OPTIM: AN OPEN PLATFORM FOR TEACHING INTERACTIVELY WITH MULTIMEDIA

Henri Delebecque

OPTIM: AN OPEN PLATFORM FOR TEACHING INTERACTIVELY WITH MULTIMEDIA

Henri Delebecque
Supelec
Computer Science Department
Plateau de Moulon
F 91190 Gif sur Yvette FRANCE
Phone: 33 1 69851491 Fax: 33 1 69851499
Henri.Delebecque@supelec.fr

ABSTRACT

In this paper, we propose an open framework for teachers and lecturers in science, to help them write their pedagogical documents with both static textual parts, and interactive and multimedia content. This framework will also allow them to factorize the structure common to all this documents into classes, and to free themselves from all other design considerations. Moreover, OPTIM allows authors to build interactive parts of these documents without any programming skills. The author can focus his attention to the pedagogical aspects, leaving the GUI generation to our engine. The computational code the teacher supplied, and that provides numerical values displayed by the GUI can be developed in various programming languages.

KEYWORDS

Authoring Structured Interactive Documents XML

1. INTRODUCTION

We propose a general authoring framework, called OPTIM, which uses HDSML, a meta-language more specifically suited for the definition of structured documents than XML [Cowan 2002]. Moreover, OPTIM will allow the author to define interactive objects into pedagogical documents. This is of great interest in the pedagogical context, where students and learners can visualize the concepts described in an alternate graphical manner, while keeping an additional textual description.

1.1 OPTIM Principles

The OPTIM acronym means «Open Platform for Teaching Interactively with Multimedia». OPTIM's objective is to give to content authors a framework that allow them to focus on contents (in all its richness) and ignore presentation's problems, while using their favorite editing tool. OPTIM uses HDML [Delebecque 2003] as structuring language that allows the description of the static part of the document. Authors can add interactive parts into their documents using first the OPTIM's GUI generator, and then its ability to link this GUI with the computational code that supplies the information displayed as curves or dynamic schematics.

To allow authors to focus their attention to content, and ignore presentation problems, a structural model is defined by the editor of the whole pedagogical document. This model defines all the mandatory or optional elements content authors have to put into their documents. HDSML 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 very 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, which help us while presenting the general principles. For more clarity, the following definitions are done in the context of a large pedagogical document, we will generalize hereafter under the *global document* term.

