



Project no. FP6-028038

Palette

Pedagogically sustained Adaptive LEarning Through the
exploitation of Tacit and Explicit knowledge

Integrated Project

Technology-enhanced learning

D.INF.05 – Second version of document authoring services

Due date of deliverable: January 31, 2008
Actual submission date: February 28, 2008

Start date of project: February 01, 2006

Duration: 36 months

Organisation name of lead contractor for this deliverable: INRIA

Project co-funded by the European Commission within the Sixth Framework Programme	
Dissemination Level	
PU	Public

Keywords: Document authoring, structured documents, document model, multimedia
Responsible Partner: INRIA

<i>MODIFICATION CONTROL</i>			
Version	Date	Status	Modifications made by
1	31-01-2008	draft	Nabil Layaïda, Jan Mikáč, Vincent Quint, Cécile Roisin
2	12-02-2008	draft	Étienne Vandeput
3	14-02-2008	evaluation	Vincent Quint based on comments from Liliane Esnault and Christine Vanoirbeek
4	26-02-2008	Final	Vincent Quint

Deliverable manager

Vincent Quint (INRIA)

List of Contributors

Émilien Kia, Jan Mikáč, Nabil Layaïda, Vincent Quint, Stéphane Rieppi, Étienne Vandeput, Irène Vatton

List of Evaluators

Liliane Esnault (AESCRA-EM Lyon), Christine Vanoirbeek (EPFL)

Summary

This document presents the second version of Palette authoring services developed in workpackage 2. It provides an introduction to the new features that were introduced in version 10 of the Amaya web editor and in version 1.0 of the LimSee3 multimedia editor. The emphasis is put on user interface improvements. Recent evolutions of the document model used in LimSee3 are also discussed, in particular its ability to export documents in various formats.

TABLE OF CONTENT

1. Introduction.....	5
2. Amaya	6
2.1. Motivations	6
2.2. Template driven editing	7
2.3. A new, customizable user interface	8
2.3.1. User interface profiles.....	8
2.3.2. New tools, buttons, and menus	9
2.3.3. Interaction modes.....	11
2.3.4. Other changes.....	14
2.4. Interoperability with other tools.....	15
2.4.1. Email messages.....	15
2.4.2. Connection with SweetWiki	15
3. LimSee3.....	16
3.1. Evolutions of the LimSee3 tool	16
3.1.1. General considerations.....	16
3.1.2. Main achievements	17
3.1.3. Current limitations and future developments.....	18
3.2. Examples of developments	19
3.2.1. User interface improvements	19
3.2.2. Specific application - Multimedia course builder.....	24
3.2.3. Conclusion on the outcomes of the two collaboration experiences	27
3.3. Evolutions of the document model	28
3.3.1. Exportation needs for LimSee3	28
3.3.2. A model for exporting LimSee3 documents	29
4. References.....	31
Appendix A - commented DTD of the intermediate format	32
A.1 General intermediate document structure	32
A.2 The header section	33
A.3 The layout section	33
A.4 The timing section.....	34
A.5 The references	35
A.6 The media section	36
A.7 Common definitions.....	37

1. Introduction

Workpackage 2 of Palette aims at providing services for information production and reuse in communities of practice (CoP). Multimedia, structured documents and presentations play a key role in CoPs, but taking full advantage of the most recent advances in multimedia document technology requires either high technical skills or sophisticated tools. The objective of WP2 is to create the efficient services and tools that are necessary for average CoP members to easily produce, use and reuse this type of information. More precisely, efforts in this workpackage focus on two services, one for multimedia content authoring, the other for structured documents reuse. This report addresses only the former. Refer to D.INF.04 [5] for more information about the document reuse service.

Regarding multimedia content authoring, the objective is to provide users with a flexible, adaptable environment for efficiently producing different types of multimedia documents, specifically targeted for use in CoPs. This goal is achieved with two complementary open source tools, Amaya and LimSee3. Given the wide range of issues to be addressed, developing a single piece of software for addressing all the needs would be risky. Each tool in itself is of significative size (Amaya is almost 1 million lines of source code) and it was considered much more efficient to develop two separate software projects, each one focusing on different issues. Also, feedback from CoPs shows that authors are used to manipulate different types of documents with different tools: they use for instance Adobe Premiere for handling video and sound, and Microsoft Word for text and images. This separation of concerns in different tools will probably last for a long time.

LimSee3 is an authoring application featuring a powerful WYSIWYG graphical user interface designed to ease the manipulation of time-based multimedia documents and to increase productivity. It focuses on the *time aspect* of documents. It handles *continuous* media objects such as video and sound. It allows users to manipulate synchronization of document components, transitions between media objects, animations, etc.

Amaya is a web editor designed to ease editing, publishing, and sharing of rich web pages containing *discrete* media objects (text, still images, vector graphics, mathematical expressions), as well as style information. It focuses on the use of the web and its specific features: links between resources, on-line distributed content, adaptable and accessible documents, separation of style from content, etc.

Although the focus is different for each tool, there is some commonality between them. Both are based on the same principle of structured documents. Both feature a graphical user interface and allow authors to interact with a graphical, familiar representation of documents, even if a more abstract structure is handled behind the scene. Both use open document formats based on XML, namely those created by the World Wide Web Consortium (W3C) for multimedia web resources: XHTML for structured text, SVG for vector graphics, MathML for mathematical expressions, SMIL for synchronized multimedia, and CSS for style sheets. This allows content created with both authoring tools to be integrated smoothly in the web environment used by CoPs [1].

The early versions of both tools, Amaya v9.54 [3] and LimSee3 v0.5 [4], were made available for two main purposes:

- to allow developers of other Palette services to experiment with the authoring service, thus making it easier to consider possible interactions and interoperability issues with other services,
- to allow CoPs to experiment and provide feedback about the authoring service and their expectations for their day-to-day practice.

Since then, multiple snapshots were published, to allow both users and developers to closely follow the development work and to react promptly to the latest evolutions. This document presents new stable versions of the tools, i.e. versions that provide a consistent set of features that have benefitted from the participatory design method adopted for the whole project. These are version 10 of Amaya and version 1.0 of LimSee3.

The document focuses on the the main changes that were made since the early versions mentioned above. These changes fit in two categories:

- new features that are part of the basic functionality of each tool and were not available with the early versions,
- evolutions inspired or requested by the developers of other Palette services to improve interoperability and by CoPs representatives to better address their needs.

Only the main changes are presented here. More details about the tools are available with the on-line documentation.

The software described here is freely available on the web:

- Amaya download site:
<http://www.w3.org/Amaya/User/BinDist.html>
- LimSee3 download site:
<http://limsee3.gforge.inria.fr/public-site/software.html>

The rest of this document is organized in two sections, one dedicated to each tool. An appendix presents the details of the intermediate format of LimSee3 introduced in section 3.

2. Amaya

2.1. Motivations

Some recent developments in Amaya are simply the finalization of the work that was initiated for the early version (9.54). Most of other developments were improvements motivated by comments and feedback from CoP representatives and tool representatives, as a result of the participatory design method.

More specifically, section 2.1 of report D.PAR.04 [7] provides a detailed discussion of the main features of Amaya and a critical analysis of its user interface as seen by CoP members. Continuous collaboration between the authors of this usability analysis and the developers of Amaya led to a completely new user interface, which is now part of version 10. Its main features are presented in section 2.3.

Scenarios defined by CoPs and presented in D.PAR.03 [6] were another strong motivation for changing Amaya. With scenarios, new requirements were identified regarding new features and interoperability with other tools and services. To summarize, the new developments requested by CoPs through scenarios were the following:

- ePrep (D.PAR.03 Appendix 4 section 4): template-driven editing had to be completed to allow users to produce various types of pedagogical content. A connection to the email service was also requested for sending documents to other CoP members and to keep them aware of the availability of new (versions of) documents.
- Did@cTIC (D.PAR.03 Appendix 5 section 4): in this scenario, Amaya is used for editing various types of documents. DocReuse is involved too. To address interoperability issues, Amaya and DocReuse take advantage of the template mechanism provided by Amaya; they are sharing common document models.
- Form@HETICE (D.PAR.03 Appendix 7 section 4): Here again templates are used to produce various types of documents, and support for email in the editor is needed for better communication and awareness within the CoP.
- @pretic (D.PAR.03 Appendix 8 section 4): This scenario involves SweetWiki and Amaya, among others, and demonstrates a need for interoperability between them, to allow documents developed with one tool to be freely edited with the other, depending on the type of task to be performed.

2.2. Template driven editing

Most of the work done for template driven editing falls in two categories: unfinished features that were reported in the "Current limitations" section of the previous deliverable D.INF.02 [3], and new features that make it easier to use templates.

Finalizing the template-driven editing feature includes:

- Full implementation of the XTiger attribute element. This allows templates to control the way XHTML attributes may be used in document instances, as specified in the XTiger language (see D.INF.02 [3]).
- Full implementation of the XTiger option element. In particular, it is now possible to remove an optional element and to reintroduce it later on.
- Use of templates within tables. All XTiger elements can be used to control the structure of tables. In particular, it is now easy to create multiples instances of the same model of row within a table, the model being defined by XTiger elements.

Making it easier to use templates:

- A contextual menu was introduced that lists all the elements that are allowed by the template at any location it controls in a document instance. Selecting an entry in the contextual menu creates an instance of the corresponding element.
- Command **File/New document** offers now an option for choosing a template that will rule the created document. The **Edit/Preferences** dialog box allows the user to create and update the list of his/her preferred templates, which can be local files as well as web resources located on remote servers. This list is proposed when creating a new document.
- Command **Save as** provides a new option for saving a document without its template. This is useful to save space when the document is supposed to be frozen. It could also make it easier to process documents with XML processors that are not yet namespace-aware and may be confused by the presence of template elements in a document.

2.3. A new, customizable user interface

The design of the new user interface was driven by three main considerations:

1. Amaya offers a number of sophisticated features for *web editing*, but when it comes to creating a simple document, average word processors users can get lost with such a tool that does not provide the same services as their usual document editor. There is a need for a simple version of Amaya that offers only the basic functions most word processors offer. This basic version is called *Amaya Lite*.
2. In addition, to allow new users to easily learn how to use this simple version, its user interface must look and feel like the interface of the most common word processors.
3. Different users may have different skills, and a user may learn how to use advanced features. To avoid reducing Amaya to the least common denominator, and to allow interested users to take advantage of its innovative features, it should be possible to incrementally add new features to the basic version. This would allow each user to tailor the tool to his/her own needs, without being disturbed by unused features.

WP1 partners specified the set of features to be put in *Amaya Lite* (item 1 above), as well as the user interface for this version (item 2). This work was based on the feedback from CoPs and on a detailed usability analysis of the complete tool. The outcome was a mockup showing a clear view of the proposed UI. Based on this mockup, the developers implemented the new interface, providing intermediate versions (snapshots) to get feedback as work was progressing. Item 3 above was addressed by introducing a profile mechanism in Amaya.

2.3.1. User interface profiles

All features available in the full version of Amaya and their corresponding UI components were grouped into several sets that can be freely combined. Each combination of feature sets is called a *profile*. In Amaya 10, the available feature sets are:

- Lite: the smallest set of features that allow untrained users to edit most documents.
- Math: everything a user needs to edit mathematical expressions within a document.
- Web: the browsing feature that allows Amaya to be used simultaneously as an editor and as a browser; it also allows users to manipulate links to local and remote resources.
- Style: allows users to create, edit, save, and link CSS style sheets. Note that the Lite profile includes a few very basic style features.

