OPTIM: PEDAGOGICAL INTERACTIVE DOCUMENT DESIGN WITH MINIMAL PROGRAMMING

Henri Delebecque

▶ **To cite this version:**

Henri Delebecque. OPTIM: PEDAGOGICAL INTERACTIVE DOCUMENT DESIGN WITH MINIMAL PROGRAMMING. CATE 2005, Aug 2005, Oranjestad, Aruba. hal-01848823

HAL Id: hal-01848823
https://hal.science/hal-01848823

Submitted on 25 Jul 2018

HAL is a multi-disciplinary open access archive for the deposit and dissemination of scientific research documents, whether they are published or not. The documents may come from teaching and research institutions in France or abroad, or from public or private research centers.

L'archive ouverte pluridisciplinaire **HAL**, est destinée au dépôt et à la diffusion de documents scientifiques de niveau recherche, publiés ou non, émanant des établissements d'enseignement et de recherche français ou étrangers, des laboratoires publics ou privés.

OPTIM: PEDAGOGICAL INTERACTIVE DOCUMENT DESIGN WITH MINIMAL PROGRAMMING

H. Delebecque
Supélec
3 Rue Joliot-Curie
F 91190 Gif sur Yvette
France
33 1 69851491
Henri.Delebecque@supelec.fr

1. ABSTRACT

In this paper, we propose an open framework for teachers and lecturers in science, to help them write their pedagogical documents using both static textual subject matter, and interactive and multimedia content. Authors can focus their attention on their pedagogical goals, leaving all aspects (static or dynamic) of the GUI generation to our OPTIM engine.

This framework will also allow them to factorize the structure common to all their documents into classes, and will free them from all other design considerations. Moreover, OPTIM allows authors to insert interactive parts into their documents without any GUI programming skills. They only have to provide the computational code that supplies the numerical values to be displayed by the GUI, and they can use whatever programming language they find the most convenient.

2. INTRODUCTION

We propose a general authoring framework, called OPTIM, which uses HDML [1], a metalanguage more suited to the definition of pedagogical documents than XML. Although one advantage of XML is the great amount of flexibility its tags offer, this flexibility also results in a great variety of semantics given to tags by the various authoring languages based on XML, and a lack of a standard in the pedagogical domain. On the contrary, the other extreme is a general authoring language like DocBook© which has its own drawbacks, since it defines only very general structural entities, like header, footer, body.

We want to help content authors, not always familiar with programming languages, and even less familiar with XML, to write their concrete documents, by defining an authoring language that will use only their domain specific terms. For example, a mathematician will greatly appreciate finding tags named «theorem», «premises», «conclusion», «proof» in the authoring language he/she uses. This is something that people have to code manually if they use languages like Latex.

Moreover, OPTIM will allow authors to insert interactive objects into their pedagogical documents. This is of great value in the pedagogical context, where students and learners can visualize the concepts described in an alternate graphical manner, while keeping an additional textual description.

The pedagogical domain helps us in our design, since the variety of graphical objects found in application is limited, as is the kind of interaction the user has with them. The purely graphical part of pedagogical software (even simulators) is commonly restricted to curves,

animated schematics or graphics, that require minimal user interaction.

2.1 OPTIM Principles

The OPTIM acronym means «Open Platform for Teaching Interactively with Multimedia». OPTIM's goal is to give teachers a framework that allow them to focus on contents (in all its richness), to use their favorite editing tool, and to ignore presentation issues. This last point contributes to the openness, a key feature in the pedagogical world, where languages and tools are frequently specific. OPTIM uses HDML as a structuring language for the description of the document's static part.

Authors can add interactive parts to their documents using first OPTIM's GUI generator, and then its ability to link this GUI with the specific code the teacher supplies for the computation of the information displayed as curves or dynamic schematics.

To help teachers that contribute to a global pedagogical document to write their sections/chapters/books in a rather uniform manner, the editor of the whole document defines a structural model. This model defines all the mandatory or optional elements authors have to put into their documents. HDML is clearly a meta-language, since this model is also used to define a markup language that we will call the content authoring language. This language defines tag names that are familiar to the content authors, since they are defined by the editor, using the words and terms authors currently use.

We will start with some definitions, to help us present OPTIM's general principles. For more clarity, lets define the following terms in the context of a large pedagogical document, which we will generalize below under the global document terms.

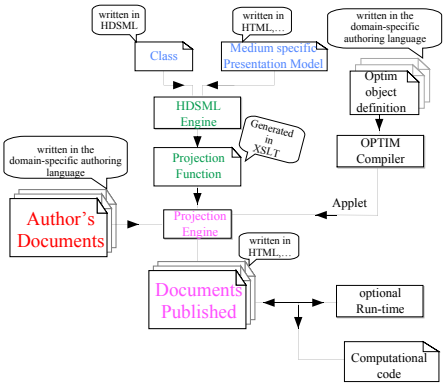
2.2 Definitions

HDML is used by the three different kinds of authors described in the next paragraph, while defining four kinds of documents. These are:

- The class (the structural model),
- The presentation model,
- The concrete document (a class instance).
- The interactive objects (OPTIM objects)

During the meta-authoring phase, the class designer, who is also the global pedagogical document editor, defines the structural model common to all the pedagogical sub-documents that share either a common look or structure, or both. Since this class defines all the elements that can be displayed, a second author, with graphical skills, will then design a presentation model, that will explain how to display all the elements described by the class for a particular medium. He will also help the teacher in designing the look-and-feel of the GUI of all interactive parts, defined as OPTIM objects.

