HDSML: a Hyper Document Structuring Meta Language

H. Delebecque
Supelec
Plateau de Moulon
91192 Gif sur Yvette Cedex France
Henri.Delebecque@supelec.fr

Abstract: In this paper, we propose an XML application for content authors, to help them write their hypertext documents in factorizing their common structures into classes, and to free them from all other design considerations. Moreover, our language allows authors to structure their documents homogeneously, and to add semantics into them at a fine grain level. The editor in charge of the global pedagogical document designs both structural models (classes), and presentation models for each sort of media (using HTML, WML…). Using these models, our HDSML engine automatically builds the projection function able to transform any existing document into publishable forms.

HDSML Definition
The HDSML acronym means «Hyper Document Structuring Meta Language». HDSML’s goal is to give to content authors the ability to focus on content and ignore presentation, while using their favorite editing tool. To achieve this goal, the editor of the whole pedagogical document will defines all the mandatory or optional elements content authors have to put into their documents. Moreover, the editor can give them the names that are the most familiar to content authors, such as the proposition, the conclusion and the proof for a theorem definition.

The main difference between HDSML and its predecessor, HDSL (Delebecque 2002), is that HDSML is clearly a meta language, since it allows the class designer to define a markup language that we will call the content authoring language. It will be used for easily writing documents related to the content authors domain. The content authoring language can be considered as an instantiation of HDSML. 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 whole web pedagogical document design, we will generalize hereafter under the global document term. The class designer, who is also the whole document editor, defines the structural model common to all the chapters/sections of the document. 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.

General Working Principle
Meta Authoring Phase
To ensure the separation of semantics and presentation, we define a global lightweight framework working with pairs of documents. For every subset of the global document (book, chapter, section…), the document editor can supply a specific class 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.

Our HDSML engine takes all these pairs of models, and then automatically generates as many XSLT style-sheets as needed to produce «displayable» forms of the concrete HDSML documents on the various presentation media. Thus, every XSLT style-sheet can be considered as 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, the authoring phase.
**Authoring Phase**
During this phase, the content authors will write the concrete documents, using any editing tools they want, even if those tools are not XML compliant. Of course, the great advantage of using XML compliant editing tools is that they are able to give help according to the grammar context of the word currently edited. For example, an editing tool able to understand the XML Schema grammar definition can display the attributes valid for the tag currently edited, or the children it will accept.

**Pre-publication Phase**
In the last phase, the projection function generated at the end of the meta-authoring phase is applied to the concrete documents that conform to the class structure, giving the documents to publish. These later documents can of course include items independent from their corresponding concrete document content (i.e. background, logo,…) the presentation model provides.

---

![HDSML Engine Principle](image)

**Figure1. HDSML Engine Principle**

**HDSML Versatility**

**Publishing Language**
HDSML accepts currently presentation models only written in HTML. Nevertheless, this language should be considered in its great diversity. One HTML «display engine» can only render text (for example Lynx), when another deals with picture, sound, movies. The browsers used by the computers of an intranet zone are commonly identically configured, and represent a very homogeneous set of HTML renderers. The HTML delivered to them can thus be very fine tuned and optimized. At the contrary, the HTML dedicated to the heterogeneous browsers of the external world wide web should be designed on the basis of the minimal requirements.

HDSML will be easily extended to many other publishing languages such as PDF, WML since its working principle require only that a presentation model would be supplied for each publishing medium, and each specific look-and-feel.

**Concrete Documents Format**
HDSML also accepts as concrete document a file that is not expressed in the authoring language at all. For this kind of files, such as Microsoft Word ones, a specific front-end is provided, which analyzes the content according to grammatical rules. These rules describe how the bookmarks and styles found in the document have to be interpreted to reveal the document internal structure. They are stored in a specific file, called the *front-end grammar file*.

**HDSML Concepts**
Let us start with the definition of the main structuring entity both the class designer and the content authors will use: the element.
The Element
These authors structure their classes or concrete documents by dividing them into 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).

The Class
The class definition is the editor’s responsibility. The structure of the concrete documents can be loosely bounded to the one 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 HDSML defines (or instantiating them in a object-oriented meaning), to describe the structure common to the global document. We can also consider the class as the definition of the content authoring language.

