

Supporting Reusable Web Design with HDM-Edit

Franca Garzotto, Paolo Paolini, Luciano Baresi
Dipartimento di Elettronica e Informazione - Politecnico di Milano (Italy)
Piazza Leonardo da Vinci, 32 – 20133 Milano (Italy)
E-mail: {garzotto, paolini, baresi}@elet.polimi.it

Abstract

As on-line and off-line electronic publishing is increasingly becoming a massive industrial activity, the design of web applications, and of hypermedia applications in general, needs to become a well-organized and efficient process. If a good design model is the first item in the agenda, immediately after design tools become necessary.

This paper describes HDM-Edit, a schema editor that supports conceptual design of web (more generally, hypermedia) applications. HDM-Edit is a component of the JWeb system, which supports all the phases of web prototyping development. HDM-Edit can be used on its own, to support the design activity only and the production of high-quality design documentation, or in conjunction with other modules of JWeb. HDM-Edit promotes design reuse by supporting several navigation patterns as built-in modeling primitives and by allowing designers to derive the schema of a specific application as a set of hyperviews on the design of a family of applications (application framework).

The paper analyzes the requirements for a design tool, sketches JWeb, describes HDM-Edit, discusses experiences of use, and finally compares the requirements with the current version of the tool.

Keywords:

Web and Hypermedia Design, HDM, Design Reuse, Hyperview

1 Introduction

The design of web applications has evolved from being art to an organized engineering activity. The most important factor to make web design efficient and effective is the availability of a suitable design model. A design model is, in essence, a set of primitives that allow designers to describe the relevant features of a web application at conceptual level by abstracting from implementation details. Several design models have been proposed in recent years, such as HDM [7, 9, 10], OOHDM [23], RMM [14], MacWeb semantic networks [18], Araneus [17], and WebML [16], just to mention a few of them.

The purpose of this paper is not to discuss merits or demerits of the different design models, but rather to argue that no model can be used proficiently, unless it is sup-

ported by a proper set of design tools. Several authors have already proposed different systems to support hypermedia design ([4, 5, 13, 18]). In this paper, we present a design-supporting tool, named HDM-Edit, which is a component of a large environment called JWeb [1,2]. JWeb supports all the different phases of web prototyping development: design, data entry, customization, presentation, prototyping, and delivery. In this paper we focus on HDM-Edit; we briefly describe JWeb only to clarify some aspects of cooperation between HDM-Edit and the different modules of the development environment. HDM-Edit has a number of original features that are examined later in the paper. To anticipate some of the most relevant aspects, we may say that:

- It supports the last version of the HDM model;
- It supports design-by-reuse and, in particular, a number of design patterns [9, 19, 20, 12];
- It supports the development of families of applications (“application frameworks” [15, 23]);
- It supports the production of high quality design documentation, as part of the design process;
- It can communicate, via XML files, with other design and implementation tools;

The remaining of the paper is organized as follows. Section 2 identifies the potential users of a design support tool and discusses conceptual and technical requirements, from the different categories of users, for such a tool. Sections 3 and 4 shortly introduce JWeb and HDM to set HDM-Edit. Section 5 introduces HDM-Edit with examples and a (simplified) case study, while Section 6 discusses its implementation. Section 7 analyzes the outcome from experimental validations, and Section 8 concludes the paper by comparing the features of HDM-Edit with the list of requirements of Section 2 and outlines the directions of our future work.

2 User Requirements

This section identifies possible users for a web design tool and discusses the requirements, both conceptual and technical, that are imposed by the different classes of users.

2.1 Potential Users

HDM-Edit has been designed and developed having in mind three main categories of users:

- *Expert designers*, who are expected to develop large and complex applications, and need to be efficiently supported from the early design stages (when several details about the application structure and functionality are still unclear or missing) to the final step of design evaluation;
- *Hypermedia design instructors*, who are likely to develop applications of average complexity, and would use the design tool to generate design examples for discussing and exemplifying the design process.
- *Students and novices*, who are likely to develop relatively simple applications, but need to be strongly guided in the design activity and would use the tool to learn-by-doing.

2.2 Conceptual Requirements

Model Adaptability. One important reason of dissatisfaction with our previous design tools was that we kept evolving the design model (HDM) to add new modeling requirements, while the design tools were frozen to a specific version of it. It is therefore important that a design tool can easily be updated to reflect the latest version of the model.

Model Integration. A specific application may require peculiar design features that the primitives of the design model do not fully express. It is important that a design tool enables the integration of model-based definitions with additional (non formalized) specifications of all the relevant aspects, including those not completely supported by the model.

Completeness Adaptability. If a design tool is part of a development environment and an application generator must manage the design specifications, a complete design must cover all application details. On the other hand, if the design tool is used to produce a "design document" for human consumption only, design specifications can be considered complete even if they are not fully detailed, but cover only the in-the-large aspects of an application. A design tool should be able to support different degrees of completeness, and to allow design specification to be refined at multiple levels.

