

Pushing Reuse in Hypermedia Design: Golden Rules, Design Patterns and Constructive Templates

Marc Nanard, Jocelyne Nanard

LIRMM, Univ. Montpellier
161 Rue Ada
F34392 Montpellier cedex 5, FRANCE
Tel: (33) 4-67-41-85-17
E-mail: nanard@lirmm.fr

Paul Kahn

Dynamic Diagrams,
12 Bassett Street,
Providence, RI 02903, USA
Tel: 401-331 2014
E-mail: paul@DynamicDiagrams.com

ABSTRACT

Reuse is increasingly strategic for reducing cost and improving quality of hypermedia design and development. In this paper, based on the design and development of a real hypermedia application, we classify and explore different types of reuse in hypermedia design. We show how constructive templates constitute a practical technique for capturing the specification of reusable structures and components and enabling the automation of the production process. We also discuss connections between constructive templates and design patterns.

KEYWORDS: Hypermedia design, golden rules, design patterns, templates, reuse, hypermedia generation.

INTRODUCTION

Reducing the design and development cost of new hypermedia documents while improving their quality is an important challenge for the information industry. All these improvements implicitly rely upon reuse.

Thus far, reuse in hypermedia design has focused on information reuse [9] and software component reuse through models embedded in hypermedia development environments and systems [11, 18]. We hope to extend the principle to include the capture and reuse of design experience as well. To this end, in [23], the authors argue in favor of discovering and describing hypermedia design patterns. Their idea is to follow the promising technique of *design patterns* used in software engineering object-oriented design [7, 21]. However, where design patterns provide reuse of architecture at a very high level of abstraction, designers also need practical techniques to capture the specification of reusable structure during the design process and easily reproduce these structures in the target hypermedia.

For this purpose, we introduced constructive templates in PageJockey®, a hypermedia design environment [6].

Permission to make digital or hard copy of all or part of this work for personal or classroo 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.
HyperText 98 Pittsburgh PA USA
Copyright ACM 1998 0-89791-972-6/98/ 6...\$5.00

These templates provide the designer with means for capturing the specification of reusable structures and components during the design process and then populating the target hypermedia. Thus, templates act as a bridge for reuse from design to implementation.

In this paper, we discuss a new classification of reuse types in hypermedia design and focus on reuse of design experience. We explore and illustrate these types of reuse through a real hypermedia design example. We show how constructive templates help push reuse into action by capturing the implementation of design patterns and automating their use, thus supporting the hypermedia design and implementation process. Finally, we discuss the relationship of reuse types to design patterns as well as newly opened questions.

REUSE IN HYPERMEDIA APPLICATION

“Reuse consists in taking advantage of any of the efforts done for previous works to reduce the needed effort to achieve a new one.”

Reuse in hypermedia design concerns data, software components within systems as well as design experience. The reuse of data such as the sharing of already produced pages, pictures, movies, and so on, has already been studied [9]. Recent works in open hypermedia have shown the interest of sharing hypermedia operating system services [18] in the perspective of hypermedia as a computing paradigm. Except for design methods [8], few works are concerned with reuse of design experience from a methodological point of view. The aim of this paper is to focus on reuse of the designer's know-how. The challenge is to design better hypermedia documents more easily and reduce the design and development cost.

Basically, we distinguish between the reuse of several major kinds of know-how, from the most general levels to the most specific ones (See Figure 1):

- **Methods (general/design):** reuse guidelines to proceed through the steps of the design activity. Design methods result from experience collected by hypermedia design pioneers and their methodological studies of design. They rely on a design model that provides concepts to represent products of the different steps. In the early nineties, the HDM model [8] was proposed to make the steps of the designer's task clearer and thus improve cost efficiency.

Together with works on hypermedia design [14], this research coined the fundamental notions of *navigation*, *presentation* and *interaction model*. Several variations of HDM have been suggested [23, 12]. These HDM-like methods typically promote reuse of general know-how.

- **Golden Rule (specific/design):** reuse of design skill which consists in recognizing situations already encountered in order to similarly handle new situations. A *golden rule* is a high level principle that can be applied to solve a problem. It provides a starting point to evaluate particular design constraints.
- **Design pattern (general/implementation):** reuse of general purpose abstract macro-structures that have been recognized as good practice in many cases. This kind of reuse, originating in architecture and applied successfully in CASE, is identified as *design patterns* [7]. Design patterns express structure and collaboration of components in the architecture of systems, applications and documents. Thereby, they aid the development of reusable components and frameworks [22].
- **Templates (specific/implementation):** reuse of specific and well-identified structures in a target document. These structures, which we call *constructive templates*, can be specified in a generic manner to automatically generate instances throughout the document.

<i>Range</i> <i>Nature</i>	<i>Towards</i> <i>Design</i>	<i>Towards</i> <i>Implementation</i>
<i>Abstract</i>	Methods	Design patterns
<i>Concrete</i>	Rules	Templates

Figure 1: Methods, golden rules, design patterns and templates as know-how reuse.

All four elements are complimentary factors in the design and implementation process. Methods and golden rules are used to develop the storyboard or planning document while design patterns and templates provide the bridge between design and production of the application.

THE EXAMPLE USED IN THIS PAPER

During summer 97, a team from Dynamic Diagrams and LIRMM designed and produced a hypermedia application to be installed in a kiosk for the Musée des arts et métiers in Paris. The brief provided by our client at the museum was flexible but imprecise: “present up to 20 antique objects from the museum collection, along with information about their inventors, and illustrate how the discoveries embodied in these objects are useful to contemporary life.” The curator suggested a preliminary title to establish the mood: “Has the scientific instrument disappeared?”

