MEDIA2MULT – A WIKI-BASED AUTHORING TOOL FOR COLLABORATIVE DEVELOPMENT OF MULTIMEDIAL DOCUMENTS

Author Name *
*Affiliation **
*Address **

Author Name *
*Affiliation **
*Address **

* Only for Final Camera-Ready Submission

ABSTRACT

media2mult is an extension for PmWiki developed at our university. It provides functionality for embedding various media files and script languages in wiki pages. Furthermore media2mult comes with a cross media publishing component that allows to convert arbitrary wiki page sequences to print-oriented formats like PDF. This article gives an overview over the offered extensions, their functionality and implementation concepts.

KEYWORDS

wiki, multimedia, cross-media-publishing, authoring tool, XML

1. INTRODUCTION

At least since the founding of the free web encyclopedia Wikipedia and its increasing popularity *wiki web*, *wiki-wiki* or just *wiki* are widely known terms in context of Web 2.0. However, their exact meaning often remains unclear. Sometimes *wiki* and *Wikipedia* are actually used synonymously. The crucial functionality of every wiki system is the possibility to edit wiki web pages directly inside a browser by entering an easy to learn markup language. Thus, manual uploads of previously edited HTML files are superfluous here. The user doesn't even have to know anything about HTML or external HTML editors.

The browser- and server-based concept makes it possible that several authors can edit and revise common documents without the necessity of exchanging independently written and updated versions. Because most wiki systems offer an integrated version management system, authors can easily merge their changes and revert selected passages to former stages. Thus, accidentally or deliberately applied changes of protected or publicly accessible wiki pages can be taken back in a second.

At our university, a group of researchers raised the question whether it is possible to write scientific texts including mathematical formulas, graphics, audio, video and footnotes with a wiki system only. A further request was an export function that should be able to convert an arbitrary set of wiki pages to a structured PDF document. As a result of this approach, the PmWiki extension media2mult was developed and is now used in about 200 wiki fields installed on our central wiki farm. On the one hand it is utilized by many lecturers to prepare lecture scripts, and on the other hand students are asked to write their term papers with media2mult.

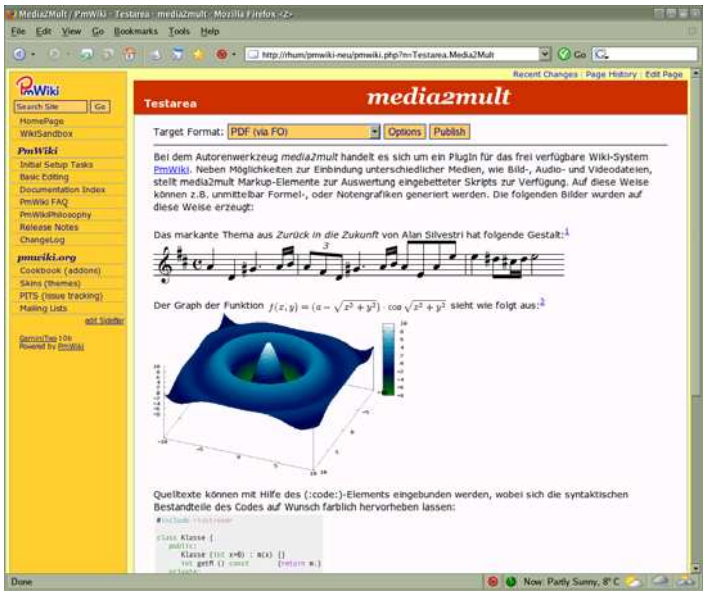

Figure 1. Screen shot of a wiki page containing several components supported by media2mult (LaTeX formulas, footnotes, code-based music notation, gnuplot graphics, colored source code).

2. PMWIKI

Because the term *wiki web* just names a fundamental concept and not a complete standard, many different wiki implementations have been developed over the last few years. The broad range of available systems covers simple, minimalistic approaches programmed with few lines of code, as well as complex and highly extensible wiki engines. The same is true for the richness of wiki input codes, concerning both syntax and available markup repertoire.

The open source implementation *PmWiki* belongs to the more comprehensive wiki variants. It is actively develop by Patrick Michaud using the script language PHP. PmWiki offers a highly configurable architecture that can easily be extended with own functionality or arbitrary new skins. Thanks to this technical base, a considerable PmWiki community has evolved which helps resolving problems and offers free extensions, the so-called *recipes*.

