An architecture model for the hypermedia engineering process

J. NANARD and M. NANARD
LIRMM, CNRS/Université de Montpellier
161 Rue Ada, 34392 Montpellier Cedex 5, FRANCE
Tel.: (33) 4 67 41 85 17, Fax: (33) 4 67 41 85 00
nanard@lirmm.fr

Abstract: We propose an architecture model for the hypermedia design and engineering process that combines incremental design of specifications and automatic generation in an experimental feedback loop. We have developed PageJockey a hypermedia design environment on this model. The designer may incrementally capture hypermedia specifications into templates by graphically interacting with the automatically produced target hypermedia and evaluate and refine the design. We introduce HLHSL, an object-oriented language which internally captures these high level specifications. We argue that the quick feedback between usage and design concerns makes it possible to address early and continuingly their mutual constraints in the development process, thereby leading to improve hypermedia quality at low cost.

Keywords: Hypermedia design, design process, specifications, prototypes, hypermedia generation, templates, architecture, reuse.

1. INTRODUCTION

Hypermedia is a highly intensive human-computer interaction technique to access documents. Designing good quality and cost effective hypermedia products for users implies to help the design process from early stage of design to full scale production.

As pointed out by the IFIPWG 2.7 (13.4), architecture is an important factor for such a purpose. In most cases, architecture concerns software or data. In this paper, we adopt a general view of architecture, as explained in (Shaw, 1995), including

The original version of this chapter was revised: The copyright line was incorrect. This has been corrected. The Erratum to this chapter is available at DOI: 10.1007/978-0-387-35349-4_22
S. Chatty et al. (eds.), *Engineering for Human-Computer Interaction*

Process models that focus on construction of target architecture. In this paper we present a model of the hypermedia design and engineering process that supports iterative refinement and abstraction, incremental design and reuse while allowing automatic generation from specifications. We have developed PageJockey, a hypermedia design environment based on this model and experienced its interest through several full scale hypermedia designs and developments. We describe also HLHSL (High Level Hypermedia Specification Language), an object-oriented specification language we have developed to internally support the process.

The paper starts by analyzing current results on architectural aspects of hypermedia design and development: no support is provided to the whole design process, especially to enable incremental design and development associated to a rigorous process based on specifications. We outline the principle of an architecture model of the hypermedia design and development process we designed to overcome this problem. We explain how PageJockey allows incremental design that benefits from an experimental feedback loop based on full-scale target product being designed and automatically produced from specifications. We detailed both the components of the design process architecture and automatic hypermedia generation from specifications. The main part of the paper describes HSHSL features.

Based on several applications designed and produced with PageJockey, we argue in the discussion on the interest of the proposed model on quality factors, such as consistency, reuse, maintainability, inclusion of good design patterns for hypermedia usability. These quality factors are important for managing high quality human computer interaction and decreasing design and development cost.

2. ARCHITECTURAL ASPECTS OF HYPERMEDIA DESIGN

Two classic architecture components are concerned during hypermedia browsing: the hypermedia "engine" (called also navigator) and the structured set of data accessed through the navigator, the hypermedia document called simply hypermedia in the following.

Works (Halasz, 1994) have introduced significant abstractions shared by typical hypermedia systems and powerful architectural principles for developing hypermedia engines and open hypermedia systems including systems relying on the web (Gronback, 1997). But they do not help building specific target hypermedia document structure, nor support the design process.

Today cost-effective manufacturing and high quality products for hypermedia presentations represent two major challenges for multimedia industry. Thus, studying the design process of hypermedia document is highly important.

The development of hypermedia documents by domain professionals is usually a sequence of two main independent steps: design and production (Figure 1). The design is done as a preliminary task which leads to a specification book. Some small scale sketching is usually performed. It is used to illustrate the specification rather than to produce an evaluable prototype.

The production steps are oriented towards intensive production without rigorous support. Very few changes are done on the final version. So, very little feedback takes place in the design process. Furthermore, most of production tools provides only low level abstractions with very poor support for incremental work. Late changes thus lead often to inconsistency.

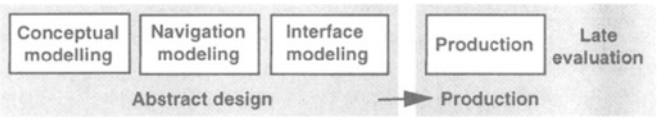

Figure 1 Approaches based on separation of design and production.

A great difficulty of hypermedia design is to produce in a straightforward manner a good design without feedback from end-users testing a full scale version of the product. But the production step of a hypermedia document is a very costly task, thereby the full scale evaluation step takes place only when the production is complete. Only very minor changes in the design can be done at this point. Very little feedback is used. To overcome this difficulty two directions are possible.

