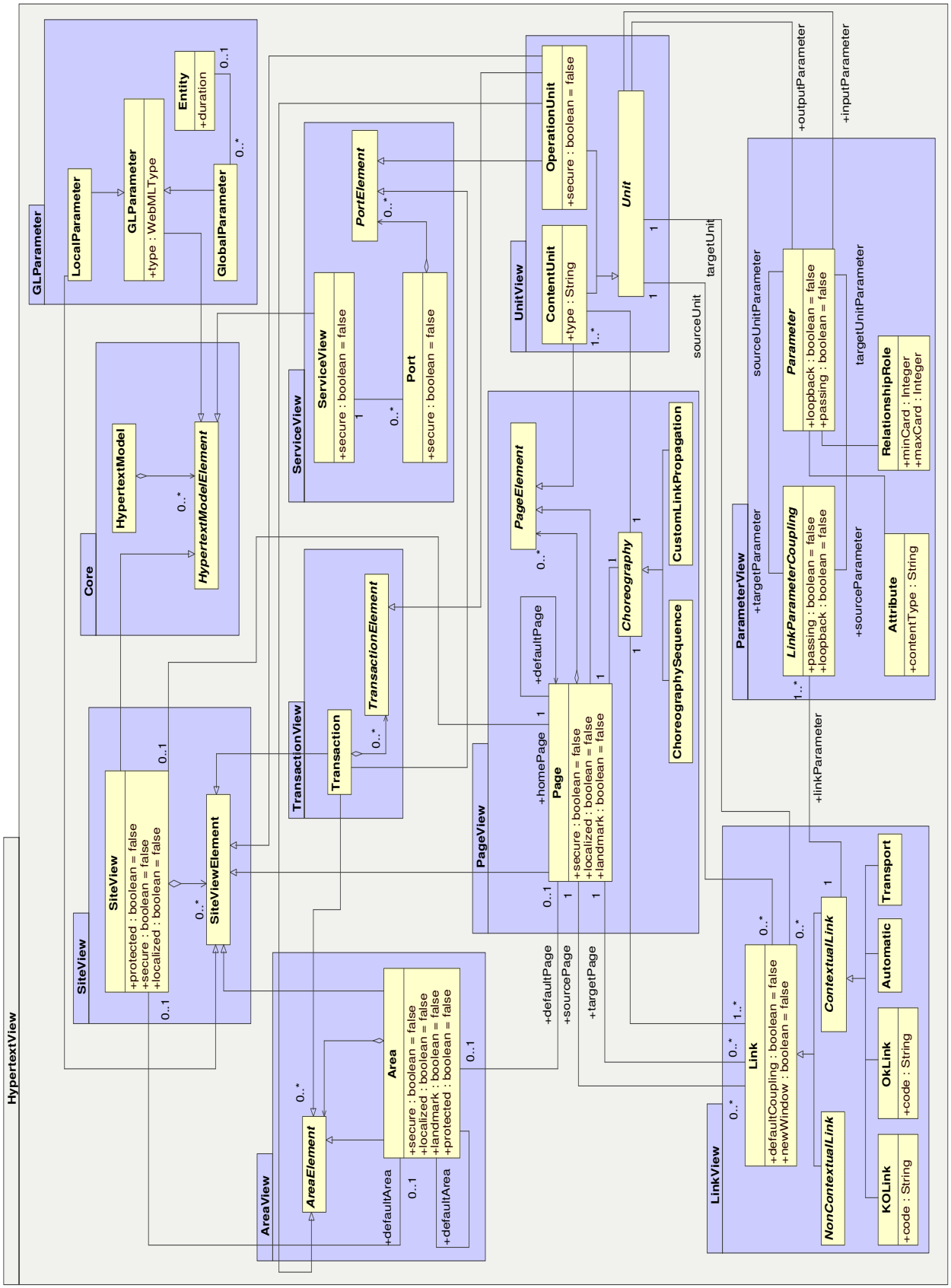

Figure 5: An excerpt of the HypertextView package

purchase by clicking the *"Add"* button associated with the item. In the same way, the customer can remove an item from its shopping cart by clicking the *"Delete"* button associated with the item. User CD selection can be inserted into a shopping cart or into a wish list associated with it. For the items inserted into a shopping cart users may issue an order, for which the payment, shipping method, and address must be specified. A subclass of CDs is defined, to cluster the instances recommended to all users at the first access. At any moment of the user session, the customer can exit the application or return to the order page to start a new shopping session.