Based on these sets, the following profiles are available with Amaya 10:

- Lite
- Lite + Math
- Lite + Math + Web
- Lite + Math + Web + Style
- Advanced (all features available).

The profile mechanism is flexible enough to allow developers to quickly define new feature sets and new profiles in upcoming releases or snapshots. Users can freely choose the most appropriate profile among those available, and they can change profile at any time through the **Edit/Preference** dialogue.

Figures 1 and 2 show the user interface of the lowest profile, Amaya Lite, and the highest profile, Amaya Advanced, respectively.

This mechanism goes beyond the usual feature that allows users to organize their graphical user interface by moving, adding or removing various toolbars and panels. This feature is indeed present in Amaya, but it comes in addition to the profile mechanism. The user is free to remove, add or move dynamically some widgets to customize the interface at a second level of customization. The first level, the profile, is used to tailor the tool itself (its features, not simply the user interface) to a particular type of task or to a category of users.

Note that the part of the user interface that is specific to templates is not affected by any level of customization, as it only appears within documents.

2.3.2. New tools, buttons, and menus

To implement these new feature sets, it was necessary to reorganize menus, toolbars and panels. Some of them were redesigned. Some others were created from scratch. The main idea was to offer the same buttons and menus as most word processors. These allow authors to create the most common XHTML elements (`h1`, `h2`, `h3`, `h4`, `p`, `ul`, `ol`, `dl`, `strong`, `em`, `code`, etc.). Some of these usual buttons are considered by word processor users as adding style to their documents. They are actually used in Amaya to create XHTML elements when the correspondence is obvious (`strong` for bold, `em` for italics, for instance). Some other buttons just create style in XHTML documents (line spacing, text alignment, indent, margins, colors). They add a CSS property to XHTML elements through a `style` attribute,

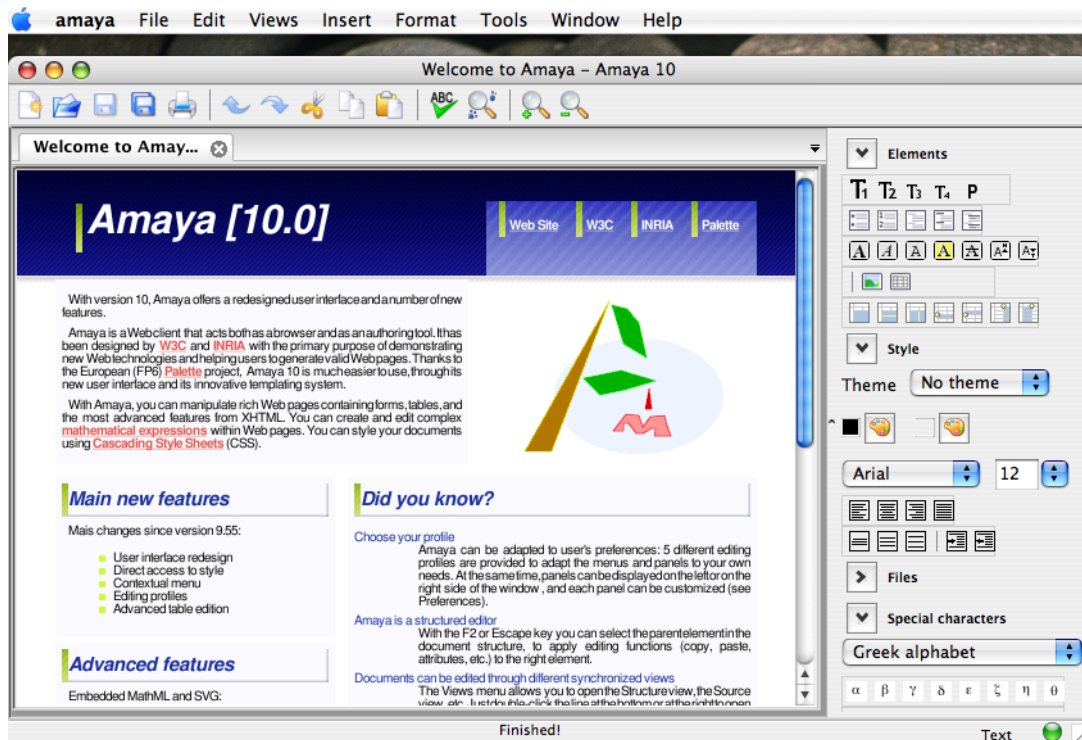


Figure 1: menus and tools in Amaya Lite

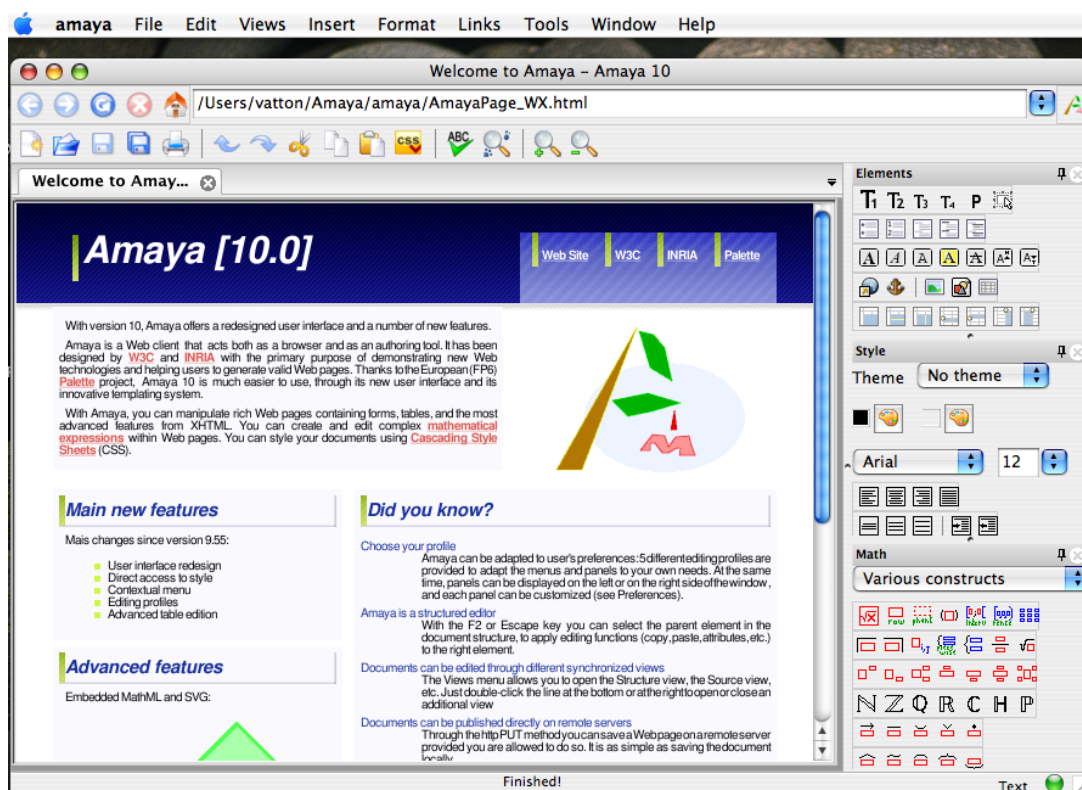


Figure 2: menus and tools in Amaya Advanced

creating a span element first when the current selection does not correspond to an entire element in the XHTML structure.

However, to encourage users to focus on the structure rather than adjusting stylistic details on every element, a **Theme** command is provided in the **Style** panel. It allows authors to associate a predefined CSS style sheet with the document. With this command, users can change style sheet at any time. The proposed style sheets specify the look of most element types available in XHTML. Thus, when changing theme (i.e. changing style sheet), the whole aspect of the document is changed at once, and in a consistent way.

This way of manipulating style sheets does not take advantage of all the features available in the CSS language, but, as experience has shown, it provides a simple mechanism that all users can understand. Users do not have to learn about CSS selectors, properties and the cascade, but they are still encouraged to separate style from structure. Like for other extensions, the **Theme** command comes in addition to the mechanisms that already exist in Amaya for handling CSS style. In particular, these mechanisms included in the Style feature set, can be used by advanced users to create new style sheets, that can easily be added to the themes already offered.

In addition to the tools and buttons shown on the right side of Figure 1 and Figure 2, reorganizing the graphical user interface led to changes in the menus. Figure 3 presents all menus available in Amaya, except the **Help** menu which was removed from the figure for the sake of clarity. Menus and entries displayed in black on a grey background are available in all profiles, starting with Amaya Lite. Menus and entries in gray on a white background are available with higher profiles. Only Amaya Advanced offers them all. Entries with a "►" call a submenu. Entries ending with "..." call a dialog box.

2.3.3. Interaction modes

Amaya is first and foremost a *structure manipulation* tool, i.e. it allows users to manipulate all the aspects of XHTML documents, which are primarily *XML structured documents*. A number of editing commands perform therefore a direct action on the document structure. This includes commands entered through the most common keys of the keyboard. The Enter key for instance is used to split elements and to create new ones in the structure, while the Backspace and Delete keys have the symmetric role of merging existing elements. Typically, when the insertion point is within an element, the Enter key splits that element and its ancestors in the tree structure, using the DTD to find the right level where splitting is allowed. When the insertion point is at the beginning or at the end of an element such as a paragraph or a heading, the Enter key creates a new element in the structure, choosing the element type and the level in the tree structure according to the DTD. More details are given in [10].

This way of interacting with the document structure through a few key strokes is called the *interaction mode*. The native interaction mode in Amaya is called the *XML mode* because it is primarily intended for manipulating XML structures. Indeed, many different XML

File	Edit	Views	Insert
New tab	Undo	Show/Hide toolbars	Paragraph
New document... New ▶	Redo	Show map areas	Heading ▶
Open document ▶		Show targets	List ▶
Reload document	Cut		Information type ▶
Read as ▶	Copy	Show structure	Character element ▶
Back	Past	Show source	Image
Forward	Delete	Show links	Horizontal rule
	Insert	Show alternate	Break
Save	Append	Show table of contents	Preformatted
Save as ...	Select all	Split view horizontally	Insert a table Table ▶
Save all	Select ▶	Split view vertically	Math ▶
Synchronize			Form ▶
	Find	Document info	Object ▶
Send by mail		Show parsing errors	Map area ▶
	Preferences		Hidden elements ▶
Setup and print		Full screen	Division
Print		Zoom in	address
		Zoom out	blockquote
Close tab			Inline frame (iframe)
Close window			ruby
Exit Amaya			

Format	Links	Tools
Theme ▶	Copy location	Check spelling...
Text color ▶	Link to previous target	Transform...
Background color ▶	Create or change link	Edit table ▶
Alignment ▶	Create target	Change title...
Line spacing ▶	Delete link or target	Change doctype ▶
Indent ▶	Add/Remove IDs...	Section numbering
Character ▶		Add/Update table of contents
		Make book
Style editor...		Insert time stamp
Create rule...		
Show applied style...		Graphics ▶
Style sheets ▶		Annotations ▶
		Cooperation ▶

Figure 3: menus

structures can be edited in that mode. The simple manipulations performed through a few keystrokes in the XML mode are generic: they can be performed for any type of element from any XML vocabulary, whatever their semantics. It works well with XHTML text, but also with MathML equations or SVG drawings.

