Open-Source Textbooks: Infrastructure for Customized Learning Materials

Dror G. Feitelson
School of Computer Science and Engineering
The Hebrew University, 91904 Jerusalem, Israel

Abstract

Writing a good textbook is akin to writing a useful software package. Both are highly creative endeavors that require a combination of sharp technical skills and artful design. In recent years, the software community has developed the open-source approach to software construction, in which improved results are obtained by sharing the talents of multiple developers. We propose to extend these ideas to the creation of study materials, specifically textbooks.

With open source textbooks teachers can build upon a repository of existing materials. If no available textbook meets their requirements, they can mix from several sources to tailor an up-to-date customized textbook to fit the needs of their specific course. They can also add new materials that were not previously available in the repository, thus contributing to the collaborative process of creating new teaching materials.

Using open-source study materials is expected to be highly beneficial both in educational terms and in economic terms. It will enable teachers to develop specialized study materials with relatively low effort and at low cost, because they will be based on reusing existing materials. This is already important in the context of traditional textbooks, and may be expected to grow in importance when moving to more advanced interactive learning materials for use in e-learning systems.

Contents

1 Introduction

Universities and students typically spend large sums of money on textbooks and other study materials. These expenditures are continuous, because study materials have to be kept up-to-date, and new textbooks (and new editions of old ones) are published every year. But despite the investment, it is often found that the obtained materials do not quite fulfill each teacher's needs. A partial (and expensive) solution is to acquire several competing books, and use a different part from each one.

The Internet has created great opportunities for new educational approaches. There is great excitement regarding on-line remote learning environments, and the option of interactive learning environments that adjust themselves to the progress of each student. In addition, more conventional course materials are also provided on-line.

The Achilles heel of most such projects is the creation of content. Unlike the technical issues involved in e-learning, the content issues cannot be solved by clever engineering. Moreover, creating content for one course does not solve the problems of other courses. Creating content is difficult: it requires thorough knowledge of the study material, and a knack for presenting it in an interesting and understandable manner. In short, even for e-learning good teachers are required, and they are a scarce resource.

To ease this situation, we recommend to create an infrastructure for open-source textbooks (initially we focus on traditional textbooks, but this can be extended to learning materials in general, including interactive ones). This infrastructure will enable teachers to build upon study materials prepared by others when creating their own courses, instead of having to do everything themselves. By pooling resources, it will be possible to create more advanced, accurate, focused, and useful study materials in a shorter time. Note that this is different from other forms of collaborative authoring: instead of having many people working on a single, shared text, we have people working on many parallel, competing versions, and borrowing from each other.

Surprisingly, as far as we know such systems are not currently available. In fact, the whole issue of document reuse and construction by multiple independent authors has received very little attention in the literature, as opposed to more technical issues of describing document structures for automatic composition. The little work that has been done has focused on abstract developments, such as specialized languages to describe document structures and meanings [13, 4, 14]. We know of only two applied efforts [3, 1], and these have not led to available and usable systems.

Note that we are not talking about e-books, where the focus is on digital delivery using the Internet. We are talking about the process of creating books, regardless of how they will be used, which can be both printing on paper and electronic access. In particular, new print-on-demand technology facilitates the economical production of low-volume printed books, as would be needed for a specialized version of a textbook (see, for example, `www.lulu.com`).

The next section reviews other projects that have similar goals or use similar terminology. Section 3 then presents our system design, and the features it supports. Section 4 comments on the similarities and differences between open-source textbooks and open-source software.

2 Background and Related Work

"Open-source" is a methodology in which one individual or organization does not develop a software system in isolation and behind closed doors. Rather, the source code for the system is made available to all, and the community at large is encouraged to contribute criticisms and additions. This is the methodology that was used to create highly successful projects such as the Linux operating system and the Apache web server. It is now used for the development of tens of thousands of other software systems (see `sourceforge.net`).

While the details depend on the exact license being used (and there are many nuances), open-source typically also implies open-content. This means that the source code can be modified by others and used as part of new projects. The only requirement is usually that the new projects also be open. Note that open-source and open-content should not be confused with open-access, which refers to no-cost access to content that is published by traditional means, with no option to re-use or modify it.