3.2 The WebML data model in UML 2.0

As illustrated in Figure 3, the data modeling is supported in WebML by means of the classical ER model with generalization hierarchies, typed attributes, and cardinality constraints. The essential elements of the ER model are entities, defined as containers of structured data, and relationships, representing semantic associations between entities. Since the semantic equivalence between the ER model and its corresponding in UML is very clear, we justify the selection of UML metaclasses in those cases in which it may not seem so intuitive for the reader.

Entities. Following a UML-based approach, each *Entity* of the WebML data model will be mapped to a UML class. In UML, classes are classifiers that have a set of features that characterize their instances. Consequently, each typed *attribute* of the entity will be considered as a typed structural feature, i.e., one of its properties. Its associated type may correspond to a predefined WebML type or a specific type defined by the user. In this last case, we will consider that each WebML *Domain* represents a UML enumeration datatype where the set of possible values are the UML literals of that datatype.

Relationships. Relationships are characterized by cardinality constraints, which impose restrictions on the number of relationship instances an object may take part in. WebML represents N-ary relationships as a combination of entities and binary relationships. Consequently, all relationships in the WebML data model are binary relationships characterized by two relationship roles, each one expressing the function that one of the participating entities plays in the relationship.

At first sight we may consider that WebML *Relationships* have the same semantics as UML associations given that they model connections between entities. However, due to the role that relationships have on both the WebML data model and the hypertext model this would be a poor solution (neither behavioral descriptions nor attributes can be associated with a UML association). Alternatively, UML association classes could be used to represent WebML *Relationships*, which does permit an appropriate semantic definition to be supplied. However, note the reader that WebML *Relationships* may be parameters of a unit, may have derivation constraints sub-setting and/or concatenating existing relationships, their instances can be created, modified or deleted as a consequence of user interactions, etc. Therefore, defining a ≪Relationship≫ stereotype for a Class can make the use of WebML *Relationships* easier.

Derivation Constraints. The value of some of the attributes or relationships of a WebML entity can be determined from the value of some other elements of the schema. The computation rule that defines the derived attribute or relationship is specified as an expression added to the declaration of the attribute or relationship. UML 2.0 derivation mechanisms for attributes and associations can be used to naturally represent WebML derivations.

Applying these correspondences to our running example, Figure 6 shows a WebML data model for the CD store application and its corresponding representation in UML using our UML profile.

3.3 The WebML hypertext model in UML 2.0

The profile must express three essential aspects of the the WebML hypertext model: the modularization structure of site views, the specification of components (content and operation units), and the interconnection of components through links supporting parameter passing. For data-centric content and operation units, it is also important to express how a component draws or updates the content of the data model objects.

Modularization. WebML groups hypertext model elements into site views and areas. The grouping functionality for better understanding and management of models is supported in UML by means of packages. Therefore, we will represent WebML siteviews and areas as UML packages. At a second level of modularization, pages are also containers of sub-pages or of units that can involve multiple individual operations. WebML suggests that the default behavior of a Web page can be defined only once and reused in the rest of the projects using it.

Moving WebML page requirements to a UML approach, we can consider it as an autonomous unit that is replaceable within its environment and which functionality is provided and/or required by means of the combination of provided and/or required funtionalities of the content units nested inside the page. In this case, we find that this concept corresponds to the UML concept of component. However, this is not exactly what represents a page in WebML. Pages in WebML act not only as containers of units but also they represent a composition of interconnected WebML elements, whose instances collaborate at run-time to achieve some common objective. Therefore, this semantics fits much better with UML structured classes. Consequently, pages are more appropriately represented as structured classes, comprising the content units nested inside them.