The HDML engine works then in three phases, using the four different kinds of documents described above.



At the end of the meta-authoring phase, the HDML engine generates a XSLT style-sheet for each pair of models, building a projection

function from the class to the presentation medium. Moreover the HDML engine produces the grammar of the authoring language, which will be used by the content author in the authoring phase.

During this phase, the teachers will write the concrete document. They can use any editing tools they want, including tools that are not XML compliant, though obviously the great advantage of using XML compliant editing tools is the syntax sensitive help they can provide.

The pre-publishing phase allows the HDML engine to produce documents ready to be published. Contents documents, which combine semantically-rich items (supplied by content authors), and other items dedicated to presentation purposes like a logo or background (provided by the presentation model) are transformed using projection functions.

During the final phase, the documents are published. Authors can activate the new functionality OPTIM adds to HDML, and add interactively to their static documents. OPTIM objects are referenced by HDML documents, and activated at publishing time by the student. This activation will produce events that are filtered by the GUI defined in the OPTIM description file, and compiled as Java© objects by the OPTIM compiler.

All events are either completely managed by the Java code, or transformed into requests sent to the code that the content author supplied, to achieve the computation required. This code can, for example, compute a new 3D-curve to be displayed, the new state of a data path into a processor, or some animation, sound, etc. This method of separating purely graphical events (such as clicking, or filling in an entry field) follows the distinction Java makes between high-level "action" events, and low-level ones, like mouseUp or others.

2.3 OPTIM Architecture

Every interactive object included in a concrete document is described in two complementary ways. First, the graphical designer has to explain how the graphical sub-objects will answer to end-user stimuli (click, keyboard events,…). And second, the computation code should be able to send refresh orders to the GUI, when new data arrives. These two aspects are managed by a classical MVC architecture evangelized by Java. This very efficient and robust way to separate the Model (in charge of all computational tasks) and the View (that displays the values in the most convenient way for a given context or user) has proved its value since its first definition in the Smalltalk [2] object-oriented language.

3. STRUCTURING ELEMENTS

The class designer builds the class using either generic blocks and objects or semantically qualified elements, such as those defining the author, the authoring platform, the document version, or others.

Every element of a class definition inherits the occurrence constraints that HDML defines for it. But the class author can modify them, according to the structural freedom he wants to give to his concrete document authors. In this way, he defines the occurrence constraints of the content authoring language elements.

3.1 The Elements

3.1.1 The Block Element

The block is the most versatile element, able to contain other generic elements, acting as sub-blocks or objects (the inclusion of semantically qualified elements is not currently supported, since they are limited to the whole page description).

A block definition in the class has a very important "name" attribute, that will give its name to the corresponding tag used by teachers when they write their concrete documents. An optional "type" attribute adds semantics by specifying the kind of block, such as: header, menu, footer, body.

The "name" attribute makes it possible to define the content authoring language so it is adapted for a specific domain. This language is fine-tuned for the content authors, since it names the structuring units they will use with the terms they are most familiar with.

Finally, a block can be structured using sub elements taken from the following non-exhaustive list: Block, Object, or OPTIM

The recursive nature of the block element makes it possible to avoid defining a limited list of structuring levels, as in DocBook, while giving it, as the same time, more descriptive power.

3.1.2 The Object Element

The object element is always a leaf of the XML node tree. It describes either non-structured elements (such as a picture, sound or movie) included by reference, or very basically structured elements such as hyperlinks or the fields of a form.

3.1.3 The Optim Element

The Optim element is the link between the actual concrete document and the interactive objects built using the OPTIM platform. OPTIM keeps the content author in a familiar context, by using the authoring language that HDML extracts from the class, for all the static elements that surround the interactive ones. Moreover, by allowing the teacher to supply the computational code in various languages (including those that require a specific runtime), we insure the maximal adaptability to most scientific domains, and the openness teachers commonly require.

3.1.4 Example

Let us give an example, with few lines of HDML defining a class block named «Theorem», followed by an example of one theorem definition in a concrete document.

```
<Blockname="Theorem" type="body">
   <Object name="Name" type="text" Occurs="1" />
   <Object name="Premises" type="text" Occurs="1+"/>
   <Object name="Conclusion" type="text"Occurs="1" />
   <Object name="Proof" type="body" Occurs="1" />
   <Object name="Keyword" type="text" Occurs="*" />
</Block>
```

The previous HDML fragment shows that the class designer has defined the generic structure for Theorem with the following sub-elements:

- a mandatory name (which is a text zone),
- a mandatory conclusion (as text zones),
- one or many premises (also as text zones),
- a mandatory proof (structured with sub-blocks)
- some optional keyword.,

as in the following example:

```
<Theorem>
   <Name>Thales</Name>
   <Keyword>Geometry</Keyword>
   <Premises>Let and be two points …</Premises>
   …
</Theorem>
```

We can see here how the content authoring language uses domain centered entity names, instead of the generic ones (like block) that HDML defines. We can also appreciate how this authoring language can be easily used by mathematicians or teachers to write their documents.