We took this opportunity to study the actual design process followed by domain experts in order to track reuse aspects in design and production. We kept a precise log of various changes in the designed sketches and versions of the kiosk and of the design rationale expressed during joint work sessions as well as in the mail exchanged between the design and

production teams in Providence and Montpellier and between the designers and the museum clients in Paris.

This real example of design¹ is used throughout this paper to analyse how different techniques can support the reuse of the various kinds of know-how required during hypermedia design. We focus on reuse through template-based techniques and show how this can drastically reduce the development time and improve the consistency of the target product.

We have chosen a few simple and relevant examples from this project. These examples should not be seen as literal guidelines but rather as illustrations of the ideas of design reuse proposed in this paper.

GOLDEN RULES

It is not our purpose in this paper to discuss design methods. We followed an HDM-like method at the start of the project. Most of the work we discuss in the paper concerns the actual presentation and interaction models.

In contrast to design methods which only structure how to proceed, golden rules guide the choice of concrete structure elements and presentation styles. Thus the application of these rules promote a type of reuse that directly effects the target document.

A golden rule have three parts: Motivation, Precondition, Action. Once the designer recognizes a situation that matches a condition to which the rule applies, she or he executes the specified action. The motivation elicits participation in the design rationale.

Example of Golden Rules

As an illustration, we list here three of many golden rules we used in our kiosk application (Figure 2) and explain in the next section why and how we used such rules. Note that the purposes and effects of rule 3 are studied in [25].

<p><i>Rule 1</i></p> <p>Rule name: Clustering</p> <p>Rule motivation: Respecting the limits of short-term memory by avoiding the presentation of more than 7 items simultaneously. Larger sets are unpleasant to access.</p> <p>Rule precondition: Providing access to a medium sized set (from 10 to 50 elements) of similar elements.</p> <p>Action: Try to elicit a sub-structure to build semantically consistent and well balanced subsets, each of them with no more than 7 elements and 7 sets.</p>
--

¹ Images of the finished kiosk application, “La science au foyer / From the laboratory to the home” are presented on the websites of LIRMM (www.lirmm.fr) and Dynamic Diagrams (www.dynamicdiagrams.com). Samples of the templates used for generating it are also presented on the Web.

Rule 2

Rule name: Logical glue

Rule motivation: Small information sets need to express a meaningful structure to avoid being perceived as an arbitrary grouping.

Rule precondition: When a small, but arbitrary, set of information (less than 10 items) has to be displayed together.

Action: Try to make apparent the reason why items in a presentation set fit together.

Rule 3

Rule name: Logical glue consistency

Rule motivation: Homologous strategies should be used in similar parts of the design in order to help the reader build up a mental model of the structure.

Rule precondition: When a structure is made visible for the reader at any point in a design.

Action: Use a homologous structure for each cluster of the given information set.

Figure 2: Some golden rules used in the kiosk application.

Putting Rules into Practice

Such rules were used in the kiosk design. The records of the design activity assert that, at a very early stage, even before working on the navigation model itself, we recognized a situation where such rules applied:

Rule 1 and 2: The museum curator expected 15 to 20 antique objects to be presented in the kiosk. These are too many entries for short-term memory and would have been difficult or boring to review without a higher-level grouping. We decided to structure the set of instruments into six clusters, each containing three 'antique' objects. The logical glue for each of the six clusters was a single 'modern' object related to each of the three antique instruments.

Rule 3: We searched for a consistent glue we could apply to each cluster to tie the antique objects to the modern one. We

created a simple narrative organized around a set of three verbs. The verbs explain the relation of the antique instrument to scientific principles behind the modern object.

Lavoisier's gasometer provides the technique used today to *measure* a volume of gas, the escapement mechanism of a Huygens clock shows how a mechanism *counts* the number of a measurement, while Pascal's adding machine uses gears to *display* on dials a cumulated decimal value just as the modern gas meter does. Furthermore, the verbs appear in relation to the antique instruments. This aims at providing a scientifically rigorous story which appeals to the reader's curiosity by unexpected visual association.

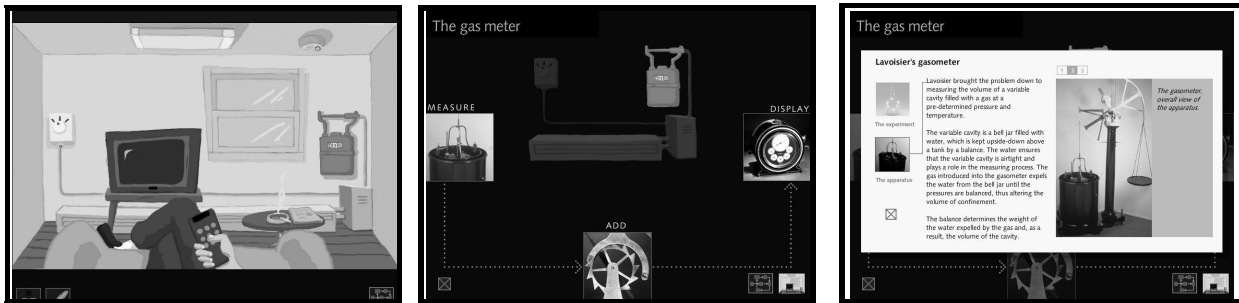
Rule 2 was applied again to glue together the 'modern' items in a first level index. This was accomplished by creating an animated drawing of a home containing the six modern items. It also makes the initial theme of the kiosk visible: "scientific instruments are hidden in our everyday life."

Figure 3 shows the result of applying such golden rules.

