

Meta-modeling Techniques Meet Web Application Design Tools

L. Baresi, F. Garzotto, L. Mainetti, and P. Paolini

Dipartimento di Elettronica e Informazione - Politecnico di Milano
Piazza L. da Vinci 32, I-20133 Milano, Italy
baresi | garzotto | mainetti | paolini@elet.polimi.it

Abstract Web-based hypermedia systems are becoming more and more sophisticated, new modeling requirements constantly arise, and design models must constantly evolve. Since design tools should complement models to support an efficient design process, model evolution raises a technological issue: Design tools must be modified when their underlying model changes. This is why the paper proposes a general approach to efficiently update design tools in response to model evolutions. The key ideas are: a) the description of a hypermedia model in terms of a general meta-model, powerful enough to express the semantics of current and future design constructs; b) the transformation of a hypermedia design tool into a meta-CASE tool, able to cope with model updates without requiring to be redefined and rebuilt from scratch.

The approach is presented by discussing a case study, that is, the feasibility study to transform our design toolkit, `Jweb`, into a meta-CASE tool (`Jweb3`). This tool will support the latest version of our model (called `W2000`), and will easily evolve with the model it supports. We discuss the adoption of the OMG meta-modeling standards MOF and XMI as enabling technology, we present a sample of the representation of `W2000` in terms of MOF, and we sketch the architecture of the under-implementation `Jweb3`.

1 Introduction

Web-based hypermedia systems are becoming more and more sophisticated: New modeling requirements constantly arise, and hypermedia design models must constantly evolve. An example of this evolution is HDM (Hypertext Design Model). Since its first definition in 1991 [8], HDM has originated a family of variants (HDM2 [7], HDM-lite [6]) and, more recently, `W2000` [2]. `W2000` has been defined in response to the transformation of Web-based hypermedia from read-only navigational "repositories" to complex applications that combine navigation and operations in a sophisticated way. `W2000` enriches the latest version of HDM with concepts and notations for modeling richer information structures as well as operations and services accessible through the Web. It exploits the Unified Modeling Language UML [13] and its customizability to ascribe `W2000` with a standard graphical syntax.

Along with HDM, we have been developing `Jweb` [3], a CASE toolbox that assists Web application designers during the whole development process: from conceptual design to semi-automatic generation of final applications. `Jweb` provides a set of tools that

enables the designer to specify, document, reuse, and prototype their design choices in an efficient way. It has been evolving with the HDM family, but always a step beyond. We have experimented that the evolution of a CASE tool is not as easy as the evolution of the model it supports, and originates an almost chronic misalignment. This is why, before starting the development of `Jweb3`, the latest version of our toolbox, which should fully support W2000, we decided to investigate new solutions.

In this paper, we present the first results of the feasibility study that we conducted to pave the ground to `Jweb3`. We started from the schema editor, which has been always the first and most important tool in the `Jweb` chain of tools ¹, trying to meet the two following requirements:

- `Jweb3` should smooth the chronic misalignment between models and tools. It should be able to absorb possible changes in the underlying W2000 model without imposing that all tools be rebuilt from scratch.
- Since W2000 is based on UML, the prototype schema editor should exploit the modeling features of commercial UML CASE tools, like Rational Rose [15] or SoftTeam Objecteering [17], without constraining the whole toolbox to adopt strange and proprietary format to store produced artifacts.

The former requirements can be tackled using meta-CASE technology. Standard CASE tools support fixed notations, whose definition is hard-coded in the tool. Even slight changes in the notation would require that the tool(set) be rebuilt from scratch. Meta-CASE tools, in contrast, separate editing facilities from the definition of the notation they support and thus foster flexibility and evolvability. Meta-CASE technology has been widely adopted by several software engineering environments, but to the best of authors' knowledge, it is new in the domain of hypermedia model: We do not know other "meta" approaches to which our proposal can be compared. The advantage in terms of flexibility is counterbalanced by the need of supplying the tool with a rigorous description of the supported notation. The latter requirement imposes to adopt UML tools that are customizable and support a vendor-independent common format to seamlessly exchange UML specifications among different tools.

In this paper, we adopt "standard" OMG (Object Management Group) to find a solution to both problems: XMI (XML Metadata Interchange [14]) solves the first requirement, while MOF (Meta-Object Facility [12]) solves the second problem. XMI is the XML-based stream format for metadata interchange. It is useful for storing artifacts in a vendor-independent format and all newer UML tools support it. MOF is the standardized model for meta-data definition, that is, for describing the fundamental concepts with which applications work. MOF can be used to define special-purpose notations for particular application domains. These definitions can then drive subsequent implementation steps, serve as basis for connecting independent software components, or be used as oracle (i.e., reference) for assessing the correctness of specifications that should be compliant with it.