When applied to textbooks, the open-source/open-content approach will enable teachers to create new study materials by combining many existing pieces. We envision a repository where teachers can deposit partial study materials of various sizes — not only complete books for whole courses, but also individual study units, figures, animations, definitions of exercises, etc. These can then be combined as needed in each case. This is similar to the notion of reusable learning objects [12], but more modest in scope as explained below.

We define open-source textbooks as supporting three features:

collaboration — the creation of an open-source textbook is a collaborative process involving contributions from multiple authors. Of course an initial version can be the work of a single author, and later a small group of authors may still dominate, but others will also contribute content or editing.

customization — an important use of open-source textbooks is to be customized by a teacher to match the detailed structure of a specific course. While there may be a single master version, as is commonly the case with open-source software, there can also be multiple additional versions with various degrees of overlap.

evolution — open-source textbooks can track progress in the field without resorting to discrete new editions. Each time a course is given, a new book can be created with the latest updates. This is similar to the concept of a continuously updated core handbook proposed by Engelbart [6].

using this definition, we can compare our ideas with other similar projects. The most common form of collaborative books are edited collections. This sort of collaboration nevertheless maintains the distinction between the contributions of individual authors, and does not allow users to modify content according to their needs. The idea of collaborative creation of content has been promoted by various web sites in recent years, but this has typically not amounted to much in terms of real collaborations and real content. Making course materials available on the web has also been done before, e.g. in the MIT OpenCourseWare project (`ocw.mit.edu`). But the current proposal

aims to go much further, by providing the *sources* of study materials, and tools to actually use the available resources to create new ones.

There are, however, several projects that are somewhat similar to our proposal. The best known and most successful are wikis [9]. A wiki is an environment for collaborative authoring using the web. The largest and most successful is probably the Wikipedia (`www.wikipedia.org`). This is a free encyclopedia in which any user can add new articles and edit existing ones. Amazingly, this has led to the creation of nearly 700,000 articles in English, and hundreds of thousands more in dozens of additional languages[1]. However, there is only one shared version, and there is limited support for typesetting and graphics; for example, mathematical formulas are either avoided or typeset separately and inserted in the form of an image. These limitations are true also of the more recent Wikibooks effort (`www.wikibooks.org`), which specifically targets the creation of topical textbooks rather than a general encyclopedia. In particular, there is no support for divergent versions of a book, where different instructors choose different structures and different pieces of text.

Another project is the "structured and intelligent documents assembly workbench" (SAW) [1]. This is superficially very similar to our proposal in its attempt to support the creation of new documents using the fragments of existing ones. However, the approach and mechanics are very different, with much emphasis on automatic classification and search for suitable materials. (It is based on earlier work that dealt mainly with parsing MS Word documents and collecting desired fragments based on massive manual tagging [2].) The system most similar to our design in terms of document structure is that of Barta and Gil [3], except for their base of personal MS Word documents without specific facilities for sharing.

In our framework for open-source textbooks, the construction of new material will be done by a combination of linking existing pieces, editing them, and adding new materials where no appropriate ones were found (naturally, such new materials should then be deposited in the repository for future use by others.) This is superficially similar to the "cut-and-paste" approach often used today. The difference is in scope and ownership. While today people typically combine materials that they have previously created and that they own, we envision a large-scale system that facilitates the *sharing* of resources.

The mechanisms of linking are related to those of hypertext. Sophisticated systems for document linking and construction have been designed in the last 30 years. We propose to borrow the most useful concepts and use them in the proposed framework. Examples include the hierarchical structuring of documents out of fragments [3, 11], and the use of paired displays to compare and edit two version of the same document [11]. However, it is quite different from the content-linking and transclusion concepts of project Xanadu [11]. That project is obsessed with keeping track of all links and copies for all time. While this may be important for the evolution of knowledge in scientific papers, it is burdensome for textbooks. Rather, in preparing course materials, it is more important to present a cohesive and comprehensive picture, at the possible expense of not recording the potentially convoluted path in which it was originally created.

[1] These numbers are for August 2005.

3 A Framework for Open-Source Textbooks

At the present time our proposal is but a paper design, and not yet a working environment. The design is based on using the facilities of various text processing and formatting environments, such as LaTeX [8]. Naturally this design may be expected to change as part of an effort to actually implement it.