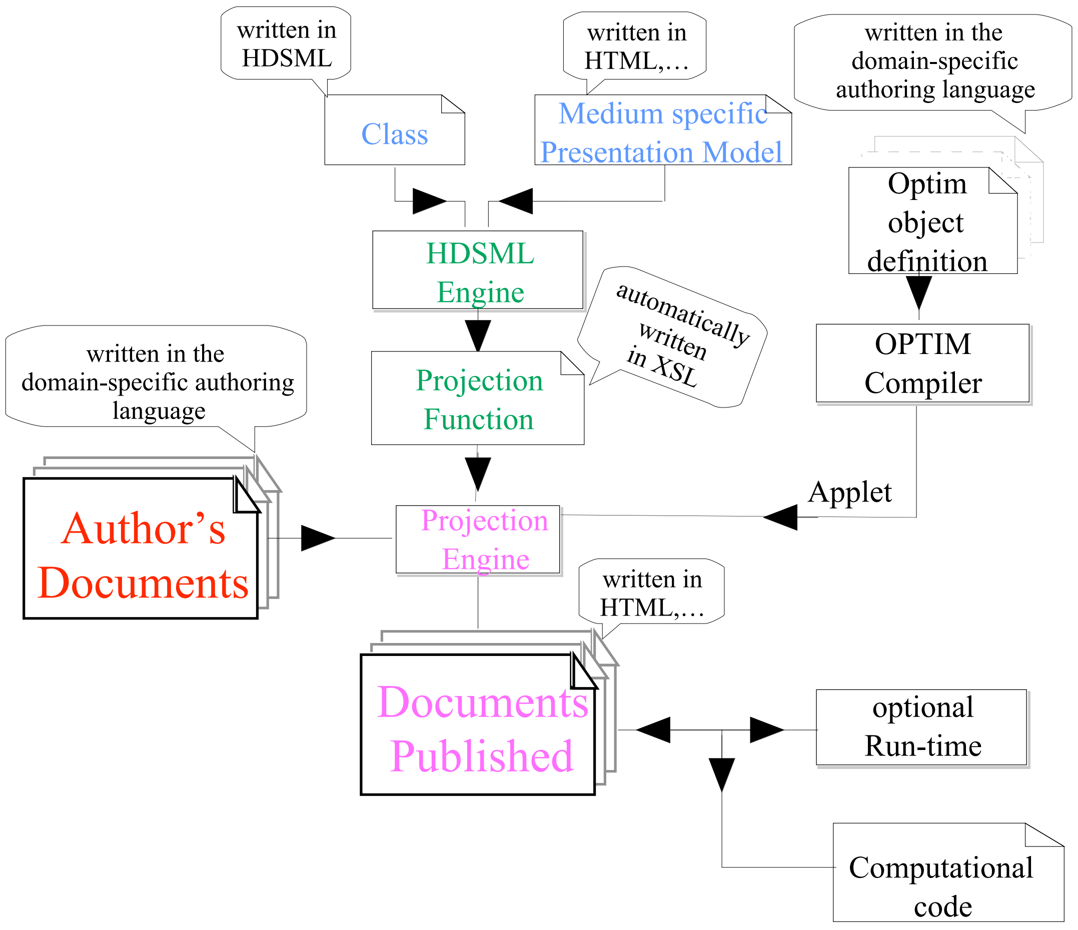
1.2 Definitions

HDSML can be used by authors following three different roles described in the next paragraph, while defining three kinds of documents, which are:

- The *class* (the structural model),
- The *presentation model*,
- The *concrete document* (a class instance).
- The interactive objects to be inserted (Optim objects)

The class designer, who is also the whole pedagogical document editor, defines the structural model common to all the pedagogical sub-documents. The graphics designer, or the ergonomist, designs the look-and-feel used to display these structure elements on a specific presentation medium, in a presentation model. Last, the content author instantiates the class in concrete documents.

The HDSML engine works in three phases, using the four different kinds of document described above.



1.2.1 Meta authoring phase

To ensure the uniqueness of the source document, whatever publishing medium is used, we define the notion of pairs of models. For every meaningful subset of the global document (book, chapter, section…), the document editor can supply a specific class (the structural model) using HDSML, and the graphics designer can provide a specific presentation model for any presentation medium. This is done during the first phase, we call the *meta-authoring phase*.

The HDSML engine takes all these pairs of models, and then automatically generates a XSLT [Adler et al 2001] style-sheets for each one, building a projection function from the class to the presentation model. Moreover the HDSML engine produces the grammar of the authoring language, which will be used by the content author in the next phase.

1.2.2 Authoring Phase

Then, the content authors will write the concrete documents, using any editing tools they want, including not XML compliant ones. The great advantage of using XML compliant editing tools is that they are able to provide context sensitive help. Obviously, an editing tool able to understand the XML Schema [Brown et al 2001] grammar definition (provided by HDSML during the previous phase) can display the attributes which are valid for the tag currently edited, or the children it will accept.

1.2.3 Pre-publication phase

In this phase, the HDSML engine applies the projection function to the concrete documents that conform to the class structure, generating the documents to be published. These later documents are a mix between items supplied by content authors, and others solely bounded to presentation (i.e. background, logo,…) and provided by the presentation model.

1.2.4 Publication phase

In this last phase, the authors can activate the new functionality OPTIM adds to HDSML, and transform their static documents into interactive ones. Since the concrete documents can reference OPTIM objects, these objects will be activated at publishing time by the student. This activation will produce events that are sent to the GUI defined by the author, and compiled as Java© objects by the OPTIM compiler. Every event is either completely managed by the Java code, or transformed into a request 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.