Design Process Flexibility. As discussed by Nanard and Nanard in [18], the design of a complex application is a sophisticated "tango-like" process that does not always proceed linearly in a predefined way. It is rather an interwoven set of activities, which alternate bottom-up and top-down steps, with backtracking and switching among several actions. Thus a design tool should support a flexible design process, which doesn't stiffen the mental process of expert designers in a fixed procedural schema. On the other end, keeping in mind the needs of inexperi-

enced designers, the tool should enforce a (possibly minimum) number of priority constraints. These constraints should define which design constituents must be specified before others (imposing for example that to define a link type the source and destination object types must be already defined). Constraints are useful to guide the design activity, especially for novice designers, helping them master the complexity of organizing the different facets of design.

Support for Design-by-Reuse. One of the best strategies to efficiently produce high quality design components is to reuse previously developed building blocks [11]. Re-using previous pieces of design is affective both for teaching purposes and for real-life applications. A design tool should therefore support the creation, inspection, and update of a library of design projects, and the possibility of partially or totally reusing them.

Support for Design Patterns. A design pattern [6, 9, 19, 20, 12] embodies the abstraction of a design solution for a design problem. Design patterns are important for improving both the quality of design and its efficiency, since they allow designers to define complex solutions in terms of high-level predefined building-blocks. It is therefore important that a design tool supports the use of design patterns within the design process and the extension of the tool with new patterns as they become available.

Support for Application Frameworks. We use the term *application framework* [15, 23] to denote a family of applications (e.g., catalogues for fashion houses) from which specific applications (e.g., a specific catalogue for a particular fashion house) can be derived. A number of operations (filtering, specializing, cut-and-paste, etc.) are needed to support an optimal use of application frameworks. A design tool must encourage their development and reuse.

Documentation Support. Software engineering prescribes well-organized design documentation. Good quality documentation is useful for a number of reasons. In a real project, design needs to be checked, discussed, and revised several times before its completion. Documentation is crucial during design revision, assessment, and evaluation to precisely understand why something has been designed in a given way. During design update, documentation helps designers efficiently recognize which design features must be modified to satisfy evolving user needs. Documentation is also important for training and education: Design examples are more easily understood if design specifications are complemented by comments or descriptions of the design rationale. A design tool should support the creation of design documentation and the production of well-organized design reports at any step of the design process.

2.3 Technical Requirements

Delivery Independence. The design process should be independent from the specific nature of the delivery medium (e.g., on-line or off-line) or architecture (e.g., different ways of organizing the roles of clients and servers). Consequently, a design tool should support the design of applications with different delivery and architectural solutions.

Integrability Independence. A design tool should be easily integrable within a large architecture (of a development environment) but also be usable on its own, with no other modules cooperating with it.

Openness through Standard. The possibility for a design tool to receive information from, or to provide information to, other tools, should be based upon well-accepted technical standards.

Availability and Portability. A design-supporting tool should be portable and easily installable on different platforms.

3 JWeb

Entirely written using the Java Development Kit (JDK) version 1.2, JWeb is a sophisticated toolkit for schema-driven design and prototyping of web applications [1, 2]. The JWeb toolkit has been conceived to respond to several goals:

- To support HDM-driven design of hypermedia applications.
- To support the definition and population of *content holding* databases (the structure of which is derived from the application schema).
- To produce fast and low-cost prototypes of web applications. As mentioned by Nanard and Nanard in [18], *light* prototyping is a well-known technique that enables designers to evaluate a design at a nearly real scale, and to get experimental feedback for checking the design choices. A prototype permits everybody, even non-expert people, to fully understand the design, changes and improvements of the design itself.
- To support semi-automatic development of complex and high quality web applications.
- To support different off-line and on-line configurations for the delivery environment with different relative burden on the client or server of the application.
- To allow designers to work with application frameworks, rather than with specific applications

JWeb consists of the *development environment* and the *execution environment*. The development environment supports the definition of the application, and the collection of its content. The execution environment allows the actual execution of the application. An overview of the overall JWeb environment is depicted in Figure 1; the main components are explained in the next sections.

3.1 JWeb Development Environment

The development environment consists of four different components: the *schema editor*, the *mapper*, the *instance editor*, the *configurator*, and the *generator*. The *schema editor* and *configurator* are packaged within *HDM-Edit*; the *mapper* and *instance editor* are packaged within the *instantiator tool*, while the *generator* is a tool on its own. The mapper enables the definition of the logical schema and of the physical structure of the relational database that supports the application. This database has an editorial purpose and is devoted to store all information items related to contents or connections. The instance editor enables schema instantiation and the population of the editorial database, and ensure also the consistency between the inserted information items and the application schema.

JWeb does not support contents production. The use of standard authoring tools is the recommended way for developing information items (text, images, graphics, video, etc.).

The generator takes the XML descriptions from the configurator as input and produces the run-time database. The latter has a structure that is optimized to support application execution, while the structure of the database managed by the mapper is more suitable for application development and for collecting and keeping organized the content items as they are produced.

3.2 JWeb Execution Environment

The execution environment is responsible for the actual execution of the final application, possibly after some customization operations. It consists of a *navigation engine* and a *presentation engine*.

The presentation engine exploits a library of presentation templates (developed with standard multimedia authoring tools) to produce high quality interfaces. A simple presentation for an application can be obtained, with no effort, by using a number of default templates stored in a template repository of JWeb.

