

First Experiences on Constraining Consistency and Adaptivity of W2000 Models

Luciano Baresi
Dipartimento di Elettronica e
Informazione
Politecnico di Milano
Milano, Italy
bares@elet.polimi.it

Sebastiano Colazzo
Dipartimento di Elettronica e
Informazione
Politecnico di Milano
Milano, Italy
colazzo@elet.polimi.it

Luca Mainetti
Dipartimento di Ingegneria
dell'Innovazione
Università degli Studi di Lecce
Lecce, Italy
luca.mainetti@unile.it

ABSTRACT

The complexity of Web applications is increasing almost every day. Besides impacting the implementation phase, this complexity must also be suitably managed while modeling the application. The paper argues that a pure modeling notation, like W2000, is not enough to cope with such complexity and proposes an approach, based on meta-modeling and graph transformation, to enforce the consistency of produced artifacts and adapt them in a controlled way. The paper describes the approach, exemplifies it on some simple examples, and sketches the supporting framework on which we are working.

Categories and Subject Descriptors

H.4 [Information Systems Applications]: Miscellaneous;
D.2.2 [Software Engineering]: Design Tools and Techniques —*computer-aided software engineering (CASE)*

General Terms

Design, Languages, and Theory

Keywords

Web applications, Graph transformation, Meta-modeling

1. INTRODUCTION

Nowadays Web applications are complex software systems. After being hypermedia repositories, they have become fully distributed applications that intertwine business processes and Web pages. The new challenge is towards adaptability and multi-channelity, where the same concepts are instantiated in a set of *related* applications. The device, the particular user profile, or even the location pose new requirements and constraints for the application [6].

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. To copy otherwise, to republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee.

SAC'05 March 13-17, 2005, Santa Fe, New Mexico, USA
Copyright 2005 ACM 1-58113-964-0/05/0003 ...\$5.00.

In many cases, practice suggests to develop and maybe design independent applications, but it is better if we conceive the set of related applications as a *family*. This concept should pervade the whole development process: from requirements to implementation and testing, but the paper concentrates only on the design phase. It argues that if we want to handle complex applications, we need to provide something beyond the pure modeling notation: better ways for organizing, validating, and transforming models. We firmly think that designers must be free to do what they want, but at the same time they must be helped and constrained.

The paper proposes a precise and formal approach to design *consistent* application models. Given our background, the starting point is W2000 [1], our modeling notation for complex Web applications. The approach is based on meta-modeling techniques. The explicit availability of the meta-model allows us to think of any model as a collection of meta-objects. If these objects comply with the meta-model, the application design is *consistent*, that is, correct with respect to the notation.

The paper also proposes *transformation rules* to modify W2000 models in a constrained way. These rules add modifiability and adaptability to W2000. They work on meta-objects to create, modify, and delete them. They serve to automatize all those tasks that do not require designers' intuition, but are mandatory to create models and derive new models by adapting existing ones. It is important to stress that designers are free to modify what produced by these rules and can always rely on the meta-model to foster consistency.

The rest of this paper is organized as follows. Section 2 describes W2000 as a proper meta-model. Section 3 presents our approach towards coherent, consistent, and adaptable models and exemplify the rules through examples. Section 4 describes the supporting toolset and Section 5 briefly surveys the related work and concludes the paper.

2. W2000 AS A META-MODEL

Meta-modeling is nothing new: It has been around for years, but it is having new life with UML (Unified Modeling Language), MOF (Meta Object Facility), and the four-layer OMG (Object Management Group) proposal [12].

According to OMG, the *objects* of an application are instances of elements specified in a *model*. The *meta-model* defines the language used to render the application and the

meta-meta-model defines MOF, that is, the unique language that must be used to specify all languages in the framework.

In this section, we describe the meta-model of W2000. Even if, for the sake of simplicity, we slightly simplified it and assume that all multiplicities are 1. .n, the class diagram of Fig. 1 presents all important aspects.