The adoption of MOF-XMI as underlying enabling technology for `Jweb3` would transform `Jweb` in a meta-CASE tool that would allow us to update all modeling tools

¹ The schema editor is the CASE component that assists the designer to model the data, navigation, presentation, and operations that belong to the application.

each time a new version of W2000 is released. A rigorous definition of the notation (i.e., of the meta-model) would provide a precise framework to accomplish all versions and customizations of W2000 and would shorten the update effort from months to days. Nevertheless, the adoption of this technology requires an in-depth and critical analysis due to its novelty and continuous evolution.

All major UML tools are good at letting designers manipulate UML models visually and export them using XMI-UML. MOF can proficiently be used to specify the W2000 meta-model, i.e., it can act as meta-meta-model for W2000. What is still missing is the semantic validation, that is, the crosscheck of a W2000 model (for a particular application) against its meta-model. This paper proposes a solution based on Schematron² ([1]) to let W2000 designers validate their models within the framework described so far. Even if the emphasis of the paper is on W2000 and Jweb, all identified solutions are general enough to be adapted to other – mainly object-oriented – design models and tools, in particular if object oriented (such as OOHDM [16], WebML [4] or similar).

The rest paper is organized as follows. Section 2 describes the feasibility study. Section 3 sketches a fragment of the W2000 meta-model, defined using the technology presented in Section 2. Section 4 describes how Jweb3 will exploit meta-modeling by proposing a first high-level architecture, and Section 5 briefly lists our current and future work and concludes the paper.

1.1 OMG Terminology

To explain our approach, it is important to clarify the OMG terminology (summarized in Table 1), and give the terms models and meta-models a different meaning with respect to traditional Web and database design. According to the OMG, a notation (e.g., UML) is based on a meta-model, which can be used to design different models (i.e., definitions of particular applications). This is against usual database or Web design jargon, where an OMG meta-model is called model (e.g., the ER model or the HDM model), and an OMG model is called schema (of a particular application).

OMG Levels	OMG terminology	DBterminology	Examples
3	Meta-meta-model	Meta-model	MOF
2	Meta-model	Model	W2000 (UML)
1	Model	Schema	Web-based conference manager
0	Object	Instance	Papers, authors, etc.

Table 1. OMG hierarchy

² Schematron is a structural schema language that differs from other schema languages since it not based on grammars but on finding tree patterns in the parsed document. This approach allows for the definition of simple and powerful validation engines.

2 Feasibility Study

Before updating (rebuilding) the `Jweb` toolset to make it become fully compliant with W2000, we decided to start a feasibility study to better understand the key features of the new toolbox. We wanted the new tool-suite to supply developers with a UML-like graphical notation for application schema design, and we wanted also to find better solutions to the severe problem of lack of flexibility and extensibility, i.e., the chronic misalignment between `Jweb` and HDM evolutions.

As to the first aspect, we decided to probe the possibilities of using existing UML CASE tools as temporary front-ends for the forthcoming schema editor. The idea was not to select “the best” available tool, but simply do some experiments and delay the implementation of a special-purpose editor to better refine W2000 constructs. As side effect, this implied that we did not want to work with proprietary formats to store models (W2000 schemas), but we needed a cross-platform standard to be able to change the front-end according to our needs.

Flexibility and extensibility required something more sophisticated than the XML-DTD solution used so far. The DTD description of HDM revealed to be insufficient and did not support tool evolution seamlessly. We did not try to find independent solutions to the two problems, but we decided to exploit the OMG novel way to meta-modeling to find a cumulative solution.

When we think of meta-modeling, we have first to recall that a CASE tool can roughly be schematized as software that supports a given notation and allows developers to create design artifacts according to that notation. Using the OMG jargon, we can say that each model (i.e., artifact) must be compliant to a given meta-model (i.e., notation). This definition allows us to decompose a CASE tool into a provider of modeling features, which depends on the supported meta-model, and a consistency checker between defined models and their meta-model. Meta-CASE tools ([18]) do a similar job at a meta-meta-level. Meta-CASE tools let the developer define his own meta-model and express the semantics of the primitives of this meta-model in terms of a meta-meta-model. Describing a new modeling feature of a meta-model (e.g., a new primitive for W2000) with a meta CASE tool would require the designer to describe the semantics of the new construct in terms of the meta-meta-model. The key point is that a meta-meta-model (MOF) supplies all modeling features to render extensions and mechanisms to (automatically) update the tools with respect to consistency checking and manipulation operations.