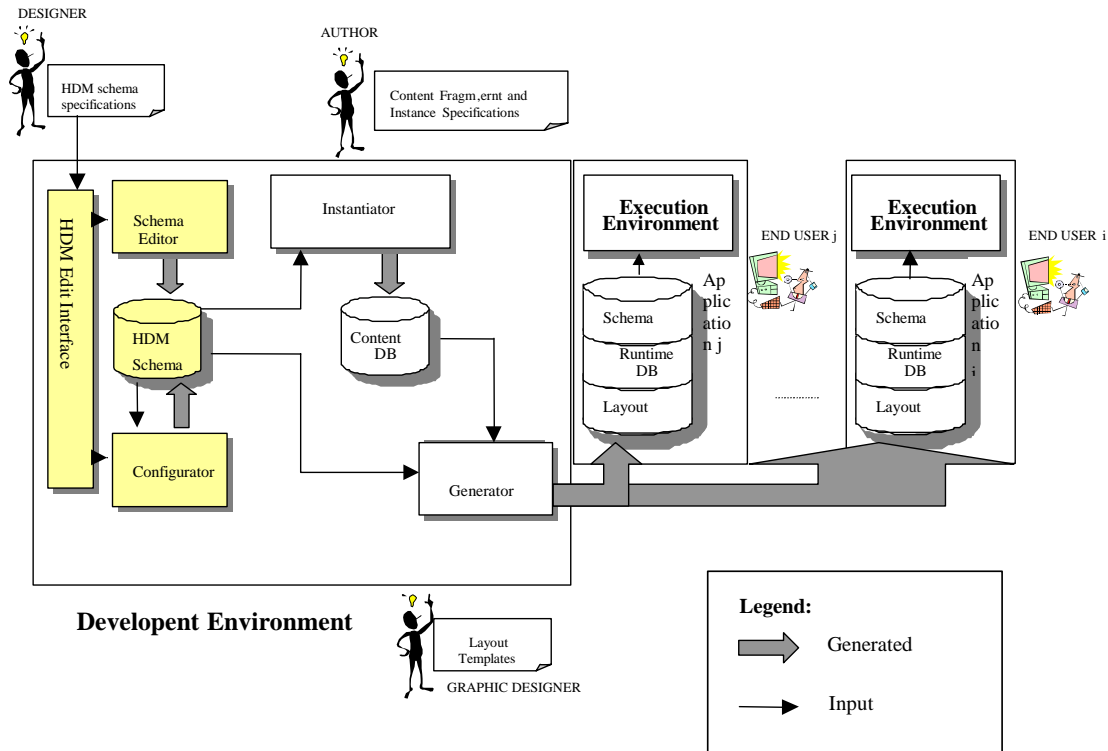


Figure 1: JWeb Software Architecture

The presentation interface intercepts user requests/actions and sends them, via the presentation engine, to the navigation engine in the appropriate format. Different presentation technologies are currently supported, including dynamic HTML, ASP, Java, and Shockwave.

The navigation engine accepts (via an XML based protocol) the requests coming from the Presentation Engine, and executes them. When a new logic page (corresponding to a node, for example) is being produced, it is delivered to the Presentation Engine, again using an XML-based format. For trivial presentation strategies the Presentation Engine will display the pages as they come from the Navigation Engine. For more sophisticated strategies, several logic pages, coming from the Navigation Engine, can be combined into a unique physical page. Logic pages are created by dynamically fetching information items from the Runtime database. Several different configurations for the navigation engine are feasible, in order to support different delivery requirements. It is possible, for example, to have both the presentation Engine and the Navigation engine on the same machine, or on different machines.

4 HDM

HDM-Edit supports design as envisioned by the HDM model, which is shortly introduced in this section. HDM

supports two sets of design primitives: to model the *structure* and *navigation* of an application. The structure provides the description of the information objects, their organization and their semantic interconnections. The navigation provides the description of the navigation paths among information objects. HDM-Edit does *not* support the design of *layout* (how the application looks alike), *interaction* (how the pieces of content react to user actions), *synchronization* (among multimedia information items), and *contents production and management*. All these aspects are addressed, directly or indirectly, by other parts of the JWeb environment.

A fundamental modeling feature of HDM supported by HDM-Edit, is the distinction between *hyperbase layer* and *access layer* of an application. The hyperbase layer consists of *entities*, which are the information objects where the main application contents is stored, and *semantic links*, interconnecting entities according to some domain dependent relationship. Entities can be structured in *components*, which, in turn, may have *sub-components*. A component with no sub-components is a *node*. A node, which is the unit of consumption, consists of elementary information items, *slots*, organized into aggregates, called *frames*.

In general, the designer does not specify the structure of each single object, but rather the structure of categories of objects, i.e., the features of *entity types* and *semantic*

link types. Moreover, there can be (a few) entities or links not belonging to any type: They are called *anomalous* in HDM.

Within the structure of the access layer, information objects are grouped into *collections* [8], to provide navigation contexts for them [23]. The objects belonging to a collection are called its *members*. Collections, again, can be grouped into *collection types* or can be anomalous (i.e., not belonging to a type).