As menus, panels and tools were redesigned in Amaya 10 to stick more closely to the user interface of a word processor, a new interaction mode, called *Text mode*, was introduced, in addition to the original XML mode. The XML mode has proven to be very efficient for manipulating different types of XML structures, but it requires from the user a good knowledge of the document structure. And for XHTML, it does not reproduce the exact

behaviour of word processors. For instance, when the cursor is at the end of a list item (`li` in XHTML), the Enter key creates a new paragraph within the list item (this is implied by the XHTML DTD), and a second hit on the Enter key is needed to create a new list item instead of the empty paragraph. This is very convenient for trained users who can manipulate different types of structured documents and data consistently. But, when manipulating only XHTML structures, the XML mode is different from the behaviour of a word processor, where a single hit creates immediately a new list item, and adding a paragraph within an item is done differently.

The new *Text mode* copes with this kind of discrepancy, allowing users to handle XHTML elements in the same way as in a word processor. To achieve this, XHTML elements are grouped in three categories (Blocks, List items, Inline elements), the most common keys are processed differently for each category in Text mode:

Blocks

XHTML elements such as `p`, `address`, `dt`, `h1`, `h2`, etc. (they are called *block* elements) are handled like paragraphs in a word processor. If the insertion point is within a block element when the Enter key is hit, the block element is split into two sibling elements of same type. If the insertion point is at the beginning or at the end of a block element and the Delete or Backspace key is hit, the whole content of the next (resp. previous) sibling element is merged with the content of the block element. When the Enter key is hit at the beginning or at the end of a block element, a new paragraph (`p` element) is created, whatever the type of the current block element.

List items

XHTML elements such as `li` and `dd` (list items) appear in lists (`ul`, `ol`, `dl`). They are handled in the same way as blocks, but the Enter key is handled a bit differently. At the beginning or at the end of a list item, a new sibling list item element of same type is created, instead of a new paragraph for a block. In addition, in an empty list item, the Enter key creates a paragraph that breaks the list.

Inline elements

When the insertion point is in XHTML elements such as `strong`, `em`, `code`, `span`, `a`, etc. (they are called *inline* elements), the Enter key splits the inline element and the event is transmitted to the parent element. As inline elements may be nested in a XHTML structure, the split may happen at several levels, until the event reaches a block or list item element, which processes the event as explained above. The Backspace and Delete keys only delete characters, crossing inline element boundaries without any action on the structure. However, when an inline element becomes empty by deletions, it is itself deleted.

These three categories are associated with the relevant XHTML elements using a configuration file, in such a way they could be used also for other XML vocabularies. For instance, DocBook also has itemized lists, and the Text mode can be easily associated with DocBook `listitem` elements.

All XHTML elements belonging to these three categories are themselves usually included in elements like the body of a XHTML page (`body`), divisions (`div`), table cells (`th`, `td`), etc. With the exception of tables, these elements are not familiar to word processor users, who do not attempt to manipulate them. In Text mode, no special commands are then required to create or change them, and advanced users may simply use the usual structure manipulation commands offered by the XML mode.

Specific commands are necessary for tables, because of their two-dimensional nature which distinguishes them from other XHTML elements. These commands are available in both Text and XML modes. Adding a column in an XHTML table requires a lot of changes to the XHTML structure, and these changes could be very complex when existing cells span over several columns. Even trained XML users have troubles manipulating tables with a plain XML editor. For this reason, specific commands have been added to Amaya for creating table columns and rows, and for merging, extending or shrinking cells. They are available as a set of buttons in the **Elements** tool and in menu **Tools/Edit table**.

Changing interaction mode is possible at any time, in both directions (from Text to XML and vice-versa). A toggle button is available at the bottom right corner of the Amaya window for that purpose.

2.3.4. Other changes

In addition to that evolution of the user interface, a few more changes have been made to address requests from users:

- A contextual menu (right click) offers the most common editing commands.
- The **Insert** and **Append** commands from the **Edit** menu can be used to insert before and after the selected structure element.
- The tabs associated with each open document can be freely reordered by drag and drop.
- These tabs have a Close button which closes the corresponding document.
- The application does not exit automatically when the last tab (i.e. document) is closed.
- Two documents can be displayed (and edited) side by side.
- The **Open document** command from the **File** menu shows the list of the recently open documents and allows these documents to be opened in a single click.
- There is a new **Select All** command.
- When a table, an image, or an object (in the XHTML sense) is created, new options are offered to choose the position of the new element.
- The **Links** view displays all elements having an `id` attribute, which makes it easier to create links to them (Web feature set only).
- Selection within tables allows any rectangular set of cells to be selected.
- A new **Merge cells** command was added to the **Tools/Edit table** menu. It takes advantage of the "rectangle" selection in tables.
- `class` attributes are displayed in the status bar, next to the element name.

- The status bar displays line and character position in source files.
- The platform language is used by default as the UI language.
- Two new toolbar buttons are available for creating and opening a document.
- A new version of the **Attributes** tool is available (Amaya Advanced only).
- Amaya is now able to remember user ids and passwords for accessing web sites, to make publication on remote web sites simpler.
- The table of contents can be automatically updated at any time (**Tool / Update table of contents**).
- The page number is not set when only one page is printed.

2.4. Interoperability with other tools

Scenarios of use have highlighted the need for exchanging documents between Amaya and other tools and services. Thanks to the choice of standard formats and languages throughout Palette, there is no difficulty for interpreting in a tool data produced by another tool. As a consequence, most interoperability issues are dedicated to interconnection issues: how could Amaya access other tools for exchanging data and trigger processing.

2.4.1. Email messages

A new feature was added to Amaya 10, the ability to send documents by email, from within the editing environment. A command **Send by email** was added to the **File** menu. It allows an author to send in a few clicks the document s/he is editing. Several complex operations are performed behind the scene to make sure the receiver(s) is (are) able to read the document. In particular, all resources used by the document itself (style sheets, images, scripts, objects) are fetched and packaged with the message, wherever they come from, and links between resources are updated according to the packaging.

The next step will be to use email messages for notifying interested people whenever a document is updated on a web site. Instead of sending the actual document and its associated resources, only a short message will be sent, containing the URI of the updated document. This will be done automatically at publishing time. The recipient of this message will just have to click the URI to access the updated document remotely. This notification mechanism is not part of Amaya 10, however.

2.4.2. Connection with SweetWiki

An interoperability experiment was made with Amaya 10 and SweetWiki. The goal is to use both tools to edit on-line information resources in different contexts, choosing the most appropriate tool for each task. During a creative phase, when several users propose different ideas, SweetWiki is the tool of choice. At the end of this phase, all these contributions make a rough document that is maintained by SweetWiki. When a final, well polished document must be generated from there, Amaya may enter into play. It is then very helpful for reorganizing some parts, rewriting some others, and adding some nice and consistent style. This may be done by a single person working with Amaya on the information

produced by the group. When the document is finished, Amaya can post it back into SweetWiki in connection with the contributions it comes from.

Thanks to the latest developments made in both SweetWiki and Amaya, most of this scenario is already possible. The document is downloaded in Amaya in exactly the same way it can be downloaded in a browser. Using a special URI offered by SweetWiki, (ending with `?templateoff=true`), only the core of the document is loaded. It is then edited locally. To upload it back, the import feature of SweetWiki is used from Amaya, thus closing the round-trip.

In this experiment, the user has to explicitly open the SweetWiki web page for import (<http://argentera.inria.fr/swikipalette//data/Tools/OfficeConverter.jsp>) and to provide the requested information. This could be made simpler for the user, by adding a new command in Amaya that would publish the current document in SweetWiki, based on the existing HTTP POST mechanism. This is not part of Amaya 10 however.

3. LimSee3

In this section we present version 1.0 of LimSee3 which is released as a part of this deliverable. We discuss the main evolutions since version 0.5 that was presented in D.INF.03 [4] and we demonstrate on some examples how these evolutions have been obtained in a participatory design way. As a more technical contribution, we present the evolution of the document model with respect to exporting documents into different presentation formats.

3.1. Evolutions of the LimSee3 tool

3.1.1. General considerations

Since D.INF.03 [4] (April 2007), the development of LimSee3 was lead in two complementary directions: on the one hand, LimSee3 is a *generic tool* for editing multimedia documents and as such it provides several general authoring mechanisms; on the other hand LimSee3 proposes some *domain-* or *application-specific* enhancements that are designed to provide more fluent authoring in some cases, but these enhancements are naturally less generic. Figure 4 illustrates this idea of LimSee3 seen as a generic authoring platform on which specific applications can be built.

To develop LimSee3, we have collaborated with several partners from two different categories:

- As a joint work with WP1, specialists of ergonomics provided us with useful feedback on the usability of our application, which allowed the graphical user interface to be improved. We present an example of such an achievement in section 3.2.1.

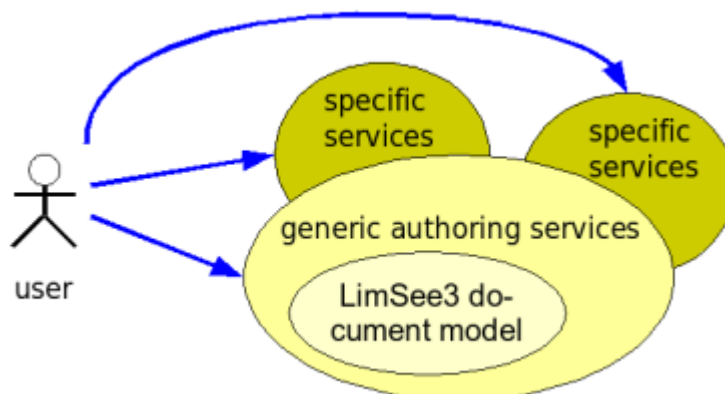


Figure 4: *LimSee3 generic/specific architecture*

- In the general settings of WP5 we have collaborated with end-users on the development of specific authoring services to instrument specific authoring scenarios. In this domain, we have worked with a partner community of practice - ePrep - on instrumentation of a pedagogical scenario (see Appendix 4, section 4 of D.PAR.03 [6]), as illustrated in section 3.2.2.

In all cases, the collaboration was lead in the spirit of the participatory design method, which implied several development loops: remarks from a partner made the application evolve, which brought in more feedback etc. Both kinds of collaboration are ongoing.

It is noticeable that especially the design-in-use experience carried out with a CoP member allowed us to go further than expected. This experience was mainly aimed at the development of two specific authoring services, yet the outcome is more general than that. In fact, having feedback from a real user with real needs allowed us to improve both specific and generic editing mechanisms, and even brought an evolution to the underlying model.

3.1.2. Main achievements

As the version number 1.0 suggests, LimSee3 has reached an important milestone. The underlying model is now managed in a mature way, the user interface was improved, a pedagogical scenario was implemented end-to-end and tested in real use. The generic part of LimSee3 was improved namely in the following domains:

- *Elementary document manipulations* are easier: text editing was improved, there is now an unlimited undo/redo feature, the underlying model can be finely tuned thanks to dedicated user interface widgets.
- Documents can be *exported* into other formats, so they be viewable in various external players. We provide exporting to SMIL [11], which is a standard format

for multimedia documents, and we propose also transformations to XHTML [12], though this feature is currently limited to producing non-interactive presentations.

- More *media formats* are supported: besides images and text, sound and video tracks can be rendered in LimSee3. All usual image and sound formats are available; video rendering is limited to a few formats, because of the lack of a clear web standard and because of numerous intellectual property rights in this domain.
- The *graphical user interface* was redesigned through our collaboration with WP1 in the field of usability. The interface is now clearer, simpler and more familiar to users. The whole application is easier to use, as several disturbing manipulations were identified and replaced by seamless gestures.
- An *automated installation procedure* was provided for LimSee3, coupled with an updating mechanism that minimizes the volume of downloaded data. Thus, users can obtain and maintain their copy of the application in a much easier way than before.