The first one consists in improving design methods and using rigorous formalisms and principles that have already proven their interest in software engineering. The stepping consists in elaborating a *conceptual data model*, an *abstract navigation model* and a *presentation and interaction model*. Object-orientation with OOHDM [Schwabe, 1996] or the Entity-Relationship Model in RMM [Isakowitz, 1995] add some interesting features but the overall underlying design stepping is globally similar to HDM [Garzotto, 1995]. In all cases the design deliverable is not a hypermedia document, but a specification book.

The second one consists in drastically reducing the production cost. Automatic structure transformation have been proposed (Kuikka, 1993, O2Web, 1996). Unfortunately they primarily rely on source document logical structure or database structure and do not take into account the reader's task to produce the hypermedia structure. Moreover, associated approaches do not take advantage of explicit evaluation and user feedback to improve design.

In this paper, we propose an architecture model for structuring the hypermedia design and engineering process that efficiently supports incremental design with user feedback loop and automatic production of target hypermedia from hypermedia specifications. This architecture is embodied in the PageJockey design environment that we have developed (Fraïssé, 1996) and used for producing several hypermedia documents (Nanard, 1998). This architecture relies on the feedback loop between the designer and an automatic production tool which spares the effort of production for the designer and enables her to focus on the incremental improvement of the work. It bridges the gap between the abstract design and the actual production.

The paper explains the architecture model of the design and engineering process, details the architecture of the automated production mechanism. The paper ends with a discussion on the main interests of the model for software quality factors directly related to usability.

3. AN ARCHITECTURE MODEL OF THE DESIGN PROCESS

The key idea of our work is to consider a design and production environment as an interface between the *designer* and the *design process* rather than as a tool to describe, edit and directly produce a target object. See Figure 2.

The model relies on three major points:

a) supporting incremental design based on full scale evaluation of the target hypermedia with real users,
b) providing a rigorous framework to specify design in order to enable automated generation for boosting the feedback loop and to make updating at full scale easy and cost effective,
c) keeping the interaction between the designer and its work as close as possible to his usual way of working.

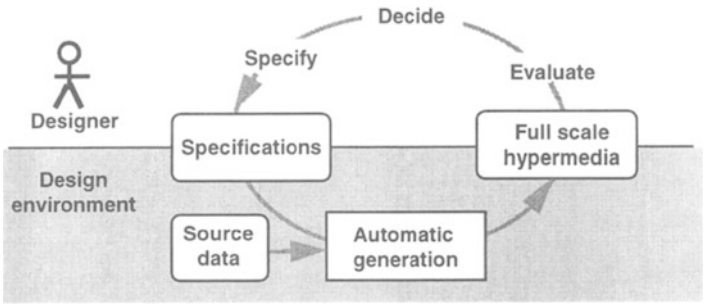

Figure 2 Cooperation between the designer and the design environment.

The design process is organized as a loop. To enable running the cycle as many times as suitable to achieve the wished quality, the cost of full scale production is made low enough so that most of the designer's effort for running a cycle only concerns the design. A *specification driven* automatic generation tool is responsible of full scale hypermedia production . Specifications are incrementally produced: the designer graphically sketches multimedia templates (Fraïssé, 1996) and the system produces their textual form and generates the target hypermedia.

Thus the designer can focus on its own expertise: create, evaluate and improve design without being hampered and slowed down by the burden of the production task. In this paper we no more develop the graphical aspect of the environment.

3.1 A rigorous framework for automated production

A rigorous framework is needed to enable specification driven automatic generation. Producing the hypermedia is specified and performed as a structure transform. The architecture of the production process relies on distinguishing three components:

- what the designer starts from: *the source data,*
- what the designer intends to produce: *the target hypermedia,*
- how the source data are turned into the target hypermedia: *the generation specifications.*

The *source data* consist in all of the raw material such as images, texts, movies and so on, that the designer uses to produce the target hypermedia. They have their own structure independently of their context of use in the target hypermedia. This structure is elicited in two parts. The *conceptual data model* and the *actual data structure*. The conceptual data model often already exists, for instance as a database schema when data are stored in a database or as a DTD (Document Type definition) when the actual data structure is elicited by a SGML markup.

The *target hypermedia* is the result of the design and production process. Its structure is organized in an object-oriented manner and relies on the well-known concepts of hypermedia: the navigation model (pages and links), the presentation (abstract contents and layout) and the behavior (animation, choreography, interaction with the user).

The *specification of the target hypermedia structure* is described in a generic manner as classes with the high level hypermedia specification language HLHSL. Any component of the target hypermedia is considered as an instance of a class whose structure is specified in a *template*. Libraries of templates and contextual libraries make it easier.