3.1 System Architecture

The first major design decision that we need to make concerns the system architecture. This is a matter of coupling between the central repository of learning materials (e.g. textbooks, sections, figures, and exercises) and its distributed users (the authors that write new materials and create new organizations). There are two main options. In the centralized option, all work is carried out directly with the central repository via the Internet. In the decoupled option, users of the system work on a local copy and only interact with the central repository to obtain new materials or deposit completed new creations.

The centralized approach has the benefit of simplifying the clients. All interaction (including editing) is carried out via a standard web browser, as was originally done in SAW [1] and is now common in all wikis [9]. However, this suffers from two drawbacks. One is that it is much more vulnerable to communication difficulties, which may hurt the interactive nature of editing to the point of making it unsatisfactory. The other is that browsers actually provide a rather poor programming and editing environment, so various advanced features that we would like to have may prove to be too difficult to implement. The decoupled approach avoids these problems at the expense of another one: users will have to download and install the editing environment on their own machines. In addition, there will be no way to ensure that users deposit the fruits of their labor in the repository for others to use.

At the present time it seems that our ideas are easier to implement using the decoupled approach, in the interest of providing a better working environment for users. Specifically, our description will be based on a Linux environment. However, a real implementation may differ.

3.2 Document Structure

The next issue that needs to be tackled is the structure of documents, regardless of where they are stored. We suggest to initially use the LaTeX system for typesetting, as it is the most popular one used in computer science academia, and enjoys a good translator to HTML [5]. Moreover, it is eminently suitable for partitioning documents into parts and combining them again, because all formatting is controlled by simple ASCII commands, and page and section numbers are assigned automatically. However, we are left with the choice between native LaTeX commands, and the newer LyX graphical front end, which regrettably uses a different file format. An initial implementation would probably use LaTeX in the interest of simplicity, while keeping the door open to migration to LyX later on.

The logical structure of educational documents such as textbooks is rather self evident — it is hierarchical, with the whole divided into chapters, sections, subsections, etc. But there are com-

plications. One is that when pieces of a document are moved from one document to another, they may be promoted or demoted in the hierarchy. For example, a section from one book may become a subsection in another. To support this cleanly, all subsections must become sub-subsections, etc. Another complication is that there may also be inserts that are independent and not part of the section hierarchy, for example figures, exercises, and definition boxes. To further complicate matters, these may have different forms depending on the viewing media. For example, a figure may be a postscript file for processing by LaTeX and printing, but an animated gif file for inclusion in an HTML version.

For all document types, a major design issue is how to store them. In the interest of simplicity and applicability, we prefer to avoid the definition of new standards [10]. Rather, we will use the conventional LaTeX format as much as possible.

If all we were interested in was the creation of printed books, we could opt for files that contain the nested structure of chapters, sections, etc. — just like a normal LaTeX file. But we intend small fragments to be interchangeable (e.g. different users will move sections around, add exercises, modify figures, etc.). It therefore seems advisable that the smallest interchangeable units be the basic storage units. Larger units, such as chapters or complete books, will actually just be an index structure that identifies the constituents that need to be stringed together.

More specifically, there will be two basic types of files: sectional units and inserts. A sectional unit file has the following generic format:

```
\sect{ section-title }
text
\include{ subsection-file-name }
```

This has three parts. It starts with a generic section header (`\sect{...}`) that specifies the section title. Note, however, that in contrast with LaTeX, this does *not* indicate the level of nesting: there are no distinct commands for sections, subsections, etc. This is followed by some introductory text, of the type that appears at the beginning of a section, before the first subsection heading. If there are no subsections, the full text of the section appears here (this is the case at the bottom of the hierarchy). At the end come a list of subsections. These are actually links to other files, denoted using the `\include` command, as in LaTeX. It is these inclusion relationships that are used to specify the hierarchical structure of the book, and to assign each sectional unit to the correct level. For example, files included in a section are understood to be one level lower in the hierarchy, that is, subsections As an immediate result of this structure, we utilize LaTeX's automatic section numbering, so users need not be concerned with this issue when they are reorganizing their textbooks. The whole structure is illustrated in Fig. 1.

