WebML modelling in UML

N. Moreno, P. Fraternali and A. Vallecillo

Abstract: In recent years, we have witnessed how the Web Engineering community has started using the standard unified modelling language (UML) notation, techniques and supporting tools for modelling Web systems, which has led to the adaptation to UML of several existing modelling languages, notations and development processes. This interest for being MOF and UML-compliant arises from the increasing need to interoperate with other notations and tools, and to exchange data and models, thus facilitating reuse. WebML, like any other domain-specific language, allows one 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 be integrated with other model-based tools. As a solution to these requirements, a UML 2.0 profile for WebML which allows WebML models to be used in conjunction with other notations and modelling tools has been described. The paper also evaluates UML 2.0 as a platform for Web modelling and identifies some key requirements for making this version of the standard more usable.

1 Introduction

The UML [1] is a general purpose visual modelling language for specifying, constructing and documenting systems, which can be used in 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 effectiveness in modelling some specific domains (e.g. the hypertext interfaces of Web applications), for which specialised languages and tools, such as for example WebML/WebRatio [2], OO-H/VisualWade [3] and so on may be more appropriate. These domain-specific languages (DSLs) tend to support higher-level abstractions than general-purpose modelling languages, and are closer to the problem domain than to the implementation domain. Thus, a DSL allows modellers to perceive themselves as working directly with domain concepts. Furthermore, the rules of the domain can be included into the language as constraints, enabling model-based checking of illegal or incorrect specifications.

Despite the potential benefits of using specific languages, in the short term we certainly see a growing interest in Meta-Object Facility (MOF), UML and UML profiles for modelling Web applications. This interest for being MOF and UML-compliant arises from the increasing need to interoperate with other notations and tools, and to exchange data and models, thus facilitating and improving reuse. And although some DSLs (and their supporting environments) might be probably more suitable for modelling specific aspects of Web applications than pure UML, they have not reached the level of acceptance and diffusion of UML modelling tools. As a consequence, developers must buy and use ‘yet-another’ modelling tool (with additional learning costs and efforts) if they want to take advantage of their favourite DSL. Even worse, proprietary DSL-based tools often do not interoperate with other tools, which fragments the workspace into a a set of isolated development environments.

Because of these driving forces, the Web Engineering community has started using standard UML notation, techniques and supporting tools for modelling Web systems, which has led to the adaptation of several DSLs, representation diagrams and development processes to UML. Several authors have proposed their approach to the definition of metamodels and UML extensions, as illustrated, for example [4–7]. Especially UML profiles have proved to be a simple and effective mechanism for representing proprietary Web Engineering modelling languages.

We provide a contribution to the effort of expressing state-of-the-practice Web Engineering modelling languages in UML, by introducing a UML 2.0 profile for WebML, which allows the WebML language and its development process (supported by the WebRatio tool) to be smoothly integrated with standard UML development environments. In addition, we also describe a metamodel for WebML, which will allow its integration with other MDA tools as soon as they are available (editors, validators, metric evaluators etc.) and also with other model-driven approaches and tools (e.g. those using KM3-or Ecore-based metamodels).

The remainder of the paper is organised as follows. Section 2 introduces a running example used throughout the paper, and Section 3 briefly presents WebML and WebRatio. Then, Sections 4 and 5 define the WebML metamodel and WebML profile, respectively. Section 6 compares the WebML and UML representations, reviews the main design choices taken in the definition of WebML and discusses how they have been impacted by the use of UML for encoding WebML, and summarises some of the lessons learnt in the process of mapping WebML to UML.

doi:10.1049/iet-sen:20060067

Paper first received 15th November 2006 and in revised form 10th May 2007

N. Moreno and A. Vallecillo are with the Department of Lenguajes y Ciencias de la Computación, University of Málaga, Spain

P. Fraternali is with the Dipartimento di Elettronica e Informazione Politecnico di Milano, Milano, Italy

E-mail: vergara@lcc.uma.es

Finally, Section 7 examines related work, whereas Section 8 draws some conclusions and outlines future works.

2 Running example

Before illustrating WebML and discussing its representation in UML, we introduce a simplified running example: the product catalogue and content management system of small furniture manufacturing company.

The catalogue is a Web application offered to the general public that advertises various types of furniture, for example chairs, tables, lamps and so on, and also special combinations of items sold at a discounted price. Individual items are described by an image, a textual description, their price, a set of technical features (dimensions, available colours etc.), and larger images. Combinations are illustrated by a photograph, advertising text and the discounted price.

Information about the store locations is also made available, including the address and contact information, an image of the store and a map. Users are expected to visit the catalogue home page containing the offer and the product of the day. From there, they can access the details of the advertised product and offer, or enter dedicated areas for navigating the catalogue, using various search and browsing facilities.

From the page describing a combination, the individual items forming the combination can be accessed, and conversely, it is possible to navigate from an item to the combinations including it. From the home page, the list of stores can also be reached.

The employees of the company can login from the home page into a protected Web application, which enables managing the content of all the objects displayed in the public site.

3 WebML and WebRatio in brief

WebML [2] describes Web applications at three levels: the content objects, the front-end organisation and the look and feel. Content objects are specified using a simplified entity-relationship (E-R)data model (or, equivalently, a simplified UML class diagram), comprising classes, entities, attributes, single inheritance, binary relationships and data derivation expressions. Fig. 1 shows the data model of the running case, with the objects mentioned in the requirements.

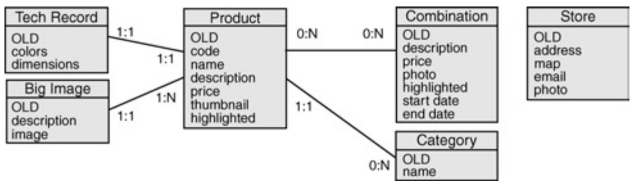

Fig. 1 *Data model of the running case*

The front-end is specified using the hypertext model, which has a hierarchical organisation: each application corresponds to a site view, internally structured into areas, which in turn may contain sub-areas and pages. Multiple site views can be associated to the same data model, for example to represent applications delivered to different actors or designed for different access devices (web, digital television, mobile phones, PDAs). Site views, areas and pages are not only units of modularisation, but also govern access, which can be selectively granted to each individual module based on the user's identity and subscription to groups. A first kind of navigation, which does not depend on page content, can be expressed over site views, areas and pages: if a page or area is marked as 'landmark' (L), 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' (D), it is assumed to be displayed by default when the enclosing module is accessed; if a page is marked as 'home' (H), it is displayed by default when accessing the application. Fig. 2 shows the hypertext model of the public site view of the running case, with the areas and pages mentioned in the requirements, annotated with the navigation markers.

