OPTIM: AUTHORING INTERACTIVE TEACHING DOCUMENTS WITH MINIMAL PROGRAMMING COST

Henri Delebecque

OPTIM: AUTHORING INTERACTIVE TEACHING DOCUMENTS WITH MINIMAL PROGRAMMING COST

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

Abstract: This paper describes an open framework for teachers and lecturers in science. It will help them write their teaching documents even if these documents include interactive or multimedia content, leaving the entire presentation burden to the OPTIM rendering engine. This framework will also allow the global editor to define the authoring language automatically, and to put the structure common to sets of documents into classes for subsequent reuse. Moreover, OPTIM gives the authors a way to insert interactive parts into their documents without any GUI programming skills.

INTRODUCTION

OPTIM is a general framework for authoring teaching material, which uses HDSML (Delebecque 2004), a meta-language more dedicated to the definition of teaching documents than XML (Cowan 2004). XML allows the definition of any markup language, a versatility that leads to a heterogeneity of XML based authoring languages, and a lack of standards in the pedagogical domain. A general-purpose authoring language such as DocBook©, however, has its own drawbacks, since it defines only very general structural entities, like "header", "footer", "body".

We want to allow global document editors to define their own authoring language using the authors' usual terms as tag names. We think that this will be of great help for content authors, who are not always familiar with programming languages, and even less familiar with XML, to write their concrete documents. For example, mathematicians will greatly appreciate finding tags named «theorem», «premises», «conclusion» and «proof» in the authoring language they use. This is something that people have to do manually if they use authoring languages such as Latex.

The OPTIM framework extends HDSML capabilities by allowing authors to insert interactive objects into their teaching documents. Students and learners that use such material can visualize the concepts described in a graphical manner, while keeping additional textual descriptions.

The pedagogical domain has several good properties that help us: the graphical objects found in applications are of a limited variety, as is the kind of interaction the learners have with them. Pedagogical software (even simulators) commonly outputs its results either as graphs or as animated diagrams, and requires minimal or no user interaction for the inputs.