Borrowing and extending this definition, we decided to study how to describe W2000 as a meta-model, how to identify what modeling features are specific to W2000 (with respect to other standard meta-models like pure UML), how to generate meta-model updates automatically, and how to store and distribute designed models in a way that would not be tight to any particular tool vendor and would reserve room for possible changes in the meta-model. As a side effect, a complete and powerful formalization of the meta-model would have served also as rigorous description of W2000 itself.

2.1 W2000 as a Profirable MOF Meta-model

W2000 can be seen as a UML extension (profile, according to the UML terminology)³. UML, in fact, supplies ad-hoc features (stereotypes, tagged values, and constraints) to extend the original UML to cope with particular needs and specific application domains. Defining W2000 as a UML profile allows us to exploit UML commercial tools, but also XMI-UML, that is, the UML instantiation of XMI⁴, which is rapidly becoming the "lingua franca" to exchange UML models in a vendor-independent way. Any XMI-compliant UML tool can then be used as graphical front-end for the schema generator; we have only to implement our UML profile using the features offered by the particular tool. Roughly speaking, we can say that no matter the tool we use to edit a W2000 schema, we should obtain an XML description compliant to an XMI-UML DTD.

The adoption of UML profiles is enough to use UML CASE tools as front-ends for the schema editor, but does not allow semantic checks. Currently, available CASE tools support profiles partially (only Objectteering [17] provides some support): Evolution and consistency checks would not be possible. This is why we decided to adopt MOF (the OMG standard way for designing meta-models) to describe the W2000 meta-model, that is, to give a precise definition of W2000 concepts in terms of a flexible meta-meta-model. This approach allows us to change W2000 the way we want and we have a better means to assess the consistency of our models. Even if MOF is the means for implementing meta-CASE technology, we did not want to get rid of the whole UML and start our description of the W2000 meta-model from scratch. We decided for a compromise solution: The W2000 meta-model reuses many UML concepts, and this means that we can see W2000 as a UML profile and all possible versions of W2000 constructs can always be mapped onto instances of UML primitives, but with respect to conventional UML profiles it satisfies also an additional condition: All concepts (both new and borrowed ones) are defined using MOF. Thus, we can say that W2000 is a profilable MOF meta-model (hereafter, p-meta-model).

The reason for introducing the concept of p-meta-model is that if the UML meta-model can be described in terms of MOF, the opposite does not hold true, i.e., not all meta-models described in MOF can be mapped onto the UML meta-model. Thus we could not be able to check the consistency of a W2000 model (compliant with a "purely UML" definition of W2000 and described using standard UML tools) against the W2000 meta-model (compliant with the MOF-defined meta-model) directly. The comparison would never be possible in the general case, but the profilability of the W2000 meta-model allowed us to think of a translator to make the comparison happen. Besides this, we had to bear in mind evolvability, thus the comparison/validation would have been possible against evolutions of the W2000 meta-model.

³ W2000 was born as a by-hand extension to UML. Its rigorous definition as a UML profile is in progress.

⁴ XMI-UML means the XMI format for the UML meta-model. XMI can be employed with any meta-model defined using MOF.

2.2 Our technological solution

All these requirements led us to define the software architecture for the schema editor of Jweb3, i.e., the editor of W2000 models, as depicted in Figure 1.

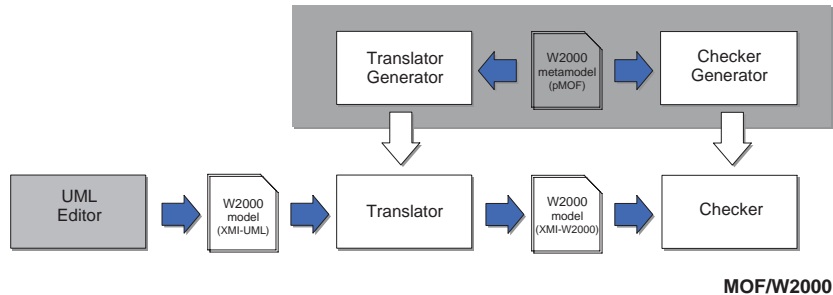


Figure 1. Our proposal

The gray box in the upper left corner identifies meta-modeling technology; all other tools belong to the usual modeling chain, which must be enabled by a set of preliminary meta-modeling steps. We start by defining the W2000 p-meta-model: Usually we do this using *Rational Rose* integrated with the *Unisys add-on for XMI* ([19]). This model, stored in XMI-MOF, is the main input to the two meta-modeling tools. The **translator generator** produces a translator from XMI-UML to XMI-W2000 as a set of XSLT rules. The **checker generator** uses the same meta-model to produce a set of Schematron rules. Schematron can be used to find specific patterns in an XML document; in this case it is used to validate W2000 models against the constraints identified in the meta-model. These tools exploit the *IBM XMI toolkit* ([9]) to produce the DTD needed by XMI-W2000.