Visibility Level. Some properties of siteviews, areas and pages, like *home*, *default*, and *landmark* properties, allow the designer to fine-tune the visibility level of these constructs inside the hierarchical structure of a site view. Other properties, such as *secure*, *protected* or *localized* provide information required for code generation. UML stereotypes have been defined to "mark" the appropriate model elements with such properties, allowing the annotation of the corresponding models with these characteristics. Alternatively, we could have considered to represent previous properties as tag definitions of their corresponding stereotypes. From our viewpoint the approach selected improves its practical appli-

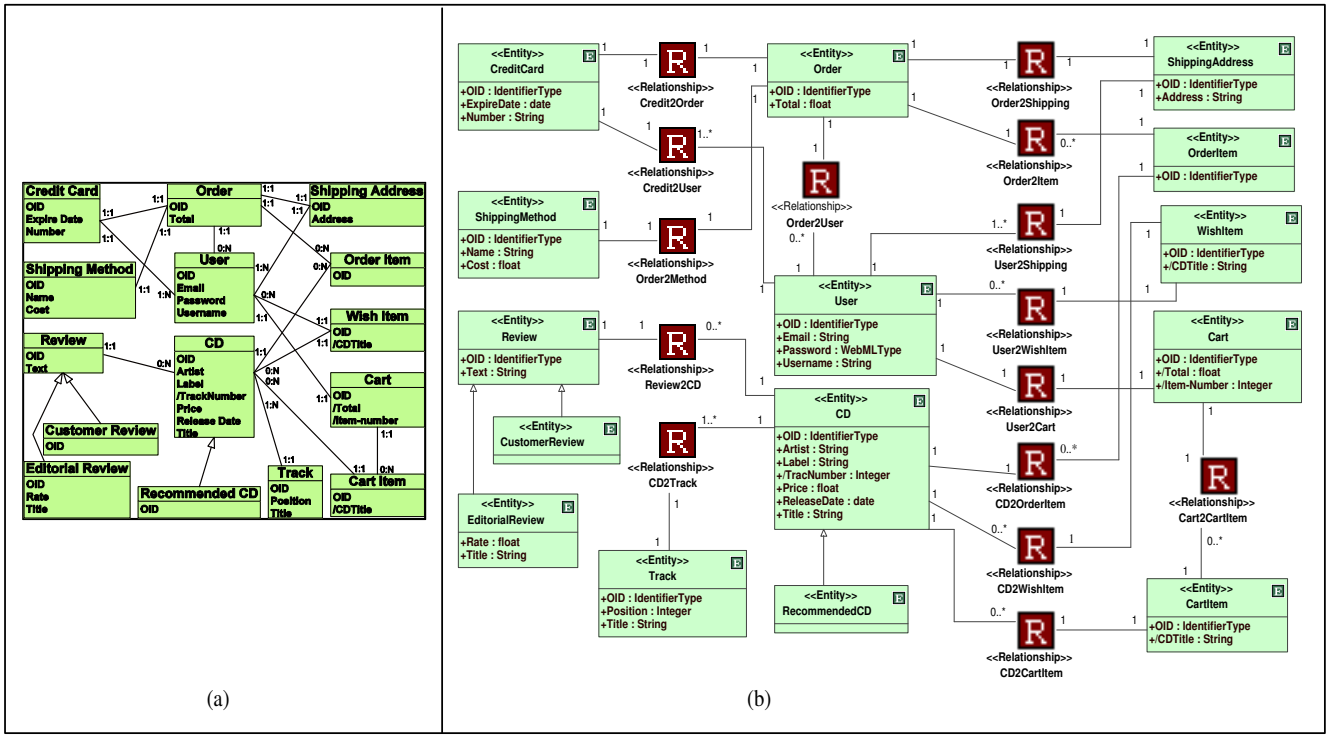

Figure 6: (a) The WebML data model of the CD store application; (b) The UML 2.0 representation equivalent to (a).

cation adding more visual clarity to UML4WebML models as WebML does by means of the (L), (H) and (D) marks.

Content and operation units. Content and operation units are components that can be assembled together to obtain arbitrary complex hypertext pages. Their essence is the capability to execute business actions and interoperate through the exchange of parameters. This notion is most closely reflected by the concept of UML 2.0 *component*. As shown in Table 9, specific stereotypes have been defined for each type of content and units supported by WebML taking as base UML element the component notion.

As a UML component, WebML units may optionally have an internal structure and own a set of ports that formalize their interaction points. Ports can formalize the input and output of a unit: one port allows the environment of the component to supply parameter values; another port allows the the environment of the component to extract parameter values.