Golden Rules as a Kind of Reuse

We observe that golden rules do not directly offer a solution but structure the way a solution can be found. Often a group of rules, such as rules 1-3 above, are considered together. Golden rules are used to elicit constraints and check solutions. It is the designers' imagination and domain knowledge that is used to develop the abstract scenario. We have observed that a golden rule may encapsulate an instance of know-how collected in previous work and can be responsible for a meta-design style in the final product. Although the use of a given golden rule in different contexts may lead to very different results, all documents where these rules are applied share a single meta-structure.

For instance, another team was working on a kiosk application about 'the photographic portrait' for the same museum. They applied the same golden rules for grouping information to structure their design: a picture of a Photograph's shop, with cameras, photos, books, and so on as anchors in the first level of their kiosk.



A 'room' metaphor is used to present the first general index to six modern instruments. Each object (a thermostat, a gas meter, a smoke detector and so on...) is an anchor to a 'modern' instrument page. Such a page is a sub index called a 'hub'. For instance, the gas hub gives access to the cluster of antique instruments which are tied to the gas meter. Lavoisier's gasometer is anchored to the verb MEASURE, which leads to an apparatus detail page, giving access to two subsections, the experiment and the apparatus, which both have the same overall look.

Figure 3: Using golden rules to structure the kiosk application.

The Orsay Museum CD-ROM also uses a clustering by painting school to reduce the number of paintings in each index page [2]. Although these hypermedia applications look very different, they all rely on the same principle: the reduction of choices to a relatively small index table. At the opposite extreme, hypermedia documents with long index tables, such as alphabetically organized encyclopedias, do not use this set of rules.

Unfortunately, golden rules are hard to automate. Producing rule-based experts systems that help the design process requires a formal description of the context and of the design choices. This is far more costly than informally applying rules. Experienced designers are so aware of these rules that they do not need to analyze the situation, but can apply these rules instinctively.

Nevertheless, golden rules are very useful to teach hypermedia design. They help elicit and reuse a part of designers' know-how and help novices to understand why, when and how to make a given choice. Golden rules are like ergonomic rules in Human Computer Interaction. As we will show, they also lead to a more precise application and selection of some design patterns. For instance the presented golden rules led to the building of some navigational contexts [20].

DESIGN PATTERNS

As defined in [7], a design pattern identifies a high level architectural abstraction that supports some function (or intent) in a context to answer some motivation (or problem) and provides a solution for building structures that conform to this abstraction. Thus design patterns define generic structures that are found in objects, applications, systems or documents, and are recognized as good solutions to solve problems which occur over and over again. This enables the designer to manage *reuse* in an overall structure at a high level of abstraction. Furthermore, several implementations may be associated with a given design pattern and thus be easily switched. Design patterns are to programming in the large what classes and objects are to programming in the small. Advantages and properties of design patterns are discussed for instance in [21].

Today, languages of design patterns are developed from design experience and propose classification such as creational, organizational, and behavioral patterns at the most general level [7]. For each pattern they generally describe its name, problem and function supported by the pattern, application context, solution (structure and collaboration of generic components), consequences. [20] is a first attempt to classify and identify some hypermedia design patterns, mainly navigational patterns and user interface patterns for hypermedia applications. A variety of common structural patterns that may prove useful for description, analysis and design of complex hypertexts is identified in [1]. These hypermedia design patterns provide high level vocabulary to describe hypermedia design.

However, because of their high level of abstraction and most often of their description in textual form, two difficulties arise. It is difficult to recognize the situations where these design patterns may be reused and how to implement concrete structures that conform to them. Golden rules help us decide

precisely where to reuse design patterns. We later show how constructive templates help capture design abstraction for implementation of design patterns and produce entire hypermedia application from these design abstractions.

Example of Hypermedia Design Pattern

To illustrate the use of design pattern in hypermedia design, we describe one of them (Figure 4) and explain how we used it to design the kiosk application. We choose this example since it is both well known and simple to understand.

Pattern name: <i>Link destination announcement</i>
Motivation: To avoid unnecessary link firing by providing information about the destination
Context: When rolling over an anchor
Intent: Provide information about the link destination
Solution: Detect roll over on anchor, enhance the anchor, display short abstract.

Figure 4: Link destination announcement design pattern.

“Link destination announcement” is a hypermedia interface design pattern we identified in many systems with very different implementations. The Netscape Navigator web browser lists the URL of a link on rolling over an anchor. Hyperties lists a short abstract [24]. One can use a pattern function in a number of different implementations. These implementations share a common scheme typical to the given pattern: for instance “detecting a roll over on an anchor,” “enhancing the anchor on roll over,” “displaying a short abstract.” In this way we can describe the architecture of a hypermedia document as reuse of metastructures at a high level of abstraction even before implementation choices are made.

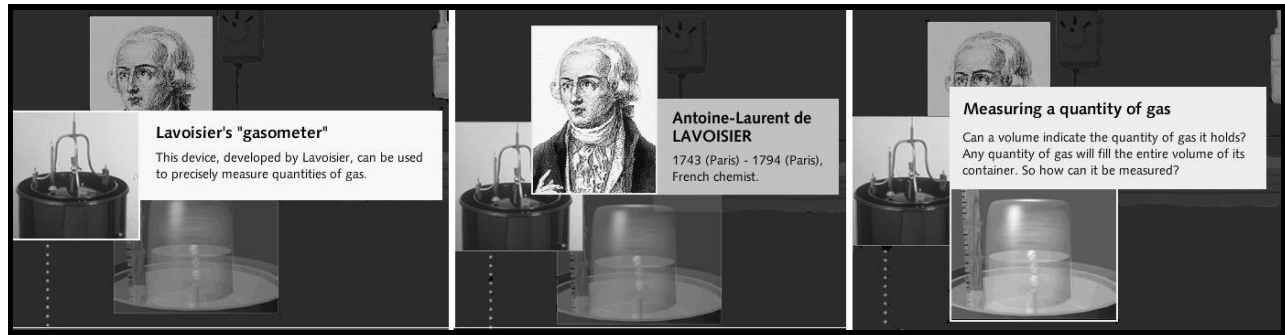
Putting Design Patterns into Practice

Our kiosk application uses two implementations of the Link Destination Announcement pattern according to the page's classes: a label with a short abstract in hub pages (Figure 5), and an animation of the modern object plus the object's name in the “room” page (see the room page in figure 3). This pattern enforces a feeling of direct exploration of the page.