Moving to the modeling chain, the designer specifies his W2000 models using his preferred UML editor and stores them in XMI-UML. The translator, through the XSLT rules, transform the XML file compliant to XMI-UML to another XML file, but compliant to XMI-W2000. This file is then the input to the checker that validates it through the Schematron rules. Model validation would have been accomplished using a MOF repository, that is, using a repository that is aware of the "format" its stored models should be compliant with. These repositories should exploit OCL as language to specify constraints and thus the rules that define a valid model. Unfortunately, none of the MOF repositories (for example, dMOF [5]) we used during our experiments fully supported all these features and thus we decided to move to a temporary solution based on Schematron. Also, other more standard technologies, like XPath, XLink, and XQuery, would be applicable, but the supporting frameworks need further attention and analysis before being usable to mimic a MOF repository.

Some excerpts of the meta-modeling process are presented in the next section, but the presented solution deserves two more remarks:

- Even if p-meta-models have been defined to solve a specific problem, that is, to define the meta-model of W2000, they are absolutely general and can be used in all those cases where either we need (want) a MOF definition of an extended UML notation or simply we want to integrate MOF-based and UML-based tools.
- We implemented all logical components of Figure 1 through special-purpose scripts (batch files and javascript) and XSLT sheets, all coordinated by HTML interfaces. We think that this implementation style should better fit evolution and simplify maintenance. More conventional solutions (for example, Java-based solutions) will be investigated in the near future as soon as both the technological and methodological landscapes are more defined and clearer.

3 W2000

This section presents an example application of the p-meta-model of W2000. We present both a simplified p-meta-model and its use for modeling the Web-based conference manager described in [2]. Lack of space obliges us to present some excerpts of both models; the whole formalizations are presented in [10]. We have also to assume that readers are proficient in HDM/W2000. Interested readers are referred to [2,8] for in-depth presentations of both notations.

3.1 A p-meta-model for W2000

The p-meta-model for W2000 (Figure 2) comprises the three standard UML packages (see the UML meta-model, [13]), and the new *W2000 package*.

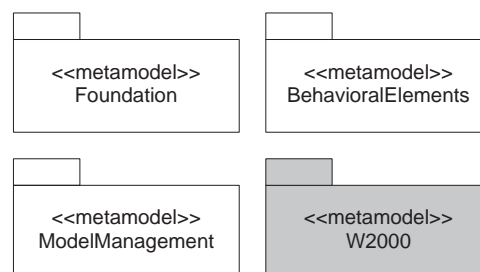


Figure 2. The high-level organization of the W2000 p-meta-model

This organization allows W2000 to share all main modeling elements with UML. The *Foundation* package supplies all basic modeling elements (i.e., classes, attributes, associations, etc.), the *BehavioralElements* package supplies the elements to specify the dynamic behavior of a model, the *ModelManagement* package supplies the means to cope with complexity and organize a model into submodels according to the different viewpoints. The new package, *W2000*, defines how we extended UML to represent W2000 constructs.

The profilability of the p-meta-model implies that all new W2000 concepts be specified as extensions to standard UML elements, that is, W2000 elements are subclasses of standard UML meta-model classes. This organization is partially shown in Figure 3, which collects some excerpts from the W2000 package. Figure 3(a) shows the main elements of the *Information Model Design*: classes `Entity` and `Component` are both subclasses of class `Class` (defined in the *Core* subpackage of the *UML Foundation* package). Both classes are abstract classes because `Entity` are always specialized in either `EntityType` or `SpearEntities`, while `Components` become `ComponentTypes` or `SpearComponents`. Types are similar to classes: They will be instantiated as many times as needed in the running application; spear elements correspond to singletons: They specify special-purpose elements singularly instantiated in the application. Figure 3(b), which shows the main elements for *Access Layer Design*, has a similar organization. A `Collection`, abstract concept, comes from a `Class` and is specialized in `SpearCollections` and `CollectionTypes`. Moreover, each collection has a `CollectionCenterType`, which is a subclass of `CenterType`. Figure 3(c) shows links and nodes, the two main elements as to *Navigation Design*. Once more, `Nodes` must be either `NodeTypes` or `SpearNodes`. Links must be either `CollectionLinks` or `SemanticLinks`, or `StructuralLinks`. Collection links relate the elements of a collection with their center, semantic links correspond to semantic associations, and structural links render the component-based decomposition of complex entities.