Links and parameter passing. Pages and units do not stand alone, but are linked to form a hypertext structure. Links express the possibility of navigating from one point to another in the hypertext, and allow the passage of parameters from one unit to another. Several alternatives have been considering for representing this WebML feature in UML. Links can be visualized as basic UML associations, dependency relationships, traces, usage relationships, etc. While they can all be labeled with suitable stereotypes such as ≪TransportLink≫, ≪NavigationLink≫ or ≪KOLink≫ to denote the classes of links, a more precise semantic definition can be provided by the use of UML assembly connectors. An assembly connector is a UML connector between two components that defines how one component provides the services that another component requires. It is defined from a provided port (i.e., an output WebML parameter) to a required port (i.e., an input WebML parameter) of the units involved.

Realization of data-centric units. Content and operation units may operate on objects specified in the data model. This is represented by specifying the internal realization structure of components, which may exploit auxiliary classes to represent a view over the data model entities. At least, for each content unit there will be two auxiliary classes: one class stereotyped as ≪focus≫ that defines core logic or control behavior of the unit and another class that selects content from the datamodel.

Appropriate OCL invariants in the auxiliary classes represent the selector condition determining the instances upon which they operate, and UML delegation connectors link the input and output ports of the component to the auxiliary classes, granting parameters flow.

Global and local parameters. WebML parameters denote small pieces of information, which can be "recorded" during the user navigation, to be later retrieved and exploited in the computation of the content of some page. A

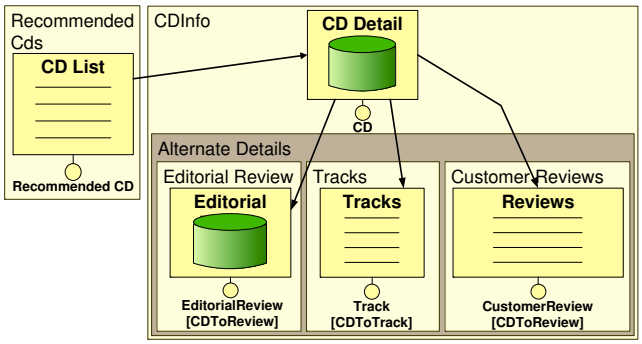

Figure 7: A fragment of the WebML hypertext model of the CD store application.

parameter will be represented as a singleton UML class that contains: a public class-scope (static) property, a class constructor declared as private so that no other object can create a new instance, and finally a class method that returns a reference to the single instance of the class.

Applying these correspondences, Figure 8 shows the UML 2.0 representation of an excerpt of the running example expressed using the WebML profile; this should be compared to the native WebML representation in Figure 7. The *List of CDs* page, represented as a structured classifier, contains an index unit component, linked by an assembly connector to the ≪AutomaticLink≫ stereotype. The internal structure of the index unit is realized by a ≪focus≫ class, comprising methods for sorting the index instances and for selecting one instance. That focus class is connected by a one-to-many composition association to class *RecomCDView1*, which represents a view over the data model entity *RecommendedCD*. Instances of class *RecomCDView1* contain the *Title* attribute, which is necessary to build the index, and the hidden attribute *OID*, needed for parameter passing. A delegation connector links the output port of the focus class to the outport port of the component, and specifies that the output value of the *select()* method is emitted by the output port of the index unit. The parameter associated with the ≪AutomaticLink≫ connector is received at the input port of the data unit component, which delegates its treatment to an inner focus class. That focus class contains an instance of class *CDView1*, which represents another view on the data model entity *CD*. An OCL invariant in the focus class enforces the contained instance of class *CDView1* to have the value of the OID attribute equal to the parameter value received at the input port.

Three alternative structured classifiers (*EditorialReview*, *Tracks* and *Reviews*) representing Web pages, offer more details about the CD selected by the user: the editorial review represented by the ≪DataUnit≫ EditorialDetails, all CD tracks represented by the ≪IndexUnit≫ TracksDetails and customer reviews represented by the ≪IndexUnit≫ ReviewsDetails. Each classifier is realized by a focus class containing the instances of a class *view* over the data model. The relationship between the *views* an the original entities in the data model are represented by means of UML dependency relationships where the attributes of the *view* classes (*EditorialView1*, *TrackView1* and *ReviewView1*) are derived attributes in UML. Ports, delegation and ≪AutomaticLink≫ connectors, and OCL constrains have been defined as above to guarantee that pages show only the information related to the CD selected.