For navigation design HDM, and HDM-Edit, distinguish among three categories of navigational links:

- *Semantic navigational links*, which define how to navigate across objects related by semantic relationships;
- *Structural navigational links*, which define how to navigate within large objects;
- *Collection navigational links*, which define how to navigate within collections.

The *Hyperbase Schema* describes entity types and semantic link types, and their corresponding structural and semantic navigational link types; the *access schema* describes collection types and collection navigational link types. For both schemas, there is a general level of description, called *in-the-large*, and a detailed level of description called *in-the-small*.

5 HDM-Edit

HDM-Edit enables designers to define all the different design features of an application as specified by HDM. There is not a predefined order of operations that must be followed by the designer, while using the tool; (s)he can work at different levels and on different parts of the design, simultaneously. There are only two obvious constraints:

- Before using an element X in a definition of element Y, X must be defined;
- There must be consistency among the different definitions.

Designs can therefore work bottom up, by identifying the basic information elements (slots) and grouping them into entities, or top down, by defining entities and progressively decomposing them.

During the design process, the designer can also associate additional informal contents to any part of the design (say an entity type or a collection definition, for example). This is made possible for two reasons: for documentation purpose, and to allow a designer to informally specify those features currently not supported by the HDM model. Formal definitions and informal descriptions can be visualized or printed at any time, both individually or in the form of design report, by collecting in an organized way all precise specifications and related documentation.

At any time, the overall result of the design activity can be saved, fetched and (possibly) modified.

Although the internal format for saving the design is proprietary, for efficiency reasons, a design can also be exported using an XML file. For such a purpose, an HDM DTD has been defined: The set of all tags that can be used to describe a HDM schema has been defined. Furthermore, an XML file can be imported as input for HDM-Edi if it satisfies the defined DTD format. Therefore the tool is able to manage the results produced by other systems, or to export application design specifications to other tools (either part of the JWeb environment, or independently developed).

HDM-Edit comprises two different modules:

- The *schema-editor*, which supports the design of a single application (either from scratch or from a pre-existing one);
- The *configurator*, which supports the development of application families, by implementing derivation and hyperviews, as we discuss in the next section.

Figure 2 shows the functional architecture of HDM-Edit.

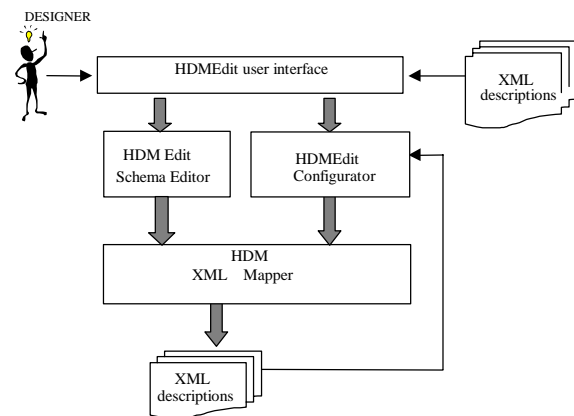


Figure 2: Functional architecture of HDM-Edit

5.1 User Interface

During the design process, the user works on a design desktop, which consists of a windows-like window, labelled with the title of the current design project. The desktop includes a menu area and a visualisation area (see Figure 3). The menu area provides some general functionality to create a new application design, to open an existing one, to save the current one, and to create or update single design elements. Since there are not rigidly predefined design procedures, at any time any (meaningful) operation can be performed on every part of the design schema.

The visualisation area displays a graphical representation of the various design elements that are progressively defined as the design process proceeds. Entity types are

represented in a tree like fashion, very similar to windows directories; intermediate nodes in the tree correspond to component and sub-component types, and terminal nodes represent slots (see Figure 3). Link types appear as arcs connecting nodes in entity type trees. The designer can display multiple views of the same schema, at different levels of abstraction (e.g., entity types names only, exploded entity types with their inner structure of components, possibly up to slot level, entity type names and semantic link types only, etc.). Introducing or updating a design specification is partially performed textually, by editing a dialogue box, and partially by manipulating visual elements in the visualization area.