Design Patterns as a Kind of Reuse

While golden rules are applied abstractly, design patterns are associated with precise structures. When a designer recognizes a situation where a given design pattern applies, all the associated abstract building blocks for reuse are known. The designer needs only to define their concrete implementation.

Design patterns provide the designer with a vocabulary to talk about and work on a problem at a very general level, before implementation choices are made. It also suggests general structures or mechanisms to implement patterns. In contrast to golden rules, which concern choices and take part to the decision process, design patterns address the issues of the overall design. They provide high-level abstractions which are building blocks for the designer to prepare the overall architecture of the document.



As mentioned previously, the kiosk navigation structure relies on sets of 3 antique instruments related to a single modern object. When rolling over the image of an antique object, a Link Destination Announcement is used both to conserve link firing and to produce a semantic glue effect. The portrait of the associated scientist and an image about the scientific relationship between the antique and the modern object pop up in the same vicinity and remain displayed while rolling over one of them. This way the reader is visually presented with the association. When rolling over any of these images, a label pops up to describe who or what is presented in a few words. The label is an announcement of the link anchored on the image and the label.

Figure 5: Detail of a kiosk application page. Samples of use of "Link destination announcement" Pattern.

CONSTRUCTIVE TEMPLATES

So long as golden rules and design pattern aim at determining 'why,' 'when' and 'what' to reuse, constructive templates provide a means to actually capture and reuse behaviors and structures, both navigational and graphical. Their purpose is to automate reuse. A constructive template specifies what to reuse, and its trigger specifies when and how to reuse it. This section presents this notion, and exemplifies its use in the kiosk application within .

Notion of Template

"A hypermedia constructive template is a generic¹ specification which makes it easier for the developer to build up an actual hypermedia structure and to populate it with its data." Constructive templates provide cost effective means to implement reuse at any level. Any general pattern as well as any document specific sub-structure which is used *repeatedly* is worth being described by templates which capture the corresponding design abstraction. Once a template is specified, any part of a hypermedia structure that conforms to this template can be automatically generated or updated just by calling the template with relevant data. The use of constructive templates implicitly relies on the separation of the hypermedia document contents from the specification of its presentation: source data is mapped into the hypertext structure as specified in the templates.

The notion of a constructive template differs from that of a model although they look closely related. A model describes a structure. A template helps produce its instances. Templates improve production since they enable the designer to put reuse into practice.

Templates take advantage and enforce the consistency of the hypermedia structure. On the one hand, since the intended hypermedia structure has to be consistent, many elements are homologous, enabling structure reuse. A template specifies how these homologous structures are actually built. On the

other hand, since a template is reused for automatically producing all of its instances, the structures it creates are consistent.

Template-based hypermedia generation

Several techniques are used to implement templates.

The simplest ones rely on programming. Actions suitable to generate sets of nodes and their relationships are described by a program. For instance Notecards used Lisp to program templates [13]. The Intermedia program supported templates that included both document and link structures [4]. Today techniques used to specify templates still are very close to programming. HTML pages may be automatically produced from object-oriented databases through templates implemented as methods of the data class they display [19, 16].

Declarative approaches aim at reducing the programming effort. They rely on executable specifications. In PageJockey, we combine a declarative approach with interactive specification of constructive templates and automatic hypermedia generation (Figure 6). Instead of programming how to parse data to turn them into a hypermedia structure, the designer graphically sketches examples of the structures to be reused. These page prototypes are turned into templates [6].

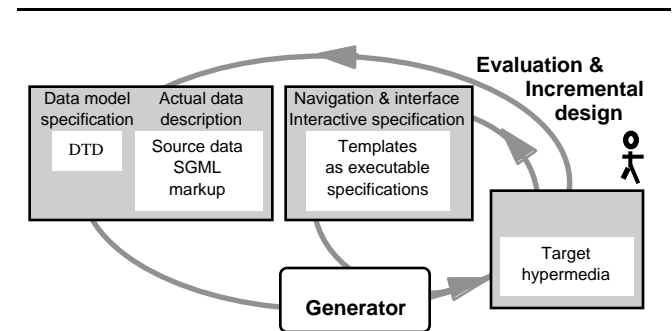


Figure 6: Declarative approach and interactive specification of constructive templates.

¹ Ethymologic sense from the latin 'generare': to give birth to.

The matching content properties are expressed in a declarative form in terms of the data model. Then, it is up to the Generator, and no longer to the developer, to determine, according to the templates specification and to the source data, which actions are needed to parse source data and to generate the target hypermedia. The generator is a single general purpose tool that can produce different hypermedia documents.

This declarative technique enables the designer to focus on design: capture design abstraction through templates, reuse, evaluate and improve design without being hampered and slowed down by programming the implementation.

Using Constructive Templates for Reuse

This section summarizes the principle of constructive templates and explains how we used them to produce a version of the kiosk application.