Moreover, two specific tools were developed during our collaboration with the ePrep CoP as a response to their needs:

- The *slideshow* creator allows users to build slideshows in a simple and efficient way. It is based on an adapted document template which guides the user through the authoring process and provides some automation in the treatment. This specific tool was designed to respond to the need for easy preparation of course material.
- The *multimedia course* builder is intended to be used after a course, to create a fully synchronized multimedia presentation out of the slideshow and the video and/or audio tracks recorded during the lesson, with the possibility to provide additional annotations into the post-produced document. This tool is our answer to the need for producing on-line viewable course presentations.

Since the initial release in April 2007, five intermediary quick releases were made during the development, so that interested users could follow the enhancements. The current 1.0 release is thus the seventh one in less than a year.

3.1.3. Current limitations and future developments

LimSee3 has evolved since its version 0.5, but the current version 1.0 is not complete with respect to our (and users') expectations. Several limitations are still present:

- The application is currently provided in English only and the user manual is outdated. These two limitations stem from the fast evolutions of LimSee3: translation in other languages and maintenance of an extensive documentation are simply left behind. The release of version 1.0 is a good opportunity to address these issues, since the new version has chances to be more stable than the preceding ones.
- Several usual media formats - especially video formats - are currently not supported in LimSee3. It means that contents in those formats cannot be rendered

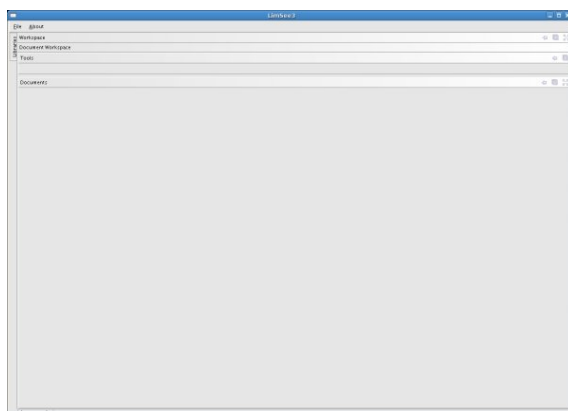


Figure 5: Start screen in version 0.5

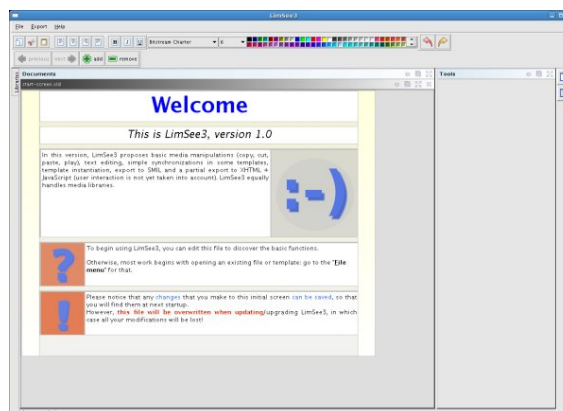


Figure 6: Start screen in version 1.0

during the authoring process, but they are correctly exported to presentation formats. This limitation is partly due to our open source strategy - since many formats are proprietary (and the corresponding codecs as well), we cannot use/distribute them directly. Other difficulties are more platform-dependent and will be addressed in later versions.

- LimSee3 proposes tools for document authoring, but template authoring is still in a very rough state. Only expert users can edit templates.

Besides addressing of the aforementioned issues, our strategy of development will include evolutions of the specific tools (slideshow and course builder, and possibly new tools) towards even more automation and better coping with user profile (beginner / proficient / expert) and corresponding adaptations of the application and its interface. Interoperability with other Palette tools and services is considered mostly in terms of data sharing, as LimSee3 is a document creation software.

3.2. Examples of developments

In this section we present the different aspects of LimSee3 through two examples.

3.2.1. User interface improvements

As we have already said, our collaboration with WP1 allowed us to create a better user interface. Before going into more details, the starting screens of version 0.5 and version 1.0 are shown on Figure 5 and Figure 6 respectively.

Besides being more friendly, the new user interface has the advantage of being better structured, with fewer but better delimited areas. The main tool bar contains familiar buttons, such as those that can be found in a text editor; specific "multimedia" buttons are scarce and located so that they catch the eye after the familiar ones do.

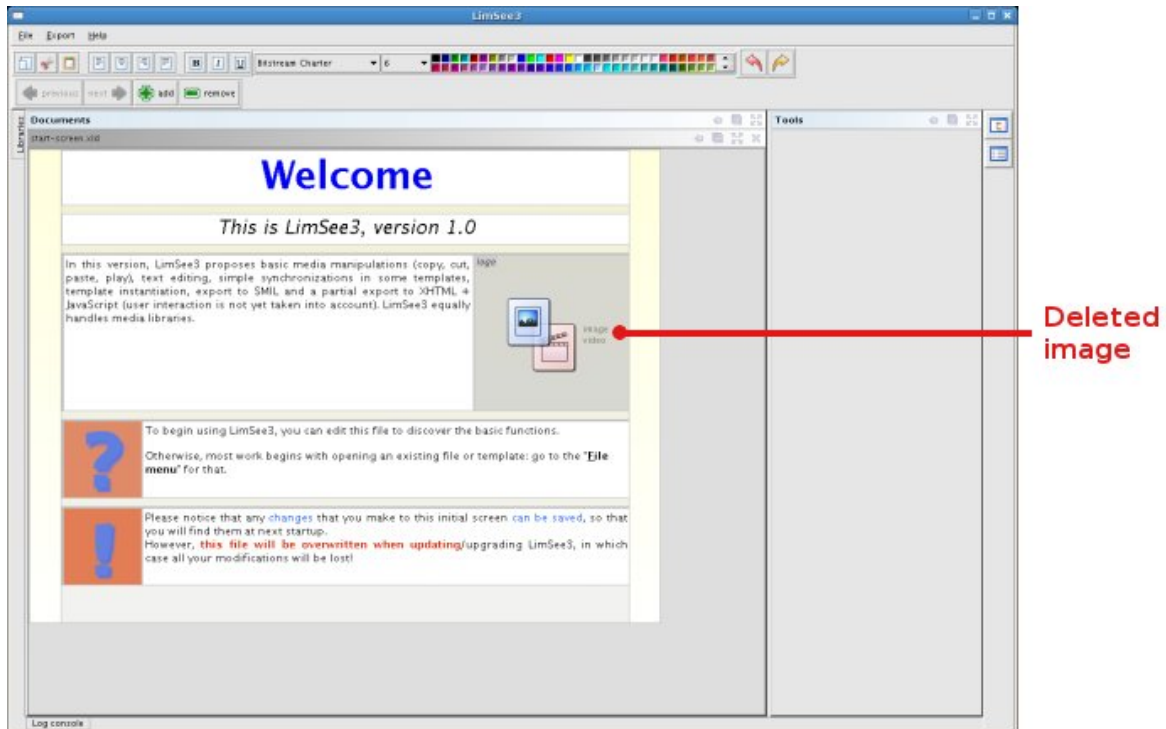


Figure 7: Start screen with deleted image

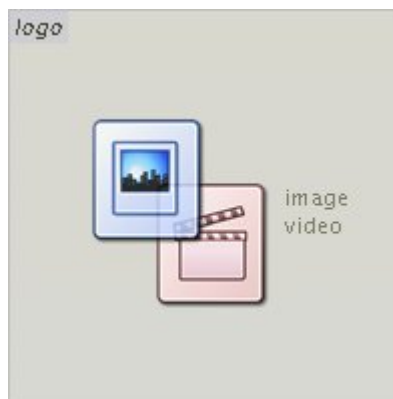


Figure 8: Detail of the deleted image area

The welcome screen is a LimSee3 document itself, so that it is editable by the user. Thus, the user has the possibility to exercise on a scratch document and to customize his/her starting screen. The document is derived from the slideshow template and behaves as such. For instance, it is possible to select an image on the screen and to delete it (Figure 7).

The image was replaced by a pair of icons (Figure 8) that describe the kinds of medias expected in that area. This behavior has existed since version 0.5, but it was pushed further in subsequent versions for enhanced ergonomics: a color was assigned to each kind of media asset and has been used across the whole application. For instance, users will find the same



Figure 9: Detail of a media library

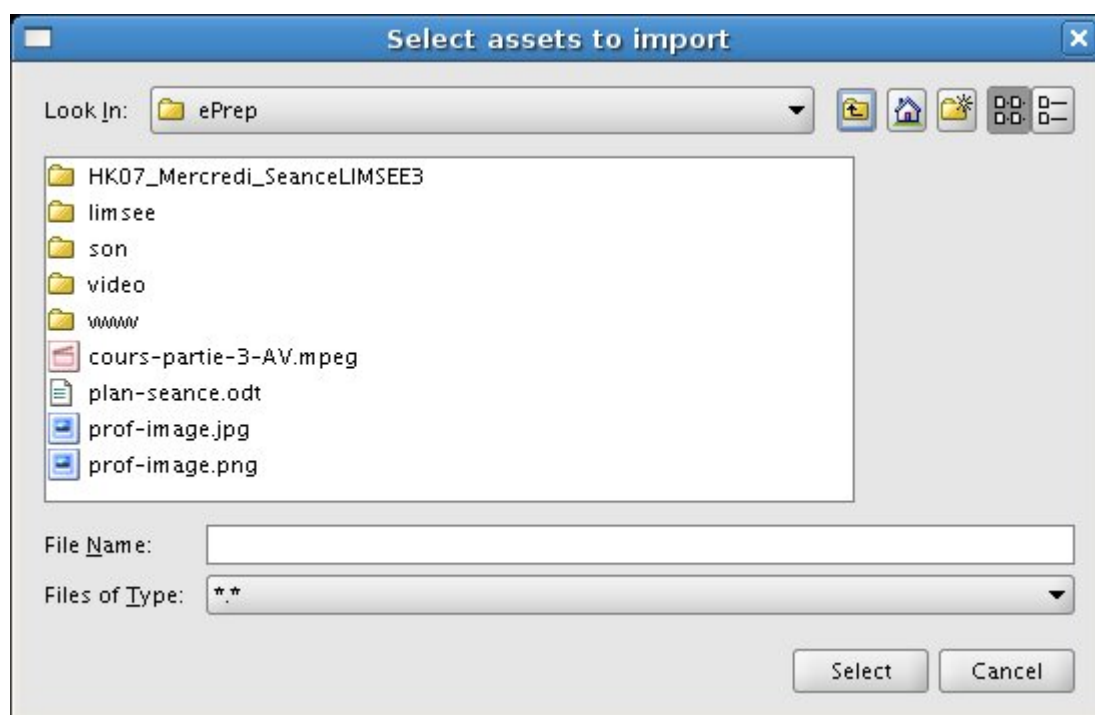


Figure 10: Detail of a file chooser

icons in the context of media libraries (Figure 9) or when browsing their file system for new assets (Figure 10).