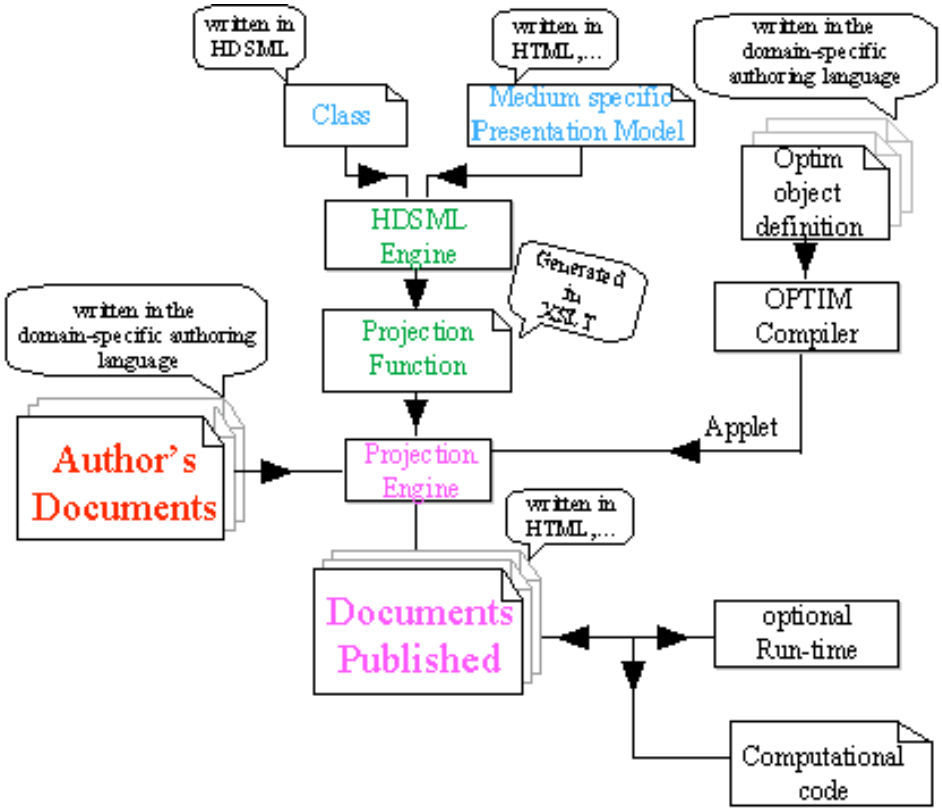
OPTIM: UNDERLYING PRINCIPLES

The acronym OPTIM stands for Open Platform for Teaching Interactively with Multimedia. OPTIM's goal is to give teachers an adaptable and lightweight framework that allows them to spend more time on content (whatever its form) and less in presentation problems, while using their favorite editing tool. This last point is a key one in the world of education, where languages and tools are usually domain-specific. It contributes to the openness of the OPTIM framework, a property that allows the author's investment to keeps its value over time and resists changes in proprietary format. OPTIM uses HDSML to describe the structure of the document's static part.

The global editor of the global teaching document will help teachers to write their sections/chapters/ books in a fairly uniform manner, by defining a structural model. It defines all the mandatory or optional elements available, which authors have to include in their document. The projection of the actual documents written by teachers onto various media is done by defining one presentation model per publishing medium. These models should be written probably by a specific designer, more familiar with ergonomics and graphics than our usual authors. HDSML has to be clearly a Meta language, since the structural model is also used to define the markup language, described above, that we will call the content authoring language.

Definitions

HDSML is used by the three different kinds of authors described in the following paragraph, while defining four kinds of documents, which are: the class (the structural model), the presentation model, the concrete document (a class instance), the interactive objects (OPTIM objects). The HDSML engine works in three phases, using the four different kinds of document listed above.



We want to guaranty the uniqueness of the source document, whatever publishing medium is used. The pair of models (one structural model and one presentation model) helps to achieve this goal by allowing the necessary separation between presentation and content as soon as possible. This dramatically reduces the authoring time spent during the meta-authoring phase that ends with the generation of a XSLT (Adler et al. 2001) file for each pair of models. Each XSLT file implements a projection function from the class to one specific publishing medium. As mentioned earlier, the HDSML engine will also define the authoring language grammar, based on the class definition.

Teachers will write their teaching documents using this grammar. This authoring task can be facilitated if they use XML compliant tools providing a syntax-sensitive help based on the grammar definition. Nevertheless, HDSML has proved its openness by working with documents written using other tools (and formats).

The HDSML engine takes the concrete documents and transforms them to publishable ones by applying the projection function devoted to the chosen publishing medium. These later documents

are a mix between items taken from concrete documents, and others dedicated only to presentation purposes like a logo or the background.

During the publication phase (and when learners use them) the published documents can activate the OPTIM objects they contain. 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 during the previous phase.

Two different kinds of event may occur: purely "graphical" events (such as mouse-down, key-down, etc.) fully managed by the Java code, or "applicative" events. These last ones are transformed into requests sent to the computational code that answers with numerical results displayed as a graphic, a diagram, which can both be static or animated. This division into purely graphical events (also called low-level events) and more semantic events follows the evolution between Java's 1.0 and 1.2 event model.

OPTIM Architecture

Every interactive object included in a concrete document should be described in two complementary ways. First, the graphical designer has to explain how the graphical sub-objects will respond to end-user stimuli (click, keyboard events…). And, second the computation code should be able to send refresh orders to the GUI, when the code changes the data displayed. These two aspects are managed by a conventional MVC architecture evangelized by Java. This very efficient and robust way of separating the Model (in charge of all the computational tasks) and the View (that displays the values in the most convenient way) has proved its value since its first definition by the Smalltalk object-oriented design team (Goldberg & Robson 1983).