After testing several wiki variants, the Center for Information Management and E-Learning of the University of Osnabrück decided to introduce PmWiki as a centrally installed and supported platform for e-learning content. One reason for this decision was the concept of *page trails*, a technique that enables the user to structure documents by defining ordered and indented lists of links to wiki pages. In contrast to MediaWiki – the wiki engine used by Wikipedia –, where pages are loosely connected by a pool of keywords, PmWiki additionally allows the description of structured tables of contents with chapters, sections and subsections. This was a crucial precondition for the planned application area of media2mult.

From a developer's point of view, PmWiki primarily gains its flexibility through two approaches: Firstly, the system uses freely definable *markup rules* that makes it possible to increase the set of supported wiki commands. The second aspect is the use of so-called *actions* which can lead to various interpretations of the same page content.

2.1 Actions

A minimum requirement of all wiki systems is that they must be able to produce two different views of the same page content: an edit view and a display view. The first one shows the page code in a text area so that it can be edited by the user, and the second view variant shows the rendered result of the interpreted code. PmWiki realizes these different views by applying different user-definable *actions*, which are passed to the

browser in the form of a GET parameter. Each action specifier is assigned to an associated action handler. The latter is automatically called when processing a wiki page and makes sure that the required HTML code is generated. In case of edit and display view the dedicated actions look as follows, whereas the explicit specification of action *browse* is optional:

```
http://www.my-wiki-server.de?n=Group.Page&action=edit
http://www.my-wiki-server.de?n=Group.Page&action=browse
```

In both cases the content of page `Group.Page` is evaluated, appropriately processed and sent to the browser.

Further examples of pre-defined PmWiki actions are *upload* for uploading of local files and *source* for retrieving the unprocessed page code. The cross media publishing component of media2mult defines an additional action which triggers the process of collecting the necessary page contents and initiates the conversion procedure.

2.2 Markup rules

Basically, the PmWiki core is nothing but a web-based text replacement engine. Its main purpose consists in converting the components of the wiki input language to HTML equivalents. The central function called by the action *browse* is called *MarkupToHTML*. At first glance it does exactly what its name implies: converting the user input code to HTML. However, that's just the default behavior. Because the text replacement is realized by iteration over an array of many user-definable replacement rules, i.e. the transformation takes place by a sequential application of regular expressions, arbitrary conversion results are theoretically possible.

The markup array is build up by so-called *markup rules* that are evaluated every time the browse handler is executed. A markup rule consists of a name, a relative position in the markup array, a search pattern, and a replacement expression. For example, the default rule for creating italic texts looks as follows:

```
Markup("''", 'inline', "/''(.*?)''/",'<em>\$1</em>')
```

The first parameter specifies the rule name -- two apostrophes in this case. The second parameter tells PmWiki where to put the rule in the markup array. The location *inline* is a special place where all rules are collected that affect single lines only. It's also possible to put a rule before or after an already defined rule. With this mechanism the order in which the rules are applied later on can be defined.

The third and fourth parameters specify search and replacement pattern, respectively. To sum up, this markup rule makes sure that text patterns of the form `''Text''` are converted to the HTML fragment `<em>Text</em>`.

Consequently, extensions to the PmWiki syntax can be realized by simply adding further markup rules. Furthermore it is possible to completely replace the pre-defined rule sets, e.g. to generate XML or LaTeX code instead of HTML. In these cases the above mentioned function *MarkupToHTML* wouldn't produce HTML, so its name is a bit misleading and conceals the enormous potential.

The first PmWiki-based version publishing component of media2mult was realized exactly this way by adding a rule set for generating DocBook-XML[1] that could in turn be transformed to various target formats by applying the Daniel Veillard's DocBook stylesheets[2]. However, it turned out that the necessary doubling of all rules complicated the migration to more recent PmWiki versions. Moreover, the strict separation of content and layout intended by DocBook turned out to be counter-productive in a wiki context because it was hardly possible to map all wiki markup elements to DokBook. Thus, most users were disappointed by the conversion results. As a consequence, we're now following a different approach described below.