The triple (*source data modeling, generation rules, target hypermedia specification)* defines a mapping between the source data and the target hypermedia (Figure 3). It specifies how the target hypermedia is populated. A *trigger* associated to each class of the target hypermedia specification defines its generation rules. It specifies which properties, defined on the source data model, any of the instances of a given hypermedia component class must match. Thereby a parser automatically fires the class instantiation whenever the specified property occurs in source data.

Once the target hypermedia specification and the generation rules are made available in HLHSL, a generator transforms the source data into the specified hypermedia.

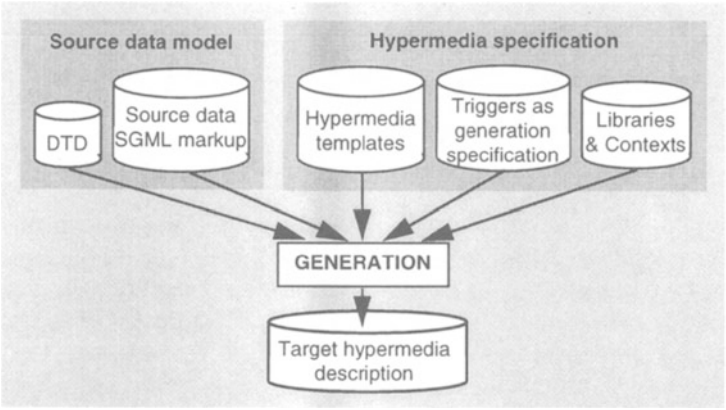

Figure 3 The architecture of the generation process.

3.2 Evaluation driven incremental design

The design process architecture is given in Figure 4. Evaluation operates on full scale generated versions of the hypermedia considered as drafts during an experimental feedback loop. According to evaluation, incremental improvements are introduced in the specifications or in the data or in their description. Since the incremental elaboration of the product is driven by its evaluation it is possible to iteratively refine the design. The speed and the extremely low cost of the automated generation process enables to check and evaluate as many design choices as needed. The designer may observe and evaluate -if needed with real users- the produced hypermedia at any stage of the design.

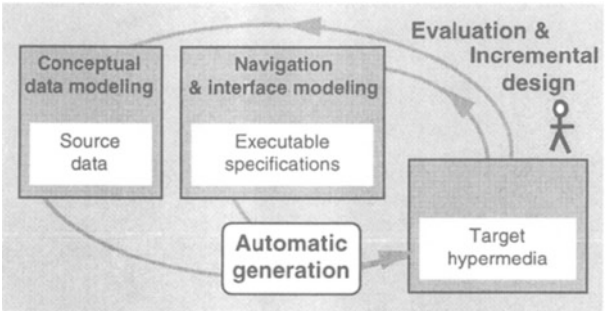

Figure 4 Generation from specifications with an experimental feedback loop.

3.3 Friendly interface for incremental template specification

The purpose of this architecture is to help the designer work at a high level of abstraction. Refining the design by simply updating specifications or data spares updating each target hypermedia component.

Nevertheless, elaborating textual specifications is often considered a tedious task for a designer. Thus, the architecture of the design environment is organized to:

- support the mainly graphical and incremental style of work of designers,
- provide the set of operations to step back and forth between graphical instances and their textual specification and to update either description,
- help capture as specifications the abstractions used by the designer.

We preserve the designer's natural way of working in the environment architecture. The approach for elaborating specifications relies on a visual description: the designer draws, the system helps him abstract, structure and reuse parts of the drawing. Specifying does not precede drawing, but results from the abstraction of the drafts which sketch the intended work.

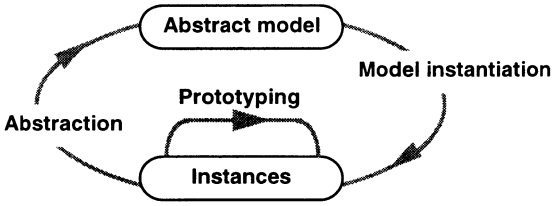

Figure 5 The interaction model for elaborating the specifications.

The formal mechanism used here relies on a visual prototype-based language. As shown on Figure 5, the designer can freely draw any object or structure, for instance some pages of the target hypermedia which are representative of the design. Such objects can be reused as prototypes by cloning[1] to build new ones. The system also helps turning any drafted objects into *templates* that are used for specifying the hypermedia structure and the generation rules. The selected object is turned (by the system with the help of the designer) into the model of a class which is later instantiated whenever needed in the design, improving by this way the overall consistency. For more details see (Fraïssé, 1996) and (Nanard, 1998).