The Presentation Model
The presentation model should probably be designed by a specific author, since it implies more competencies in ergonomics and graphics than in the document content domain. This author can choose the presentation language according to the presentation medium, as soon as it is supported by the HDSML engine (currently all the HTML dialects). He has also a complete freedom for the publishing tool he will use for the presentation model authoring. This model organizes the look and feel of the all the class building blocks.

The Concrete Document
The HDSML engine can manage a large family of concrete documents, written by various content authors. This family builds the class instance set, and the global document. Each content author can freely edit his concrete document, using his favorite publishing tool, without the burden of sharing a common document with other content authors. 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 projection by the HDSML engine.

Another great advantage of this concrete document set is its uniqueness, stressed as a key point for Content Management Systems in (Bergstedt et al. 2003) , even if it has to be published on multiple media, thanks to the HDSML engine projection functions. The content authors gain the insurance that the information published on every medium will conform to the content they have defined, even if it is very dynamic.

Inheritance in HDSML
A document family definition can itself be derived from another wider family. This is useful for example if the former is a chapter of the book described by the later, or the web pages of a department in the university web site. XML Schema (Davidson et al. 1999) provides a simple but useful inheritance feature among types. HDSML goes many steps further.

Structural Inheritance
A class can inherit its structure from another, which itself can do it from another, as deeper as required. In this case, we call the class inheriting a subclass, and the set of classes from which the subclass inherits the superclasses, as in object-oriented languages. The subclass can add new elements, overload or delete some elements one of the superclasses defines.

This removing capability can have multiple practical applications. First, it will simplify the authoring task, by removing useless structural parts. The content author can thus focus himself on the elements relevant for him. Last, it raises the global document coherency by insuring its editor that critical element values (such as the version, date…) are hidden to content authors, and thus not modified by concrete documents.

As in object-oriented languages, the various elements described by a class are unordered, even if they are inherited from a superclass. That allows the content author to organize the internal structure of his concrete document in the easiest way for him, and not for the tools that will process it after authoring.

Behavioral Inheritance
The concept of method, familiar in object oriented languages, could seem more strange in HDSML, due to its lack of programming features. Let us define it by analogy: we know obvious operations that can be applied to documents, such as the concrete document initialization or its validation versus a XML Schema grammar…
Related Work

HDSML versus Tag Libraries or CSS
One can think that the powerful capabilities tag libraries like with Struts (Apache Jakarta Project 2000) add to HTML, allowing an easy creation of a coherent set of HTML pages, even if they are dynamically produced, can be an alternative to our language. We think that tag libraries are clearly a good solution, but limited to the HTML production, and tend to stress on the look-and-feel, when we want to free the authors from such considerations. We can note that CSS share the same drawback.

HDSML versus XSL Stylesheets
The solutions XML currently offers, such as pure XSL sheets, has multiple drawbacks: you have to fully master XML and XSL. And this mastering has to be regularly exercised for two reasons. First, you have to write a XSL sheet for every presentation medium you use, and for every subset of your pedagogical document. Then, this XSL writing task must be redone every time you change the look and feel, and for every presentation medium, and for every subset of the global pedagogical document.

Moreover, this way of producing web pages doesn’t allow re-utilization or genericity, which our tool includes, 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.

HDSML: Another Hypertext Language?
The more or less HTML familiar writer can have fear to throw away all this heavily acquired knowledge and to learn another hypertext dialect. We have multiple answers.

First, a great number of excellent XML editors exist, for every platform, allowing everybody to find the one that meets its requirements. Such tools can configure themselves with the HDSML XML-Schema syntax definition we supplied, allowing a very high learning curve of HDSML, even for beginners. 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 described previously. These tools will be able to extract the structure of documents produced by the most common word processing tools available.

HDSML: Another Meta Language?
It can be argued that the differences between XML and HDSML are not sufficient to justify its definition. We think that mastering XML and all its powerful associated tools is not a simple task. As mentioned above, XSL is not an intuitive language, and the good definition of an XML-Schema grammar requires time and competence. Thus, our goal is to simplify the tasks of both editors and content authors, while keeping the power of a meta-language. This can perhaps avoid the emergence of a great number of multiple and incompatible authoring languages based on XML, which is clearly not desirable.

References

ED-MEDIA 2002 Denver CO USA  
http://www.hdml.org

http://www.w3.org/TR/NOTE-SOX

Apache Jakarta Project (2000) *Struts*  
http://jakarta.apache.org/struts/

ICALT 2003 Athens Greece