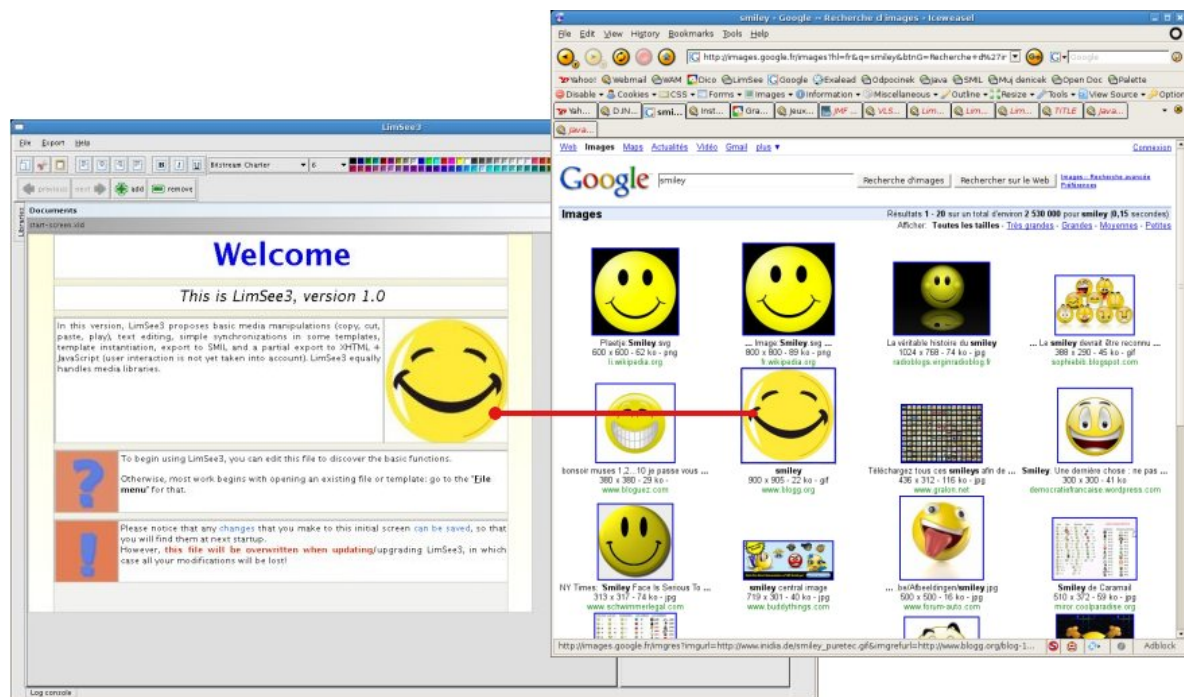


Figure 11: Start screen with new content

Media icons in a template area provide a first guideline given by the application to the user: in the example, it says that "only images or videos can go there" (Figure 8). Another indication comes from the name of the area which is visible in the top-left corner: "logo" could make think of a simple visual element that merely goes with the text besides.

As described in D.INF.03 [4], such a media area can be populated by content copied or dragged from a media library, but the current version of LimSee3 allows also authors to import media objects directly from another application, for instance from a web browser (Figure 11).

The document template provides some automatic treatments. For instance, one can see on the previous example that the image put in the "logo" area is stretched to fill the whole area. Such a behavior is provided as a default - the author of the template esteemed this behavior satisfactory in most cases - but it is possible to modify it thanks to a fine tuning tool (Figures 12, 13, 14).

In this example, we have presented several user interface enhancements: the ability to import/export media objects from/to other applications, more informative media areas, redesigned general layout, revisited tool bars and fine tuning of document behavior. Other, more technical results of collaboration with WP1 on ergonomics can be consulted in section 2.3.3 of D.PAR.04 [7].

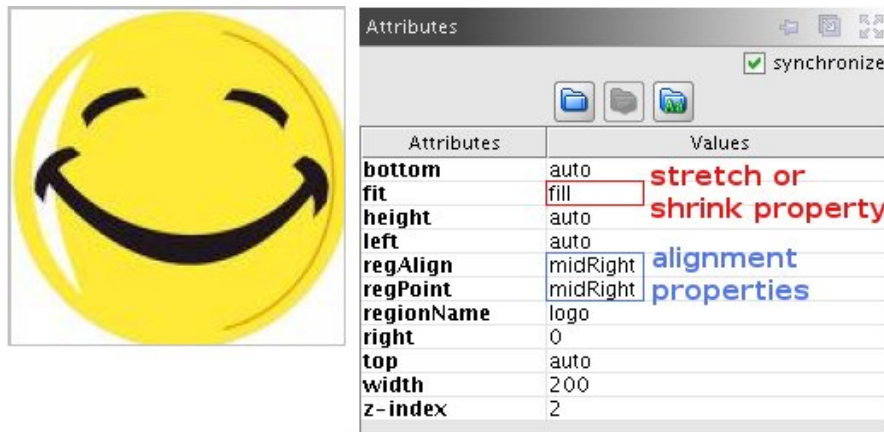


Figure 12: An image with its default layout and the corresponding properties

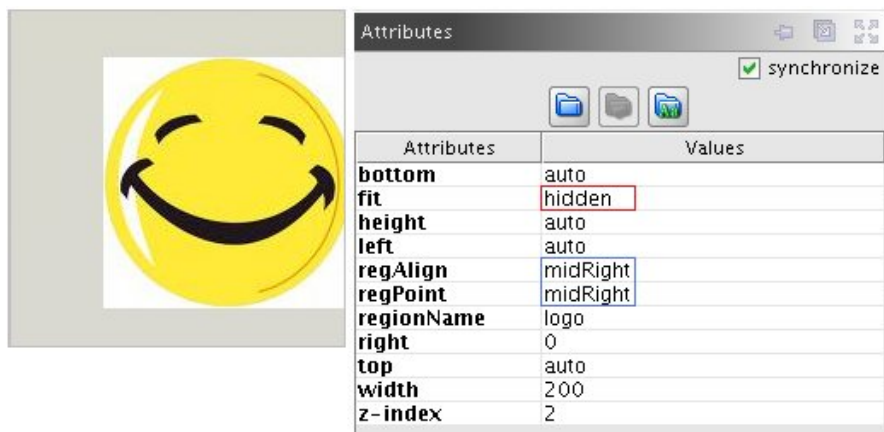


Figure 13: An image and its properties with a "proper size, right alignment" layout

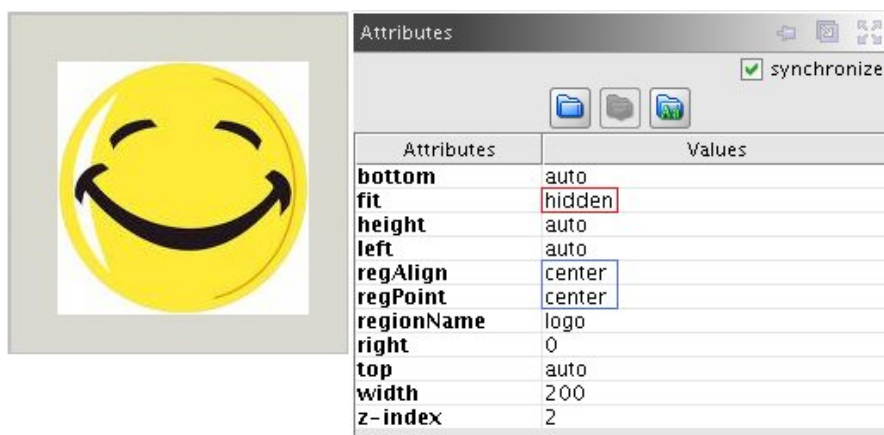


Figure 14: An image and its properties with a "proper size, center alignment" layout

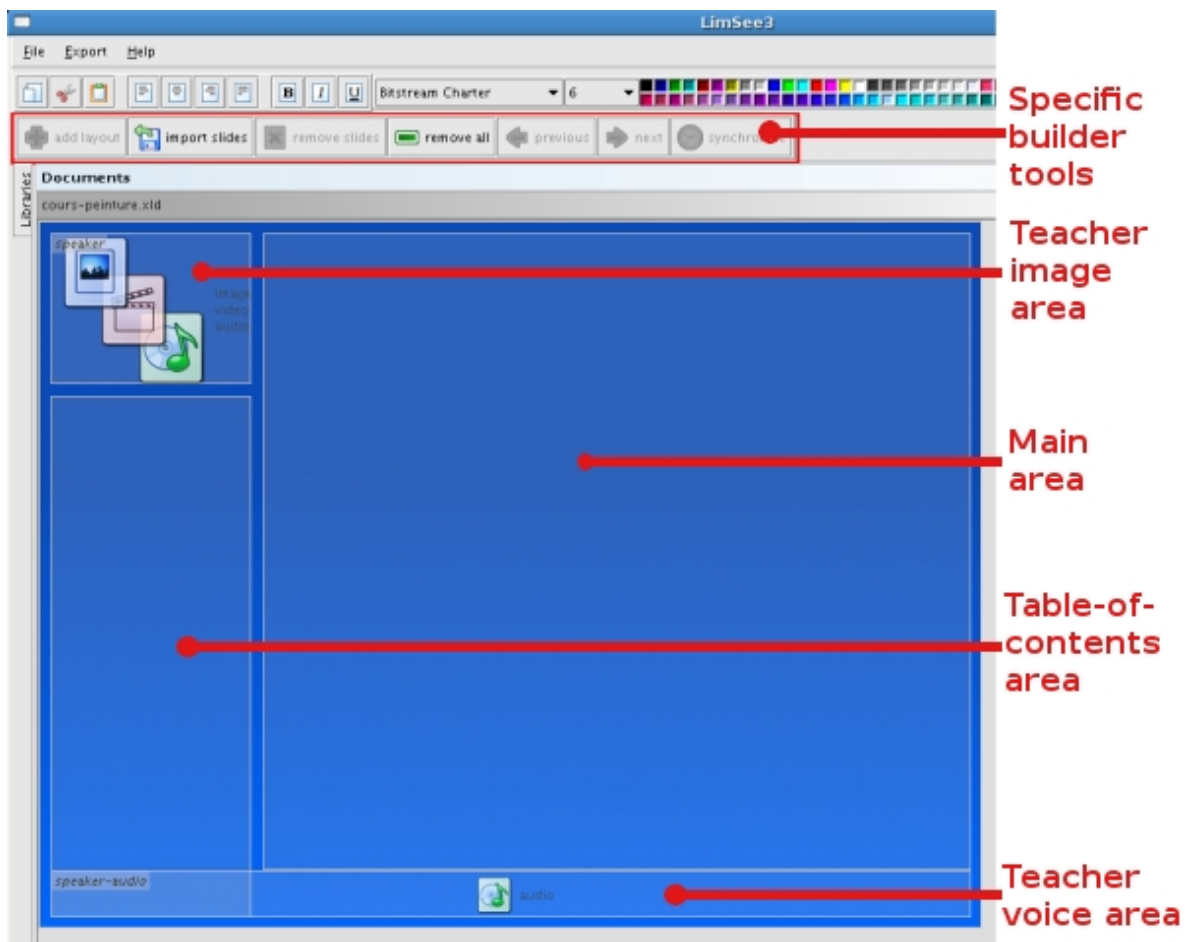


Figure 15: Detail of LimSee3 - empty course document with its dedicated tools

3.2.2. Specific application - Multimedia course builder

The multimedia course builder is a specific application built on top of the LimSee3 generic platform. It has been developed in collaboration with the ePrep CoP as a tool for creating a fully synchronized, interactive course presentation including slides, teacher's voice (and possibly his/her image), annotations and a table of contents.

The development followed the general scenario described in Appendix 4, section 4 of D.PAR.03 (ePrep and LimSee3). A detailed description of the several-month-long experience of participatory design in use can be found on the Palette Wiki [9] and the user's point of view is available in section 2.3.4 of D.PAR.04. We present here several steps of the course building process, to illustrate how it enhances the generic layer of LimSee3 and in which way it takes into account the needs and the requirements of the CoP.

The course builder is based on a specific document template. An empty document created from this template contains several media areas and is accompanied by some specific tools in the tool bar (Figure 15).