The whole W2000 package comprises some 40 different elements, which are special-purpose refinement of UML elements. Besides adding new properties (attributes) to these elements, we defined also some 60 constraints among the elements in the new package. These constraints range from very simple constraints, like: each W2000 model must comprise at least an entity, to more complex constraints. For example: An `AssociationEnd`, which defines the connections of either a `SemanticAssociation` or a `SemanticAssociationCenterType`, must refer to an `EntityType` or a `ComponentType` or a `CollectionType`, or a `CenterType`.

To formalize this constraint as a Schematron rule, we have first to recall all dependencies in the p-meta-model:

- `SemantiAssociation` is a specialization of `Association`;
- `AssociationClass` is a specialization of `Association`;
- `SemanticAssociationCenterType` is a specialization of `AssociationClass`;
- `EntityType`, `ComponentType`, `CollectionType`, `CenterType` are specializations of `Class`.

All these specializations, together with the knowledge of how XMI works, permit the construction of the Schematron pattern of Figure 4. The name of the pattern simply identifies its meaning. The context defines the "scope" of the rule and identifies the types of the `AssociationEnds` connected to either `SemanticAssociation` or `SemanticAssociationCenterType`, that is, the two specializations of `Association`. The test statement checks that the types of the previously identified ends are compliant with the constraint and outputs a warning in case of violation.