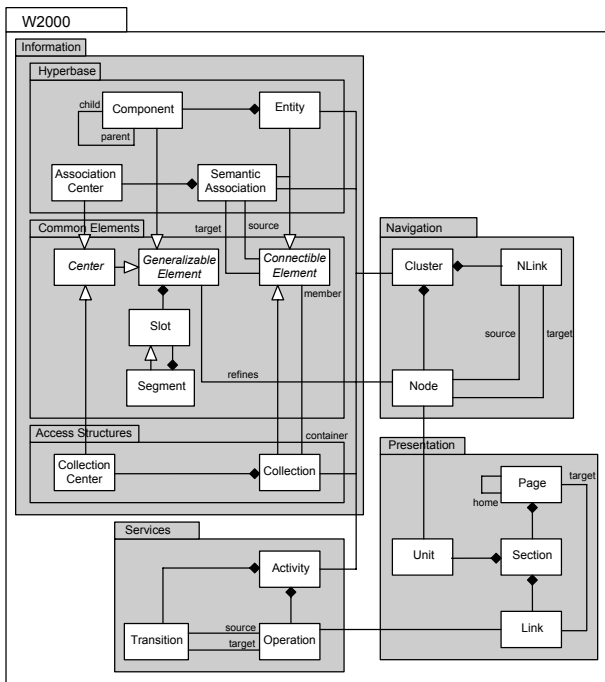


Figure 1: The meta-model of W2000

W2000 fosters *separation of concerns* and adopts a *model-view-control* approach. A complete model is organized in four sub-models: *information*, *navigation*, *services*, and *presentation*. *Information* defines the data used by the application and perceived by the user. *Navigation* and *Services* specify the control, that is, how the user can navigate through the pieces of information and modify them through suitable business processes. *Presentation* states how data and services are presented to the user, that is, it specifies pages and activation points for the business services.

The meta-model of Fig. 1 renders all these concepts using special-purpose packages. Package *Hyperbase* identifies the *Entities*¹ that characterize the application. They define conceptual “data” that are of interest for the user. *Components* are then used to structure the *Entities* into meaningful fragments. They can be further decomposed into sub-components, but the actual content is associated with leaf nodes only. Since a *Component* is also a *Generalizable Element*, from package *Common Elements*, it is further decomposed into *Segments* and *Slots*.

Slots identify primitive information elements and are the typed attributes that specify the contents of leaf components. *Segments* define “macros”: they identify sets of slots

¹All W2000 elements can be either *typed* or *single*. They belong to the first group when define *classes* of similar elements; they are part of the second group when define singletons, that is, elements that are available as single instances only.

that can be reused in different elements. Both *Slots* and *Segments* belong to package *Common Elements*.

Semantic Associations identify navigational paths between related concepts. Their sources and targets are *Connectible Elements*, that is, *Entities*, other *Semantic Associations*, or *Collections*, which are explained later in this section.

An *Association Center* – subclass of the abstract class *Center* of package *Common Elements* – describes the set of “target” elements identified by a *Semantic Association*. In a 1 to n association, it defines how to identify either the entire set of targets as a whole or each individual element in the set.

Package *Access Structures* organizes the information defined so far. It specifies the main access points to the application and comprises only *Collections*, which define groups of elements that are perceived as related by the user. *Collections* organize data in a way that complies with the mental processes of the application domain. *Collections* can have special purpose *Centers* called *Collection Centers*.

Package *Navigation* defines how the user can browse the application. It reshapes the elements in the previous packages to specify the *actual* information elements that can be controlled. *Nodes* are the main modeling elements and define atomic consumption units. Usually, they do not define new contents, but render information already defined by *Generalizable Elements*. *Clusters* relate sets of *Nodes* and define how the user can move around these elements. *Nodes* and *NLinks* identify the navigational patterns and the sequences of data traversed while executing *Activities*. This leads to organizing *Clusters* in²: *structural clusters* if all their elements come from the same *Entity*; *association clusters* if they render *Semantic Associations*; *collection clusters* if they describe the topology of a *Collection*, and *transactional clusters* if they relate the set of nodes that the user traverses to complete an *Activity* (i.e., a business transaction).

Package *Services* describes the *Operations* that can be performed by the user on the application data. Each *Operation* can be part of a business process, which is identified by an *Activity*. *Transitions* identify the execution flow. *Activities* must be rendered in the navigation model through suitable *Clusters*.