1.3 OPTIM versatility

1.3.1 Computational language

OPTIM accepts currently Java computational code. But it will be quickly extended to many other domain specific languages, since its working principle requires only basic exchanges between the GUI and the domain-specific code, as explained in the section devoted to the OPTIM architecture. We plan to interface languages like Mathematica and Matlab.

1.3.2 Concrete documents format

HDSML also accepts as concrete document any file not expressed in one of its authoring language at all. We have designed for a special kind of files (namely Microsoft Word © [Microsoft 2004] ones) a specific front-end. It analyzes the content according to grammatical rules, stored in a specific file, called the front-end grammar file. These rules describe how the bookmarks and styles found in the document have to be interpreted to reveal the document internal structure.

One advantage of XML is the great amount of flexibility its tags offer; however, 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 standard in this domain. On the contrary, a general authoring language like DocBook © [Walsh et Muellner 1999] have its drawback, 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 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 to find tags named «theorem», «premises», «conclusion», «proof» in the authoring language he/her uses. This is something that people have to code manually if they use languages like Latex.

1.4 HDSML Concepts

Let us start with the main structuring entity that both the class designer and the content authors will use: the element.

1.4.1 The Element

All the authors structure their classes or concrete documents by dividing them in meaningful parts they will put into elements. For HDSML, an element is a XML tree node, such as a header or footer structure description (in the class), its content (in a concrete document) or its appearance (in the presentation model, actually using HTML nodes).

1.4.2 The Class

The class definition, (also called the structural model) shared by all the concrete documents, is the editor's responsibility. The actual structure of the concrete documents can be loosely linked to the model that the class defines, since the class designer can describe some elements as optional or mandatory, unique or multiple (like using the Kleene operators).

The class design is done customizing the building blocks that HDSML defines (or instantiating them in a object-oriented meaning), to describe the structure common to the global document.

1.4.3 The Presentation Model

The presentation model should probably be designed by a specific author, since it implies more know-how in ergonomics and graphics than in the normal content domain. This author can choose the presentation language best suited for his/her presentation medium, as long as it is supported by the HDSML engine (currently all the HTML dialects). The author also can use any authoring tool for this task. This model defines the appearance of the all building blocks defines into the class, and that should or can be displayed on that medium.

1.4.4 The Concrete Document

Content authors can freely edit their concrete documents, using their favorite publishing tool, without the burden of sharing a common document. Despite this complete freedom, the global document editor can always compile all the concrete documents in a single one which references them, and publish it after the projection by the HDSML engine.

Another great advantage of this concrete document set is its uniqueness, even if it has to be published on multiple media, thanks to projection functions generated by the HDSML engine. This uniqueness gives to the content authors the insurance that the information published on every medium will conform to the content they have defined, even if it is very dynamic.

1.5 OPTIM Architecture

Every interactive object included into a concrete document should be described in two complementary ways. First, the author have to explain how the graphical sub-objects will answer to end-user stimuli (click, keyboard events,…). On the opposite, 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 [Goldberg et Robson 1998] object-oriented language.

2. STRUCTURING ELEMENTS

The class designer builds the class using generic blocks, objects and 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 HDSML defines. But the class author can modify them, according to the structural freedom he wants to give to his concrete document authors. This way, he will define the occurrence constraints of the content authoring language elements.

2.1 The Elements

2.1.1 The Page Element

The page is the XML root node of every HDSML document. A page includes the semantically qualified elements that describe the whole page, and generic ones, that define a common document structure.

2.1.2 The Block Element

The block is the most general element, able to hold other generic ones, like (sub) blocks or objects. The inclusion of semantically qualified elements is not currently supported, since they are theoretically limited to the whole page description.

A block owns a very important «name» attribute, that will give its name to the corresponding tag used by the content author when he writes his concrete document. An optional «type» attribute specifies the kind of block as for example: header, menu, footer, body or form.