Pages are the basic interface containers: they can be structured in sub-pages and comprise content units. A content unit is defined as a component that publishes some content in a page; the published content can be extracted dynamically from the objects specified in the data model or specified statically in the hypertext model (e.g. an entry form consisting of multiple input fields). In addition to content units, WebML comprises operation units, defined as components for executing arbitrary business logic. Operation units, unlike content units, do not publish content and thus are positioned outside pages. Components (content and operation units) may have input and output parameters (e.g. the OID attribute of the object to display or to modify, the username and password

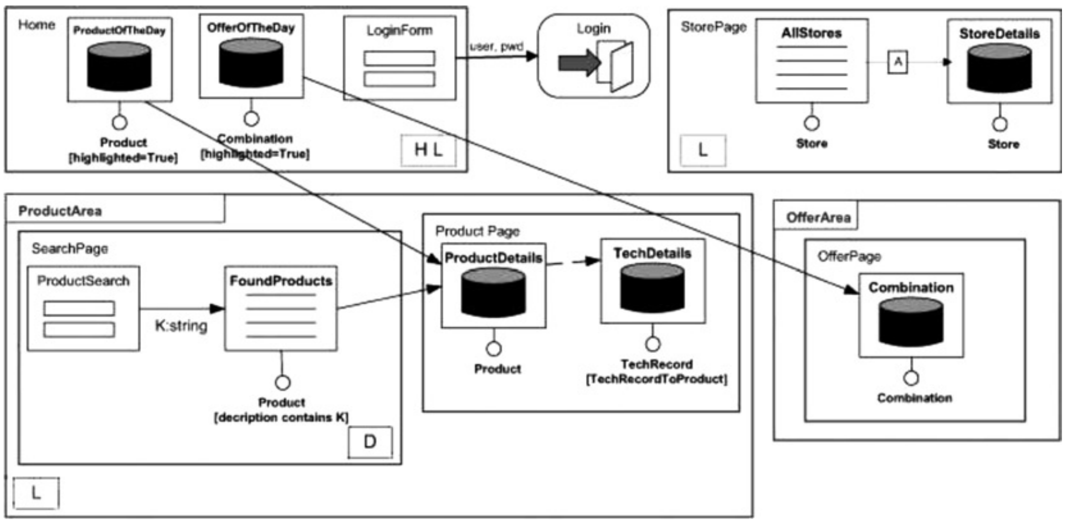

Fig. 2 *Hypertext model of the public site view*

for authenticating the user etc.). Parameter passing is expressed as a side effect of navigation: components are connected by links, which have a threefold purpose: enabling the user's navigation, supporting the passage of parameters and triggering the execution of components. Therefore a WebML hypertext can essentially be described as a graph of parametric components, connected by links, in which some components publish content and are contained within pages, some other components perform business actions, and are triggered from links emanating from pages. Links express the 'wiring' of the application. Five kinds of link are defined: normal links, denoted by solid arrows, allow both navigation and parameter passing; transport links, denoted by dashed arrows, allow only parameter passing and are not rendered as navigation devices; automatic links, denoted by the symbol [A], are normal links automatically 'navigated' by the system on page load; OK and KO links are output links of operations, respectively, followed after execution success or failure. In Fig. 2, the home page contains two data-driven content units (for displaying the product and offer of the day) and one static content unit (an entry form); the entry form is connected to a login operation (placed outside the home page), for authenticating the employees before accessing the protected content management site view. The outgoing link of the entry unit shows the parameters transferred to the login operation units. Application development with WebML is supported by WebRatio, a tool enabling the editing of data and hypertext models and the generation of the application source code. WebRatio also supports all the technical development tasks: declaring the data sources and Web services used in the project, forward and reverse engineering of the data model, presentation specification through page mockups, and code generation and deployment for different platforms.

4 Metamodel for WebML

A metamodel is the 'perfect' way to model a dynamically evolving notation and maintain it in a homogeneous and comprehensive way [5]. There are different metamodelling languages and notations, among which the combination of MOF and the object constraint language (OCL) constitutes the Object Management Group (OMG) proposal for specifying metamodels.

Mapping a DSL like WebML to MOF involves representing each element of the domain – its syntax and semantics – as a MOF artefact and performing a refactoring process in order to introduce further MOF-elements such as compositions/aggregations or abstract classes, which preserve the original semantics while allowing a better grouping of concepts with similar attributes.

This task is relatively simple when there is a formal description of the language syntax and grammar (although the MOF metamodel generated must be augmented with several OCL constraints to cover the semantical features and structural rules of the language). In the WebML case, a formal Backus-Naur Form (BNF) grammar is available only for its derivation language (WebML-Object Query Language, OQL).

A possible starting point for deriving the metamodel could be the set of document type definition (DTDs) employed by the WebRatio development tool for storing WebML projects (see [8]), as done by other authors (cf. [9]). However, the WebRatio DTDs are low level because they comprise many auxiliary concepts, necessary to the tool but redundant with respect to the formal definition of the language. Therefore we decided to build the MOF

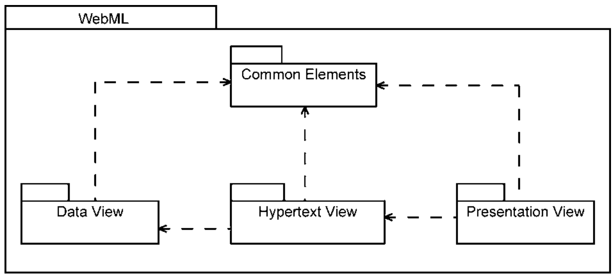

Fig. 3 *WebML metamodel packages*

metamodel from scratch, starting from the definition of the basic concepts of the language. Such a metamodel is described in the remainder of this section.

4.1 WebML metamodel structure

We have represented the WebML concept space by means of four metamodel packages, as illustrated in Fig. 3: CommonElemets, DataView, HypertextView and PresentationView. Complying with a package requires to comply with its abstract syntax, well-formed rules, semantics and notation.

- The CommonElements package comprises core concepts used when metamodelling 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 generalisation relationships.
- The DataView package addresses WebML data model concepts, such as entities, attributes, relationships, ISA hierarchies and so on, 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 and so on, 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 [8].

In the next subsections, each package is described, and all the entities, relations and constraints instantiated within the WebML models are progressively introduced.

4.2 CommonElements package

As shown in Fig. 4, the abstract ModelElement metaclass is the central element of the abstraction. It represents any WebML modelling element from both the data, hypertext and presentation models. Its abstract sub-metaclasses represent the fact that any WebML element may have a name, an identifier, a comment, a property, a type or a derivation constraint associated to its definition.

WebML data types are represented by the Type metaclass. In this way, WebML enumeration types and their corresponding values have been realised by metaclasses Domain and DomainElement, whereas WebML primitive types such as Integer or String have been meta-modelled by the WebMLType metaclass.

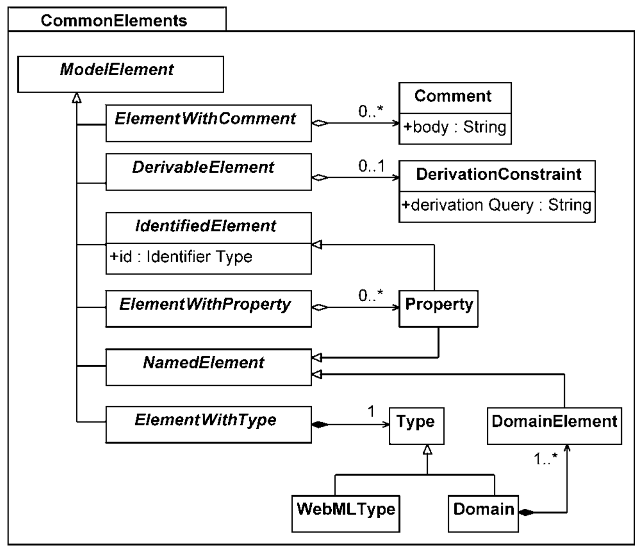

Fig. 4 *CommonElements package*

4.3 DataView package

All WebML specifications contain a DataModel describing the data structures used to represent the information (content) handled by the application. Content can be modelled using an 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 organised in generalisation 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 characterised by two RelationshipRoles that can be annotated with minimum and maximum cardinality constraints.

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.

Fig. 5 represents the E-R data schema metamodelled from the WebML viewpoint. We would like to point out that although there are other proposals for metamodelling the E-R model, they do not reflect precisely the usage of the data model in WebML and WebRatio. For example, if we metamodelled RelationshipRoles as meta-attributes of the Relationship metaclass, we would miss part of the semantics of WebML.