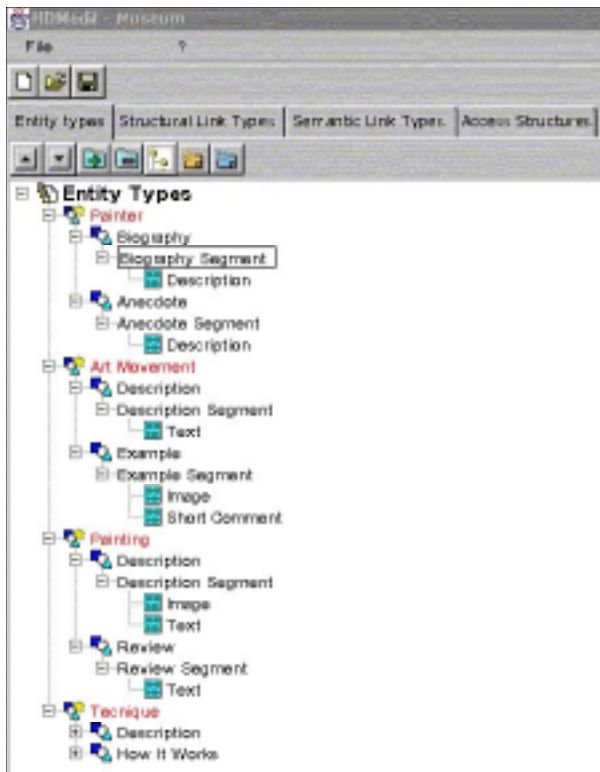


Figure 3: Detailed view of Entity types

In many situations, the design proceeds by invoking appropriate design patterns, as we discuss later. Let us suppose, for example, that the designer needs to create a (simplified) hyperbase schema for a virtual museum of modern art web site, as the one depicted in Figure 4. It includes four entity types: painter, painting, art movement, and technique, each one with an inner structure made of two component types, and a number of structural and semantic links. For simplicity, we can assume that the designer works top-down, first specifying entity types and their inner structures, then defining semantic links and finally describing the navigation links across and within properties of the various structures.

Entities are initially defined by providing, via a dialogue window, a set of attributes: name, used to univocally identify the entity type, description, which can be used to explain the purpose of the entity type and any relevant modelling comment, minimum and maximum number of expected instances, which are useful to plan the size of the final application, a tag (*structured* or *unstructured*) to know if the entity type is decomposed in several components, or it is a mono-component. As the attributes of a new entity type are defined, a visual feedback is provided to the designer, by updating the design desktop to include the representation of the new type.

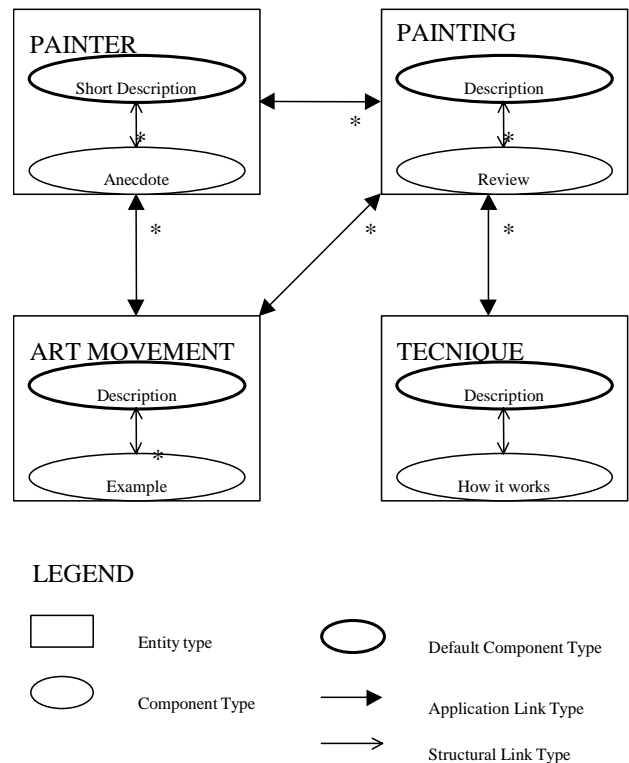


Figure 4: Hyperbase Structural and Navigational Schema in-the-large of the Virtual Museum

The inner structure of an entity type is defined by specifying its component and sub-component types. After visually selecting where, in the entity type tree, a new (sub) component type must be placed, the designer can specify its attributes: name, informal description, cardinality, the tag to indicate if it is a *default* component and another tag to indicate if it is structured in sub-components. Within an entity, a default component acts as the entry point to navigate to the entity and the starting point for structural navigation. As a new (sub) component type is defined, a new node is visualized in the proper position within the corresponding entity type structure tree. The

structural definition of entity types should be completed by defining their structure in-the-small, i.e., the actual data values – slots in the HDM terminology, assigned to the various terminal components. Their definition follows a very similar style and is omitted for the sake of simplicity.

A semantic link type is defined by visually selecting the (entity or component) types that it connects, and by defining its attributes: name, description, and cardinality (one-to-one, one-to-many, or many-to-many). The navigation properties of the hyperbase describe the types of actual navigation links that users can traverse to browse across entities related by a semantic link (semantic navigation), or to navigate within entities (structural navigation). Navigation properties of a given structure can be defined in HDM-Edit either link-by-link, or, more efficiently, by invoking a navigation pattern [9, 12, 20], i.e., by selecting a predefined design solution, which embodies all the navigational links needed in a given situation. If the semantic link is one-to-many, for example, there are at least three different design solutions for navigating across its interconnected structures (with a number of possible variants) [12]:

- From the source directly to each destination element (which can be selected from an index of destinations) and back to such an index, to access a different destination; this navigation pattern is called *index* in HDM-Edit
- From the source to the first destination, and from here to the next/previous one; this navigation pattern is called *guided tour* in HDM-Edit
- Both styles; this navigation pattern is called *mixed* in HDM-Edit

The possibility of using design patterns (and of defining new ones) provides several advantages:

- It speeds up the design process, since it avoids describing complex solutions piece by piece, in a time-consuming way. To define index navigation, for example, the designer needs only to invoke the corresponding pattern, instead of defining the details of all the many links that correspond to this navigational solution
- It improves the quality of the design, since the design solutions offered by patterns have already been tested and proved effective.