Finally, package *Presentation* offers *Units* that are the smallest information elements visualized on pages. They usually render *Nodes*, but can also be used to define forms, navigable elements, and labels. *Sections* group related *Units* to better structure a page and improve the degree of reuse of page fragments. They can be divided in *content sections*, which contain the actual contents of the application, and *auxiliary sections*, which add further contents (for example, landmark elements). *Pages* conceptually identify the screens as perceived by the user. *Links* connect *Pages* and identify the actual navigation capabilities offered to the user. *Links* can also “hide” the enabling of computations (i.e., *Operations*).

2.1 OCL constraints

The pure class diagram shown in Fig. 1 is not enough to fully specify the notion of *consistent model*. Many constraints are already in the diagram (e.g., a user model is correct if it complies with the multiplicities associated with each association), but others must be stated externally by means of OCL (Object Constraint Language, which is part

²This specialization is not rendered with subclasses, but is specified using a simple flag associated with class *Cluster*.

of UML).

Topological constraints identify restrictions on the topology of meta-models. They complement those already embedded in the class diagram and must always hold true. *Special-purpose constraints* impose specific restrictions to the notation and identify new dialects. These constraints must only hold true on those models that comply with the specific dialect.

In the first set, one obvious constraint is that each element must be unique in its scope. For example, the following OCL invariant:

```
Context Entity
  inv: allInstances -> forall(e1, e2 |
    e1.Name = e2.Name implies e1 = e2)
```

imposes that Entity names be unique in a model. If two Entities have the same name, they are the same Entity. `inv` defines an invariant, i.e., a property that must always be satisfied for all the objects of the class (Entity, in this case). Notice that we used the keyword `allInstances` to predicate on the set of all instances of class Entity. Similarly, we defined invariants for all the other elements of W2000 [8].

An example of special-purpose constraints, in a simplified version of W2000 for small devices, might impose that each Page renders exactly one Section. This condition can be easily stated as an invariant associated with class Page:

```
Context Page
  inv: sections->size = 1
```

In this case, we use the attribute `sections` to refer to the aggregation between Page and Section of Fig. 1.

The meta-model, along with its constraints, supplies the means to assess the consistency of designed models. We can cross check every model against its definition – the meta-model – and see if the first is a proper instance of the second. The meta-model is important, but it neither helps transform models nor simplifies their design. This is the role of the *transformation rules* presented in the next section.

3. TRANSFORMATION RULES

The meta-model paves the ground to *coherence* and *adaptability*. Given the organization of W2000 models, there are two kinds of relationships between W2000 sub-models. *Horizontal relationships* relate different versions of the same model. For example, the *Presentation* for a PC-based application and that for a PDA-based system define a horizontal relationship. *Vertical relationships* relate two models in the hierarchy. For example, the *Information* and *Navigation* for a PC-based application define a vertical relationship.

Both relationships are implemented by means of *transformation rules* that work on instances of the meta-model. Vertical relationships can be used to add model elements automatically. All modeling activities that are intrinsically automatic can be rendered through rules. They help the designer save time and produce consistent models. For example, rules can add a component to each entity, a node for each component, and a cluster for each entity, association, and collection in the model.

Rules are also useful to transform and adapt (part of) models. Adaptation (horizontal relationships) can be imposed by new requirements or by the need of delivering a

new member of the family by modifying some model elements (e.g., we can support a new device by reshaping navigation and presentation models). This is a way to enforce the use of modeling patterns. Instead of relying on the ability of designers to embed significant patterns in their models, rules offer a ready-to-use means to exploit them.

Even if users exploit these rules, they are free to modify their artifacts by hand to change and complete them. As already stated, we want to ease the modeling phase and not completely substitute design intuitions with machine-based rules. This is also supported by the idea that the approach can be adopted in different ways. At one end, it can be used to define a first framework for the application and leave plenty of room to the designer to complete it. On the other end, it could offer a complete library of rules to produce the application almost automatically. In both cases, the meta-model oversees the correctness of produced models.