The design environment provides the designer with the necessary operations for running the incremental abstraction / specialization cycle and to edit or reuse the graphical instances as well as their textual specifications (Nanard, 1995). The specification of the target hypermedia is a set of templates that the designer has abstracted from drafted objects produced to make the ideas emerge. Although

[1] Cloning is more than copy and paste: it induces inheritance between objects and thus makes consistency within the design explicit.

templates are stored as text for readability the environment helps to graphically specify a large part of them in order to minimize the programming effort.

The template mechanism offers a modular approach to encapsulate all of the information needed to specify and handle each abstraction of the design. A template captures the specification of reusable structures and components during design and rules for populating the target hypermedia. Thus changes are made easier since they can be done locally.

4. THE ARCHITECTURE OF THE GENERATION PROCESS

PageJockey internally uses HLHSL, a hypermedia description language. Owing to object orientation, HLHSL is both a language for describing a target hypermedia structure and a language for specifying it. HLHSL enables inheritance, and by the way, reuse. A classless approach (prototype-based) is also supported. Furthermore some hypermedia specific notions such as choreography are supported by the language.

Although HLHSL descriptions of either target hypermedia or hypermedia specifications are more often generated and interpreted by the machine rather than handwritten by the designer, the HLHSL syntax is easily understandable by humans.

A design issue in HLHSL was to provide support for expressing the structural relationships that are graphically sketched by the designer. Thus most of the constructs reflect how the designer graphically builds objects. The language elicits the dependencies between objects, supports incremental design and enforces reuse.

We explain how architectural concepts of the different components of the generation process are expressed in HLHSL and processed.

4.1 Target hypermedia description in HLHSL

The target hypermedia description produced by the generator is expressed in HLHSL. It is further interpreted to display the generated hypermedia.

4.1.1 Creation of pages and actors

In PageJockey a hypermedia is structured as a set of "pages". Each page represents a scene like in a theater play. A page is a dynamic and interactive document. Its components are *actors* which are texts, sounds, graphics, images, movies, and so on. Pages as well as actors are objects in the sense of object-oriented languages. Each page has a *Choreography* which specifies the behavior of the actors during the scene and the navigation or the sequencing of the scenes. The notion of choreography allows to group as a single entity all of the behavior of a page rather than disseminating it in many scripts.

```
<Page Named = Pg1>
  <Circle Named = C1>
```

```
  <Label Named = T1;  Hidden > This is a circle
  <Choreography>
     <When entering C1 do T1 appears during 3>
     <When Clicks T1 do LoadPage "Demonstration" during 3>
  </Choreography>
</Page>
The page named Pg1 contains a circle, when the mouse rolls over it,
a label displaying "This is circle" pops up during 3 seconds. if
the user clicks in the label, a link to page demonstration is
fired.
```

Any object, page as well as actor, is produced in one of the following ways:

- as an instance of a given class,
- as a clone of any other object used as a prototype,
- by grouping as a single entity several objects.

A set of primitive objects and classes are provided in the environment libraries to bootstrap the development. Thus a simple graphic object such as a circle simply is a clone of a primitive model of circle. An actor may be specified as a clone:

```
<Circle; Named = GlassLens;
LineSize = 3; LineColor = GoldenMetal;
FillColor = TransparentGlass;
>
The actor named GlassLens is specified as a clone of circle -a
primitive object in the libraries- overridden with specific color
attributes.
```

As soon as the designer names an object or an object property in the graphic window, the object or the property becomes an abstract prototype that can be reused and handled anywhere else. *Cloning* and *instantiating* are the key operations since they enable overall consistency and make specification simpler and more generic. They also better fit the designer's way of working.

Grouping allows to describe compound objects and to keep their structure visible. So, a compound object is described in terms of its components.

```
<group named = Glasses> <LeftLens><RightLens><Arms> </group>
A new actor called glasses is defined in an abstract manner.
```

4.1.2 Incremental design of objects

Any compound object can be modified in the following ways:

- *adding* new actors,
- *discarding* some inherited actors (they no longer belong to the new one),
- *overriding* some inherited actors to change their attributes,
- *substituting* an inherited actor by a new one which replaces it but still plays its role in the scene (basically this enables dubbing: the substituted item may even belong to a different class, but it keeps the links and the behavior of the initial one; this makes it simple to define place holders with specified behaviors. Substituting is not discarding plus adding!).

For instance:

```
<Glasses named = monocle ; group>
<Discard Arms> <Handle>
</group>
Discarding enables to define the notion of monocle. A monocle is a
kind of glasses without arms but with a single handle.

<Page Pg1 named Pg2>
<Substitute C1 Diamond>
<Override T1> This is a diamond
</Page>
The page Pg2 is simply a clone of the page Pg1 in which the circle
C1 is substituted by a diamond which have the same behavior as the
circle in Pg1. Overriding the text T1 enables to change the label
contents.
```

The features described above provide a simple, clearly readable, but expressive language to describe an actual hypermedia document. An important aspect of the language in terms of human computer interaction is its ability to preserve the intentions of the author in the structure description itself.