Design patterns in HDM-Edit are used in a number of different situations, for example, during the definition of the collection types that describe the access layer of the application. A collection type is specified by its name, the types of its members – entity types, components types, or, for nested collections, collection types – and the criteria adopted to group (and order) the members. Figure 5 shows the visualization of a number of collection types in

the Virtual Museum. Paintings are grouped by period, by painter, and by art movement (these collections are further grouped into a higher level nested collection), and also by more subjective criteria, such as because they are related to a given theme (see the collection of paintings *the war in the painters eyes*).

Collection design patterns defined so far concern navigation properties (very similar to the ones discussed for one-to-many semantic links) as well as structural aspects, such as the definition of the entry point elements to access collection members (called *center* in HDM). The reader is referred to [12] for a more complete discussion of some of the collection patterns supported by HDM-Edit.

Finally, beside pure modeling features, HDM-Edit supports technical features such as zoom in/out (to change the level of detail in the visualization), creation (deletion update) of elements, update, copy, cut-and-paste, save/retrieve, drag-and-drop, etc. These technical features enable the easy fetch of a previous design and the adaptation to new exigencies (i.e., design by reuse).

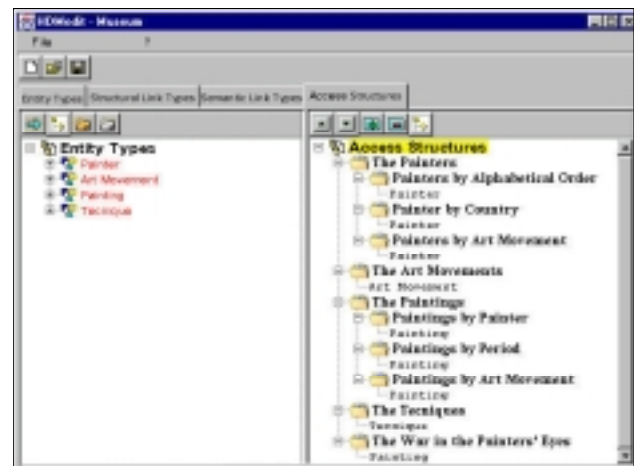


Figure 5: Detailed view of access structures

5.2 Advanced features

HDM-Edit supports two approaches to design the structure and navigation of an application. One consists in creating a new design from scratch by incrementally defining the relevant features of the application. Examples of this design style have been discussed in the previous section. A second approach allows designers to define a new application as a version or a specialization of an existing one. The advanced features of HDM-Edit support this second approach, and allow designers to derive new applications from a previously defined comprehensive schema. The main module in the tool architecture, which is responsible for supporting these advanced features, is the component configurator (see Figure 3).

The idea is that the designer defines the schema for a family of applications (i.e., (s)he defines an application framework [15]) and then derives each application when needed. For a large industrial consortium [16], for example, we are defining an application framework for producing commercial catalogues (off-line and on-line) for fashion and garment industries. Each single catalogue, derived from the framework, is considered a *hyperview* of the general schema. Another possible use of hyperviews is to allow each category of users of a given application (say, for example, of an on-line fashion catalogue) to have its own customized view of the application.

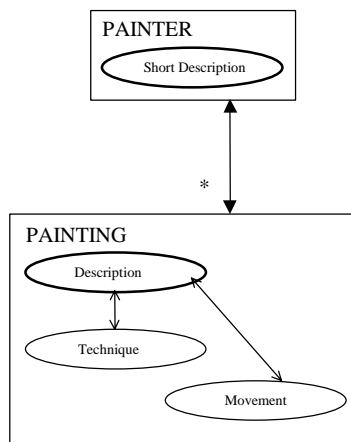


Figure 6: Hyperview of the Virtual Museum hyperbase schema

Figure 6 shows a hyperview defined on the application schema depicted in Figure 4. *Painter* is now a simplification of the original type and contains only the component type *Biography*. The entity type *Painting* derives from three original types of Figure 4. It draws the component *Description* from the original entity type *Painting*, *Movement* from the original entity type *Art Movement* and *Technique* from the original entity type *Technique*.

Another important advanced feature of HDM-Edit is the possibility of defining more than one version for the same application design. The main envisioned use for this functionality is the design of multi-language applications. A German version of an English site, for example, can contain its own version of the text items, which need to be translated, but it does not contain its own version of the images, that do not need any translation, of course. The tool is able to support flexible versioning to optimize the designer effort.

6 Implementation

HDM-Edit has been designed using the UML specification language [21], which favors the development of well-defined and accurate designs. It has been developed

using the Java Development Kit (JDK) version 1.2, which guarantees the complete portability of the tool on different platforms.

The system architecture is modular to favor the future expansions and modifications of the software. It is a *document-view* architecture type, which separates the data structure management from the graphic interface.

The hearth of the graphic interface is the class `HDMedit.java`, which defines the main window and the toolbar from which it is possible to activate some general functions, such as saving a project, opening a project, or printing the documentation.

The main application class is `Project.java` contains the data structure and the methods to manage an HDM design project specification, i.e., to visualize, modify, import, and save it. In particular, the data structure has been implemented using the Package `com.sun.java.swing.tree`, which provides the management methods and also generates its graphic representation. The HDM model primitives are coded through an XML DTD, which is managed and validated by the inner class `XMLParser`.