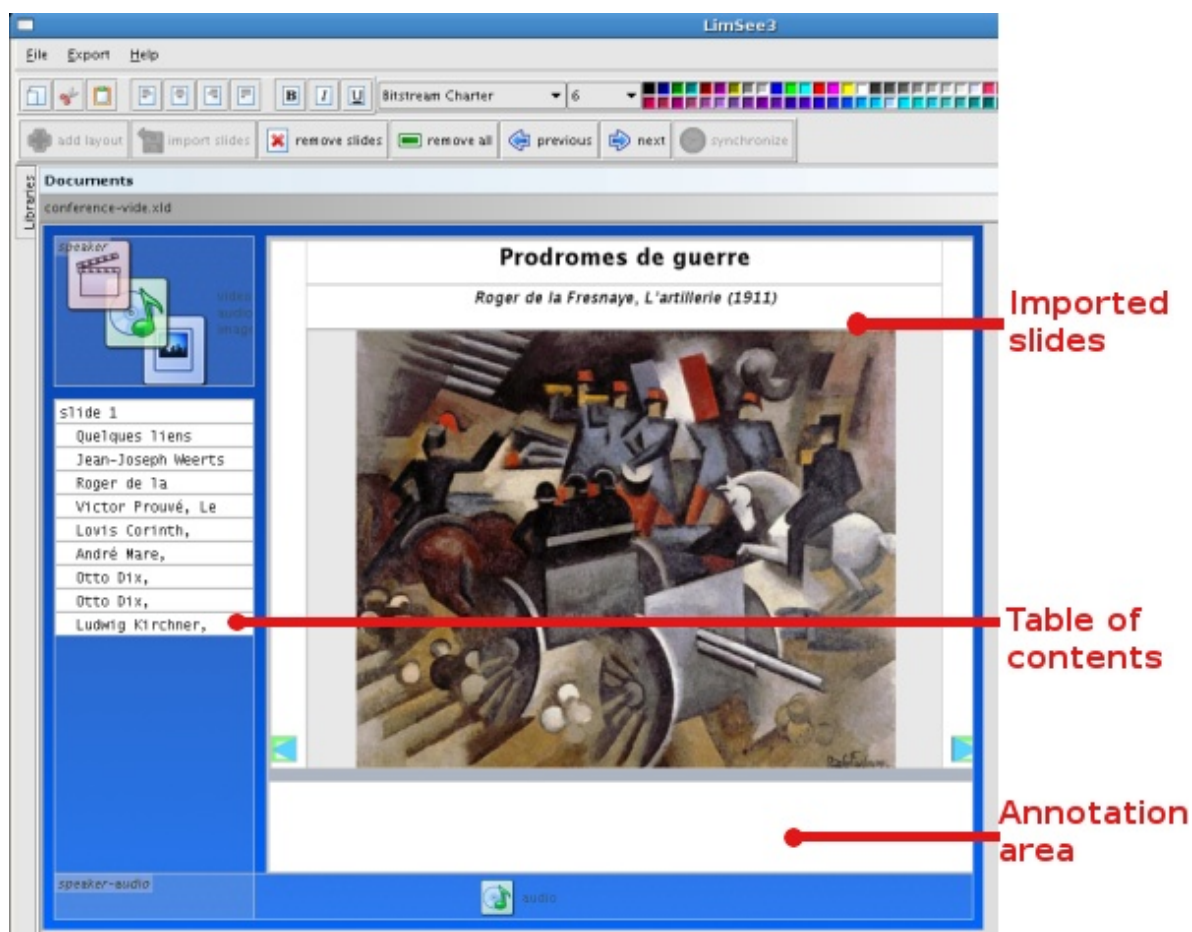


Figure 16: Course under construction - slides imported and table of contents automatically built

To start the authoring process, the user can import the slides that (s)he has used during the lesson. The procedure is automated - it is enough to provide LimSee3 with the name of the slideshow and it will be imported into the document under construction (Figure 16).

The table of contents is created during the import procedure, one item per slide. The items are initialized with the title of the corresponding slide (when available), but they remain editable if needed. Under each slide an empty annotation area is created: it can be freely edited with any contents that the teacher might want to attach to that particular slide.

The next step is to instantiate the other medias: in our example, we drop in a sound track and an image of the teacher (Figure 17).

Now, all that remains is to provide synchronization between the slideshow and the continuous media(s). In other words, it is necessary to indicate for each slide when it should be shown (and when hidden) with respect to the flow of speech of the teacher.

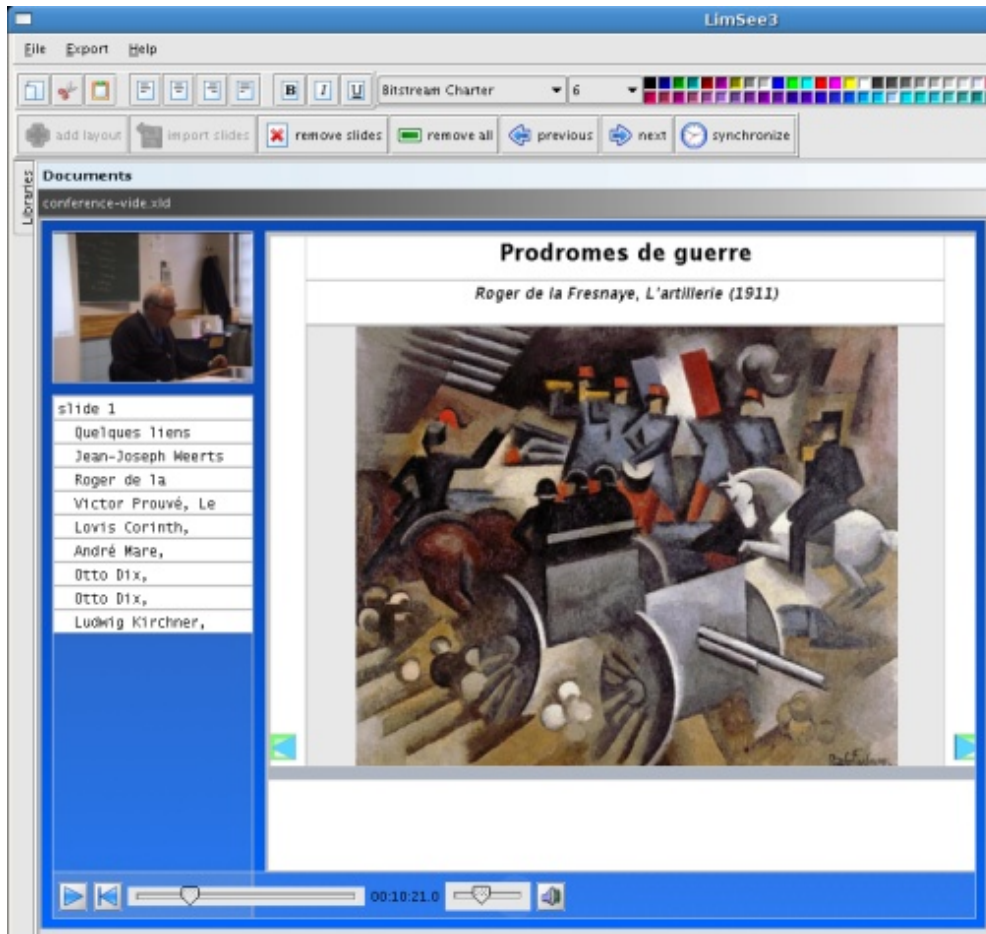


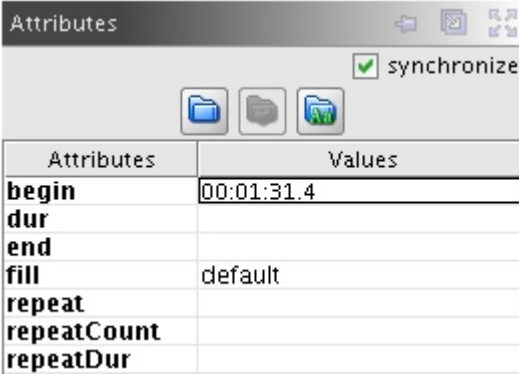
Figure 17: Course under construction - medias instantiated



Figure 18: Detail of the synchronization tool

To achieve this, an interactive synchronization tool can be used (Figure 18). This tool basically plays the continuous medias and the user is required to indicate (by a click on the synchronization button) when the next slide should appear. The clicks are saved and the synchronization is written into the document.

At the end of this step, all timing information has been gathered. When rendered in a player, the document is fully synchronized: a click on the table of contents makes the corresponding slide appear and the speech jumps to the corresponding section; conversely, fast forwarding the speech makes fast forward the slides and their annotations.



Attributes	Values
begin	00:01:31.4
dur	
end	
fill	default
repeat	
repeatCount	
repeatDur	

Figure 19: Manual tuning of the timing

If needed the timing information can be tuned up to fractions of seconds (Figure 19).

The last step consists in exporting the course document into some presentation format (e.g. SMIL). This is what we have done with the history lesson given by an ePrep CoP member - the result is now part of the teaching materials available on the "Plate-forme Francophone" by ePrep.

3.2.3. Conclusion on the outcomes of the two collaboration experiences

Both collaborations we have presented (WP1, WP5) have lead to substantial evolutions of LimSee3 in terms of software development. Moreover, on several occasions, the participatory design method brought changes even to the "philosophy" of the application:

- Usability analysis made us abandon the idea of a constantly adapted user interface, where tools would appear and disappear as different parts of a document are selected and edited. Deemed too confusing, adaptation was limited, so that several common tools are now permanently present, be they are active or not depending on the current context (e.g. text editing buttons).
- The need for interactivity in the documents produced with ePrep made us aware of the limitations of the underlying document model, which led to the definition of more complex ways of expressing internal links.

The two examples above show how collaboration could involve modifications outside of their expected range. Conversely, the changes we made to LimSee3 lead to interesting reactions among users, possibly modifying their current habits (see for instance section 2.3.4 of D.PAR.04).

Generally speaking, user comments and suggestions - e.g. those collected after a LimSee3 tutorial session - provide us with useful insights into the real needs. Thus, LimSee3 is bound to evolve towards even more automation in course material building, exploiting and publishing, in order to simplify those time-consuming tasks.

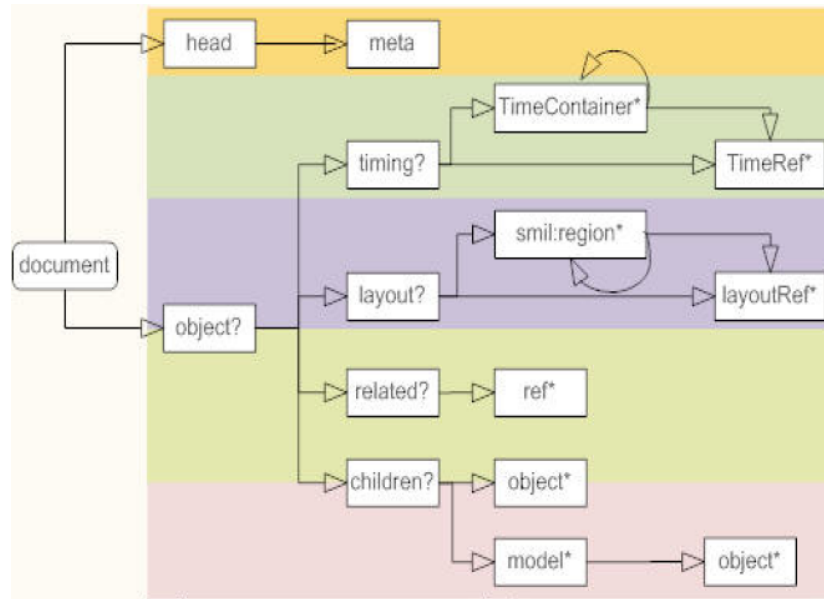


Figure 20: Main components of the LimSee3 document model

As another example of collaboration outcome, document D.PAR.04 [7] describes LimSee3 in a very user-oriented way. We can benefit from this deliverable to write an easier to read LimSee3 manual, since it provides us with precious indication on how and in which order the various application features have to be presented.