Inserts are a piece of text that can be incorporated in other text, such as a figure, a table, a definition box, or an exercise. In the case of figures, this is a wrapper around the actual figure that is in some graphical format, e.g. postscript. For both figures and tables, an appropriate caption is included. For example, a figure insert will have the following format:

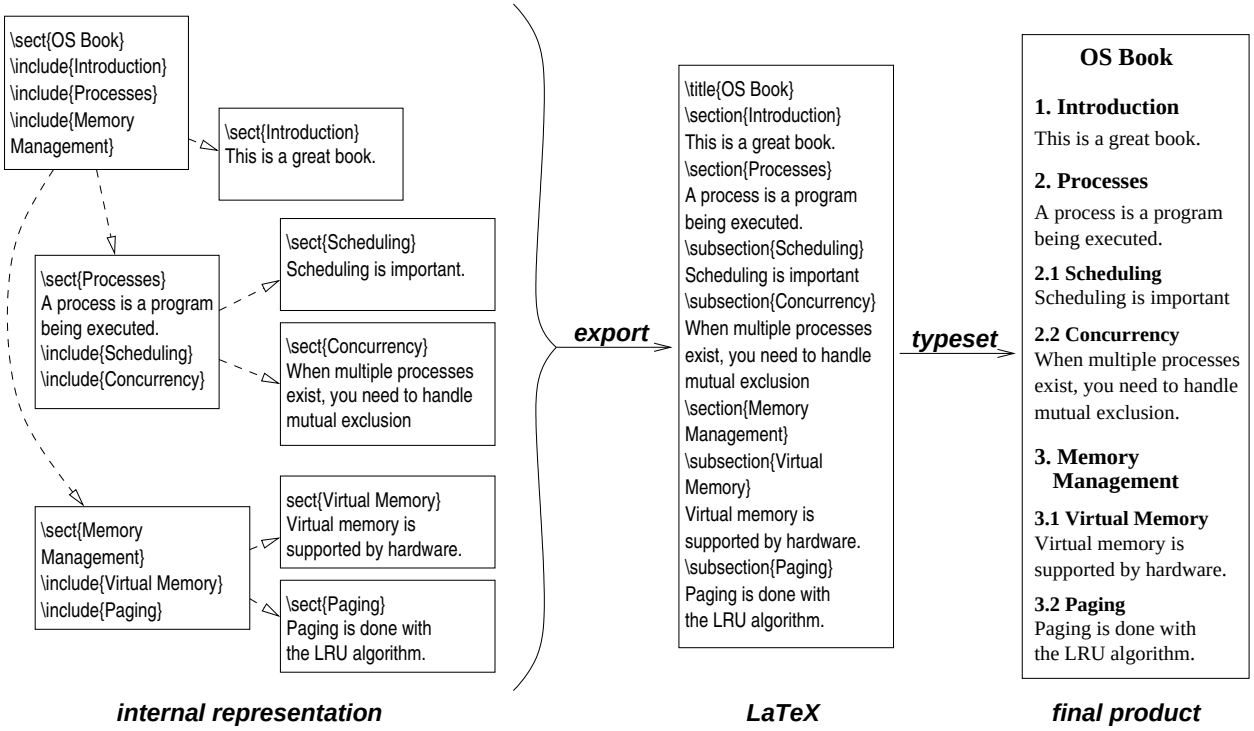

Figure 1: *Relationship between the hierarchical structure of a book as reflected in the internal storage, the exported LaTeX document, and the final product.*

```
\begin{ figure }
\includegraphics{  graphic-file-name }
\caption{ caption-text }
\end{ figure }
```

which is exactly as in LaTeX. This insert file will be included in some sectional unit text, using the same `\include` command that is used to include subsections. This naturally facilitates automatic numbering depending on the inclusion pattern, which is a basic feature in LaTeX. In particular, figures and tables will be numbered consecutively within each chapter.

The actual content of tables is easily incorporated in the insert file itself, using LaTeX's table formatting commands. But figures pose special problems because of how they are generated. To qualify as open source, it should be possible for users to modify the figures. But this typically requires additional data and files that are not part of the final product — for example, a gnuplot script and its associated data files. Each figure should therefore be accompanied by an archive containing the files needed to generate and modify it.