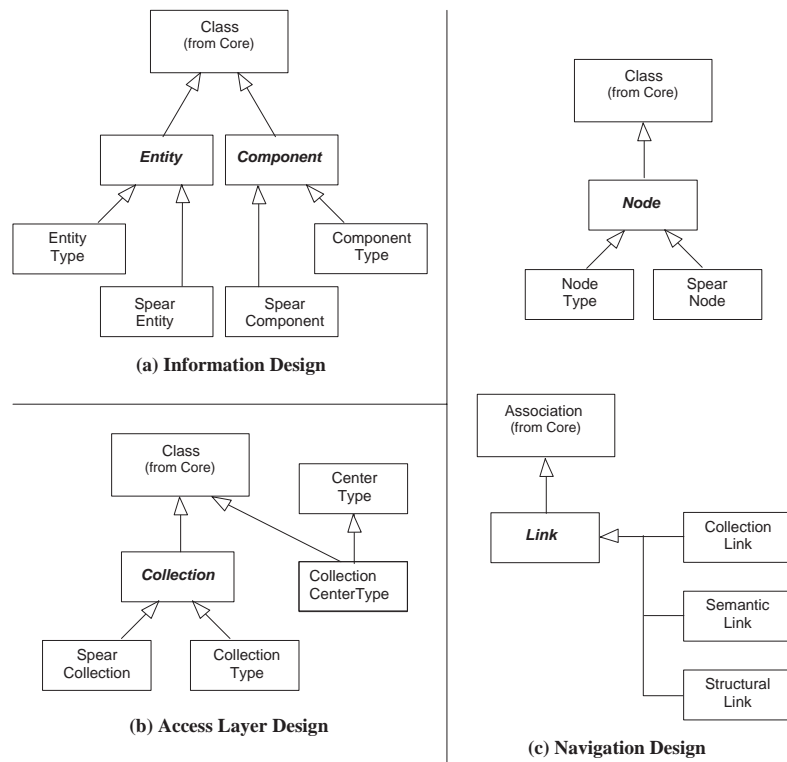


Figure 3. Excerpts from the W2000 p-meta-model

This solution, even if should only be a temporary one, supplies an interesting way for assessing the correctness of XML files with respect to external constraints. It would be redundant in our solution if all technology were available, but could be an interesting way for checking the consistency of XML files that simply come from a DTD.

```
<pattern name="SemanticAssociation">
  <rule context="
    //W2000.SemanticAssociation
    /Foundation.Core.Association.connection
    /Foundation.Core.AssociationEnd
    /Foundation.Core.AssociationEnd.type/*
    |
    //W2000.SemanticAssociationCenterType
    /Foundation.Core.Association.connection
    /Foundation.Core.AssociationEnd
    /Foundation.Core.AssociationEnd.type/*
  ">
  <assert test="
    (local-name(id(@xmi.idref))=
      'W2000.EntityType') or
    (local-name(id(@xmi.idref))=
      'W2000.SemanticAssociationCenterType') or
    (local-name(id(@xmi.idref))=
      'W2000.CollectionType') or
    (local-name(id(@xmi.idref))=
      'W2000.ComponentType')
  ">A SemanticAssociation link only EntityTypes, ComponentType,
  SemanticAssociationCenterType, and CollectionType</assert>
  </rule>
</pattern>
```

Figure 4. Sample Schematron rule

3.2 An example W2000 model

After defining a simplified p-meta-model for W2000, we can show some excerpts of the model of the Web-based management system described in [2]. The goal is not the presentation of W2000 as modeling means for Web application, but rather we want to describe how well known concepts are rendered using the p-meta-model. Interested readers are referred to [10] for a complete specification of the application.

Figure 5 presents the *information model* for the paper entity type. The entity is structured in three main components: The *abstract* is always part of the paper and it presents some basic information, like the paper title, authors, and affiliation, along with other information required by the conference, that is, paper id (number) and review status.

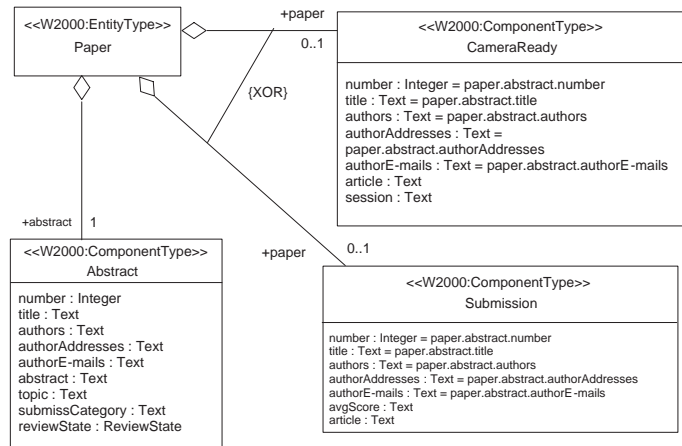


Figure 5. Information model for the *paper* entity type

The other two components are mutually exclusive: Either we have a *first submission*, or after accepting the paper, we have its *camera-ready* version. The XOR label between the two aggregations codes this dependence. These components reuse also some information already defined in the abstract. Each time an attribute (slot) is reused in a component, we simply refer to its original definition, instead of re-specifying it. This way, we avoid inconsistent redefinitions and we maximize reuse. For example, the paper id is repeated in all components, but the abstract defines it and the others reuse the definition.

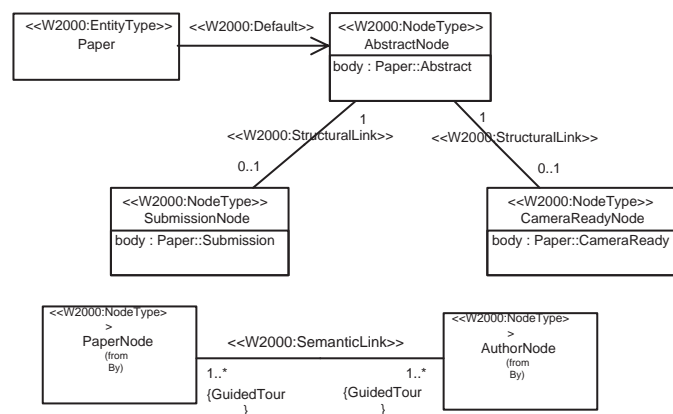


Figure 6. Excerpts from the navigation design

When we move to *navigation design*, Figure 6(a), we see that the paper has been decomposed in three nodes. Once more, the main one contains all base information, while the other two nodes correspond to the first submission and camera-ready respectively. The actual contents of these nodes is specified using the keyword body, which make readers refer to the information design to understand the actual contents. For example, the contents of the `AbstractNode` node is defined by the body of the `Abstract` component. The correspondence between nodes and components is not mandatory, but it helps modularize the design. Users are free to specify the contents of nodes as lists of attributes taken from the entities and components that they embody.