3.3. Evolutions of the document model

Section 3 of deliverable D.INF.01 [2] has presented the first version of the LimSee3 document model which is implemented in the initial version of the LimSee3 authoring tool (as described in D.INF.03 [4]) and in version 1.0 (previous section of this deliverable). The core of the LimSee3 document model has not significantly changed since D.INF.01, and it is being validated through the current implementation. The evolution has merely concerned the part of the model that copes with document exportation.

This section introduces first the exportation requirements, then it presents the model and the architecture of the exportation service in LimSee3.

3.3.1. Exportation needs for LimSee3

As stated earlier [2], the LimSee3 model is oriented towards document description needs (object structuring coupled with template definitions) in order to provide simple multimedia authoring features. That is the reason why the main entity in the model is the "object", which can not only be spatially and temporally placed but can also be defined as a hierarchy of components (through its "children" part), as shown in Figure 20.

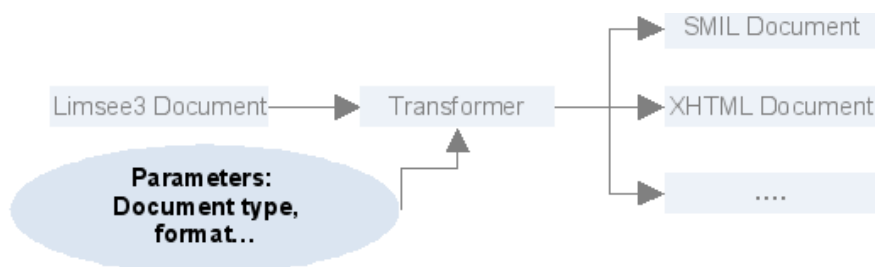


Figure 21: LimSee3 Export Process

The resulting authoring services that are provided by LimSee3 are therefore completed by a set of publishing services, which are also called exportation services. The goal is to cope with the different publication formats in which users may want their multimedia documents to be accessed.

This approach brings up two main benefits:

- better authoring services, as stated in D.INF.01 [2],
- independence of the authoring tool from the publication formats.

The latter item is of primary importance, because multimedia document formats are continuously evolving, even the most widely used formats defined by standard organizations such as W3C (SMIL, SVG, HTML) or ISO/IEC (MPEG4). This independence insures therefore a more stable perspective for the work being done in LimSee3.

Another interesting advantage for users is the separation between the authoring model and the publication format, which allows authors to choose the output process adapted to each CoP where multimedia documents will be used. The choice can also be driven by the kind of multimedia documents used in the CoP. For instance, lightly-synchronized documents can be exported to XHTML+javascript in order to provide a wider access (only a plain web browser is required), while a SMIL-based solution is required when more complex scheduling is necessary.

3.3.2. A model for exporting LimSee3 documents

The exportation process sketched below covers the following objectives:

- multiple targets, to cover user needs regarding publication and access formats,
- optimization of the resulting document,
- efficiency in the development of export modules,
- extensibility, to cope with future formats.

The exportation process (Figure 21) is illustrated by the case of publishing a LimSee3 document in the SMIL format. As can be seen by comparing the SMIL document structure

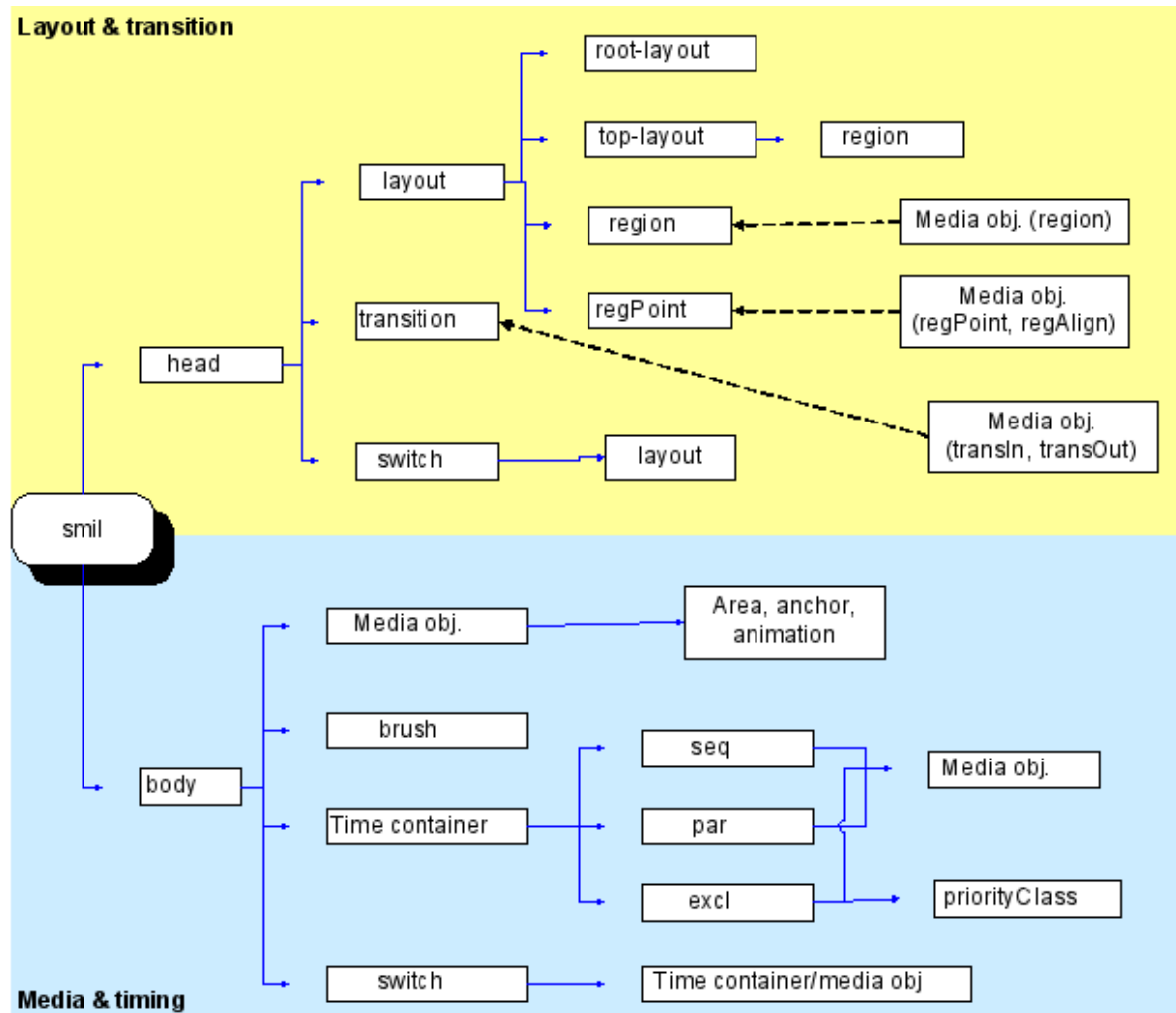


Figure 22: SMIL document structure

(Figure 22) with the LimSee3 document structure (Figure 20), the transformation requires to completely compute and reorganize both the time and spatial components (Figure 23).

Several solutions have been studied (see [8] for a complete analysis), from a pure XSLT-based one to a pure Java-oriented one. The former seems to be straightforward as both the LimSee3 syntax and the targeted publication formats are XML languages. However it has proven to be complex and inefficient because the XSLT code is not relevant for the required treatments, such as time and spatial position computations. Moreover it does not facilitate reusing and sharing of existing export services. On the other hand, pure Java solutions benefit from the power of a general-purpose programming language but imply that all new export developments have to be done by a "LimSee3 developer". Finally, the proposed architecture for exportation takes advantage of both approaches: an intermediate format has been defined to record all the structures and formatting parameters that can be computed by the LimSee3 core modules; each document format can then be produced by an XSLT transformation (or a Java module) as shown in Figure 24.

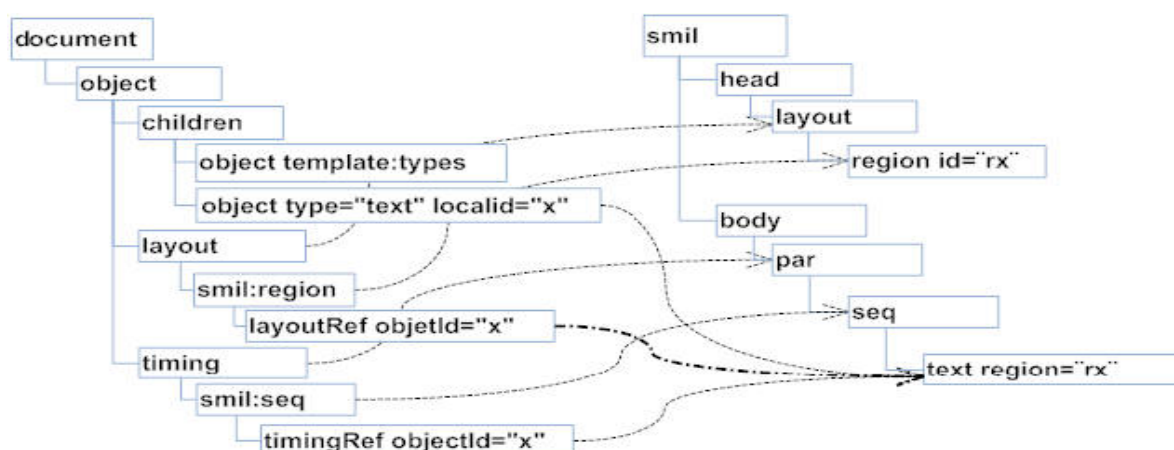


Figure 23: Transformation of LimSee3 document structure to SMIL document structure

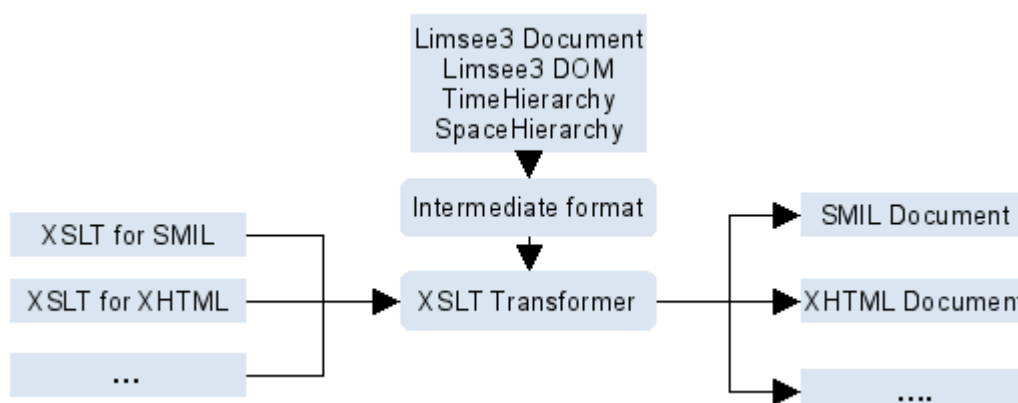


Figure 24: LimSee3 export architecture using an intermediate format

The precise definition of this format is given in Appendix A, which provides a commented presentation of the corresponding DTD.