4.1.3 Behavior of actors

Another important point is the choreography section of pages which differs from most of other approaches. In many hypermedia production tools, the interactions are specified in scripts which are directly attached to objects and thus distributed in the whole document. Especially in Macromedia director, scripts are attached to movies, to frames, to cast members, to sprites..., making harder for the designer to remind why some sprite has a given behavior. In PageJockey, the explicit definition of page choreography enables the designer to catch at a glance the dynamic aspects of a page, like a choreographer does. The choreography is performed by individual on the stage but planned in a centralized manner by the choreographer.

4.2 Hypermedia templates specification in HLHSL

HLHSL contains features suitable to capture abstractions during design: for instance class specification, symbolic names for values, variable expressions to represent the values of attributes, and so on.

4.2.1 Attributes specification

Simple values for attributes often are represented by names:

```
<define GoldenMetal RGB% ( , , )>
```

The abstract notion of GoldenMetal color becomes available just by assigning it a name and adding it in the palette. Changing this color specification updates all of the objects which use this abstract color, not those who had the same RGB color. Remark that the designer does not write this sentence, she just operates on the palette representation.

4.2.2 Page template specification

The most important aspect concerns class specification. For instance:

```
<Page Named = Pg1>
…
</Page>
is the description of an actual page.
```

Conversely,

```
<Template Page Pg1; Parms =(&x, &surname, &dest)>
  <&x Named = C1>
  <Label Named = T1;  Hidden > This is a &surname
  <Choreography>
     <When entering C1 do T1 appears during 3>
     <When Clicks T1 do LoadPage &Dest during 3>
  </Choreography>
</Page>
is the specification of a page template which can be instantiated
by assigning actual values to the parameters &x, &surname, &dest of
its constructor.

<Instance Pg1,  Named = MyPg; Parms =(&x=Circle, &surname="Big
round", &dest="Pg12")>
is a page instance of that class.
```

Turning a prototype description into a template is done with the help of the designer who is the only one to know where are the variable parts of sketch. The designer graphically sketches a prototype of the object and then defines the variable parts.

The use of variable expressions in the specification of classes templates makes them play the same role as the constructors in O.O. programming languages. HLHSL also includes most of familiar notions such as iteration, alternative, dynamic arrays, and so on, which provides with the same power of expression as any programming language, though this language is especially shaped to describing hypermedia structures.

4.2.3 Generic navigation model in HLHSL

HLHSL uses the HTML like HRef tag to denote links. Defining expressions in a link description within a template provides with a powerful means to specify the abstract behavior part of the navigation model. The expression is elaborated according to the context in each instance.

4.3 Source data model and description

The source data consists of all of the raw material used by the designer and the producer to build a hypermedia document. By essence, the structure of these data is different from the target hypermedia structure, otherwise the hypermedia document would already exist. Most of the data which are present in a hypermedia document exist before being included in the hypermedia structure. Often, even when data are prepared especially for the document, they also exist as external data, for instance as

Photoshop images, or Adobe Premiere Movies. In order to define how to map this set of data into a hypermedia structure we need to describe the available data.

4.3.1 Source data conceptual modeling

Regardless to any presentation purpose, a semantic description of available data is needed in order to identify which role these data play in the communication process, and thus to be able to place them at the relevant place in the target hypermedia. The source data structure description elicits the logical and semantic modeling of data, it is not a description of the target hypermedia document like done in HTML. It is a detailed description of the relationships between the data. Descriptors of non textual objects such as images, video and sound files are also built in order to make their semantic available and handle them in the data model.

4.3.2 SGML-based Data description in PageJockey

We have chosen to use SGML for its flexibility. This enables at lower cost evolution of the source data during the design process. The readability of the markup language is an important advantage for the designers who are rarely computer scientists. It is easier to add tags into a marked up description than to write a program to fill up or update a database. On the other hand defining a SGML DTD or a database schema remains tasks which require some help from specialists.