Note that the above is less ambitious than general reusable learning objects [12]. By focusing on printable textbooks based on LaTeX, and assuming some editing by users who want to create a textbook from existing components, the basic units are simplified. In particular, they need not be reusable without modification in different contexts, and they need not be completely independent of media. These simplifications should greatly reduce the doubts that our concept is realizable.

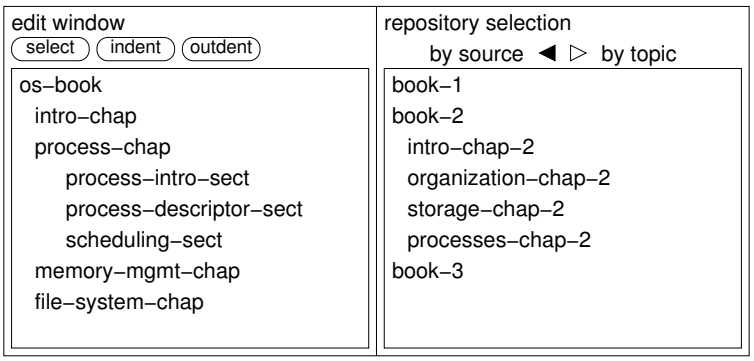

Figure 2: *User interface for skeleton construction, enabling the choosing and organization of components from the repository.*

3.3 Editing Environment

The interface to using the repository of available sectional units and inserts is the editing environment. The editing environment has to support two main functions: construction of a skeleton from existing components, and editing the final combination to ensure that it flows smoothly. The final edit is essentially just like any other editing task, so conventional text editing tools are expected to suffice. But constructing a document from components is more novel.

The basic design we envision for document construction is based on two windows: a new document window and a resources window (Fig. 2). The resources window is used to display the available fragments from existing documents. It can be fashioned like a hierarchical file browser. At the top level are alternative "books". Opening them reveals chapters, and then sections and subsections. All this can be displayed in two orthogonal organizations: a source-oriented organization, in which the branches in the hierarchy are consecutive parts of the same book, or a topic-oriented organization, in which branches are alternatives of the same part from different sources, as identified by a search procedure.

To construct a new document, a drag-and-drop mechanism will be used to select document fragments from the resources window. These will be placed in the desired location in the new document window, thus creating the desired sequence. An indentation mechanism will be used to specify the desired level of each unit — whether it is the at the same level as the preceding unit, a subunit of it, or a new unit at a higher level.

Each unit incorporated in the new book can be edited in a separate buffer. This applies only to the direct text of this unit, not to subunits or other surrounding text. The main function of such editing is to incorporate inserts into the flow of the text. Normally, the selection of a sectional unit from the repository implies the selection of all the inserts included in it as well.

Sometimes it may be necessary to compare fragments to decide which one to select, or to see how they relate to other fragments that have already been selected. This can be supported by enabling the user to load any fragment from the repository into another editor window. This will lead to a situation in which two windows are used to display the new document and some other fragment side by side. It is desirable that these will share a common scrolling mechanism, with highlighting to enhance the coupling between the two documents (similar to the mechanisms of

transpointing windows suggested in the Xanadu project [11]). However this will only be possible if the two versions are indeed closely related to each other, and if the user uses the system's native editor rather than employing the editor of his choice.

When the desired structure is achieved, it will be exported to create a single conventional LaTeX document that can be further edited. The exportation process will automatically adjust the level of nesting, i.e. turn the generic `\sect` header into `\section`, `\subsection`, or `\subsubsection` headings — using one level deeper than that of the including file each time (Fig. 1).

The inverse procedure will also be possible, i.e. to import a standard LaTeX file into the system for additional structural modifications. To do so, the editing environment will parse the file and partition it into its sectional units and inserts.

3.4 Versioning and Attribution

Two related problems with collaborative authoring of open-source textbooks are versioning and attribution. Versioning is the problem of producing multiple competing versions of the same text. This is problematic because users are then faced with too many choices, and may end up wasting too much time on comparing versions. Attribution is the problem of attributing contributions to their correct authors. This is important because one of the main motivations for joining open-source projects is recognition.