Transformation rules define a graph transformation system. Specifically, a *typed graph transformation system* $\mathcal{G} = \langle TG, C, R \rangle$ consists of a type graph TG , a set of structural constraints C over TG , and a set R of rules $p : L \Rightarrow R$ over TG . A graph transformation rule $r : L \Rightarrow R$ consists of a pair of TG -typed instance graphs L, R such that the intersection $L \cap R$ is well-defined (this means that, e.g., edges which appear in both L and R are connected to the same nodes in both graphs, or that vertices with the same name have the same types, etc.). The left-hand side L represents the pre-conditions of the rule while the right-hand side R describes the post-conditions.

The application of a graph transformation rule comprises three steps: 1) We find an occurrence o_L of the left-hand side L in the current graph G . 2) We remove all the vertices and edges from G which are matched by $L \setminus R$. The remaining structure $D := G \setminus o_L(L \setminus R)$ must be a legal graph: no edges are left dangling because of the deletion of their source or target vertices. In this case, the *dangling condition* is violated and the application of the rule is prohibited. 3) We glue D with a copy of $R \setminus L$ to obtain the derived graph H . We assume that all newly created objects, links, and attributes get fresh identities, so that $G \cap H$ is well-defined and equal to the intermediate graph D . Usually, rules are composed to perform significant transformations.

3.1 Example rules

Given the meta-model presented in Fig. 1, we can design different rules to both help the designer automatize design decisions and transform models. The paper does not describe a smart and extended library of rules; instead we want to demonstrate the feasibility of the proposal.

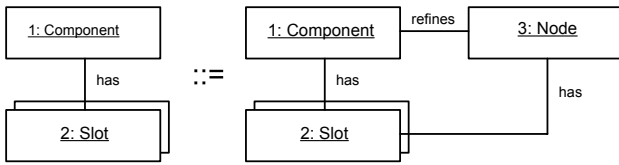
Rules are rendered here as pairs of UML object diagrams. For the sake of understandability, they do not deal with the structure (packages) of the model, but assume a simple flat organization. Hierarchical models would simply need more complex rules.

To start with a simple and general purpose rule, Fig. 2³ presents a vertical transformation. It helps define the *Navigation model* by refining the components of the *Information model*. It adds a new Node element that corresponds to a leaf Component and the new Node inherits all Slots that define the Component. Notice that since the cardinality of the set of Slots that belong to the Component can vary, we

³As general solution, compositions (black diamonds) of Fig. 1 are rendered with `has` labels in the rules.

use the UML *multiobject* to identify a variable collection of objects.

1.leaf == true



3.name = 1.name + "Node"
3.comment = "automatically generated";

Figure 2: The rule that creates a new Node given a leaf Component

The rule comprises two object diagrams and two text blocks. The one before the diagrams constrains the attribute values of the left-hand side elements to enable the rule. The block after defines how to set the attributes of the right-hand side elements. In this case, the rule imposes that the Component be a leaf one and shows that the name of the new Node is the name of the Component augmented with the suffix Node. The comment says that the Node is generated automatically.

This rule allows the designer to apply it to as many leaf Components as he wants. The tool support will allow the user to either pick the elements of the left-hand side, and apply the rule selectively, or apply it as many times as possible by discovering all possible redexes (left-hand side parts) in the current model.

Similarly, if we wanted to create a new Cluster, along with its Nodes, from a single Entity, with different Components, we could apply the rule of Fig. 3, where textual annotations are left unspecified since they have no impact on the rule. The left-hand side matches the Entity and its Components, while the right-hand side preserves them, adds the new Cluster, and creates as many new Nodes as the number of Components. The rule could have been more detailed and structured, but here we prefer a compact representation.

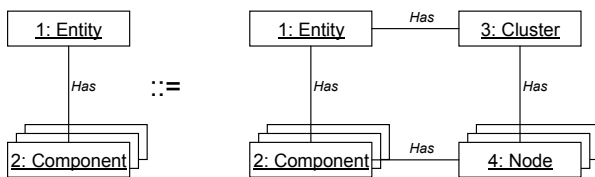


Figure 3: The rule that creates a new Cluster given an Entity

Figure 4 exemplifies the rule and shows an Entity (Figure 4(a)) and the generated Cluster (Figure 4(b)). Notice that nodes are not connected: this is an explicit design decision, but more sophisticated rules could also specify the links among nodes.