Unlike HTML tags, the SGML tags used in the description have no presentation role nor hypermedia structure description function. No links are expressed in the source data description. For instance, here is a sample of a data file used for describing the data we used to produce a kiosk for the Musée National des Technologies in Paris.

```
<PERSON IMG =Lavoiser.face.Jpeg SHORTNAME=Lavoisier>
Antoine Laurent de LAVOISIER
<FUNCTION> French chemist .
<BIRTHDATE>1743 (Paris)<DEATH> 1794 (Paris)
<INSTRUMENT IMG ="Lavoisier.Gazo.Jpeg"
SHORTNAME=gasometer> Lavoisier's "gasometer"
<DESCR> This device, developed by Lavoisier,
can be used to precisely measure quantities of gas.
<EXPERIMENT >By isolating mercury and oxidizing it through heating,
Lavoisier noticed that some of the air disappeared, and that the
oxidization process then stopped. What remained was a gas that was
unable to support life: nitrogen. In contrast, when the mercury
oxide was taken and broken down, the initial quantity of gas
reappeared and the air recovered its usual properties. The gas in
question was oxygen, which can oxidize mercury and maintain
combustion.
<APPARATUS NAME=gasometer> Lavoisier brought the problem down to
measuring the volume of a variable cavity filled with a gas at a
pre-determined pressure and temperature. The variable cavity is a
bell jar filled with water, which is kept upside-down above a tank
by a balance. The water ensures that the variable cavity is
airtight and plays a role in the measuring process. The gas
introduced into the gasometer expels the water from the bell jar
until the pressures are balanced, thus altering the volume of
confinement.
```

Let us remark the semantic role of the tags in the description. The items above take place in several pages of the hypermedia document and some of them are reused in several places.

4.4 Generation specification

To better fit the designer's mental scheme, the generation specification is associated to the target hypermedia specification rather than to the source data description. This architecture choice is rather unusual since most of generation tools usually attach the generation actions directly to the grammar that defines the structure of the source data. As a consequence, the actions concerning a specific target item often are distributed in the code, making their maintenance more complex.

At the opposite, in HLHSL, a template encapsulates all information needed to produce and manage any instance of the described class. Generation rules are directly described as a specific part within the templates, improving maintainability.

Expressing generation specification from the target side is an architecture choice that has the same purpose as most of PageJockey's features: be an interface to the designer's work. The designer expresses any thing on the closest side and the environment acts as a functional core which converts the designer's intentions into the needed actions.

4.4.1 Specifying generation rules in HLHSL

To each template is associated one or several triggers which are a specification of the conditions in the source data files that lead a page of the class defined by the template to exist. A trigger specifies:

- an activation condition which expresses the invariant properties that the set of variables of the template must match in any of its instances,
- a rule for computing a unique identifier (UID) for each node instance according to the data,
- the way to compute the actual values of the template parameters.

```
For instance,
<Template page PGAboutScientist>
<Trigger Cond=&PERSON,   UID = &("Scientist-".&Shortname)>
<Attrib &ScientPortrait=&Img>
<Attrib &ScientLabel=&(&PERSON , \n,
&BIRTHDATE, &DEATH, /n, &FUNCTION>
…
</Trigger >
is a trigger in a page template.
```

Figure 6 shows a generated page with different user interactions.

Figure 6 Page generation example.

4.4.2 Generation rules processing

As shown in Figure 7, the source data SGML description is first parsed in order to build a derivation tree. The values of each attribute are referenced by their path name along the derivation tree. The generator explores the tree and checks for each node if a triggering condition fires in the current context. If so, the template attributes are assigned as specified from the actual values collected in the context of that node in the derivation tree. Then the template class is instantiated. The instance name is computed as declared in the template UID. If an instance with this name already exists, its attributes are just updated accordingly. This allows the designer to declare several triggers for a single template, or to fire a given trigger several times for the same page. For instance, indexes are incrementally filled-up in this way whenever relevant information is found in the source file.

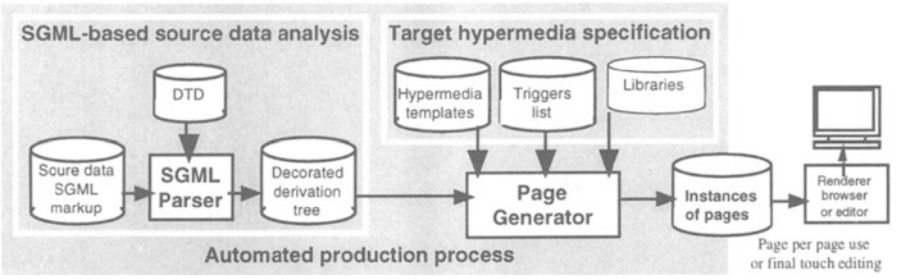

Figure 7 The internal production stages of the template-based generator.

5. DISCUSSION

The architecture choices of PageJockey design rely on the following ideas:

- A designer's task is to design, not to produce, but often a designer sketches drafts for designing.
- A good design needs validation and improvement by evaluation feedback.
- A design environment is an interface between a designer and the designed work. Very few systems take these simple and obvious ideas into account.

5.1 Generating: from source data to target structure

A trend in software engineering is to consider that the designer should focus on design, not on final code production (Budinsky, 1995, Winograd, 1995). For this purpose, PageJockey strongly dissociates design and generation. Generation is only the environment concern. Moreover this enables the designer to take advantage of automated generation to support incremental design.