The usage of XML for describing an application schema offers several advantages:

- It allows designers to import (fragments of) application schemas, created by using HDM-Edit or other tools;
- It enables exporting of (fragments of) application schemas to HDM-Edit or to other tools.
- It enables the management of a large design effort subdividing it into more manageable modules.
- It enables the integration of the HDM-Edit tool into several possible toolboxes (JWeb is just an example of this possibility)
- It allows implementers to hide any change of the internal representation of a schema, maintaining a stable interface with the outside modules.

7 Empirical Validation

HDM-Edit has been used in different environments and for different purposes:

- It has been adopted by several classes of students, for doing homework in WWW design; the students come from two different universities: the Engineering Faculty at Politecnico di Milano e Communication Science Faculty at USI-Lugano (Switzerland);
- It has been used to design several real-life applications, by the industrial consortia of the European SITMOON project and of the national MURST TA2000 project;
- It has been used, in conjunction with other tools, to build some application prototypes, implemented afterward with different authoring platforms.

When the tool was used in a training/education environment, we could compare the results achieved by the students using HDM-Edit with the results of the classes of the previous years, not equipped with the tool:

On the positive side:

- In any situation of use of HDM-Edit, the design process has been faster than it was without the tool, and the design documentation has been more complete and of better quality.
- The design produced by the students using HDM-Edit has been more consistent and formally correct. Furthermore, the tool has greatly enhanced the student capability of understanding the HDM modeling features and the various details of the design process.

On the negative side:

- Sometimes the designers worried more about the tool itself than about the design process. We expect that this drawback can be overcome by extending the preliminary training of the tool, and by providing better user manuals.
- The fact that the model did not cover all the needed features made sometimes hard to understand how to use the tool. For example, if experienced designers learned immediately how to use the comment facility to overcome the current limitations of the design model, inexperienced designers tried, unsuccessfully, to bend the tool to their modeling needs. Again, this problem could be overcome by anticipating it to users, and by providing them with better examples of use in the tool documentation.

8 Conclusions and Future Work

Let us compare now HDM-Edit with the requirements illustrated at the beginning of this paper.

Expert designers. They find the tool very efficient for the initial phases of design or for fast prototyping; the tool has still to be improved to support sophisticated detailed design. Furthermore, our feeling is that a greater number of advanced design patterns is desired to improve the efficiency of the design process.

Hypermedia design instructors. The tool helps a lot (mostly when the students come from a non-technical background) in making the modeling concepts clear and unambiguous; the only danger is to make apparently less evident the conceptual part of design.

Students and novices. They can greatly benefit from using the tool, both in terms of efficiency than in terms of quality of design.

Let us examine now, more specifically, if and how the different technical and conceptual requirements has been

met by the current version of the tool, and which are the needed improvements.

Model Adaptability. The implementation of HDM-Edit has been reworked to enable a much faster evolution of the model being supported; this is still a crucial area of improvement, in order to make the tool acceptable in the long run.

Completeness Adaptability. The functionality of the tool has been revised, in order to allow different levels of completeness for the design. Still there are areas such as completeness check, in-the-large only (allowing the designer to explicitly skip all the details), documentation control (allowing the designer to specify which part of the documentation should be produced), and others, that need further developments.

Model Integration. The requirement has been satisfied in two ways: using comments as informal extension to the model; introducing design patterns (discussed later) to extend the descriptive power of the model.

Design Process Flexibility. A major area of intervention has been to make the tool very “un-prescriptive” about the sequences of operations that should be performed and the order to do things; in a sense we have done too much. If the experienced designer is satisfied of the current freedom of action, we have realized that students and inexperienced designers need a little more guidance. We are currently trying to figure out ways to provide “guided assistance” to novice designers.

Support for Design-by-Reuse. This is an area well supported, today, if a whole design must be reused. We are working at defining new features that will enable a better cut-and-paste of pieces of design.

Support for Design-Patterns. As we have already said we believe that this is a major issue, mostly for expert designers. Reusing, over and over, well known patterns often produces high quality design. For such a reason we are currently enhancing the design patterns currently available within HDM-Edit, and are trying to support also the creation of a well-organized user defined pattern library.

Support for Application Framework. This is mainly related to different ways of reusing design in-the-very-large. The configurator component has been added, to HDM-Edit, for such a purpose. A good level of hyper-viewing has been made possible, but this is not powerful enough for really advanced applications.

Documentation Support. The current document generation is satisfactory in several ways; still we need to make it more flexible, in terms of letting the designer to decide which parts should be generated and which formats should be used. Also we need to be able to import, within

the documentation, not only textual descriptions but also multimedia contributions such as pictures, sound, video.

Availability and Portability. Being implemented in Java (with only standard features having been used), the tool is quite portable and easily installable almost everywhere.

Integrability and Independence. HDM-Edit can export the design specifications in XML and also to import design specs in the same way; it is, therefore, very easy to integrate it with other tools.

Openness through standard. As we mentioned above, HDM-Edit is very open to cooperation with other design tools, since the design is described in XML, using a public DTD.