Another transformation that deals with clusters is presented in Fig. 5. We do not describe the rule itself, but its application to transform a cluster from being conceived

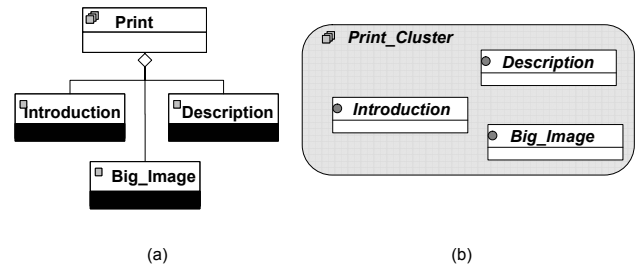


Figure 4: Example application of the rule of Fig. 3

for a PC (Figure 5(a)) and its counterpart for a PDA (Figure 5(b)).

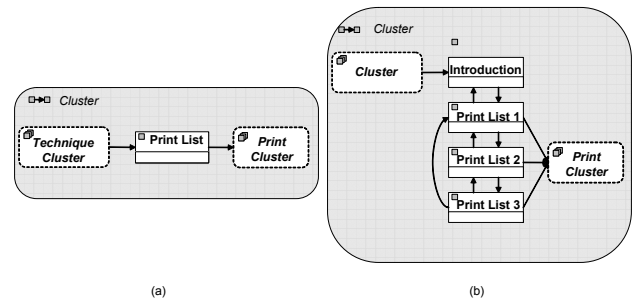


Figure 5: An example of PC- and PDA-based clusters

The PC-based cluster uses a single Node, which renders an association center, and contains both the slots that describe the list and the whole print list (i.e., a representation of the elements of the list). The same elements for a PDA are split onto different Nodes. The node *Introduction* contains the slots that describe the list and is the first node from which we start the navigation. The other nodes are used to visualize the elements in the list, five elements for each node, and implement a *guided tour* among them. Needless to say, the rationale behind this transformation is that we want to organize the information in smaller chunks because of the limited display capabilities of the device.

Transformation rules, which predicate on the elements of the meta model, leave W2000 models in a consistent state. The user can always edit and complete produced artifacts and exploit the meta-model to ensure the consistency. Intermediate steps can be inconsistent, but the final result must be consistent.

3.2 Lessons learned

Rules are usually simple and light, but if applied consistently can really help the designer. Currently, we have no empirical evaluation of the approach to “demonstrate” how good the approach is, but we have already applied it to a couple of projects. Even if the tool support is still limited, we applied these concepts to the design of the OpenDrama application⁴, a multimedia and multichannel application for presenting contents related to operas, and also to the development of the Munch project⁵.

⁴The experiment has been carried out within the homonymous EC-funded project.

⁵In cooperation with the Staatliche Museen of Berlin (Ger-

These two pilot applications have raised a couple of interesting arguments. Intra-model transformation rules (horizontal relationships) simplify the design process and make the final artifact much more homogenous. The exercise has been carried out mainly by hand, but even the simple specification of these rules helped the designer focus on the different transformations and save time while implementing them. Inter-model transformation rules (vertical refinements) demonstrated to be even more important for an application that must support PCs, PDAs and iTV as primary channels. The rules to produce collections from entities, nodes from components, and clusters from entities, associations, and collections have interesting (from a technical point of view) and really useful during the actual design.

To find further inputs for our work, we have also tried to ask our students to model some (excerpts) of the same applications and in many cases, we have been able to discover the same (or other) transformation rules by following a reverse-engineering approach.

4. TOOL SUPPORT

The toolset exploits available MOF technology and graph transformation tools. *Eclipse* acts as *Editor* and supplies the framework, *MDR/netbeans* [10] is the *MOF repository*, and *AGG* (Attributed Graph Grammar System, [3]) is the *Rule Engine*.

According to the architecture of Figure 6, the user interacts with the toolset using the *Editor*, which is a W2000-specific graphical editor, implemented as an add-in to *Eclipse* [2]. It also exploits *GEF* as graph editing library. This component exists and is already used internally.