Dealing with these problems is more of a cultural issue than a technical one, but the technology can help in fostering a desirable solution. In the context of our framework for open-source textbooks, there is a need to enable two types of editing. One is minor editing for correcting typos and other non-consequential modifications. It is highly desirable that anyone would be able to make such corrections without branching a new version of file. A simple mechanism to achieve this is a wiki-like editing system. The other type of editing is making a major revision, in which the file is drastically changed. This should lead to a new version that possibly competes with the original one. However, such branching should not be taken lightly, and cultural pressure should be applied to limit it. And indeed, in open-source software development forking off new versions of a project — while possible — is generally frowned upon [15].

Branching new versions is obviously related to authorship and ownership. Based on the analogy with open-source software, it seems advisable that each file (representing a sectional unit or an insert) have a single main author, who is its owner. Anyone can make small corrections to this, as noted above, but if you want to make a major modification you need to start from scratch and create your own file. When inserting a file into the repository the author is noted, and this enables the list of contributors to be generated automatically when a new book is created by collecting existing fragments (and even counting exactly how much each one contributed).

An important additional benefit of this organization concerns labeling and cross references. LaTeX provides convenient support for cross references by using two commands: `\label` to create a label that refers to the nearest enclosing structure (typically a figure caption or a sectional unit), and `\ref` which prints the number associated with the referenced label. The problem with using this mechanism in multi-authored collaborative texts is that labels need to be unique. The solution is that each author should only be responsible for labels in his own text. The global

labels are then generated by concatenating the author name with the label, thereby guaranteeing uniqueness.

A result of this approach to ownership is that open-source textbooks will be created as collections of fragments that are each created by a single author, rather than as the result of multiple editing sessions of authors that modify each others work. The person who makes the collection (i.e. the instructor who is creating a customized book for his course) then becomes an editor who needs to smooth the boundaries and make the different fragments fit together.

4 Comparison with Open-Source Software Development

The success of open-source software has been used by others to argue for open-source learning materials in general and open-source textbooks in particular. Many of the considerations and procedures of open-source software development seem to carry over directly to open-source textbooks. For example, one can have a core group of developers (read authors or editors) that retain control over the project, while other contributors make focused contributions and suggestions. At the same time, the periphery can help with debugging (read finding typos and inconsistencies). Licensing issues are also similar, e.g. using "copyleft" to allow dissemination and modification while preventing direct commercial use. In fact, the modification-enabling licenses of open-source software are more appropriate for open-source textbooks than the open-access licenses more common for literature. Acknowledgments can be used to keep track of contributors, however minor.

In addition, software engineering concepts can been applied to the building blocks of such study materials. For example, it has been suggested that the reusability of learning objects will be improved if their authors attempt to reduce the coupling among them, and to improve their internal cohesion [7].

Both open software and open text suffer from versioning problems. Ownership of text can be defined in the same way as ownership of code: the owner is whoever has the right to make modifications in the "official" version [15]. But this doesn't imply that others are not allowed to make modifications. It just implies that if they are not satisfied by the official version, they need to work on a personal branch. Such branching is frowned upon in the context of general development of large software projects, as it leads to competing versions. However, it is commonly used to demonstrate or perform research on new functionality, possibly with the intention of proposing that it be incorporated later into the main branch. The same can happen with textbooks. For example, a number of authors may collaborate on a single version of a textbook, while other authors use it as a basis for diverging versions that better fit their specific needs.

While these and other similarities exist, text and software are actually not the same. A major difference is the intended consumer. Text is intended to be read by humans, whereas software is intended for a compiler.

Humans may be more forgiving than a compiler, but may also be more sensitive. On one hand, humans can read and use text that is far from perfect. They can compensate for gaps in the flow and for spelling errors. They can work with incomplete drafts and even benefit from their use, as exemplified by the Wikipedia and Wikibooks. Thus open-source textbooks allow much more flexibility in mixing and matching — there is no minimal requirement of having all components

for the whole to function. You can create a partial book for an ad-hoc specific need by combining parts of two existing books. You can't do this when combining two software systems; you need to first reconcile the details of the interfaces, and even then, chances are that subtle differences in the semantics will require some debugging. In general, compilers require a much higher level of perfection than humans — a single syntax error will cause the whole program not to compile. With text, one part cannot break another except for a possible effect on cross references.