4. References

- [1] D.IMP.01 – First Guidelines for Development, 30 October 2006
- [2] D.INF.01 – Report on the design of extension mechanisms for creating templates, using templates for editing and customizing the user interface, and of extensions to be integrated in the information reuse tool, 15 May 2006
- [3] D.INF.02 – First version of Amaya offering a template mechanism, 19 February 2007
- [4] D.INF.03 – First version of LimSee3 offering a template mechanism and a customized user interface, 16 April 2007

- [5] D.INF.04 – First version of the information reuse service, 11 June 2007
- [6] D.PAR.03 – Description of 6 scenarios and of the results of 6 validated trials, 14 September 2007
- [7] D.PAR.04 – «User centered» description of the PALETTE tools and services, and first analysis of their usability, February 2007
- [8] Duc Bao Le, Modèle d'édition de document multimédia (in French), September 2007, <http://hal.inria.fr/inria-00232724/>
- [9] Palette Wiki, ePrep & LimSee3 - participative design in action, <http://argentera.inria.fr/swikipalette/data/Mediators/EPrepLimSee3.jsp>
- [10] V. Quint, I. Vatton, Techniques for Authoring Complex XML Documents, Proceedings of the 2004 ACM symposium on Document Engineering, DocEng 2004, ACM Press, Milwaukee, USA, October 2004, pp. 115-123.
- [11] SMIL, The Synchronized Multimedia Integration Language, W3C, <http://www.w3.org/TR/SMIL2>
- [12] XHTML 1.0, The Extensible HyperText Markup Language (Second Edition), W3C, <http://www.w3.org/TR/xhtml1/>

Appendix A - commented DTD of the intermediate format

In this section we provide the full Document Type Definition (DTD) of the LimSee3 intermediate format. For readability reasons, the entity definition section, which is normally placed at the beginning of the DTD, is placed at the end of this appendix. The actual DTD is located at <http://ns.inria.fr/limsee3/intermediate/intermediate.dtd>.

It is a feature of the intermediate format that some information is duplicated. The reason is to be able to retrieve it as quickly as possible, without multiple indirections.

A.1 General intermediate document structure

An intermediate format document is a valid XML document conforming to the following DTD: <http://ns.inria.fr/limsee3/intermediate/intermediate.dtd>. Elements describing the intermediate format are declared in the <http://ns.inria.fr/limsee3/intermediate> namespace. The root element is the document element:

```
<!ELEMENT document (head?, layout?, timing?, references?, media?) >  
<!ATTLIST document %namespace; >
```

An intermediate format document contains five main sections:

head

contains document meta-data

layout

describes the document spatial layout in terms of containers

timing

describes the timing structure as a tree of time containers and contained objects

references

links objects to the corresponding spatial and time containers

media

lists the basic media contained in the document

In the sequel, we present each of these sections in more detail.

A.2 The header section

```
<!ELEMENT head (meta)* >
<!ATTLIST head %common.attr; >

<!ELEMENT meta EMPTY >
<!ATTLIST meta
  name      CDATA #REQUIRED
  content   CDATA #REQUIRED
  %common.attr;
>
```

The intermediate document header contains a list of `meta` elements, each of which expresses a piece of meta-information formed by a property name and the associated content.

A.3 The layout section

The layout section describes the tree-like spatial structure of a LimSee3 document as a hierarchy of containers (called "regions"). Each region may contain other regions and registration points. A registration point defines some particular point inside its container.

Regions have two kinds of attributes: "normal" ones and "resolved" ones. The former contain the same values as they do in a LimSee3 document, the latter are resolved with respect to the parent region's values.

```
<!ELEMENT layout (region) >
<!ATTLIST layout type      CDATA #FIXED 'text/smil-basic-layout'>

<!ELEMENT region (region|regPoint)* >
```

```

<!ATTLIST region
  id          ID          #REQUIRED
  bottom      CDATA      'auto'
  left        CDATA      'auto'
  right       CDATA      'auto'
  top         CDATA      'auto'
  height      CDATA      'auto'
  width       CDATA      'auto'
  resolvedBottom CDATA    #REQUIRED
  resolvedLeft CDATA    #REQUIRED
  resolvedRight CDATA    #REQUIRED
  resolvedTop  CDATA    #REQUIRED
  resolvedHeight CDATA    #REQUIRED
  resolvedWidth CDATA    #REQUIRED
  showBackground (always|whenActive) 'always'
  backgroundColor CDATA    #IMPLIED
  resolvedBackgroundColor CDATA    #REQUIRED
  z-index       CDATA    #IMPLIED
  resolved-z-index CDATA    #IMPLIED
  fit (hidden|fill|meet|scroll|slice) #IMPLIED
  regPoint      CDATA    #IMPLIED
  %regAlign.attr;
>

<!ELEMENT regPoint EMPTY >
<!ATTLIST regPoint
  name      CDATA    #REQUIRED
  top       CDATA    'auto'
  left      CDATA    'auto'
  right     CDATA    'auto'
  bottom    CDATA    'auto'
  %regAlign.attr;
>

```

A.4 The timing section

```
<!ELEMENT timing (timeRoot, timeSequences) >
```

There are two distinct parts in the timing section:

- `timeRoot` and its descendants describe the time-container hierarchy much in the way the layout section does for space containers,
- `timeSequences` provides precise sequencing inside each of the time containers.

The time hierarchy is a tree formed by three different kinds of time containers: `par` (parallel-time container), `seq` (sequential-time container) and `excl` (exclusive-execution container).

```

<!ELEMENT timeRoot (par | seq | excl) >

<!ELEMENT seq (par | seq |excl)*>
<!ATTLIST seq %timing.attrs; >

<!ELEMENT par (par | seq | excl)*>
<!ATTLIST par
    endsync                CDATA            'last'
    %timing.attrs;
>

<!ELEMENT excl ((par | seq | excl)* | priorityClass+) >
<!ATTLIST excl
    endsync                CDATA            'last'
    skip-content            (true|false)    'true'
    %timing.attrs;
>

<!ELEMENT priorityClass EMPTY >
<!-- to be defined more precisely in future versions -->

```

The time sequence section is a list of sequences. There is one sequence per time container in the hierarchy (the corresponding container is uniquely determined by the `containerId` attribute in sequence). Each sequence is a list of references to temporal objects, which are [temporal] children of the associated time container. Temporal objects can be other time containers (par, seq, excl) or media objects. They are uniquely determined by the `ref` attribute. The order of temporal objects in a sequence is important.

```

<!ELEMENT timeSequences (sequence*) >

<!ELEMENT sequence (temporalObject+) >
<!ATTLIST sequence
    containerId    CDATA            #REQUIRED
    containerType (seq | par | excl) #REQUIRED
>

<!ELEMENT temporalObject EMPTY>
<!ATTLIST temporalObject
    type (seq | par | excl | object) #REQUIRED
    ref  CDATA                        #REQUIRED
>

```

A.5 The references

This section is formed by a list of references. Each reference links an object (uniquely determined by its `objectId`) to a space container and a time container.

```
<!ELEMENT references (ref*) >
```

```
<!ELEMENT ref EMPTY >
```

```
<!ATTLIST ref
```

```
  objectId  CDATA  #REQUIRED
```

```
  regionId  CDATA  #REQUIRED
```

```
  timingId  CDATA  #REQUIRED
```

```
>
```

A.6 The media section

The last intermediate document section is formed by a list of media objects.

```
<!ELEMENT media (text | img | video | audio | animation)* >
```

There are five types of medias, each of which can define some areas. Moreover, text media objects may provide some actual text content.

```
<!ELEMENT text (area*, content?) >
```

```
<!ATTLIST text %media.attrs; >
```

```
<!ELEMENT content (#PCDATA) >
```

```
<!ELEMENT img (area*) >
```

```
<!ATTLIST img %media.attrs; >
```

```
<!ELEMENT audio (area*) >
```

```
<!ATTLIST audio %media.attrs; >
```

```
<!ELEMENT video (area*) >
```

```
<!ATTLIST video %media.attrs; >
```

```
<!ELEMENT animation (area*) >
```

```
<!ATTLIST animation %media.attrs; >
```

Finally, an area just defines a portion of a media object.

```
<!ELEMENT area EMPTY >
```

```
<!ATTLIST area
```

href	CDATA	#REQUIRED
begin	CDATA	#IMPLIED
end	CDATA	#IMPLIED
dur	CDATA	#IMPLIED
repeatCount	CDATA	#IMPLIED
repeatDur	CDATA	#IMPLIED
repeat	CDATA	#IMPLIED
min	CDATA	'0'
max	CDATA	'indefinite'

```

%system.attrs;
customTest          CDATA          #IMPLIED
>

```

A.7 Common definitions

The following entities are normally defined at the beginning of the DTD:

```

<!ENTITY % namespace 'xmlns CDATA #FIXED
    "http://ns.inria.fr/limsee3/intermediate"'>
<!ENTITY % common.attr "xml:lang CDATA #IMPLIED " >
<!ENTITY % system.attrs
    "systemBitrate          CDATA          #IMPLIED
    systemCaptions         (on|off)       #IMPLIED
    systemLanguage          CDATA          #IMPLIED
    systemOverdubOrSubtitle (overdub|subtitle) #IMPLIED
    systemRequired          CDATA          #IMPLIED
    systemScreenSize        CDATA          #IMPLIED
    systemScreenDepth       CDATA          #IMPLIED
    systemAudioDesc         (on|off)       #IMPLIED
    systemOperatingSystem   NMTOKEN       #IMPLIED
    systemCPU               NMTOKEN       #IMPLIED
    systemComponent         CDATA          #IMPLIED
    system-bitrate          CDATA          #IMPLIED
    system-captions         (on|off)       #IMPLIED
    system-language        CDATA          #IMPLIED
    system-overdub-or-caption (overdub|caption) #IMPLIED
    system-required         CDATA          #IMPLIED
    system-screen-size      CDATA          #IMPLIED
    system-screen-depth     CDATA          #IMPLIED"
>
<!ENTITY % regAlign.attr "regAlign (topLeft | topMid | topRight | midLeft |
    center | midRight | bottomLeft |
    bottomMid | bottomRight) #IMPLIED " >
<!ENTITY % timing.attrs "
    id          ID          #REQUIRED
    fill (remove|freeze|hold|transition|auto|default) 'default'
    dur         CDATA       #IMPLIED
    repeatCount CDATA       #IMPLIED
    repeatDur   CDATA       #IMPLIED
    begin       CDATA       #IMPLIED
    end         CDATA       #IMPLIED
    repeat      CDATA       #IMPLIED
    min         CDATA       '0'
    max         CDATA       'indefinite'
    restart     (always|whenNotActive|never|default) 'default'
    restartDefault (inherit|always|never|whenNotActive) 'inherit'
    syncBehavior (canSlip|locked|independent|default) 'default'
    syncTolerance CDATA       'default'

```

```

        syncBehaviorDefault (canSlip|locked|independent|inherit)      'inherit'
        syncToleranceDefault      CDATA      'inherit'
        fillDefault      (remove|freeze|hold|transition|auto|inherit) 'inherit'
        %system.attrs;
        customTest      CDATA      #IMPLIED"
>

<!ENTITY % media.type "(image | audio | video | text | animation)" >

<!ENTITY % media.attrs
    "id          ID      #REQUIRED
    preferredHeight CDATA #REQUIRED
    preferredWidth  CDATA #REQUIRED
    preferredTop    CDATA #REQUIRED
    preferredBottom CDATA #REQUIRED
    preferredLeft   CDATA #REQUIRED
    preferredRight  CDATA #REQUIRED
    actualType      %media.type; #REQUIRED
    fit             (hidden|fill|meet|scroll|slice) #REQUIRED
    src             CDATA #REQUIRED"
>

```