The fact that most WebML concepts can be represented by UML 2.0 concepts without changing their original semantics (maybe imposing some additional constraints on them, at most) enables the use of a UML profile as an appropriate mechanism for our purposes. In summary, Table 9 shows the most important stereotypes defined in the UML profile for the WebML proposal. Finally, it is important to notice that the use of profiles to develop applications does not have any effect on the development process supporting the original DSL. In our case, the straightforward application of the profile to develop a Web application is based on the same steps being followed to develop a Web application using WebML: (1) Create a class diagram (the Data WebML model) describing the conceptual data organization; (2) Extend the previous model to enable the specification of calculated data; (3) Create a composite structure diagram (the Hypertext WebML model) describing the front-end of the application; (4) Extend the hypertext model with simple constructs such as operations and transactions, enabling the invocation of predefined operations (e.g., insertion, deletion and modification of objects) and the integration of external services.

4. DESIGN PRINCIPLES REQUIRED FOR BUILDING A UML-BASED DSL

Based on our experience defining and using the WebML profile, we agree with [7] that at least four criteria need to be considered for building a UML-based DSL: 1) *Semantic match* between the UML constructs and the features of the domain; 2) *Visual clarity*, i.e., the capability to avoid visual clutter and highlight key details; 3) *Completeness*, i.e., the ability to capture all relevant features; and 4) *Tool support*.

Semantic match. Regarding the *semantic match*, very few works address the validation of the semantics equivalence between UML profiles and the original DSLs from which they arise. From our point of view, it is necessary to bridge the gap between these UML profiles and their equivalent DSLs, in such a way that it is possible to automatically convert DSL models into their corresponding UML diagrams and vice-versa. We have addressed this issue by means of a set of XSLT rules formalizing the correspondences between the mapped models and comparing how implementations generated starting from both approaches are equivalent.

Visual clarity. WebML native representations are far more compact than their UML counterpart. This is obviously unavoidable, when comparing a general purpose language like UML and a DSL. However, the disproportion shows that there is ample room for making UML and OCL expressions more concise and suitable.

Completeness. The definition of a UML 2.0 profile equivalente to WebML demostrates in a practical way the suitability of UML for encoding the requirements and generating the complete code of complex Web applications.