---

[1] http://docbook.sourceforge.net

[2] http://wiki.docbook.org/topic/DocBookXslStylesheets

3. THE EXTENDED MARKUP OF MEDIA2MULT

A crucial prerequisite for acceptance of a newly introduced authoring tool is the initial support of functions usually available in traditionally used applications, such as footnotes, mathematical formulas and arbitrary graphics. All these aspects are normally omitted by a wiki system, especially when it comes to supporting vectorial versions of the document components. The latter are important for high quality PDF generation.

That is why the plug-in media2mult offers the new universal markup element *embed* that can be used to integrate several media types into wiki pages. The set of supported formats includes EPS, FIG, SVG, SWF, VRML, WAV, MP3, MIDI, MOV, MP4, and more. The file to be embedded has just to be uploaded to the server using PmWikis attach function. Afterwards, it can be referenced, such as

```
(:embed file=graphic.eps:)
```

The conversion routines of media2mult make sure that possibly necessary format transformations take place. In case of an EPS graphic the file is converted to PNG for web display and to an embeddable PDF snippet for PDF document creation.

As you can see in the above list of supported file types, media2mult allows the embedding of audio and video files as well. These media are directly supported by most web browsers, so it's easy to reference them in wiki pages. The more difficult part is its conversion to PDF because there is of course no print-oriented equivalent of audio and video files. In this case, the converter tries to automatically extract a preview image (see figure 2). If this attempt wasn't successful, a warning message is generated and the user can explicitly give an alternative image used for PDF creation:

```
(:embed file=audio.wav print-file=bild.gif:)
```

For embedding mathematical formulas, media2mult offers the common LaTeX syntax. As in LaTeX the user can distinguish between in-line and display formulas. Each formula is converted to a PNG image and aligned with the baseline of the surrounding text. Since this conversion process takes some time and can distinctly delay the page display, especially if the page contains many equations, an MD5-based technique is deployed to avoid multiple conversions of the same formula. This procedure assures that a formula image is only generated if it is displayed for the first time or if the corresponding LaTeX code has been changed.

The technique of embedding code snippets of external declarative languages like LaTeX has been generalized and can now be applied to arbitrary script languages and their corresponding processors. To demonstrate this interface it has been exemplarily implemented for gnuplot, Metapost, POV-Ray, and the music notation generator *p2i*. The code for these processors can be either directly typed into the wiki page or uploaded in a separate file. The latter must be referenced by the mentioned *embed* statement, whereby the file extension specifies the actual file format.

Besides the media file support media2mult offers the markup element *code* for displaying source code excerpts that can optionally be numbered as well as colored syntactically. The code language is thereby selected by a facultative attribute:

```
(:code file=bubblesort.java lang=java:)
```

The footnote creation syntax is similar to those of LaTeX. The footnote text must be given at the place where the footnote number is to be displayed. The text itself will be moved to the bottom of the page, even on the wiki page:

```
(:fn This is a footnote.:)
```

Since chapter and page sequence are defined by wiki trails, and since there is intentionally no inherent wiki page order, the footnote numbering starts at 1 again on every wiki page. The generated PDF documents however contain continuously numbered footnotes. At that point, online and PDF version must obviously differ because of their different nature. Depending on a unique trail it would be also possible to generate

continuously numbered footnotes across wiki pages, but we classify the limitation to a single trail per wiki field as too restrictive.