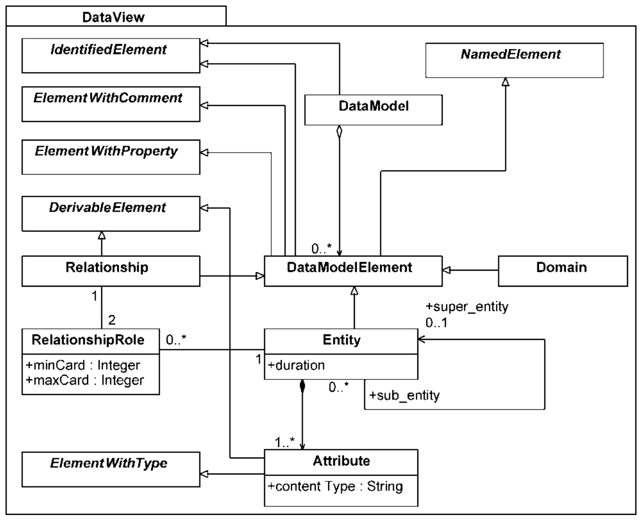

Fig. 5 *DataView package*

4.4 HypertextView package

The HypertextView package is sub-structured into the Core, SiteView, ServiceView, AreaView, PageView, OperationUnitView, ContentUnitView, TransactionView, GLParameter, ParameterView and LinkView sub-packages (see Fig. 6). The former contains the basic core modelling elements for organising the hypertext structure of Web applications. The latter depends on the Core package and it contains further sub-packages for exploiting specific hypertext elements.

Because of space limitations, we provide only a brief overview of the HypertextView package (see Fig. 7). The interested reader can download the complete MagicDraw model from [8].

The SiteView metaclass of this package represents a collection of SiteViewElements allowing users to perform a set of activities. From the WebML viewpoint, a SiteViewElement can be a Page, an Area, an OperationUnit, a GLParameter or a Transaction.

Pages comprise PageElements, an abstract metaclass representing other Pages and ContentUnits. Each Page has associated a Choreography that describes the chain of WebML operations or actions to be performed as a consequence of the invocation of the Units. This Choreography can be both a predefined sequence represented in the metamodel by the ChoreographySequence metaclass and a custom sequence established by the designer and metamodelled by the CustomLinkPropagation metaclass.

Pages can be clustered into Areas, dealing with a homogeneous subject (e.g. the Amazon Web Store includes a book area, a music area etc.). Features related to how SiteViews, Areas and Pages can be reachable from other SiteViews, Areas or Pages have been metamodelled as metarelationships between the participant metaclasses.

Units can be classified into two metaclasses: ContentUnits and OperationUnits. The content displayed in a ContentUnit 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 OrderingClauses. OperationUnits, unlike ContentUnits, do not publish content and may have input and output

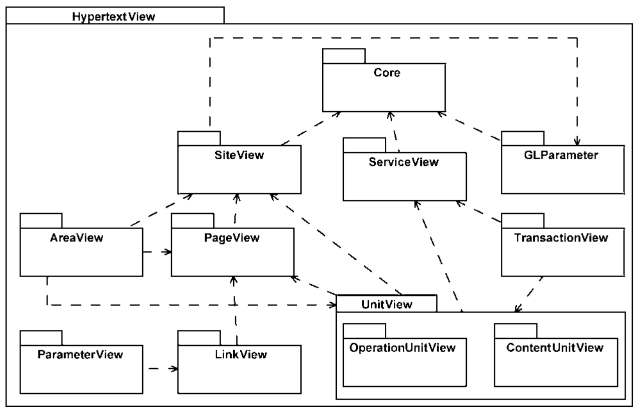

Fig. 6 *Package Structure of the Hypertext WebML Metamodel*

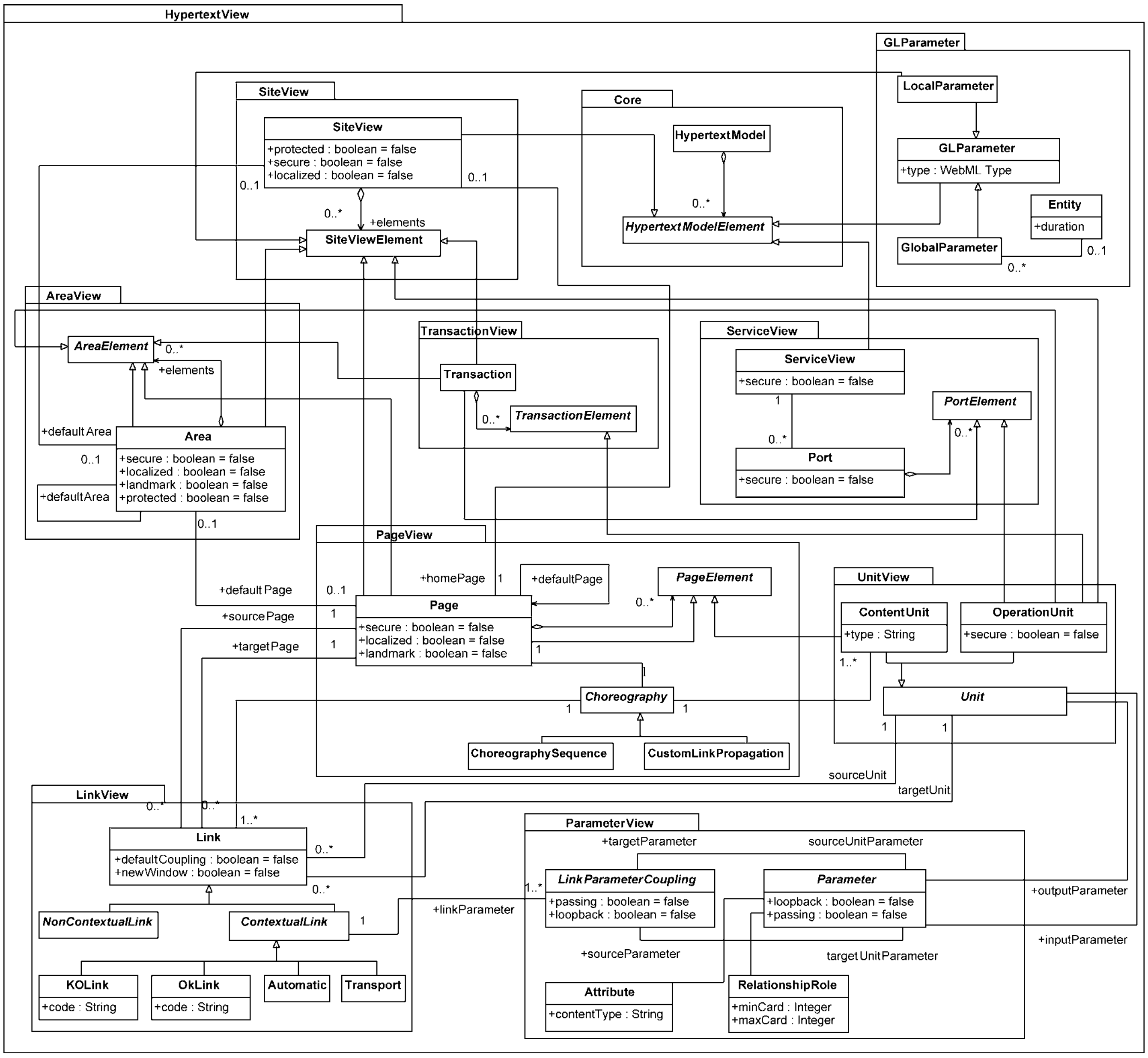

Fig. 7 *Excerpt of the HypertextView package*

Parameters. The link parameter coupling defines the flow of parameters from the source unit to the destination unit of the link, which output parameter of the source unit provides a value to which input parameter of the destination unit.