The «name» attribute allows the domain centered definition of the content authoring language. This language will be very fine tuned to content authors, since it named the structuring units they will use with their most familiar terms.

Finally, a block can be structured using sub elements taken in the following non exhaustive list:

- Block
- Object
- Keyword
- Import
- Optim

The block recursive nature avoids the definition of a specific structuring levels, as defined by some structuring languages like DocBook, while giving it more descriptive power.

2.1.3 The Object Element

The object element is always a leaf of the XML node tree. Its purpose is to describe either non-structured elements (such as picture, sound or movie) or very basically structured elements (such as the «link» or the «field» objects).

The «link» object is a regular association between a text part (the link source) and an URI (the link target). The «field» object is also an association, between a text part (the field name) and an un-typed value (the field value), mostly used in forms. Many other object types exist, and more can be added by class designers.

2.1.4 The Keyword Element

The keyword is a powerful tool for the content author, who can use it to add semantics to potentially every document element.

These fine grain defined keywords are of great utility for information retrieval systems or more generally the Semantic Web Activity [Bourda et al. 2003] applications. The current full text search tools often return irrelevant documents, since they are unable to use the word definition context for filtering. We all know that the same word can have a lot of meanings, depending on this context: for example, «Washington» can be understood as the last name of a person, a state's name, a city name, …. HDSML allows the definition of keywords in an element context, allowing finer semantically based searches.

2.1.5 The Import Element

The import element allows content authors to factorize document parts (either structured or non structured) in external files, dramatically reducing the global document maintenance cost by removing redundant information. The same feature is of course available to the class designer, to factorize without inheritance, reaching multiple inheritance power.

2.1.6 The Optim Element

The Optim element is the link to interactive objects build using the OPTIM platform. Since the description language of Optim elements is still the authoring language, we keep the content author in a familiar context. Moreover, by allowing the user to supply the computational code in many languages (including those that require specific run-time), we insure the maximal adaptability to most scientific domains.

2.1.7 Example

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

```
<Block        name="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="text" Occurs="1" />
      <Object name="Keyword" type="text" Occurs="*" />
</Block>
```

The previous HDSML fragment means that the class designer defines the Theorem generic structure with the following sub-elements:

- a mandatory name (which is a text zone),
- a mandatory proposition, conclusion and proof,
- some optional keywords,

as in the following example:

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

We can see that the content authoring language uses domain centered entity names, instead of the generic ones (like block) HDSML defines.

3. RELATED WORK

3.1 HDSML versus Tag Libraries or XSLT

One can think that the powerful capabilities tag libraries like those defined by Struts [Apache group 2000] add to HTML, allowing an easy creation of a coherent set of HTML pages, fulfill our objectives. We think that tag libraries are clearly a good solution, but limited to the HTML production, and probably more oriented towards dynamic HTML. Moreover, they tend to stress on the look-and-feel, when we want to free the authors from such considerations, and stress on semantics and document's structure. CSS share the same drawback.

The XSL sheets offered in the XML tools family, has multiple drawbacks for a content author: he has to fully master XML and XSL. And this mastering has to be regularly exercised 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 XSL writing task must be redone every time the user changes the look and feel, and for every presentation medium, and for every subset of the global document.

Moreover, this way of producing web pages doesn't allow the factorization and genericity our tool allows, nor any possibility for handling interrelations between documents. It doesn't provide either any way to manage the document versioning the HDSML engine supports, and which is of great help in managing the production of many contents author.

3.2 OPTIM versus current IDEs

One can think that existing Integrated Development Environments, like JBuilder©, or Visual Studio©, are solutions to our problem. We think that these products are really powerful, and maybe too powerful for the rather simple graphical objects authors need. The complexity of the programming task should not be measured in the GUI's design, but rather in the simulation part. And we think that it is very important to leave our content author using his/her favorite programming language in that case. Moreover, IDEs impose often the mastering of many specific programming skills, such as animation techniques, graphical algorithms, which are clearly not trivial, and require a great investment.