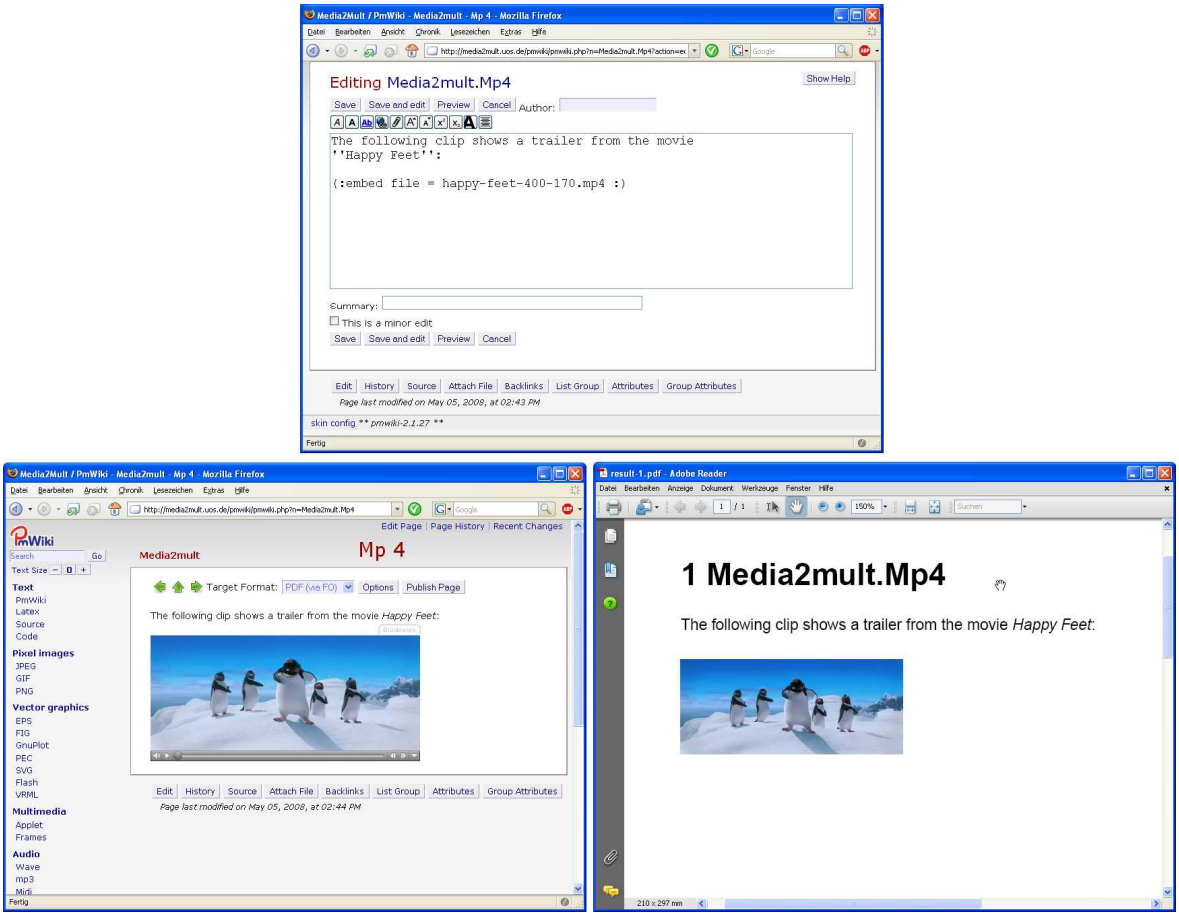

Figure 2. media2mult's media converter extracted a preview image from the embedded MP4 video that is used as a replacement in the created PDF document (bottom right).

4. REALISATION OF THE MEDIA EMBEDDING FUNCTIONALITY

The automatic processing of various media types belongs to the more complex programming aspects of media2mult. In order to ensure the extensibility of the plug-in a unified class has been developed. On this basis it is possible to integrate new media format handlers with minimal effort. Usually, only a few lines of code have to be written to support a new file format.

As shown in figure 3, the file types have been divided into the four categories *image*, *audio*, *video* and *script*. Category *script* represents all script languages, like gnuplot or LaTeX, that describe images, audio or video objects. All four classes are derived from the base class *MediaObject* which provides the fundamental, by means of template methods implemented conversion function *convert*. In case of success it returns a new MediaObject according to the specified parameters. The specialized methods *createMO* of the child classes specify the steps to be executed to create a media object of the particular type.

In order to take advantage of these methods, the first thing to do is to create an appropriate media object of the file referenced in the *embed* statement. This task is handled by the class *MediaObjectFactory*. It provides a static method that determines the file format and creates a corresponding object. A following call

to the object's *convert* method produces a new object of different type. For example, if we want to extract a preview image in PNG format from a Flash video we can use this code:

```
$video = MediaObjectFactory.createByExtension('video.flv');
$image = $video->convert('png');
```

The actual conversion tasks are not directly handled by media2mult but delegated to various open source command line tools, like ImageMagick's *convert*, the video tool *ffmpeg* or the SVG converter *batik*. Their conversion results are finally stored in separate folders segmented by wiki pages and media types.