Separating Source Data from Hypermedia Specification. Constructive templates rely on the principle of separating source data from hypermedia presentation. It is a very cost effective technique since it enables the designer to work on the structure independently of the actual data, reducing the burden of actual production. Unfortunately, separating data from structure, a common feature of databases interfaces, is often related to poorly designed pages. So, data separation is associated with low quality production techniques, not aesthetically pleasing web-based or CD-ROM applications. Our experience producing the same kiosk application both manually with Macromedia Director and automatically with PageJockey constructive templates tells us that this claim is untrue. The two versions of the application are so similar in appearance as well as in behavior, that only experts of both systems could recognize the differences.

```
<MODERN HUBNAME=Gaz>The gas meter
<WHO IMG =Lavoiser.face.Jpeg NAME=Lavoisier>
Antoine Laurent de LAVOISIER
<FUNCTION> French chemist .
<BIRTHDATE>1743 (Paris)<DEATH> 1794 (Paris)
<WHAT IMG ="Lavoisier.Gazo.Jpeg
NAME=gasometer> Lavoisier's "gasometer"
<LABEL> This device, developed by Lavoisier,
can be used to precisely measure quantities of gas.

In the kiosk application source file, for instance, the SGML
tags denote the semantics of data such as the name of
antique instruments, the birth date of scientists, the file
names of portraits and so on. The 'who', 'what' and 'how'
tags only denote the logical structure of the document. No
presentation is associated with them in the source file.
```

Figure 7: Example of source data description.

In this paper we call *source data description* an ASCII file of the full storyboard¹ with embedded SGML marks which fully identify the elements needed for production. The markup

¹ A storyboard is a textual specification book of the contents of a multimedia document to be produced. This one contains all of the texts which need to appear in the kiosk, all the file names of images and videos, and so on.

denotes the logical and semantic modeling of data, it is not a description of the target hypermedia, but only a detailed description of its contents. It just materializes the HDM data model onto the actual data. The source data includes non-textual objects such as images, video and sound files described and referenced in the storyboard. Unlike HTML tags, the SGML tags used here have no presentation role nor hypermedia structure description function. No links are expressed in the source data description. For instance, see Figure 7.

Navigation and Presentation Specification. In PageJockey, a *page template* is a generic description of a hypermedia page, including all aspects of its navigation, presentation and interaction models. Variables or expressions stand for the actual contents. When assigning actual values to variables, the template is turned into an actual page description that the renderer displays. Conditional expressions, iteration as well as template calls enable powerful specifications whenever needed. For more details see [6]. By way of example, Figure 8 presents a generic specification of a slide-show loop design pattern and is used for specifying a generic page sequence about a scientist.

```
{First : assigning application dependent variables}
<&CurName = &("Scientist-",&NAME,&(&#LPgNbr+1))>
<&PG1Name = &("Scientist-",&NAME,"1">
{The two following lines are a reusable specification of
the navigational pattern 'Slide-show', 5 sec per slide.
The sequence loops when next page is missing }
_if_ ExistNode &CurName
    <At 5 do load &CurName >
_else_ <At 5 do load &Pg1Name > _endif_
```

Figure 8: A diaporama loop design pattern specification and use in a template.

In PageJockey, prototypes of pages are first graphically sketched by the designer who operates just as she or he would with any graphical tool. In parallel, PageJockey produces and updates a textual description of the page. The description of a sketched page is turned into a generic template just by replacing variable parts by variable names or expressions.

Generation Specification. The *generation trigger* is a specification of the conditions in the source data files that lead a page to exist associated with a template. It expresses the invariant properties that the set of variables of the template must match in any of its instances. It specifies in a declarative manner the conversion between the source data and the actual pages generated according to the template.

Whenever the generator finds, within the source data, an SGML sub-structure matching the specified properties, it automatically generates a new page according to the template. The generation trigger also provides a rule for elaborating a unique identifier (UID) for each node instance.

```

<Template page PGHub>
<Trigger Cond=&WHO ,
UID = &("Hub-",&MODERN.&Hubname)>
<Attrib &*ScientPortrait=&Img>
<Attrib &*ScientLabel=&(&Who, \n,
&BIRTHDATE, &DEATH, /n, &FUNCTION>
...
</Trigger >

```

An instance of page of class PGHub is created or updated for each WHO structure in the source file. The specified page name will be "Hub-gas" with the example of source data. Elements of the array variables called ScientPortrait and ScientLabel are built as specified. See Figure 5 for the look of this part of generated "Hub-gas" page.

Figure 9: A trigger in a page template.

How Declarative Templates are Implemented

As shown on Figure 10, the source data SGML description is first parsed in order to build its actual 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 attributes of the template 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 that 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 time for the same page. For instance, indexes are incrementally filled-up in this way whenever relevant information is found in the source file.

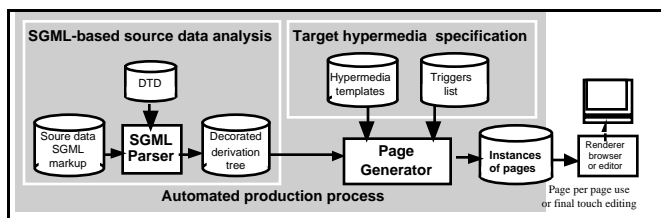


Figure 10: The internal production stages of the template-based generator.

Once page instances are automatically produced, the designer can graphically edit them to improve small details if suitable, or just browse them like a reader.

Using sketchable templates for reuse

Sketchable templates allow a very efficient kind of reuse.