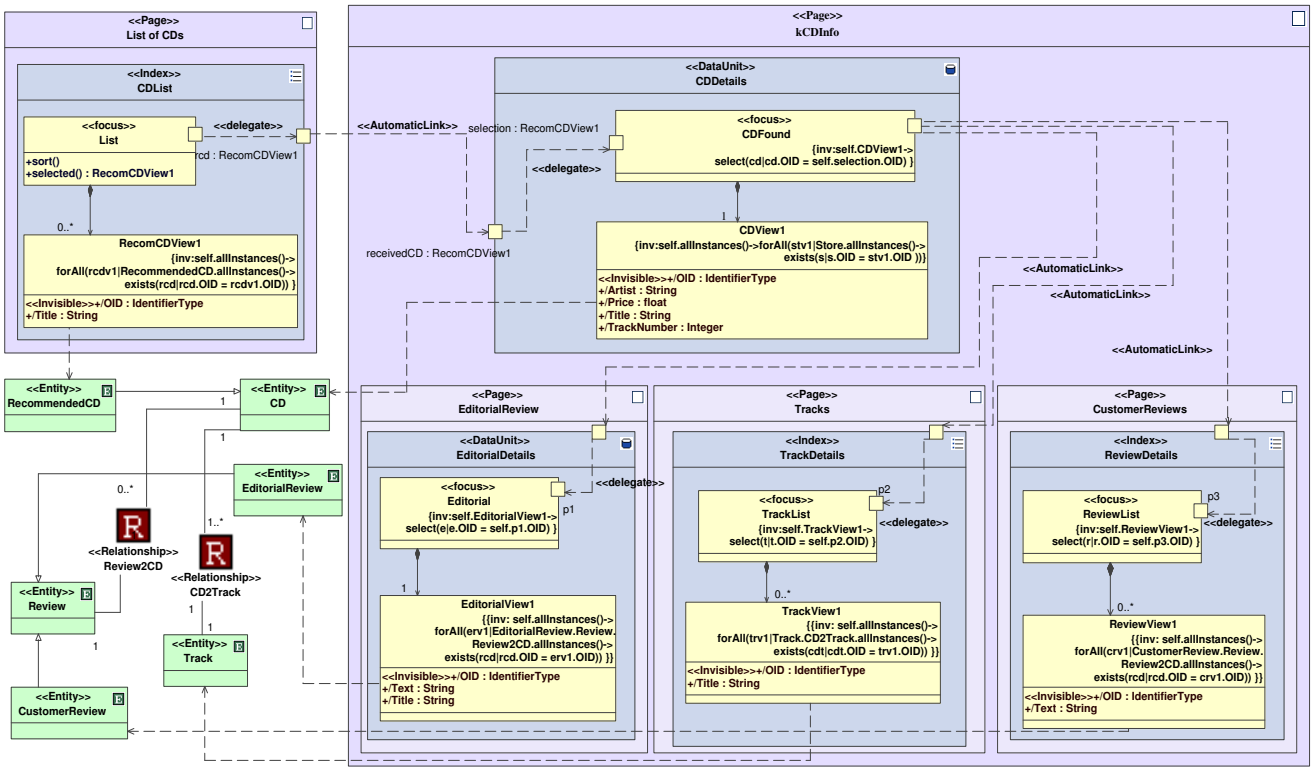

Figure 8: The UML 2.0 representation equivalent to the hypertext model of Figure 7.

Tool support. In addition, *tool support* is interesting not only during the design stage, but also to assist during the code generation and deployment phases. Although many UML-based tools successfully achieve code generation in other domains such as realtime embedded systems, this is still to be proven in the case of Web applications, where the interplay of dynamic behavior and adaptation with user interfaces and business logics is more intricate.

However, there are many different problems that we have found in representing WebML with UML that need to be addressed in order to deal with UML 2.0 and OCL in the context of Web applications like:

- *Complexity,* the number of concepts that need to be used to "completely" model and implement a web application is not that big (on the contrary, it is rather small). UML contains too many concepts and mechanisms, a fact which becomes a drawback more than an advantage. In this sense, we have shown how UML Profiles provide the right mechanisms for tailoring UML to a specific domain, identifying only the concepts that are required for that domain.

- *Required skills,* using UML requires a certain degree of specialized knowledge about the UML concepts and mechanisms, which are far from being intuitive in many cases (e.g., what is a port, a component, a structured class, a collaboration, etc.);

- *Semantics.* The semantics of the UML elements and models are sometimes rather loose. In our approach, we decided that, in case of variation points of a UML concept representing an WebML concept, the semantics of the WebML concept should be used.

- *Behavior modeling.* Modeling the structural aspects of an application is fine, and can be easily done with UML because the structuring models and diagrams of UML have a clear semantics, and can be easily customized to fulfill the structural requirements of most application domains. However, something which is not completely solved in UML is how to represent (executable) behavior. One of the lessons we have learnt is that the behavioral semantics can be defined in an implicit and natural way as part of the domain-specific language concepts. In this way, all the data-centric content and operation units have a well-defined behavior: they are simply the basic CRUD (Content Read, Update, Delete) operations familiar to any data designer, and thus it would make no sense to represent their behavior in a high-level diagram. The same is true for user-defined components: their semantics are known to the developer and need not be explicitly expressed when specifying their *usage* in a hypertext model.

5. CONCLUSIONS AND FUTURE WORK

In this paper we have presented a UML profile for representing WebML models. The definition of a UML 2.0 profile equivalent to WebML demonstrates in a practical way the

suitability of UML for encoding most requirements of complex Web applications.

Our future work will concentrate on completing the UML profile of WebML with presentation and architecture features, and to adapt the code generation process currently implemented by WebRatio as a development strategy based on MDA. In this way, it will be possible to eliminate the strong dependencies between the WebML models and the final implementation technologies (currently Java and JSP). In addition the existence of the WebML metamodel will allow us to tackle other objectives such as formal reasoning on WebML models in order to validate the event occurrence and properties on them.

6. ACKNOWLEDGMENTS

This work has been partial supported by Spanish Research Project TIN2005-09405-02-01 and the Italian grant FAR N. 4412/ICT.

7. REFERENCES

[1] WebML resources. http://www.webml.org/webml/page5.do.

[2] L. Baresi, F. Garzotto, and M. Maritati. W2000 as a MOF Metamodel. In *Proc. of World Multiconf. on Systemics*, volume 1, 2002.