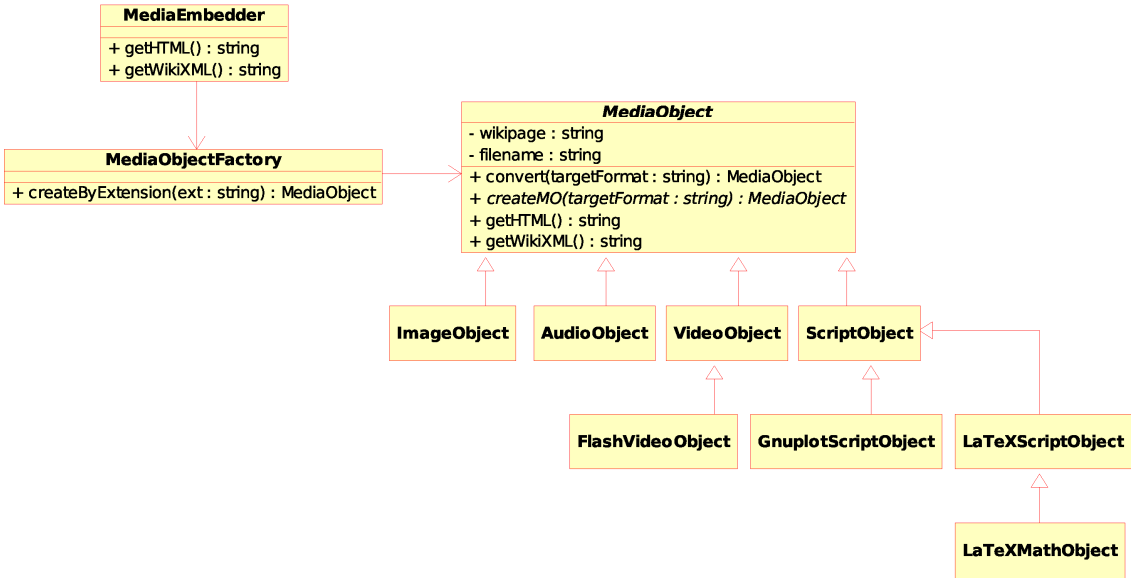

Figure 3. Excerpt of the class hierarchy used in media2mult to uniformly process different media types.

Example: Embedding of an EPS file

To illustrate the conversion process a bit more, we now take a closer look at the embedding procedure of an EPS graphic. As stated before, the user just has to upload the file using PmWiki's attach mechanism. On the wiki page he or she adds the *embed* statement that references that file:

```
The following figure shows the rendering pipeline of OpenGL:
 (:embed file=rendering-pipeline.eps transparent=white:)
```

The first line of this example consists of text only. Because there are no special markup rules for plain text, it will be displayed unchanged in the browser. In the second line the text replacement engine finds an *embed* statement. The corresponding markup rule initiates the conversion process.

media2mult now recognizes that the file format of the referenced graphic cannot directly handled by the browser an must be turned into a bitmap image. For this purpose the command line tool *convert* is called:

```
convert -transparent white graph.eps graph.png
```

The optional transparency parameter is passed to *convert* and makes sure that all white areas of the graphic become transparent. In the second step the actual text replacement takes place: PmWiki substitutes the *embed* statement by the HTML fragment `<img src="graph.png"/>`. The result is shown in figure 4.