Figure 6(b) shows how the user of the Web application can navigate among papers and authors. `PaperNode` nodes have been defined in Figure 6(a); we assume we have a similar definition for `AuthorNode` nodes. These two node types are connected through a *semanticLink* object, that is, a kind of association with a navigational semantics. The diagram specifies that given a paper, we can navigate its authors (at least one, but possibly many) through a guided tour. The same is true if we consider an author and we would like to know his papers. We could navigate them using another guided tour.

The few excerpts presented in this section are not enough to demonstrate if W2000 is powerful enough, but they show how each W2000 concept has a clean and neat representation in the p-meta-model and how the concept can be rendered using an UML-like syntax to supply designers with a usable means. Moreover, simple annotations or the use of the dotted notation allow us to trace and relate all concepts to define consistent specifications.

4 Impact on Jweb

Before describing how meta-modeling could affect Jweb and its tools, we have to briefly introduce the toolset and clarify its main components. Jweb assists designers during the whole design process: from information design to prototype and enactment. Figure 7 shows its main logical components: The *editor* lets designers design their applications using HDM/W2000 and produces a HDM/W2000 schema. The *mapper* takes this schema and automatically generates the relational schema for the editorial repository, suitable interfaces for populating the repository, and a general description (XML mapping) of the relational schema. The *configurator* reads the HDM schema, the description of the editorial repository, and the editorial contents and allows for the creation of special-purpose filters to select data; the description of these filters is rendered in XML. The *generator* uses the HDM/W2000 schema, the description of the editorial repository and its contents, together with the filters defined to select data, and generates the run-time database, populating it with the editorial contents suitably filtered. The *engine* reads the contents from the run-time database, presents this contents to users and is responsible for managing the interaction between the user and the application.

The work done so far concentrated on the *editor*, but since its output (application schema) is then used by all other components, its being meta affects also these other tools. The ideal foreseen situation is that all the meta-tools interact with a centralized repository, which contains the current definition of W2000, that is, its p-meta-model, and through suitable interfaces it allows for the generation of all data needed by the

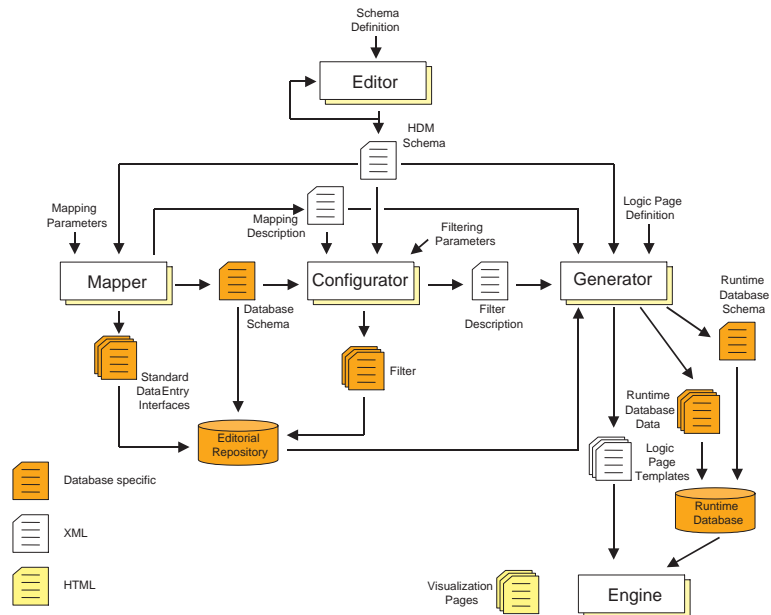


Figure 7. Logical architecture of Jweb

chain of tools. In other words, all tools would play a dual role: They would have a part that belongs to the design process, but their meta component would be in charge of generating/customizing the element according to the specific W2000 p-meta-model in use.

5 Conclusions and Future Work

Our experiments in using meta-modeling technology to support design tools are so far limited to the editor, but starting from the p-meta-model presented in the paper, we induced a non-trivial modification and we studied the impact of these modifications. The main bottleneck was the definition and test of all (new) constraints in Schematron, but all other changes were almost trivial and they could be obtained automatically. This exercise suggested two interesting considerations:

- The availability of a real MOF repository together with a friendlier notation for specifying constraints would further shorten the maintenance process.
- If we had done the same exercise using our standard way, we would have used the following process:
 1. The new meta-model would have been coded directly using a DTD, with suitable comments to explain the meaning of the new/changed features;
 2. The implementer in charge of working on the new editor (component) would have taken this definition as the reference for his work.