Some systems (Kuikka, 1993, Feng, 1993) also propose automated generation approaches but they directly associate generation specification to the source data, thus making difficult to follow a task-centered design (Paterno, 1997). In PageJockey, templates result from user's task elicitation. Other systems enable to produce target documents for different platforms or to adapt cosmetic aspects to the end users taste. For instance, style sheets enable a separation between the contents and the presentation (Rutledge, 1997). But the hypermedia structure remains the same and is often already made explicit in the data description. Work by (Kristensen 98) uses the current servlet mechanism (Sun 97) to implement specific kernel for controlling dynamic page production in response to http request. They introduce TML, a template markup language, but it does not support object-orientation. Furthermore their notion of template only concerns documents and there is no environnement for designing high level abstractions such as navigation structure. Thus, such techniques are helpful only to the back end of the production process and not to the design step.

At the opposite, PageJockey fully separates source data from the target hypermedia specification and claims that these structures are independent. Generation specifications, as an intermediate layer, make the separation clearer. PageJockey generates any level of structure. Moreover, one may produce several hypermedia from the same data, or reuse a subset of hypermedia structure specification to turn other data into a hypermedia which shares the same structure. This enables incremental design. The fast generation tool enables the designer to update at will either the data or the hypermedia structure specification and evaluate her design. A real application of 500 nodes is generated in one minute on a G3.

5.2 Elaborating specifications

Bottom up approaches in design are suitable as well as top down ones (Nanard, 1995). PageJockey supports both. The designer can work at the level of instances and use them as clonable prototypes without needing to define abstract classes. Conversely, the system helps turning prototypes into templates. Once classes are defined as templates, top-down design becomes easier. Alternating between both is simple and supported by the system. This enables to reduce the gap between expression of specifications and production: the designer expresses abstract specifications but directly observes their real effect at full scale.

5.3 Maintainability, consistency, reuse

Templates is an efficient mechanism to capture and express at a high level of abstraction shared structure, look and behavior that a designer sketches on the early drafts. It enables to evaluate them at a larger scale and thus improves the efficiency of the evaluation process, by providing earlier feedback.

Separation between source data, templates and their generation rules promotes reuse and solves the problem of information reuse context (Garzotto, 1996) since source data modeling does not include specification of use in the target hypermedia. The templates mechanism and the underlying object-oriented approach helps improve consistency by reuse. Inheritance is responsible for vertical consistency. Inherited elements have the same presentation rules and the same behavior whatever be the context in which they appear. Instantiation is responsible for horizontal consistency. All instances of a class share the same structure. Consistency is needed in hypermedia design in order to help the reader build up a mental model of the explored structure. This is an important factor for usability [Nielsen 89].

Templates provide good and reliable maintainability conditions. Any change done in a template is propagated wherever the template is used. So consistency is preserved. This clearly is a key factor for improving quality of the target hypermedia. Systems which do not provide with mechanisms to automatically propagate changes, raise the risk of introducing inconsistencies and require a heavy maintenance effort which is incompatible with incremental design.

6. CONCLUSION

A design and production environment is not simply a tool for recording the result of a design that a designer elaborates only in his mind. It must support the design activity (Winograd, 1995), especially the evaluation feedback which is a characteristic of all human creative activities, and relieve the designer from the burden of producing for evaluating the design choices. The described architecture is used in the PageJockey environment and has proven its usefulness in several real scale developments: "The computer science pioneers" (Fraïssé, 1996), "From the laboratory to the home" for the National Museum of Technology in Paris (Nanard, 1998). Beyond the currently available features, many issues are open:

- Use directly the components of the design process loop as a support for design rationale (Bœhm, 1995): it is possible to use the same technique to manage both the product and external information used during the design process and to organize information about the design as a hypermedia easy to explore.
- Generate features for recording users behavior and analyzing it during the evaluation steps directly into the generated target hypermedia. This enables evaluation to rely upon objectified data. The template mechanism makes it possible to specify control and record users actions wherever suitable.
- Extend the template mechanism in order to use it at a higher level of abstraction to handle design patterns (Gamma, 1995). Many hypermedia share the same

design patterns. Beyond identifying these patterns (Rossi, 1997) providing a simple mechanism to efficiently implement them (Nanard, 1998) is a very promising topic. Templates constitute one of the ways to study for promoting reuse and developing frameworks (Schmidt, 1997).

7. REFERENCES

Bœhm, B., Bose, P., Horowitz, E., Lee, M.J. (1995) Software Requirements Negotiation and Renegotiation Aids: A Theory-W Based Spiral Approach. *Proc. ICSE'95.*

Budinsky, F.J., Finnie, M.A., Vlissides, J.M., and Yu, P.S. (1996) Automatic code generation from design patterns. *IBM Systems Journal*, Vol. 35, N°2.