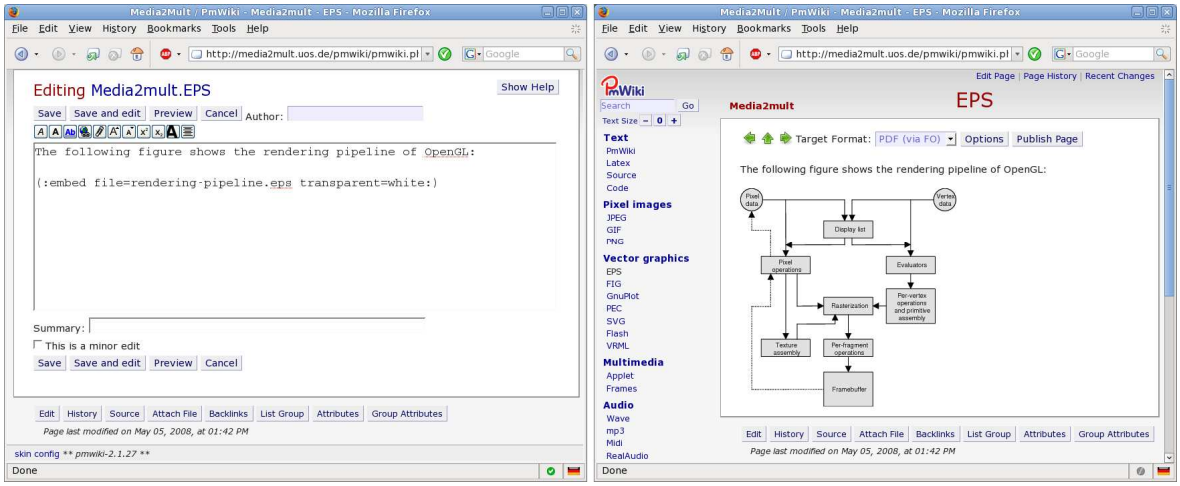

Figure 4. Edit and browse view of a wiki page containing an EPS graphic.

5. THE CROSS MEDIA PUBLISHING MODULE

The cross media publishing module of media2mult provides a function to convert single wiki pages or an arbitrary number of page sequences to PDF, PostScript or RTF. Before such a feature can be implemented we have to answer the question how to define an order on wiki pages. In contrast to printed documents, wiki fields are comparable to loose-leaf collections with many independent pages devoid of a unique order.

5.1 Wiki trails

However, if we want to transfer the contents of a wiki field to a print-oriented format, we must inevitably define a linear order on the pages. Furthermore, a structured document needs information about hierarchy levels so that chapters, sections and subsections can be represented. In PmWiki, the definition of such a structured order is realized by so-called *trails*, a list of links to wiki pages. By using nested lists with items and sub-items a section structure similar to a table of contents can be built. A trail in PmWiki syntax has the following form and leads to the result shown on the right:

```
* [[Introduction]]
* [[Sort algorithms]]
** [[Bubblesort]]
** [[Quicksort]]
* [[Search algorithms]]
```

1. Introduction
2. Sort algorithms
   2.1 Bubblesort
   2.2 Quicksort
3. Search algorithms

When media2mult finds a trail on a wiki page, it displays a button labeled *Publish Trail* that can be clicked to start the conversion process. On all other pages this button is missing and only a function to convert the current page is offered (see figure 4 on the right).

5.2 XML-based conversion

A click on button *Publish Trail* triggers the action *m2m-publish* that firstly collects all page contents in the order they appear in the wiki trail. Afterwards, the whole collected code is passed to the already mentioned function *MarkupToHTML* and converted to WikiXML, a slightly extended XHTML variant. For this purpose, special markup rules are applied. Besides converting the media files these rules also produce XML

elements describing the document structure and the generated media files, so that the following conversion process receives all necessary information.

The decision for XHTML as a base for all further conversion steps has the advantage that all markup rules already defined by PmWiki or additional plug-ins can normally be used without adaption. Especially, there is no need for doubling the elementary rule sets in order to get both wiki pages in HTML and XML files for PDF creation. That actually would complicate not only the progression of media2mult but also the update procedure for the whole PmWiki system. The XHTML attempt reduces the required adaption effort to writing markup rules for all new wiki statements like *embed* or *code*.

We already mentioned above that the first version of media2mult relied on DocBook. Even when we moved to another approach for the stated reasons, using XML technology proved to be very reliable. Particularly, the complex but powerful XSL-FO standard enables a fully automated creation of high-quality documents. An important advantage over LaTeX – which is usually chosen in order to create professional scientific documents – is the better network support. So it's possible to reference images located on different servers without the need of explicit downloads.

Furthermore, LaTeX's relatively inflexible table handling is a big problem. It is nearly impossible to translate a given HTML table to LaTeX without analyzing its actual content because LaTeX expects absolute width parameters for each column in order to break long lines properly. None of the currently available extension packages is able to completely solve this problem. Overall, the edit-compile-revise paradigm of TeX doesn't fit too well in an automated one-pass document creation process. In contrast XSL-FO has been designed for this scenario and works more reliable even if there's room for further improvements.

The XSL-FO files needed for PDF, PostScript and RTF creation are derived from the WikiXML files by applying a parameterized XSLT stylesheet. Since we wrote templates for nearly all XHTML elements the majority of all layout information included in the wiki pages is transferred to the XSL-FO file. This is true for simple text formatting as well as more complex layout data like text flow around images and table cell formattings. Big images that would cross at least one page margin are automatically downscaled to fit into the print area.