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.

The block is the most versatile element, able to hold either 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 the teacher when the concrete document is written. The «name» attribute allows the domain-centered definition of the content authoring language. Finally, a block can be structured using sub-elements taken from the following non-exhaustive list: Block, Object, Keyword, OPTIM. The recursive nature of the block avoids the definition of a limiting list of structuring levels, as in DocBook, while giving it more descriptive power.

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 basic structured elements. The keyword is a powerful tool for the content author, who can use it to add semantics to (potentially) every document element, even the smallest. The OPTIM element is the link between the actual concrete document and the interactive objects built using the OPTIM platform.

RELATED WORK

OPTIM Versus Current IDEs

One can think that existing Integrated Development Environments, like JBuilder©, or Visual Studio©, are the better choice for developing the interactive parts of our teaching 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 measured in the GUI's design, but rather in the development of the simulation part. And we think that versatility and openness is crucial here, allowing our teachers to use their favorite programming languages. Moreover, IDEs often impose the mastering of many specific programming skills — such as animation techniques, graphical algorithms, which are clearly not trivial— which requires a considerable investment, and constant updates.

Programmingless Development Tools

New products, like Gatonero appeared recently that allow the rapid design of interactive applications by non-programmers. It offers the ability to describe sophisticated prototypes using the specific language Gatonero defines, 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 prototypes re-usability, as multi-platform deployment, since it is Java based. We think that this kind of products does not fully meet the requirements of the teaching community. These products tend to propose Rapid Application Development tools for business purposes. Teachers , on the other hand, ask for a framework that allows them to present data produced by a regular computational code in a more pedagogical way. Moreover, like the SimTools project mentioned below, Gatonero suffers from a 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 highly portable client-side software to interact with a computational code located on a dedicated remote server. This also requires the teacher to convert the computational code into a foreign programming language, at a great cost.

E-Learning Development Tools

We have found active projects, like the one described in (Matsuno et al. 2004), that try to give teachers a way to design interactive teaching documents without programming. This project is component- ware based, and has the same openness property as OPTIM, since it offers links with databases, e-mail, etc. But this project is currently based on the Delphi© programming language, and seems to have no capability to deal with other programming languages. Moreover, it lacks the regular model given by HDSML to OPTIM, and which guarantees a high level of adaptability to upcoming application domains.

Another project, SimTool (Sassen et al. 2004) shares some of our goals. Experiments conducted using SimTool have shown that students greatly appreciate interactive counterparts to textual presentations. One of the major limits of SimTool is the choice of Java for both the GUI construction and the computational tasks. We think that these two complementary parts are better implemented by different authors, and that one should leave greater freedom in the choice of the computational language. The mastering of computer animation, graphic algorithms, and the full knowledge of the GUI's object library are required to design a good graphical user interface. On the contrary, the teacher who 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 achieve it. Moreover, SimTool's goal is the building of simulation applets, and it does not cover the whole design of their teaching document, as does HDSML.

REFERENCES

Adler S., Berglund A.(2001) The Extensible Stylesheet Language http://www.w3.org/TR/xsl

Cowan J. (2004) Extensible Markup Language 1.1 http://www.w3.org/TR/xml11/

Delebecque H. (2004) HDSML: a Hyper Document Structuring Meta Language. *Proceedings of ED-MEDIA 2004* Lugano Switzerland

Goldberg A., Robson D. (1983) *Smalltalk-80: The language and its implementation* 1983 Addison Wesley

Matsuno R., Tsutsumi Y., Gilbert R. (2003) A Tiered Approach Utilizing Reusable Componentware Methodologies for Multimedia Educational Software Creation and Development *Proceedings of ED-MEDIA 2004* Lugano Switzerland

Sassen I., Reiners T., Paschilk B., Voß S. (2004) Instructional Design and Implementation of Interactive Learning *Proceedings of ED-MEDIA 2004* Lugano Switzerland