Units are connected to each other through Links. For each kind of link (NonContextualLinks, Transport, Automatic, OK and KO) identified in Section 3 a MOF metaclass has been declared inheriting from the Non-ContextualLink and ContextualLink metaclasses, respectively. While the NonContextualLink metaclass captures the semantics and behaviour of WebML links allowing only the navigation, the abstract ContextualLink metaclass represents those others allowing also the parameter passing.

WebML OperationUnits can be grouped into Transactions describing sequences of operations that are executed atomically, as they were one unique big operation. In the WebML, this concept is represented by the Transaction metaclass, inside which the TransactionElements to be executed transactionally can be placed.

The context information needed to calculate units that is not transferred point-to-point during navigation, but must be available ‘globally’ to all the pages of a site is metamodelled by the GlobalParameter metaclass. LocalParameters are similar to GlobalParameter, but are locally to a specific SiteViewElement.

Additionally, well-formed rules complete the semantics of the metamodel and its elements. We show some of them but the interested reader can visit the WebML site [8] for a complete description of the metamodel constraints.

– A page cannot be nested in a SiteView
– and in an Area at the same time.
context Page **inv**:
  self.**isOclKind**(AreaElement)
  xor self.**isOclKind**(SiteViewElement)

– Any page marked as ‘landmark’ must be
– nested in a SiteView or in an Area.
context Page **inv** :
  self.landmark **implies**
    self.siteView- > **notEmpty**()
    or self.area- > **notEmpty**()

5 Representing WebML using a UML profile

Early attempts at encoding WebML in UML 1.X did not succeed in producing a completely equivalent profile, because of the lack of appropriate structuring concepts and mechanisms of UML, and the semantic gap between specific Web application concepts and the UML constructs.

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 modelling the structure and behaviour of Web applications. In addition, UML 2.0 provides enhanced profiling capabilities, which allow the precise definition of MOF-based domain-specific languages.

The OMG offers three approaches for defining a DSL. The first solution is to develop a metamodel, as we have done in the previous section. This means creating a new domain language alternative to UML, using the MOF metamodelling facilities. In this way, the syntax and semantics of the elements of the new language are defined to faithfully match the domain's specific characteristics. The problem with metamodel definition is that standard UML tools are not able to deal with the new language (e.g. to edit models that conforms to the metamodel, compile them etc.). This approach has been followed by languages such as CWM (common warehouse metamodel) and W2000, because some of their constructs do not match the semantics of the corresponding UML model elements.

The second and third options provided by the OMG are based on extending UML. Extensions of the UML can be either heavyweight or lightweight. The difference between lightweight and heavyweight extensions comes from the way in which they extend the UML metamodel. Heavyweight extensions are based on a modified UML metamodel with the implication that the original semantics of modelling elements are changed and therefore the extension might no longer be compatible with UML tools. Lightweight extensions are called UML profiles and exploit the native extension mechanisms of UML (stereotypes, tag definitions and constraints) for specialising its metaclasses, without breaking their original semantics. UML profiles may impose new restrictions on the extended metaclasses, but they should respect the UML metamodel, without modifying the original semantics of the UML elements (i.e. the basic features of UML classes, associations, properties etc. will remain the same, only new constraints can be added to the original elements). Syntactic sugar can also be defined in a profile, in terms of icons and symbols for the newly defined elements. One of the major benefits of profiles is that they can be handled in a natural way by UML tools. In UML profiles, stereotypes define particularisations of given UML elements, adding them some semantics. Perhaps the best known example of profileable UML-based Web modelling language is UWE [10].

In this section, we describe the UML profile for WebML. In building the profile, we have pursued two main goals:

1. To provide UML modellers with a UML profile that can help them structure their Web application specifications according to a mature Web Engineering proposal. In this way, UML modellers can reuse their knowledge of standard UML, exploit any tool that supports UML for editing the specifications and export the application model to WebRatio for generating a complete implementation.
2. To provide WebML modellers with a UML profile for expressing their specifications in a standard way, decoupling the abstract syntax and semantics of the modelling element from its specific representation, hence allowing its use within any standard UML tool environment.

Table 1 shows the main elements of the profile for WebML that we are going to describe in the next subsections.

5.1 WebML data model in UML 2.0

As illustrated in Fig. 5, the data modelling is supported in WebML by means of the classical E-R model with generalisation hierarchies, typed attributes and cardinality constraints. The essential elements of the E-R model are entities, defined as containers of structured data, and relationships, representing semantic associations between entities. Since the semantic equivalence between the E-R 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.

5.1.1 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 characterise their instances. Consequently, each typed attribute of the entity will be considered as a typed structural feature, that is, 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.

5.1.2 Relationships: Relationships are characterised 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 characterised 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, because of to the role that relationships have on both the WebML data model and the hypertext model this would be a poor solution (neither behavioural 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 subsetting and/or concatenating existing relationships, their instances can be created, modified or deleted as a consequence of user interactions and so on. Therefore defining a ≪Relationship≫ stereotype for a Class can make the use of WebML Relationships easier.

5.1.3 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, Fig. 8 shows a WebML data model for the Acme application and its corresponding representation in UML using our UML profile.

Table 1: Subset of the UML Profile for WebML

WebML Concept	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	Package, Component, Struct·Clasiffier	≪Default≫
Landmark	Package, Component, Struct·Clasiffier	≪Landmark≫
Secure	Package, Component, Struct·Clasiffier	≪Secure≫
Localized	Package, Component, Struct·Clasiffier	≪Localized≫
Protected	Package, Component, Struct·Clasiffier	≪Protected≫
DataUnit	Component	≪DataUnit≫
MultiDataUnit	Component	≪MultiDataUnit≫
IndexUnit	Component	≪IndexUnit≫
MultiChoiceIndexUnit	Component	≪MultiChoiceIndexUnit≫
HierarchicalUnit	Component	≪HierarchicalUnit≫
Level	Class, Association	≪Level≫
EntryUnit	Component	≪EntryUnit≫
CustomContentUnit	Component	≪CustomContentUnit≫
ValidationRule	Constraint	≪ValidationRule≫
CreateUnit	Component	≪CreateUnit≫
DeleteUnit	Component	≪DeleteUnit≫
ModifyUnit	Component	≪ModifyUnit≫
ConnectUnit	Component	≪ConnectUnit≫
DisconnectUnit	Component	≪DisconnectUnit≫
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, Association	≪AutomaticNavigation≫
TransportLink	Connector, Association	≪Transport≫
LinkParameterCoupling	Port	≪LinkParCoupling≫
Parameter	Port	≪UnitParameter≫
Condition	Constraint	≪Condition≫
SelectorCondition	Constraint	≪SelectorCondition≫
Property	Property	≪WebMLProperty≫

5.2 WebML hypertext model in UML 2.0

The profile must express three essential aspects of the the WebML hypertext model: the modularisation 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.