Reuse from Drafts. Reuse can start directly from the designer's early drafting work. In this way, it drastically reduces the effort for producing the final version. Any designer always starts by drafting page samples, even only for evaluating emerging design ideas. Drafted pages need little effort to be turned into templates. For instance, the file name of an actual image used in a draft is substituted by a variable name. Generic links are defined similarly. Then the generation trigger is added to the template. This task is far simpler than

writing Perl scripts. No regular expression notation, and so on, are needed since the SGML parser extracts the attributes values of elements in the actual derivation tree of the document. Just naming these attributes and exhibiting their relationships — often very simple — is needed.

We used intensively sketchable templates to implement most of the PageJockey version of the kiosk application. Nearly 90% of pages are produced in this way. Only pages with non-reused animations or unique graphics are hand-crafted.

Putting in Practice Sketchable Templates to Implement some Design Patterns. To exemplify the use and the interest of templates, let us consider the Link Destination Announcement pattern previously described for the 'hub' page class. This pattern is reused 54 times in the actual kiosk. It is specified only once.

The chosen implementation has been evaluated in detail on several drafts. The look and the behavior of any instance should be perceived as similar at first sight, although they are not similar at the detailed level:

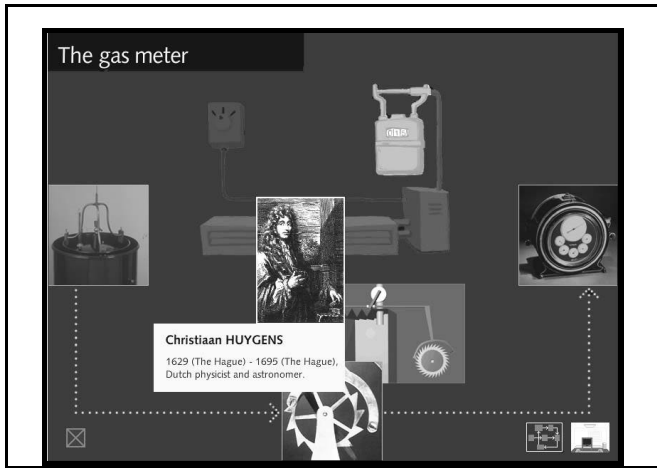
- The images are not exactly the same size, some portraits are taller, some are larger.
- The associated texts for link announcement are not the same length. They have from two up to four lines. Scientist names are longer or shorter and so on. For instance using "Samuel Morse" or "Heinrich Daniel Ruhmkorff" should change the label length accordingly.
- The layout interaction dynamics spreads the label toward the right when the instrument is presented on the left side of the screen, toward the left when at the right side, and in either direction when at the bottom (see Figure 11).

This variation is needed to avoid an imbalanced and mechanical appearance in the repeated geometric layout structures. In order to keep a good balance between consistency, clarity and diversity, the template specification introduced constraints between objects, thus allowing compatibility between flexibility and automated production.

For instance, for any placement:

- The label size is determined by the size of the source text, plus margins.
- The size of any image comes from its file.
- The bottom line of the scientist image is the mid-point of the instrument image.
- The scientist image is located above the explanation image.
- For a left placement, the bottom left corner of the label is tied to the bottom right of the scientist image.
- For a right placement, the bottom right corner of the label is tied to the bottom left of the scientist image.

This specification is given only once, the generator software uses it for producing a page of the PGHub class for each set of 3 antique instruments described with a <Hub> tag in the source file. Thanks to this technique, several placement rules or interaction rules have been studied just by changing their specification, since they can be easily recreated.



```

</trigger Cond=&MODERN>
UID = &("Hub-",&MODERN .&Hubname)>
<Attrib &*Placement "left"> {the first one}
<Attrib &*Placement "bot"> {the second one}
<Attrib &*Placement "right"> {the third one}
</trigger>

```

In the template of a PCHub page, arrays of variables are used to store the attributes collected when the trigger fires on encountering WHO, WHAT, ... tags. In all cases, on encountering the MODERN tag, the page is prepared, the first group is assigned a left side, the second one a bottom, and the third a right side presentation. Note that no such information is present in the source file. Since it concerns only the presentation of the target document, it is given only in the template trigger. This ensures the independence of source data from the target hyperdocument. Just switching the placement attributes in the trigger would exchange all elements in the Lavoisier group with the Huygens group on the page without any more effort. The image illustrates the generated behavior for the bottom placement, which can be compared with left placement behavior in Figure 5.

Figure 11: Reuse of the Link Destination Announcement pattern in a page.

DISCUSSION

Complementarities and specificities

The four kinds of reuse presented here are not exclusive. Each of them contributes to a reduction in efforts when designing and producing a hypermedia title. While not limited to a specific step in the work process, we can say that methods and golden rules are more like guides whereas patterns and templates are more like tools. Methods and golden rules address the planning of the work and the design of the scenario rather than its production. They help the designer benefit from the previous experience. In this way they promote reusing skill rather than elements or structures.

Design patterns constitute a broad paradigm that applies to any step in the work, design as well as production. They provide a conceptual framework for handling higher level design abstractions and connecting solutions to problems, thus leading to better design solutions. We must remember that most of general works on design patterns [7] classify them in three categories: organizational (including strategy)

creational, and behavioral. The first category mainly concerns the organization of the target document. But the two other categories provide means to build the architecture and to actually produce it. Currently, most of design patterns are described in libraries, for two reasons. First, most patterns are so general that they can be easily identified. For instance, an index or a guided tour is a general notion which can be described as a navigational context design pattern [20]. Second, most of design pattern languages still are very informal and are of little help to move from design to implementation. No language is currently available to let a developer specify original design patterns and their possible implementation in an entirely general way.