Topological constraints are embedded directly in the meta-model, while special-purpose constraints are checked using an external *Constraints validator* based on *xlinkit* [9], which allows the definition of constraints on XML documents and supplies the needed degree of flexibility. The *MOF repository* is released and works in conjunction with the *Editor*, while the *Constraints validator* is still under development. This solution is flexible since both the *Editor* and the *MOF*

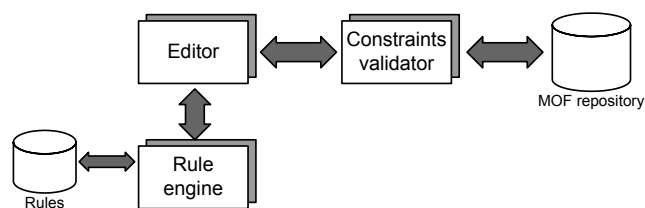


Figure 6: High-level architecture of the tool support

repository support XMI (XML Metadata Interchange [11]) as XML-based neutral format for exchanging artifacts and leave room to the integration with other components (for example, automatic generators of code and documentation).

AGG is an interpreter for graph transformation rules that – in this context – applies transformation rules on the instances of the meta-model. This is not fully integrated with the *Editor*, but we have already conducted experiments with some rules.

many)

5. CONCLUSIONS AND FUTURE WORK

The paper presents our approach and the first results for fostering consistency and coherence on W2000 models by means of meta-modeling and transformation rules. Besides introducing W2000 as a proper MOF meta-model, we describe *transformation rules* as means to automatically transform instances of the meta-model to shorten the modeling process, enforce coherence, and guide designers while adapting their models to specific needs, like channels, profiles, and accessibility issues.

The context of this work is the set of design/specification notations – for Web applications – characterized by a precise definition in terms of either a UML profile or a MOF meta-model. Among these, the most significant proposals are the work by Nora Koch [7] and the work done within the OO-H approach [4].

We must also mention graph transformation systems and the work by Heckel et. al. [5], which applies graph transformation to the design of the back-end of a Web application.

6. REFERENCES

- [1] L. Baresi, E. Bianchi, L. Mainetti, M. Maritati, and A. Paro. UWA Hypermedia - User Manual. Technical Report UWA-15, Politecnico di Milano, 2001.
- [2] Eclipse consortium. Eclipse – Home page. www.eclipse.org/.
- [3] C. Ermel, M. Rudolf, and G. Taentzer. The AGG approach: Language and tool environment. In G. Engels, H.-J. Kreowski, and G. Rozenberg, editors, *Handbook of Graph Grammars and Computing by Graph Transformation, Volume 2: Applications, Languages, and Tools*, pages 551 – 601. World Scientific, 1999.
- [4] J. Gómez and C. Cachero. OO-H Method: Extending UML to Model Web Interfaces. pages 144–173, 2003.
- [5] R. Heckel and M. Lohmann. Model-based development of web applications using graphical reaction rules. In *Proc. Fundamental Approaches to Software Engineering (FASE'2003), Warsaw, Poland*, LNCS. Springer-Verlag, 2003.
- [6] G. Kappel, B. Proll, W. Retschitzegger, W. Schwinger, and T. Hofer. Modeling ubiquitous web applications - a comparison of approaches. In *Proceedings of the Third International Conference on Information Integration and Web-based Applications and Services (iiWAS2001), Linz, Austria*, pages 163–174, sep 2001.
- [7] A. Knapp, N. Koch, and G. Zhang. Modeling the structure of web applications with argouwe. In *Proceedings of the Fourth International Conference on Web Engineering*, volume 3140 of *Lecture Notes in Computer Science*. Springer Verlag, 2004.
- [8] M. Maritati. Il Modello Logico per W2000. Master's thesis, Università degli Studi di Lecce - Politecnico di Milano, July 2001. In Italian.
- [9] C. Nentwich, L. Capra, W. Emmerich, and A. Finkelstein. xlinkit: a Consistency Checking and Smart Link Generation Service. *ACM Transactions on Internet Technology*, 2(2):151–185, 2002.
- [10] netBeans.org. Metadata Repository (MDR) home. mdr.netbeans.org/.
- [11] Object Management Group. Xml metadata interchange (XMI) specification. Technical report, OMG, 2002.
- [12] Object Management Group. Meta Object Facility (MOF) Specification - v.1.4. Technical report, OMG, March 2002.