Delivery independence. HDM Edit can support schema-driven design toward all possible delivery environments. Other tools, within the JWeb environment, take care of the specific needs of different delivery architectures and environments.

Overall we think that HDM-Edit can be very effectively used as it is, for both introductory and advanced training of in-the-large (general) design and, integrated with JWeb, for fast prototyping. It has still to be improved to support more exhaustively all the detailed features of in-the-small design, and to provide also a number of auxiliary services. A help facility, for example, should be included, to explain modeling concepts or to suggest design examples in a context-dependent way.

Acknowledgments

This work has been partially supported by the European SITMOON project, by the MURST TA2000 project, and by the CNR "Museo Virtuale della Storia dell'Informatica Italiana" Project.

References

- [1] Bochicchio M.A., Paolini P., "An HDM Interpreter for On-Line Tutorials", in Proceedings of Multimedia Modeling 1998 (MMM'98), N.Magnenat-Thalman and D. Thalman, eds. IEEE Computer Society, Los Alamitos, Ca, USA, 1998, pp. 184-190.
- [2] Bochicchio M. A., Paiano R., Paolini P., "JWeb: an HDM Environment for fast development of Web Applications". In Proceedings of IEEE Multimedia Computing and Systems 1999 (ICMCS '99), Vol.2, pp.809-813.
- [3] Diaz, T. Isakowitz, "RMCASE: Computer-Aided Support for Hypermedia Design and Development", in Fraisse S, Garzotto F., Isakowitz T., Nanard J, Nanard M, (eds.) "Hypermedia Design (Proceedings of the International Workshop on Hypermedia Design (IWH'D'95)", Springer, 1996, pp 3-15.
- [4] Fraternali P., Paolini P., "A Conceptual Model and a Tool Environment for Developing More Scalable Dynamic and Customizable Web Applications", In Proceedings of Extending Data Base Technology '98 (EDBT98), Valencia (Spain) pp. 421-435.
- [5] E. Gamma, R. Helm, R. Johnson, J. Vlissides, "Design Patterns. Elements of Reusable Object-Oriented Software", Addison-Wesley, 1996.
- [6] Garzotto F., Paolini P., Schwabe D., "HDM - A Model Based Approach to Hypermedia Application Design", ACM Transactions on Information Systems, 11, 1 Jan, 1993, pp. 1-26.
- [7] F. Garzotto, L. Mainetti, P. Paolini, "Navigation Patterns in Hypermedia Data Bases", in Proceedings 26th IEEE Int. Conf. on System Sciences, Maui (HW, USA), IEEE Press, January 1993
- [8] Garzotto F., Mainetti L., Paolini P., "Hypermedia Design, Analysis and Evaluation Issues", Communications of the ACM, Vol.38, No.8, Aug.1995, pp. 49-56.
- [9] Garzotto F., Mainetti L., Paolini P., "Information Reuse in Hypermedia Applications", In Proceedings of ACM Hypertext'96 ACM, Boston (Ma, USA), March 1996, ACM Press.
- [10] Garzotto F., Paolini P., Bolchini D., Valenti S., "Modeling-by-Patterns" of Web Applications", In Proceedings of the First International Workshop on the World-Wide Web and Conceptual Modeling, Springer, 1999.
- [11] Gellersen H., Wicke R., Gaedke M., "WebComposition: An Object-Oriented Support System for the Web Engineering Life Cycle", in Electronic Proceedings of the 6th WWW Conference, Santa Clara, USA 1997.
- [12] Isakowitz T., Stohr E.A., Balasubramanian, "RMM: A Methodology for Structured Hypermedia Design", Communications of ACM, Vol.38, No.8, Aug.1995, pp. 33-44.
- [13] Johnson R. E., "Frameworks = (Components + Patterns). How frameworks compare to other object-oriented reuse techniques", in Communications of the ACM, Oct 1997, vol. 40, No. 10, pp. 39-42.
- [14] Fraternali P, Ceri S, Bongio A., "Web Modeling Language (WebML): a modeling language for designing Web sites", Proc. Int. Conf. WWW9, Springer Verlag LNCS, May 2000
- [15] Mecca G., Atzeni P., Masci A., Merialdo P., Sindono G., "The Araneus Guide to Web-Base Management System", Proceedings ACM SIGMOD Conference 1998, ACM Press, pp. 544-546.
- [16] Nanard J., Nanard M., "Hypertext Design Environment and the Hypertext Design Process", Communications of the ACM, Vol.38, No.8, Aug.1995, pp. 49-56.
- [17] Rossi G., Schwabe D., Lyardet F., "Improving Web Information Systems with Design Patterns", in Proceedings of 8th International World Wide Web Conference, Elsevier Science, 1999
- [18] Rumbaugh J., Jacobson I., Booch G., "The Unified Modeling Language Reference Manual", Addison Wesley Longman, 1999.
- [19] Schwabe D., Rossi G., "An Object Oriented Approach to Web-Based Application Design", Theory and Practice of Object Systems, 4 (4), J. Wiley, 1998.
- [20] Schwabe D., Rossi G., L. Emerald "Web Design Frameworks: An Approach to Improve reuse in Web Applications". In Proc. WWW9 Web Engineering Workshop, Springer Verlag LNCS, May 2000