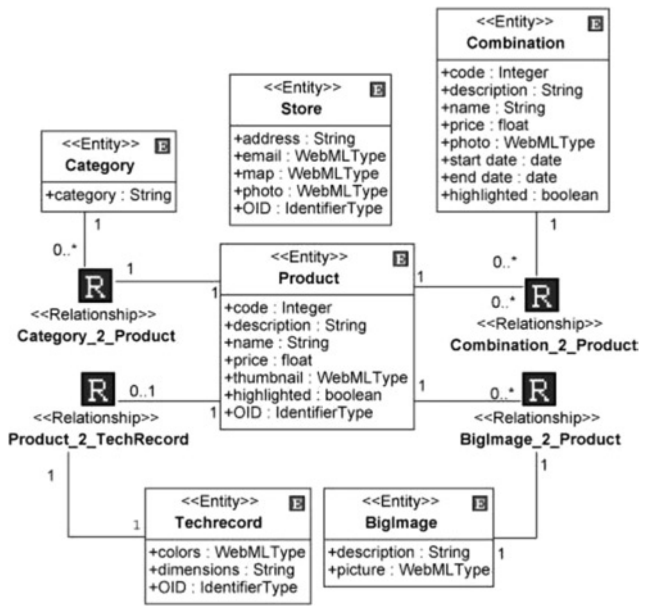

Fig. 8 *UML 2.0 data model of the Acme application*

5.2.1 Modularisation: 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 site views and areas as UML packages. At a second level of modularisation, pages are also containers of sub-pages or of units that can involve multiple individual operations. WebML suggests that the default behaviour 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.

5.2.2 Visibility level: Some properties of site views, areas and pages, such as 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 application adding more visual clarity to UML4WebML models as WebML does by means of the (L), (H) and (D) marks.

5.2.3 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 of executing 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 1, 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 formalise their interaction points. Ports can formalise 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.

5.2.4 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 visualised as basic UML associations, dependency relationships, traces, usage relationships and so on. Although they can all be labelled 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.

5.2.5 Realisation of data-centric units: Content and operation units may operate on objects specified in the data model. This is represented by specifying the internal realisation 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 behaviour 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.

5.2.6 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 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.

The above-mentioned mappings are pictorially illustrated in Fig. 9, which represents the UML specification equivalent to the WebML page of Fig. 10.

The StorePage, represented as a classifier, contains an index unit component and a data unit component, linked by an assembly connector with the ≪AutomaticLink≫

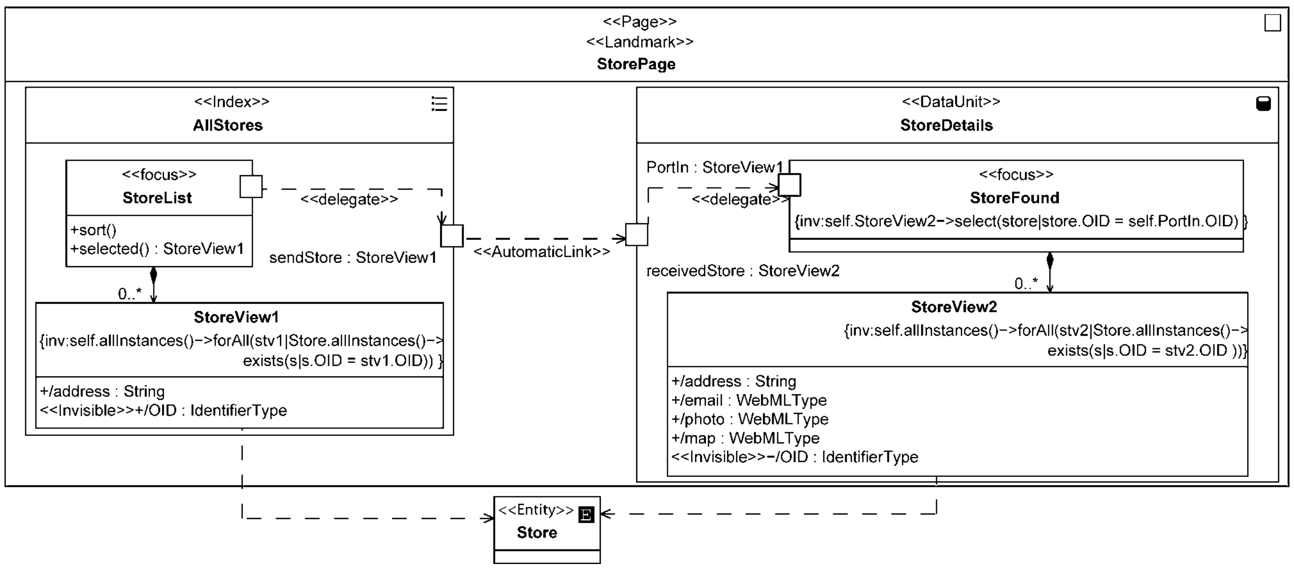

Fig. 9 *UML 2.0 representation equivalent to the page of Fig. 10*

stereotype. The internal structure of the index unit is realised by a focus class, comprising methods for sorting the index instances and for selecting one instance. The focus class is connected by a one-to-many part of association to class StoreView1, which represents a view over the data model entity Store. Instances of class StoreView1 contain the address attribute, necessary to build the index and the hidden attribute OID, necessary 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 index unit component's output port. 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. The focus class contains on instance of class StoreView2, which represents another view over the data model entity Store. An OCL invariant in the focus class enforces the contained instance of class StoreView2 to have the value of the OID attribute equal to the parameter value received at the input port.

As a conclusive example, Fig. 11 overviews the UML 2.0 representation of the public site view of the running case expressed with the WebML profile; this should be compared to the native WebML representation of Fig. 2.

6 Comparison of the WebML and UML representations

This section compares the WebML representation in UML against its native encoding. We first compare in Section 6.1 how compact the two representations are, using the running application as an example. Then, Section 6.2

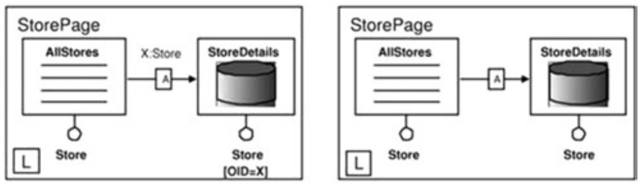

Fig. 10 *Page with an index unit for selecting a store and a data unit for displaying its details (left). Same page with abbreviated notation (right)*

reviews the main design choices taken in the definition of the WebML model and tool. Finally, Section 6.3 discusses how these design principles have been impacted by the use of UML for encoding WebML, and summarises some of the lessons learnt.

6.1 Compactness and complexity comparison

The proposed encoding of WebML as a UML 2.0 profile has been validated and used to compare the complexity of the two approaches.

Validation was performed by implementing an Extensible Stylesheet Language Transformation (XSLT) translator from the XMI representation [Note: In the experiment, we used the XMI output of MagicDraw version 10.5.] of the model encoded in the UML profile into the native WebML format (which has its own XML Schema) and then: (1) verifying the correctness of the synthesised WebML specifications by means of the model checker of WebRatio; (2) generating the Java2EE code of the application from the the synthesised WebML specifications and comparing the obtained code with the one produced by using native WebML. Both assessments proved the equivalence of the UML 2.0 profile and the native WebML representation.

Next, the complexity of the UML representation was compared to that of native WebML, with respect to multiple dimensions. Table 2 shows the result of the comparison. [Note: The editing time has been estimated by assigning fixed time in seconds to the base editing actions: insertion of a node, arc and editing of a property.]