[3] S. Ceri, P. Fraternali, M. Brambilla, A. Bongio, S. Comai, and M. Matera. *Designing Data-Intensive Web Applications*. Morgan Kaufmann, 2002.

[4] J. Conallen. *Building Web applications with UML, 2nd ed*. Addison Wesley, 2002.

[5] I. Garrigós, J. Gómez, and C. Cachero. Modelling Dynamic Personalization in Web Applications. *Proc. of ICWE'03*, LNCS 2722:472–475, 2003. Spain.

[6] J. Gómez and C. Cachero. *Information Modeling for Internet Applications*, chapter OO-H Method: Extending UML to Model Web Interfaces, pages 144–173. Idea Group Publishing, Hershey, PA, USA, 2003.

[7] J. Ivers, P. Clements, D. Garlan, R. Nord, B. Schmerl, and J. R. O. Silva. Documenting Component and Connector Views with UML 2.0. Technical Report CMU/SEI-2004-TR-008, Carnegie Mellon Univ., 2004.

[8] N. Koch and A. Kraus. Towards a common metamodel for the development of Web applications. *Proc. of ICWE'03*, LNCS 2722:495–506, 2003. Oviedo, Spain.

[9] A. Kraus and N. Koch. A Metamodel for UWE. Technical Report 0301, Ludwig-Maximilians-Universität München, January 2003.

[10] Object Management Group. *UML 2.0 Superstructure Specification*, 2005. http://www.omg.org/cgi-bin/apps/doc?formal/05-07-04.pdf.

APPENDIX

The following figure shows the main elements of the profile for WebML.

WebMLConcept	UML Base Element	Stereotype
DataModel	Model	<<DataModel>>
Domain	Enumeration	<<Domain>>
DomainElement	EnumerationLiteral	<<DomainElement>>
Entity	Class	<<Entity>>
Attribute	Property	<<Attribute>>
Relationship	Class	<<Relationship>>
RelationRole	Property	<<RelationRole>>
HypertextModel	Model	<<HypertextModel >>
SiteView	Package	<<SiteView >>
ServiceView	Component	<<ServiceView >>
Port	Port	None
GlobalParameter	Class	<<GlobalParameter >>
Area	Package	<<Area>>
Page	StructuredClasiffier	<<Page>>
HomePage	StructuredClasiffier	<<HomePage>>
Default	StructuredClasiffier , Package, Component	<<Default>>
LandMark	StructuredClasiffier , Package, Component	<<LandMark >>
Secure	StructuredClasiffier, Package, Component	<<Secure>>
Localized	StructuredClasiffier, Package, Component	<<Localized>>
Protected	StructuredClasiffier , Package, Component	<<Protected>>
DataUnit	Component	<<DataUnit>>
MultiDataUnit	Component	<<MultiDataUnit>>
IndexUnit	Component	<<IndexUnit>>
MultiChoiceIndexUnit	Component	<<MultiChoiceIndexUnit >>
HierarchicalUnit	Component	<<HierarchicalUnit >>
Level	Class	<<Level>>
Index	Component	<<Index>>
EntryUnit	Component	<<EntryUnit>>
CustomContentUnit	Component	<<CustomContentUnit >>
ValidationRule	Constaint	<<ValidationRule >>
CreateUnit	Component	<<CreateUnit>>
DeleteUnit	Component	<<DeleteUnit>>
ModifyUnit	Component	<<ModifyUnit >>
ConnetUnit	Component	<<ConnetUnit>>
DisconnetUnit	Component	<<DisconnetUnit >>
LoginUnit	Component	<<LoginUnit>>
LogoutUnit	Component	<<LogoutUnit >>
ChangeGroupUnit	Component	<<ChangeGroupUnit >>
SendEmailUnit	Component	<<SendEmailUnit>>
AdapterUnit	Component	<<AdapterUnit >>
SetUnit	Component	<<SetUnit>>
OKLink	Connector	<<OKLink>>
KOLink	Connector	<<KOLink>>
AutomaticLink	Connector	<<AutomaticLink >>
TransportLink	Connector	<<TransportLink >>
LinkParameterCoupling	Port	<<LinkParameterCoupling >>
Parameter	Port	<<Parameter>>
Condition	Constraint	<<Condition>>
SelectorCondition	Constraint	<<SelectorCondition >>
Property	Property	<<Property>>

Figure 9: The main elements of the UML Profile for WebML.