3.3 E-Learning Development Tools

We can found currently projects, like the one described in [Matsuno et al 2004], that tends to help teachers in designing interactive pedagogical documents. This component-ware based software allows the teacher to create very sophisticated graphical applications without programming. It offers links with databases, and many other services. But it is based on a component-ware architecture, and lacks of a general paradigm, like HDSML. Moreover, this software is currently based on the Delphi© programming language, and seems to have no capability to cooperate with other programming language, for the computational requirements of the simulated objects.

Another project, named SimTool [Sassen et al 2004], embedded in the more ambitious VORMS project, shares some of our goals. Experiments done using SimTool have proved that students are highly motivated by interactive counterparts to textual presentations. One of the major differences we have found between SimTool and OPTIM is the implementation language, Java, which is, in the case of SimTool, the unique programming language, used for both the GUI construction and the computational tasks. We think that these two complementary parts will be better implemented by different authors. The GUI design involves the mastering of computer animation, graphic algorithms, and the full knowledge of the GUI's object library available. On the opposite, the computational part has to be written by the teacher, which is, in this case, the knowledge reference. Moreover, SimTool is only a tool for building simulation applet, and does not try to help the teacher throughout the whole design of his/her pedagogical document, as HDSML.

3.4 Another Hypertext Language?

The author familiar with HTML can have fear to throw away all this heavily acquired knowledge and to learn another hypertext dialect. We have multiple answers.

First, many excellent XML editors exist, for every platform, allowing everybody to find the one that meets its requirements. Such tools can be configured using the HDSML syntax we supplied, allowing a very high learning curve of HDSML, even for beginner editors. Such editing tools tutor users and help them by supplying only the tags allowed in the current context of definition.

The content author can also find all the legal attributes of every element, with a list of their possible values. Moreover, we plan to work on more conversion tools, expanding the way opened with the Word front-end processor already operational [Delebecque 2003]. These tools will be able to extract the structure of documents produced by the most common word processing tools available. Our goal is to simplify authors work and no to ask them to learn a new and strange language.

3.5 Is OPTIM universal?

We do not think to solve all the problems that communication between GUI and applicative code can rise, nor pretend to define an universal tool for building interactive applications. We try to describe an architecture that allows an easy separation between GUI and applicative code in an educational context, that have two main characteristics. First the GUI used in a pedagogical context are rather simple, and use a small number of objects. Lastly, the educational world uses a great variety of languages, even very specific ones, that should be interfaced with the GUI's objects. We have chosen an architecture that tends to solve the later problem, and defines a limited but extensible set of GUI objects. We think that these options can be of great help in authoring interactive and structured pedagogical materials of scientific courses.

BIBLIOGRAPHY

Sharon Adler, Anders Berglund, Jeff Caruso, Stephen Deach, Tony Graham, Paul Grosso, Eduardo Gutentag, Alex Milowski, Scott Parnell, Jeremy Richman, Steve Zilles *The Extensible Stylesheet Language* http://www.w3.org/TR/xsl

Apache Jakarta Project *Struts* http://jakarta.apache.org/struts/

Allen Brown, Matthew Fuchs, Jonathan Robie, Philip Wadler *XML Schema: Formal Description* http://www.w3.org/TR/2001/WD-xmlschema-formal-20010320/

Yolaine Bourda, Bich-Lien Doan, Henri Delebecque Learning Resources and the Semantic Web *Proceedings of Ed-MEDIA 2003*, Honolulu, USA 2003

John Cowan *Extensible Markup Language (XML) 1.1* http://www.w3.org/TR/xml11/

Henri Delebecque HDML: an Object-Oriented Hyper-document Describing Meta Language for E-education. *Proceedings of Ed-MEDIA 2003* Honolulu USA http://www.hdml.org

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

Microsoft Corporation http://www.microsoft.com/office/techinfo/productdoc/

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

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

Norman Walsh, Leonard Muellner *DocBook: The Definitive Guide* O'Reilly http://www.docbook.org/