6.2 Design of WebML

The original principles adopted in the design of WebML put special emphasis on three aspects, which were considered prominent for achieving developers' acceptance: (1) expressive power: the model should be capable of expressing Web applications comparable in complexity to industrial-strength systems developed manually; (2) ease of use: the model should be easy-to-learn for developers not skilled in software engineering; (3) implementability: the model should encompass enough information to permit the generation of code for all the tiers of a dynamic Web application. Code generation should produce optimised code as far as

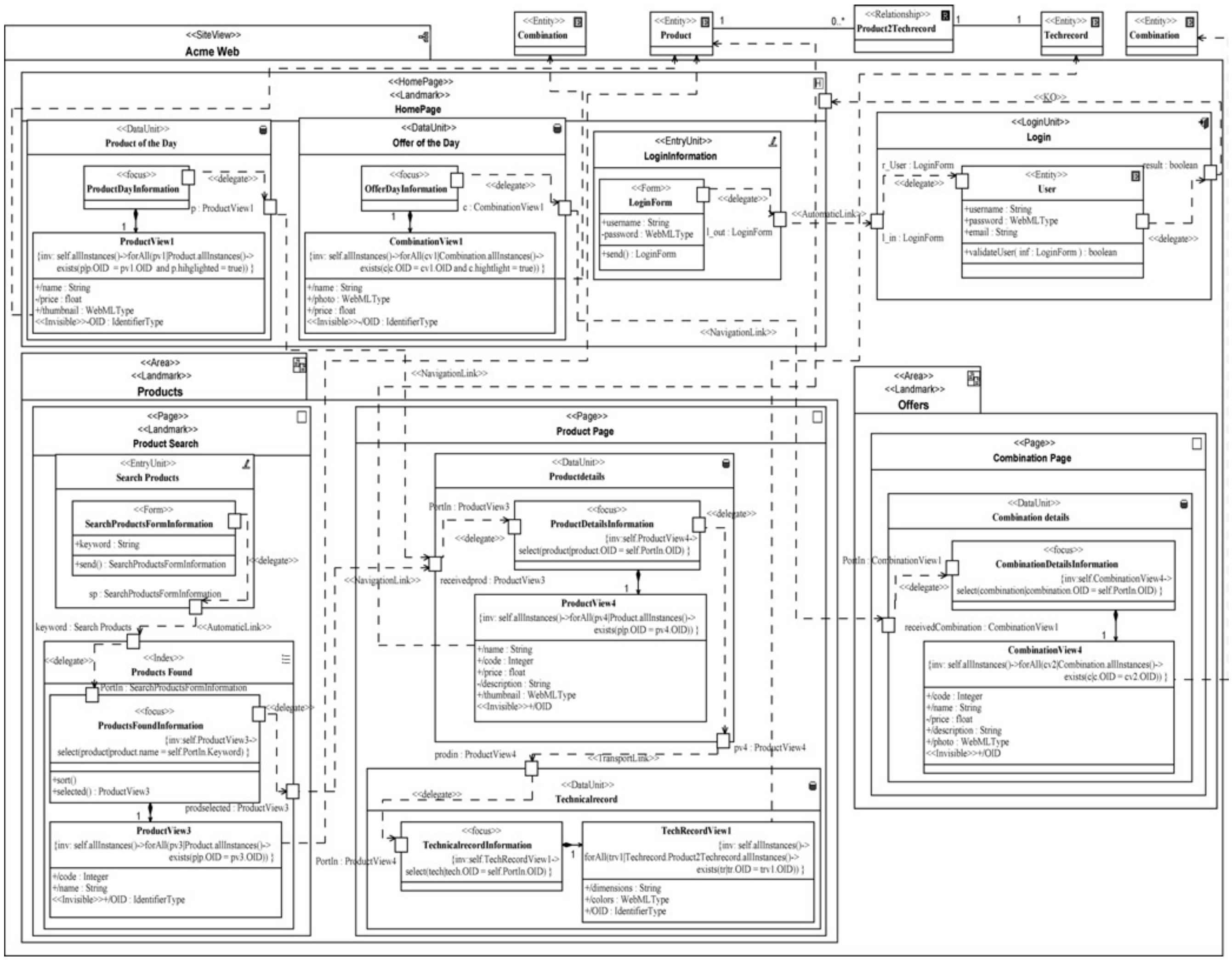

Fig. 11 *Hypertext model represented with the WebML UML profile*

possible. In the following paragraphs, we comment on the essential choices taken in the design of the language.

6.2.1 Model boundaries: The approach to define model boundaries has been rather crude: the model contains the minimal number of concepts necessary to generate code. This led to the partition of the concepts into the three perspectives of the data model, hypertext model and presentation; the latter perspective gathers all those aspects that have to do with aesthetics and interface usability and is not expressed diagrammatically, but by means of annotated examples.

A notable feature of WebML is the absence of a separate sub-model for the business logic. Server-side business logic is partly encoded in the data model (in the form of declarative specifications of derived data) and partly in the hypertext model (in the form of black-box content and operation units, which can represent components with arbitrary functionality. This choice simplified dramatically the model, to the price of two (not necessarily negative) unforeseen phenomena: (1) the proliferation of custom units for encoding non-standard server-side business logic; (2) the complexity growth of the presentation model, which was used to capture, beside aesthetics, also client-side behaviour (e.g. JavaScript event handling), not expressible otherwise.

This restriction of the model boundaries reduced the model concepts necessary for building real-world dynamic web sites. For instance, the initial version of the Acer-Euro web applications was specified and automatically built using only simplified E-R diagrams and 14 WebML units (six content units: index, data, multidata, scroller, entry, get; and eight operation units: create, delete, modify, connect, disconnect, set, login, logout). Clearly, not all applications can be represented by the original WebML components; in time, a wealth of *ad hoc* components has been developed to manage further requirements (messaging, web service interaction, sophisticated input etc.), but they have been treated as black-box plug-ins to the three existing sub-models, rather than constituents of an independent modelling layer. As a further simplification,

Table 2: Comparison of the running case model in native WebML and in the UML WebML profile

Parameter	Native WebML	UML 2.0 WebML Profile
number of modelling concepts	10	9
number of instances of the modelling concepts	23	90
number of constraints on model elements	6	10
size of XML file of the model	22 kbytes	505 kbytes
estimated editing time	345 s	1350 s

no explicit architectural model is provided. Physical resources (data sources, application servers, web services, identity repositories etc.) are not modelled diagrammatically, but simply declared as project resources in the development tool, and referred to by the model concepts that use them.

Another essential aspect of model definition is the compact representation and efficient management of inter-model references, which are ubiquitous: the hypertext diagram references the data model (e.g. for data extraction and update), and the presentation refers to the hypertext model (e.g. for content positioning in pages). Fig. 10 shows a reference from the hypertext model to the data model: the AllStores index unit and the StoreDetails data unit refer to entity Store, which provides the components' content. The reference to the data model element is simply realised by means of a typed property in the hypertext model.

6.2.2 Level of abstraction: A WebML specification is far more abstract than an object-oriented representation of the generated code. A model concept is normally mapped onto multiple software arteifacts, possibly residing at different tiers. For example, a WebML page, its unit and links are indeed a compact representation of multiple arteifacts: (1) the data extraction code in the data tier (stored procedures, queries, Enterprise JavaBeans (EJB) finder methods etc.); (2) the object(s) storing the content in the business and/or presentation tier (entity EJB, javabeans); (3) the action class decoupling the request from the business tier objects; (4) the business service orchestrating the computation of the page content; (5) the JavaServer Pages (JSP) executable tags translating object content into page markup.

The decision of keeping a high level of abstraction traded realism for compactness. To the developer's eye the WebML model of a page mixes things that in reality stand apart. However, if the ultimate purpose of modelling is to generate the code, the ease of building up a complete model should prevail on the realism of the representation, provided that the meaning of each abstract concept is known and the behaviour of element composition predictable. Considering that a small-size project may typically consist of tens of pages comprising hundreds of components, it is easy to see that raising the level of abstraction, while preserving semantics, is crucial to achieve practical usability.