Thus, no automatic steps can easily be identified and misunderstandings and errors would be highly possible. In contrast, the new approach should pay both in terms of avoiding misunderstandings and errors due to different interpretations, and in the possibility of making several maintenance activities automatic. In conclusion, the feasibility study gave encouraging results as to the use of meta-CASE technology, but it revealed also the extreme youth of available implementations of OMG standards. While waiting for more robust implementations and completing the whole design of Jweb3, we can summarize the feasibility study as follows:

- It gave us the opportunity to identify new technologies for Jweb3 and permitted a thorough analysis of problems (known) and possible solutions (still to be completely identified).
- We defined the family of profilable MOF meta-models (p-meta-models), which are characterized by their intrinsic UML extended notation.
- We exploited XML, UML, MOF, XMI, XSL, and Schematron to implement the software components that support p-meta-models.
- We applied p-meta-models to a first prototype of the new schema editor.
- We defined the first p-meta-model for W2000 and also a first evolution to better understand and evaluate the soundness of our proposal as to evolution and misalignments.

Even if we applied p-meta-models to W2000, they are much more general and can be exploited each time we have a UML extension defined using MOF and we want to integrate MOF-based and UML-based tools.

In the new future, we will continue working on these ideas, trying to simplify our approach and to identify the right compromises between technological and methodological solutions ([11]). We will complete the design and start the implementation of Jweb3. But we are also investigating other possible uses of p-meta-model technology to improve the potentialities of Jweb. For example, we have ideas for two new components: a simulator and a report generator. The former should allow the designer to probe the quality of his models by playing with them before generating the real application. The latter should allow for the customizable generation of design documentation. Both these components will exploit p-meta-models and all needed information will be coded using special-purpose tags in the W2000 p-meta-model.

References

1. Academia Sinica Computer Center. *The Schematron*, www.ascc.net/xml/resource/schematron/schematron.html
2. L. Baresi, F. Garzotto, Paolo Paolini. From Web Sites to Web Applications: New Issues for Conceptual Modeling. In Proceedings WWW Conceptual Modeling Conference, Salt Lake City, October, 2000.
3. M. Bochicchio, R. Paiano, P. Paolini. JWEB: An Innovative Architecture for Web Applications. ICSC 1999: 453-460
4. S. Ceri, P. Fraternali, S. Paraboschi. Web Modeling Language, (WebML): a modeling language for designing Web sites. Proceedings of the 9th. International World Wide Web Conference, Elsevier 2000, pp 137-157

5. DSTC. dMOF User Guide, 2000, v1.01, www.dstc.edu.au/Products/CORBA/MOF/
6. P. Fraternali and P. Paolini. A Conceptual Model and a Tool Environment for Developing More Scalable, Dynamic, and Customizable Web Applications. EDBT 1998: 421-435
7. F. Garzotto, L. Mainetti, P. Paolini. HDM2: Extending the E-R Approach to Hypermedia Application Design. In Proc.12th Int'l Conf. on the Entity-Relationship Approach, Arlington, Tx, Dec. 1993
8. F. Garzotto, P. Paolini, D. Schwabe. HDM - A Model-Based Approach to Hypertext Application Design, TOIS 11(1) (1993), pp.1-26
9. IBM Alphaworks. IBM XMI Toolkit v1.15, www.alphaworks.ibm.com/tech/xmitoolkit
10. V. Miazzo. Strumenti di supporto per Modeling Ipermediale ad elevata dinamicità: ambiente di meta-modeling basato su MOF e UML. Laurea Thesis. Politecnico di Milano, February 2001. In Italian.
11. J. Nanard and M. Nanard. Hypertext Design Environment and the Hypertext Design Process, Communications of the ACM, Vol.38, No.8, Aug.1995, pp. 49-56.
12. OMG. Meta Object Facility Specification, version 1.3, March 2000.
13. OMG. Unified Modeling Language Specification version 1.4, Beta 1, November 2000.
14. OMG. XML Metadata Interchange (XMI) version 1.1, November 2000.
15. Rational Software. Rational Rose. www.rational.com/rose
16. D. Schwabe, G. Rossi. An object-oriented approach to web-based application design. Theory and Practice of Object Systems (TAPOS), Special Issue on the Internet, v. 4#4, pp.207-225, October, 1998
17. Softeam. Objecteering UML Modeler and Profile Builder, www.softeam.fr
18. J. Tolvanen. ABC to Metacase Technology. MetaCase Consulting Ltd, white paper, July 1999
19. Unisys. Unisys Rose XMI Tool v1.3, www.rational.com