Conversely, constructive templates constitute a simple and effective means to let the developer capture and make design abstraction operational, as well as reuse parts of the work to build others which rely on the same structure, in terms of navigation as well as look or behavior. By encapsulating structural elements constructive templates can be handled as abstractions during the design and the production. This helps clearly separate use from the actual implementation. In this sense, constructive templates are a technique for describing and implementing some design patterns. The main interest of templates is to let the developer capture and reuse design patterns which are specific to the application and not only use some predefined ones. It enforces reuse at a higher level of abstraction than cloning and substitution techniques currently used to produce pages one by one.

Other means of reuse: sharing and cloning

A comparative experiment was conducted with two development technologies for producing the kiosk, one page-by-page with Macromedia Director, the other using automated template-based generation with PageJockey. Templates is not the only way to handle reuse in the implementation. For example, Director allows the sharing of scripts and casts to promote the reuse of behavior and of data as well as cloning parts of the score to replicate similar structures. Nevertheless, we find it easier and more reliable to use templates.

Cloning a structure is a simple form of design reuse. Replicated structures are easily built by manually exchanging contents in clones. Unfortunately, since the cloned files are just copies that do not support inheritance, any later design change has to be repeated separately in each copy. There is no way to automatically change and regenerate a new instance of a design by modifying a high-level description. Thus, some structure weaknesses due to irrelevant early design choices become increasingly difficult to locate and modify.

Conversely, templates enforce the separation of implementation details from their use, thus enabling the designer to express changes once for all at the specification level. The interpreter of the page description languages automatically takes into account any new change wherever a template is instantiated at page load time. Furthermore, PageJockey supports inheritance between templates.

Automated generation

Template triggers specify the generation of the target document as a structural transformation between the semantic structure of source data and the hypermedia structure

(navigation, presentation, interaction) of the target document. This has some light similarity with CSS in the principle, but deeply differs on two points. First, the CSS mechanism is dedicated to restructure web pages that are already organized as hypermedia documents whereas template triggers transform data structures which are not hypertext into complex hypermedia structures. Second, currently the expression power of HTML and of CSS is very poor to describe precise placements and rich animation and interactions which are needed in hypermedia applications.

The technique of triggers in constructive templates looks similar to dynamic approaches which rely on queries to databases to produce web pages. Nevertheless web pages are produced on demand whereas triggers produce the whole hypermedia structure, making it possible for a designer to evaluate the generated pages and incrementally improve the overall design.

Ease and reliability of maintenance

The independence of data from hypermedia specification is another very important feature of constructive templates. This strongly improves ease and reliability of maintenance especially at the late stages of design, when user feedback from beta testing typically leads to final changes.

We observed in the page-per-page production that the stress of handling complex last-minute changes often caused inconsistencies to appear when pages are updated one by one: it is simply too easy for some information, links or scripts to be updated improperly. Enforcing consistency requires an additional checking-revision-checking process that is very time consuming.

In contrast to this, with the constructive template-based approach the designer can be confident that all the pages produced by a given template share the same overall look and behavior. If a mistake occurs, it is visible on every instance. In this way errors can be very quickly detected and fixed throughout an application. When one page of a class is correct, one can reasonably hope that all other pages of that class are also correct. Furthermore, the cost or the risk of complex changes such as those involving the navigational or behavioral model, is no higher than those of simple changes, and their reliability is also assured.

The designer can update the generic specification of a template, in its trigger as well as in its body, without editing the data used in the instances. Running the generator on the same data produces a new set of pages updated accordingly. For instance, adding a Slide Counter Animation in any page of any slide-show is described only once in a template, in terms of look as well as in terms of behavior. Placing the new specification in an ancestral class of the templates which use it even makes the change simpler. The new document typically reuses in many places this part of the design captured and described as an abstraction. Doing the same change without templates nor generator implies that changes be manually propagated to each page in each cloned file. This process is much more time consuming and less reliable.

Incremental early design

The ease and the speed of maintenance of template generated documents enable full scale incremental design [17], even at the earliest stages. Any draft of a page design can be turned into a template with little effort. Instead of evaluating a sketched draft of a single page design, one can try any idea about interaction, structure or presentation issues on a large scale. At any stage of the design process, small prototypes can be developed quickly to evaluate a design choice and even to experimentally study user interaction.

It is not necessary to markup the full storyboard at first. An incremental approach allows the designer to iteratively check ideas about the data description and ideas about the target document. For instance, in our kiosk application several images and comments were added later in the source, generating new pages.

The design process we ran was incremental for several reasons. First, we had to take into account the feedback from our client¹. About three major releases of the kiosk have been studied. The page-per-page approach did not provide us with an efficient support to incremental design, since early choices in the implementation proved to be difficult to modify later. Thus an increasing number of programming problems appeared as changes were made, slowing the design process and making revisions harder and harder to manage.

Cost reduction versus quality improvement

Since the template-based approach automates a large part of the production, its cost is drastically reduced. The saving can be used either to reduce the overall production cost or reinvested to iterate on design in order to improve the quality and usability of target documents. For instance, several versions of a given document can be evaluated at full scale to choose the best one.

CONCLUSION

In this paper we have studied a real-life example of how reuse can be put into practice in design and production and discussed some of its advantages. We have shown how constructive templates help implement design patterns to automate reuse. The comparative experiment asserts that the template approach is able to produce documents of the same look and quality than common manual techniques at a lower cost and a higher reliability. Enforcing the reuse of design patterns also helps improve hypermedia quality by reusing good design experience.

Many questions remain open and provide new directions for investigation.