6.2.3 Behavioural semantics: A non-obvious aspect of model design, which distinguishes WebML from traditional OO modelling, is the absence of a separate behavioural model. Behavioural aspects are embedded in the hypertext model, which has a fixed operational semantics that applies to all WebML applications, and therefore need not be specified explicitly by the designer. In essence, a hypertext model is represented by an extended finite state machine [11], in which events capture user's interactions and system generated events, and transitions describe the propagation of computation from one component to another one. Conditions on transitions express the flow of control, for example the order in which components must be executed depending on the availability of user's input or of system-provided parameters. The provision of a standard, application-independent operational semantics is a cornerstone of the design of WebML: the 'in the large' semantics describes the general functioning of the Web application as a network of cooperating components; the 'in the small' semantics describes how individual components work. The former is standardised and developers need not address it; the latter is treated as a plug-in to the model: developers are requested to understand how the predefined WebML components work and define their own components by respecting a few basic rules requested by the 'in the large' semantics (essentially, each component should declare its input and output parameters and optional default rules). This approach proved quite effective. The cases in which the default 'in the large' semantics proved inadequate for capturing application requirements are very limited (we estimate less 1% of the cases) and we coped with these situations in the design tool, where the developer that understand the computation semantics can alter the components' execution sequence. Another lesson we learned is that standardising the model semantics does not reduce the expressive power. The range of implemented applications demonstrates this claim; developers were always able to partition requirements into what can be achieved using standard components and behaviour and what requires the construction of *ad hoc* components. This approach resembles the notion of framework at the coding level, raised to a more abstract level.

6.2.4 Ergonomics: The success of a model-driven method is tightly connected with its practical usability. If editing the model is more cumbersome than implementing functionality in the code, developers will hardly switch to model-driven development (MDD). The evolution of WebML is characterised by innumerable revisions aimed at optimising the performance of modelling, whose common principle is 'everything that can be reasonably inferred from the context must not be specified explicitly'.

A notable example occurs with parametric components and parameter passing along links. Such a feature is the backbone of hypertext modelling and occurs repeatedly in any application model. To alleviate the developer's task, WebML provides default rules whereby 80% of parameter passing is inferred from the context. The notation on the right part of Fig. 10 exemplifies this idea. The AllStores index unit has a single output parameter (`X`) of type (Store), the entity underlying the component; the StoreDetails data unit has a parametric selection condition (i.e. a query) determining the object to display (`[OID=X]`), which makes use of the parameter associated with the input link: in this way, the data unit shows exactly the object chosen in the index unit. By inference, the model in the left-hand side of Fig. 10 is equivalent to that in the right-hand side of Fig. 10, in which the parametric selection condition and the parameter passing on the link are omitted. This simple default rule spares two editing actions per link, which results into a substantial reduction of the model editing effort.

6.2.5 Openness: The last far-reaching design choice concerned the evolution of the model. Being a DSL, WebML was initially conceived as a closed language, comprising all the primitives deemed necessary for building Web applications. This choice soon proved ineffective: the extension of application functionality constantly demanded for language revisions. A major breakthrough was achieved with the definition of a standard model extension mechanism, which transformed WebML into an open component-assembly language. The semantics was revised so that developers could define their own components, by wrapping any existing piece of software, and mix them freely with the built-in WebML units. This required a simple standardisation of the component external behaviour to make it comply to the (very minimal) requirements of the standard component orchestration semantics; each content and operation unit must simply declare its input and output interfaces, and the rules for inferring parameter passing and content selection conditions. After such revision, the standard component orchestration semantics, coupled to libraries of domain-specific components, proved adequate to the modelling and automatic implementation of applications in the most disparate domains: mobile, digital television, Web service interaction, workflow management and so on.

6.3 Lessons learned on UML as a Web modelling language

The definition of an UML 2.0 profile equivalent to WebML demonstrates practically the suitability of UML for encoding the requirements and generating the complete code of complex real-world Web applications. However, several issues were encountered in representing WebML with UML 2.0, both of general nature and specific to the encoding of WebML.

6.3.1 Expressive power: The UML profile for WebML provides the same expressiveness as the native representation: all WebML concepts were mapped into the corresponding UML (stereotyped) elements and the resulting UML profile proved sufficient to describe complete Web applications that could be successfully processed by the WebRatio model checking and code generation engine. This result, which was not achieved in the previous attempts at encoding WebML in UML 1.X, has been granted by the new features of UML 2.0. For example, ports and connectors as defined in UML 2.0 have provided the necessary expressive power to represent the interconnection of components and the associated flow of parameters, which is essential to hypertext specification. Furthermore, OCL invariants, involving ports, realisation and view classes, can express the semantics of data-centric Web components in quite a natural and compact way. The notion of connector, although criticised elsewhere [12] for its lack of behavioural semantics, proved sufficient for expressing the linking of Web components, where parameters are simply transferred from one component to the other.

6.3.2 Complexity: Only a limited number of UML concepts proved necessary to model and implement a WebML application. UML seems to contain far more concepts and mechanisms than needed for Web modelling. The main advantage of the UML profiling mechanism has not been found in the extension of the UML metamodel (which is already too large and complex to be used in full), but in the possibility of 'restricting' the set of UML elements that are relevant to a given domain, tailoring their semantics so as to capture the behaviour and well-formedness rules of the domain-specific concepts they represent. In this respect, the complexity of UML is greatly alleviated by the use of a well-designed profile.

6.3.3 Required skills: Using UML properly requires deep knowledge of specialised concepts (e.g. ports, connectors, structured classes, collaborations etc.). These concepts are mostly alien to the Web engineer, who thinks in terms of Web-specific entities (Web pages, links, forms etc.). The use of stereotypes, possibly represented with icons that exploit visual objects familiar to the domain expert, lowers the skill-level necessary to make practical use of UML in implementing Web applications, because it helps bridging the gap between the domain concepts and the UML constructs and mechanisms.

6.3.4 Behaviour modelling: Representing behaviour in UML, and generally in MDD, is much more complex than representing structure. UML structural diagrams have a clear semantics and can easily be customised to fit most domains. At present, UML dynamic models (collaboration, sequence and activity) are more frequently used as a documentation for the human reader than as a specification usable for code generation. Other models (statecharts) are more formal and naturally suited for code generation, but are hard to master for practitioners.

There is a current lack of a concrete syntax for actions in UML, which hinders the specification of executable code. The progress of tools in the field of executable UML and action semantics [13] may change this situation, by allowing the direct execution or compilation into code of arbitrarily sophisticated behavioural models. This approach, which demonstrated effective in such application domains like real-time embedded systems, is still be proven in the Web domain, where the interplay of dynamic behaviour with user interfaces and business logics is more intricate.

In this sense, the approach taken by WebML of not representing explicitly the behaviour of their elements (as discussed in Section 6.2) has proved its benefits, exploiting the domain-specific meaning of business components without loss of expressive power. For instance, all the data-centric content and operation units have a well-defined behaviour: they are just the basic CRUD (content read, update, delete) operations familiar to any data designer, and thus it would make no sense representing their behaviour in a high-level diagram; their transformation into platform-dependent code can easily be delegated to a tool. The same is true for user-defined components: their semantics is known to the developer and need not be expressed when specifying their usage in a hypertext model. They can be modeled and coded elsewhere and treated as a plug-in in the Web design model.

6.3.5 Usability: WebML native representation is far more compact than its UML counterpart (see Section 6.1). This is obviously unavoidable, when comparing a general purpose language such as UML and a DSL that has been refined for years in search of compactness and notational economy. However, the disproportion shows that there is ample room for making UML more concise. One of the areas of possible improvement is in the cross-references between models. One aspect in which UML 2.0 proved rather cumbersome is the specification of the internal realisation of set-oriented content units (e.g. index units). Here, three classes (the component realisation class, its focus class and a data view class) were necessary to express the trivial concept of 'select-project' view (a subset of the objects of a class, possibly with a restricted set of attributes). Clever mechanisms for abbreviating the expression of view relationships between classifiers in different models could dramatically reduce the complexity of diagrams. Another area where usage proves rather cumbersome in the specification of parametric conditions, a fundamental issue in component-based applications. Suitable abbreviations and a better scope mechanism for OCL variables within structured classifiers could greatly simplify the task.