A. Feng & T. Wakayama. Simon: a Grammar-based transformation system for structured documents (1993). *Electronic Publishing 6(4)*, 361-372.

Fraïssé, S., Nanard, M., Nanard, J. (1996) Generating hypermedia from specifications by sketching multimedia templates. *Proc. Multimedia'96*, ACM Press, 120-124.

Gamma, E., Helm, R., Johnson, R., Vlissides, J. (1995) Design Patterns: Elements of Reusable Object-Oriented Software. Addison-Wesley, Reading.

Garzotto, F., Mainetti, L., Paolini, P. (1995) Hypermedia design, analysis and evaluation issues. *CACM 38(8)*, 74-86.

Garzotto, F., Mainetti, L. Paolini, P. (1996) Information reuse in hypermedia applications. *Proc. Hypertext'96*, ACM Press, 93-104.

Gronbæk, K. (1997) Designing Dexter-based hypermedia services for World-Wide Web. *Proc. Hypertext'97*, 146-156.

Halasz, F. & Schwartz, M. (1994) The Dexter Hypertext Reference Model. *CACM 37(2).*

Kristensen, A. (1998) Template resolution in XML/HTML. *WWW7*, Elsevier Publisher.

Kuikka, E. & Penttonen, M. (1993) Transformation of Structured Documents with the Use of Grammar. *Electronic PublishingVol. 6(4)*, 373-383.

Nanard, J. & Nanard, M. (1995) Hypertext Design Environments and the Hypertext Design Process. *CACM 38(8)*, 49-56.

Nanard, M., Nanard, J., Kahn, P. (1998) Pushing Reuse in Hypermedia Design: Golden Rules, Design Patterns and Generic Templates,*Proc. HT'98,* ACM Press, 11-20.

Nielsen, J. (1989) Tools for generating consistent user interfaces in Coordinating User Interfaces for Consistency, Academic Press, 107-130.

O2Web (1997) *User Manual.* O2 Technology.

Paterno, F., Bucca, M.F. (1997) Task-oriented Design for Interactive User Interface in Museum Systems. *Proc. ICHIM'97*, Le Louvre, 271-279.

Rossi, G., Schwabe, D., Garrido A. (1997) Design Reuse in Hypermedia Applications Development. *Proc. Hypertext'97*, ACM Press, 57-66.

Rutledge, L., van Ossenbruggen, J., Hardman, L., Bulterman, D. (1997) A Framework for Generating Adaptable Hypermedia Documents. *Proc. Multimedia'97*, ACM Press.

Schmidt, D.(1997) Object-Oriented Application Frameworks,*CACM 40(10)*,32-38.

Schwabe, D., Rossi, G., Barbosa, S.D.J. (1996) Systematic Hypermedia Application Design with OOHDM. *Proc. Hypertext'96*, ACM Press, 116-128.

Shaw, M. (1995) Abstractions for Software Architecture and Tools to Support Them. *Proc. of the 1st Int. Workshop on Architectures for Software Systems.*

Sun (1997), The Servlet API, Sun MicroSystems, *http://jserv.javasoft.com/products/java-server/servlets/*

Winograd, T. (1995) From programming environments to environments for designing, *CACM 38(6)*, 65-74.

8. BIOGRAPHY

Jocelyne Nanard is professor at Universite' de Montpellier (France). Her research interest covers HCI aspects of Hypermedia design, Hypermedia modelling, and design patterns.
Marc Nanard is professor at Centre National des Arts et Me'tiers of Paris and Universite' de Montpellier. His research is in Interaction with documents, Hypermedia design methodology and development tools. He is chair of the ACM SigWeb, Special Interest Group on Hypermedia and the Web.

Discussion

Fabio Patterno: What assumptions do you make to build a page, because, for example, I don't think your environment can work with applets.

Jocelyne Nanard: The environment works with SGML implementation only.

Joelle Coutaz: What happens if the source data does not match the constraints expressed in the template?

Jocelyne Nanard: We must look at consequences of design decisions. We can modify data descriptions or the data.

Joelle Coutaz: Do you support temporal constraints such as Allen's operations?

Jocelyne Nanard: Yes. We can specify them by visual programming on concrete objects and then replace references to concrete instance objects in the generated code by variable names when abstracting the template.

Franck Tarpin-Bernard: For the moment, designers explicitly decide to view a page or an item as a pattern. Is it possible for the system to automatically identify patterns?

Jocelyne Nanard: No, but it is a future issue.

Stephane Chatty: Have you given thought to the extent of the systems your language can describe; in other words, what would you need in addition to your language to describe an interactive system?

Jocelyne Nanard: We don't handle computation properly. Apart from that, we handle user actions pretty well.