5.3 Output formats

Most of the currently available FO processors fortunately support various output formats, i.e. from a programmer's view, the main task is done by creating the FO file. Then the FO processor finishes the remaining work. Because of the complex XSL-FO standard a lot of time has been invested in writing flexible stylesheets for this intermediate format. However, there are planned further stylesheets for static HTML and LaTeX (see also fig. 5). The latter can be used as a base for further manual additions and/or layout optimizations.

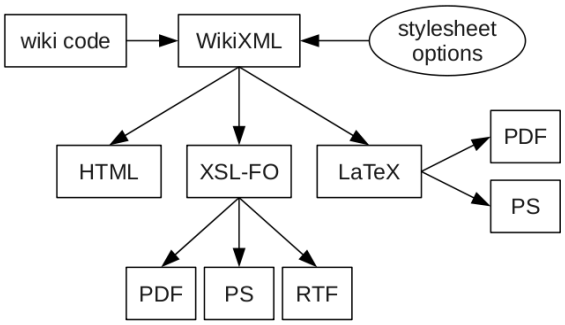

Figure 5. Conversion steps from the initial wiki source code to various target formats.

5.4 Stylesheet options

To increase the flexibility of the stylesheets and to give the user the opportunity to change some pre-defined layout settings, the stylesheet templates can be influenced by several parameters. They include options to adapt page size, margins, font style and sizes, as well as information about the handling of web links.

The stylesheet parameters can be altered through a web form. All settings are transformed to variable definitions and stored in a local XSLT file. This file also serves as a wrapper for the actual static stylesheet. The combination of WikiXML file and local stylesheet leads to an XSL-FO file indirectly adapted by the user (see fig. 6).

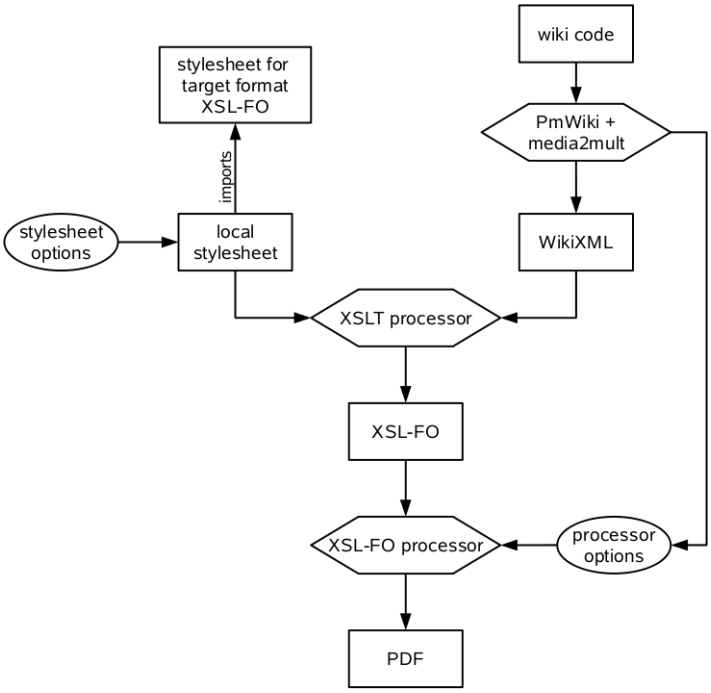

Figure 6. Schematic view of the cross media publishing component.

6. CONCLUSION

A PmWiki system extended by media2mult provides producers of e-learning content with a server-based authoring tool that enables them to collaboratively write multi medially enriched documents with minor effort. The documents are immediately available on the web and can be directly converted to print-oriented formats like PDF or RTF. For this purpose it offers transparent mechanisms for media file conversions. Among other things, they make it possible to embed natively unsupported files, like EPS, into wiki pages without the need of manual conversions. In conjunction with video files media2mult also extracts preview images for PDF documents.

The cross media publishing component converts sequences of wiki pages to PDF or RTF files using an XML-based conversion procedure. While able to influence the conversion process by stylesheet parameters the user doesn't come in touch with the XML, XSLT and FO files. The user input is confined to the wiki input language and the layout parameter web form.

At our university, media2mult is successfully used in many seminars of different subjects -- especially in non-technical fields as well. There are currently over 200 independent wiki fields on our farm server. Lecturers and students are able to productively use the tool after a short time of practice. The field of application ranges from student term papers to whole books including lecture materials.

A short media2mult document with the most important syntax and media examples can be found at http://www.media2mult.de.