7 Related work

Several authors in the Web Engineering community have exploited metamodelling and UML extensions in defining their DSLs and Web models [4–7, 9, 14]. The pioneering work of Conallen [4] uses, in conjunction with the standard

UML diagrams, the Web application extensions (WAE), which comprise the typical Web arteifacts (server pages, client pages, forms, frames, etc.). The main advantage of WAE proposal is that its simplicity allows an efficient reflect the design artefacts of a Web Interface, as well as to drive the implementation of such interface. On the other hand, this same simplicity evidences a lack of elements for modelling navigation and representing technology and device independent requirements. Basically, WAE is a too low-level proposal for capturing the complexity a high level of abstraction of the navigation and presentation viewpoints.

In order to solve these limitations, Góomez and Cachero proposed in [7] a set of new views that extended UML diagrams to provide a Web interface model. Built up departing from a UML-compliant use-case and a business class diagram, the navigation and presentation models are described using a mix of UML and proprietary primitives, not compatible with current UML tools.

Also similar in goals to our work, Koch [6] and Baresi [5] independently define a conservative extension of the UML 1.4 metamodel, aimed at allowing the easy definition of profiles. The shortcomings of using UML 1.4 make it difficult to use the resulting profiles for modelling complex distributed Web systems. However, these works show that 'although alternative Web models use different notations, they could be based on a common metamodel, which could help the unification of modelling constructs and allow easier comparison and integration' [6].

UML 1.4 has been criticised also for its scarce support to represent architectural styles. In the Web domain, architecture specification is elaborated in [14], where a UML 2.0 profile based on the new composite structure model is exploited to specify the architecture of Web applications. As in our work, [14] uses UML components and connectors for Web modelling. However, the authors exploit these constructors to express the architectural view of the system, which adds up to the functional view embodied in the structure and navigation diagrams. The result is a proliferation of models and stereotypes, which can be difficult to integrate into a well-structured code generation process.

Generally, both the definition of metamodels and UML profiles are very laborious tasks made by hand by an expert on the corresponding DSL. There are not many works that address the automatic generation of metamodels or profiles from DSLs descriptions. In this regard, we find very interesting the proposal of [9] which present a semiautomatic approach that allows to generate MOF-based metamodels from DTDs. Also very interesting results can be found in [15]. The author addresses the validation of the semantics equivalence between UML profiles and the original DSLs from which they arise (e.g. by means of rules formalising the correspondences between the mapped models). Bezivin described a tool that can bridge an UML profile and an equivalent DSL, whereby it is possible to produce automatically the UML profile from the DSL, and vice versa. Such tool could automatically convert WebML models into their corresponding UML diagrams, following the mapping rules described.

8 Conclusion and future work

We have presented both a metamodel and a UML profile for WebML, with the aim of integrating WebML into the MDA approach – hence allowing UML modellers to make use of a Web Engineering mature proposal for designing and developing Web applications, and also allowing WebML modellers to use their notation and tool (WebRatio) within a UML modelling environment.

This work clearly shows that UML 2.0 has the potential to be used for industrial-strength Web development, especially if some simplifications on model cross-references and declarative semantics specifications are introduced. The standardisation of Web development platforms and technologies, rather than making MDD/MDA less attractive [16], multiplies its potential, making the long-envisioned benefits of software engineering more concretely attainable. The definition of an UML profile alleviates the issues of model complexity, required skills and semantics, because a profile helps tailoring the subset of UML used to model a specific application domain, makes UML elements and models closer to the language of the domain expert, and helps customising the 'loose' semantics of UML to better fit the meaning of domain concepts.

Our future work will concentrate on completing the UML profile of WebML with presentation and architecture features, and complementing the code generation process currently implemented by WebRatio with a novel approach closer to the declarative model-transformation paradigm advocated by MDA. In this way, it will be possible to eliminate any dependency between the WebML models and the final implementation technologies (currently Java 2 Enterprise Edition), and realise more flexible code generators capable of exploiting a declarative specification of the target architecture. In addition, the WebML metamodel will allow us to perform formal reasoning and transformations on WebML models for solving a broader classes of problems, related to other phases of the development process, for example, the estimation of the size and cost of a project from its conceptual model and the derivation of optimal test sets directly from the model of a Web application.

9 Acknowledgments

The authors would like to thank the anonymous referees for their thorough comments and suggestions that have helped us improve the paper. This work has been partialy supported by Spanish Research Project TIN2005-09405-02-01 and the Italian grant FAR N. 4412/ICT.

10 References

1 Object Management Group 'UML 2.0 Superstructure Specification', 2005, http://www.omg.org/cgi-bin/apps/doc?formal/05-07-04.pdf
2 Ceri, S., Fraternali, P., Brambilla, M., Bongio, A., Comai, S., and Matera, M.: 'Designing data-intensive web applications' (Morgan Kaufmann, 2002)
3 Garrigós, I., Gómez, J., and Cachero, C.: 'Modelling dynamic personalization in web applications'. Proc. Int. Conf. on Web Engineering, Oviedo, Spain, July 2003, pp. 472–475
4 Conallen, J.: 'Building web applications with UML' (Addison Wesley, 2002, 2nd edn.)
5 Baresi, L., Garzotto, F., and Maritati, M.: 'W2000 as a MOF Metamodel'. Proc. 6th World Multi-Conf. on Systemics, Cybernetics and Informatics - Web Engineering Track, Orlando, USA, July 2002
6 Kraus, A., and Koch, N.: 'A metamodel for UWE'. PhD thesis, Ludwig-Maximilians-Universität München, January 2003.
7 Gómez, J., and Cachero, C.: 'Information modeling for internet applications', 'Chapter OO-H method: extending UML to model web Interfaces' (Idea Group Publishing, Hershey, PA, USA, 2003), pp. 144–173
8 WebML resources. 'The WebML Metamodel', 2006, http://www.webml.org/webml/page5.do
9 Schauerhuber, A., Wimmer, M., and Kapsammer, E.: 'Bridging existing web modeling languages to model-driven engineering: a metamodel for WebML'. Proc. of Model Driven Web Engineering, Palo Alto, CA, July 2006
10 Koch, N., and Kraus, A.: 'Towards a common metamodel for the development of Web applications'. Proc. Int. Conf. on Web Engineering, Oviedo, Spain, July 2003, pp. 495–506

11 Brambilla, M., Comai, S., and Fraternali, P.: ‘Hypertext semantics for Web applications’. Proc. Sistemi Evoluti per Basi di Dati, Isola d’Elba, Italy, January 2002, pp. 73–86

12 Ivers, J., Clements, P., Garlan, D., Nord, R., Schmerl, B., and Oviedo Silva, J.R.: ‘Documenting component and connector views with UML 2.0’, Technical Report CMU/SEI-2004-TR-008, Carnegie Mellon University, 2004

13 Mellor, S.J., and Balcer, M.J.: ‘Executable UML: a foundation for model driven architecture’ (Addison Wesley, 2002)

14 Meliá, S., Gómez, J., and Koch, N.: ‘Improving Web design methods with architecture modeling’. Proc. Electronic Commerce and Web Technologies, Copenhagen, Denmark, August 2005, pp. 53–64

15 Abouzahra, A., Bézivin, J., Del Fabro, M., and Jouault, F.: ‘A practical approach to bridging domain specific languages with UML profiles’. Proc. Best Practices for Model Driven Software Development at OOPSLA’05, San Diego, CA, October 2005

16 Haywood, D.: ‘MDA in a Nutshell’, 2004. http://www.theserverside.com/articles/article.tss?l=MDAHay-wood