4. RELATED WORK

4.1 HDML vs Tag Libs or XSLT

One can argue that our objectives can be fulfilled using tag libraries like Struts [3], which are based on the powerful capabilities such tag libraries add to HTML. We think that tag libraries are clearly an efficient solution in certain instances, but limited to HTML production, and probably more oriented towards dynamic HTML. Moreover, they tend to stress the look-and-feel, whereas we want to free the authors from such considerations, and stress the semantics and document structure. Finally, the customization of Struts features supposes skills that content authors and editors usually don't have.

The XSL tool is also very powerful, but has one major drawback for a teacher: the mastering of XSLT and X-Path. And if X-Path has to be used only rarely in our context, XSLT has to be used regularly for two reasons. First, the author has to write a separate XSL sheet for every presentation medium used and for every subset of the document. Then, this XSLT authoring must be redone every time the user changes the look and feel, for every presentation medium, and for every subset of the global document.

Moreover, this way of producing web pages does not allow any factorization and genericity, nor any possibility for handling interrelations between documents. It does not provide any way to manage the document versioning that the HDML engine supports, and which is of great help in managing the production of multiple content authors.

4.2 OPTIM versus current IDEs

One could think that existing Integrated Development Environments, like JBuilder©, or Visual Studio©, are a better choice for developing the interactive parts of our pedagogical documents. We think that these products are too complex for the rather simple graphical objects authors need. The complexity of the programming task should not be included in the GUI's design, but rather in the development of the simulation part. And we think that versatility and openness is crucial here, letting our teachers use their favorite programming languages. IDEs often impose the mastering of many specific programming skills, such as animation techniques and graphical algorithms, which are clearly not trivial, require a great investment and constant updates.

4.3 Programmingless Projects

Recently new products, like Gatonero have emerged that allow the quick design of interactive applications by non programmers. Gatonero offers the ability to describe sophisticated prototypes using its own specific language, or the Gatonero Studio tool with drag-and-drop capabilities. Gatonero's architecture relies on a framework, which is announced as "fully-programmable and object based". Gatonero obviously offers inheritance capabilities, but also the reusability of prototypes, as well as multi-platform deployment, since it is Java based.

We think that the goals of products like this one are slightly different from ours. They tend to propose Rapid Application Development tools rather than a framework that makes it possible to integrate data produced by a regular computational code into a pedagogical document.

Moreover, like the SimTool project mentioned below, Gatonero suffers from the lack of openness, since it offers only one programming language. Clearly, this gives it the ability to be multi-platform, but OPTIM can reach this goal in a different way, by allowing the client-side software, highly portable, to interact with a computational code located on a dedicated remote server.

4.4 E-Learning Development Tools

We have found active projects, like the one described in [4], that plans to give teachers a way to design interactive pedagogical documents without programming. It is a component-ware based project that has the same openness property as OPTIM, in that it offers links with databases, e-mail, etc. But this project is currently based on the Delphi© programming languages and seems to have no capability to cooperate with other programming language, for the computational requirements of the simulated objects. Moreover, it lacks the uniform model OPTIM defines with HDML, and which guaranties a excellent adaptability to new application domains.

Another project, SimTool [5], part of the more ambitious VORMS project, shares some of our principles. Experiments done using SimTool have shown that students greatly appreciate interactive counterparts to textual presentations. One of the major limitations of SimTool is the choice of Java for both the GUI construction and the computational tasks. We think that different authors will better implement these two complementary parts, and that one should leave more freedom in the choice of the computational language. To design a good graphical user interface it's necessary to master computer animation, graphic algorithms, and to have extensive knowledge of the GUI's object library. On the other hand, the teacher, which is, in this case, the knowledge reference should only describe the computations required to supply data to be displayed. This task is complex enough, and we have to let teachers use their favorite programming language to complete it. Moreover, SimTool's goal is the building of simulation applets, and it does not cover the whole design of a teacher's pedagogical document, as does HDML.

5. REFERENCES

[1] Henri Delebecque: HDML a lightweight authoring metalanguage with object-oriented features IADIS-WWW/Internet 2003

[2] Adele Goldberg, David Robson Smalltalk-80: The language and its implementation 1983 Addison Wesley

[3] Apache Jakarta Project Struts http://jakarta.apache.org/struts//s

[4] Ryoji Matsuno, Yutaka Tsutsumi, Richard Gilbert: A Tiered Approach Utilizing Reusable Componentware Methodologies for Multimedia Educational Software Creation and Development, Proceedings of Ed-MEDIA 2004, Lugano Switzerland

[5] Imke Sassen, Torsten Reiners, Björn Paschilk, Stefan Voß: Instructional Design and Implementation of Interactive Learning Tools, Proceedings of Ed-MEDIA 2004, Lugano Switzerland