Design patterns provide authors with powerful abstract notions for describing and handling the design at a high level of abstraction. Nevertheless, these promising techniques still have to be improved to be efficiently put into action.

¹ The museum plans the production of a series of 24 kiosks. A preliminary series of four have been produced to study their usability in the exhibition. The feedback from the museum was important for us and caused many changes during design.

On the one hand, recent works in CASE suggest how to generate applications specified in terms of design patterns [3], but the proposed techniques still require heavy descriptions and do not take into account the design steps. Given their potential advantages, design patterns will surely take a larger place in hypermedia design in the future.

On the other hand, constructive templates constitute an effective technique for putting reuse into action. Any repeated structure or pattern can be specified once and then reused whenever suitable. This benefits not only the production stages, but also the whole life cycle of a hypermedia document ranging from early design to maintenance. Furthermore, automated reuse improves the overall consistency and reliability of the document. We have experienced the implementation of some design patterns with templates (Only one has been described in this paper). Complex macro-structure specification still have to be studied to help generation from any kind of pattern.

The support of reuse in collaborative design within large or distributed teams [9] should also benefit from constructive templates and design patterns since they enable handling elicited design abstractions separately. This enforces modularity in design and even in production.

ACKNOWLEDGMENTS

We are grateful to C. Hosoe, B. Jacomy, M. Kasman, K. Lenk, D. Negel and Y. Petit who all contributed significantly to the design and development of this kiosk.

REFERENCES

1. Bernstein, M. Patterns of Hypertext, in Proc. Hypertext'98, ACM Press, (1998).
1. Brisson, D. Musée d'Orsay, promenade interactive au cœur de l'art du XIX siècle, Montparnasse Multimedia et Réunion des Musées Nationaux, 1996.
2. Budinsky, F.J., Finnie, M.A., Vlissides, J.M., and Yu, P.S. Automatic code generation from design patterns, IBM Systems Journal, Vol. 35, N°2, (1996).
3. Catlin, K.S, Garrett, L.N., Launhardt, J.A. Hypermedia templates: An author's tool, in Proc. Hypertext'91 (San Antonio, December 1991), ACM Press, 147-160.
4. Chiu, C.M., Bieber, M. A Generic Dynamic-Mapping Wrapper for Open Hypertext System Support of Analytical Applications, in Proc. Hypertext'97, (1997).
5. Fraïssé, S., Nanard, M., Nanard, J. Generating hypermedia from specifications by sketching multimedia templates, in Proc. ACM Conf. Multimedia'96 (1996), 120-124.
6. Gamma, E., Helm, R., Johnson, R., Vlissides, J. Design Patterns: Elements of Reusable Object-Oriented Software. Addison-Wesley, Reading, (1995).
7. Garzotto, F., Paolini, P., Mainetti, L. Hypermedia Design, Analysis and Evaluation Issues. CACM 38(8) (August 1995), 74-86.
8. Garzotto, F., Mainetti, L., Paolini, P. Information Reuse in Hypermedia Application, in Proc. ACM Conf. Hypertext'96 (1996), ACM Press, 93-101.
9. Garzotto, F. Paolini, P. Team-Based Coordinated Development of Hypermedia: Lessons Learned from Piero Della Francesca's "Agostinian Polyptych", in Proc. ICHIM'97 (Paris, 1997), Le Louvre, 1-12.
10. Gronbæk, K., Trigg, R. Toward a Dexter-based model for open hypermedia: Unifying embedded references and link objects, in Proc. Hypertext'96 (1996), 116-128, ACM Press.
11. Isakowitz, T. et al. A Methodology for Structured Hypermedia Design. CACM 38(8) (1995), 34-44.
12. Jordan, D., Russell, D., Jensen, A.M.S., Rogers. R.A. Facilitating the development of representations in hypertext with IDE, in Proc. Hypertext'89 (Pittsburgh, 1989), ACM Press.
13. Kahn, P.D. Global and Local Hypermedia Design in the Encyclopaedia Africana, in Hypermedia Design, Fraïsse, Garzotto, Isakowitz, Nanard & Nanard (eds), Springer, (1995), 109-122.
14. Lewis, P.H. *et al.* Media-Based Navigation with Generic Links, in Proc. Hypertext'96, 215-223, 1996.
15. Matisse, User Manual, Euriware, 1997.
16. Nanard, J., Nanard, M. Hypertext Design Environments and the Hypertext Design Process, CACM 38(8) (August 1995), 49-56.
17. Nürnberg, P.J., Leggett, J.J., Schnase, J.L. Hypermedia Operating Systems: a New Paradigm for Computing, in Proc. Hypertext'96, (1996), ACM Press, pp. 194-202.
18. O2Web, User Manual, O2 Technology, 1997.
19. Rossi G., Schwabe D., Garrido A. Design Reuse in Hypermedia Applications Development, in Proc. ACM Conf. Hypertext'97 (1997), ACM Press, 57-66.
20. Schmidt, D.C. Using Design Patterns to Develop Reusable Object-Oriented Communication Software. CACM 38(10) (October 1995), 65-74.
21. Schmidt, D.C. Object-Oriented Application Frameworks, CACM 40(10) (October 1997), 32-38.
22. Schwabe, D., Rossi, G., Barbosa, S.D.J. Systematic Hypermedia Application Design with OOHD, in Proc. Hypertext'96 (1996), 116-128.
23. Shneiderman, B. User Interface Design for the HyperTIES Encyclopedia, in Proc. Hypertext'87, (1987), 189-194.
24. Thüring, M., Hannemann, J., Haake, J. Hypermedia and cognition: Designing for comprehension, CACM 38(8), (1995), 57-66.