On the other hand, humans are more sensitive to style. A compiler doesn't care if different modules are coded in different styles, as long as the interfaces match. Even convoluted code that is hard to understand can compile into a correct and useful executable. But humans attach much more importance to style, and the flow of the text. In fact, a human assessment of a textbook's quality may be influenced more by the style than by the actual content. This may be problematic for textbooks created using an open-source methodology, because different sections and subsections might be written by different authors, leading to a choppy reading experience.

5 Conclusions

A number of attempts to jumpstart open-source textbooks have been made in recent years. Regrettably, this movement has so far not taken off, with the possible exception of the Wikibooks effort. Part of the problem may be overambitious goals and an emphasis on sophisticated technology and legal issues. Rather than foster a collaborative environment, such actions may actually hinder it, by creating a higher learning barrier that has to be overcome by potential contributors. This may also explain the relative success of wikis, which are extremely simple to use. We believe that our proposed framework is also simple enough to be viable, while providing enough added benefits in terms of typesetting and the option of producing real printed books to be attractive.

Of course a real test of these ideas would require an implementation. This needs to incorporate two elements. The first is technical, i.e. an implementation of the repository and editing tools described in Section 3. The second is enough initial content to make such a repository attractive for potential users. This can be based on lecture notes from several courses on the same subject matter, by different instructors, in an attempt to achieve the required critical mass.

Open source software has been especially successful in creating software infrastructures — software tools that are used by many other developers to create more software; one of the reasons for this success being the prestige that developers get when the products of their work is used. It can therefore be hoped that a similar success can be achieved for educational infrastructure, in the form of open-source textbooks.

References

[1] H. Ahonen, B. Heikkinen, O. Heinonen, J. Jaakkola, P. Kilpeläinen, and G. Lindén, "*Design and implementation of a document assembly workbench*". In 7th *Intl. Conf. Electronic Publishing*, pp. 476–486, Springer Verlag, Apr 1998. Lect. Notes Comput. Sci. vol. 1375.

[2] H. Ahonen, B. Heikkinen, O. Heinonen, and P. Kilpeläinen, *A System for Assembling Specialized Textbooks from a Pool of Documents*. Technical Report C-1997-22, University of Helsinki, Department of Computer Science, Mar 1997.

[3] D. Barta and J. Gil, "*A system for document reuse*". In 7th *Israeli Conf. Computer systems and Software Engineering*, pp. 83–94, IEEE Computer Society Press, Jun 1996.

[4] L. K. Branting and J. Lester, "*Justification structures for document reuse*". In 3rd *European Workshop on Case-Based Reasoning*, pp. 76–90, Nov 1996. Lect. Notes AI vol. 1168.

[5] N. Drakos, "*From text to hypertext: a post-hoc rationalisation of LaTeX2HTML*". *Computer Networks and ISDN Systems* **27(2)**, pp. 215–224, Nov 1994.

[6] D. Engelbart and J. Ruilifson, "*Bootstrapping our collective intelligence*". *ACM Comput. Surv.* **31(4es)**, Dec 1999. Article 38.

[7] R. Jones, "*Designing adaptable learning resources with learning object patterns*". *J. Digital Information* **6(1)**, Dec 2004. Article 305.

[8] L. Lamport, *LaTeX: A Document Preparation System*. Addison Wesley, 2nd ed., 1994.

[9] B. Leuf and W. Cunningham, *The Wiki Way: Quick collaboration on the Web*. Addison-Wesley, 2001.

[10] D. M. Levy, "*Document reuse and document systems*". *Electronic Publishing — Origination, Dissemination, and Design* **6(4)**, pp. 339–348, 1993.

[11] T. H. Nelson, "*Xanalogical structure, needed now more than ever: parallel documents, deep links to content, deep versioning, and deep re-use*". *ACM Comput. Surv.* **31(4es)**, Dec 1999. (Online electronic supplement).

[12] P. R. Polsani, "*Use and abuse of reusable learning objects*". *J. Digital Information* **3(4)**, Feb 2003. Article 164.

[13] R. Rada, "*Hypertext writing and document reuse: the role of a semantic net*". *Electronic Publishing* **3(3)**, pp. 125–140, Aug 1990.

[14] J. C. Ramalho, J. J. Almeida, and P. Henriques, "*Algebraic specification of documents*". *Theoretical Comput. Sci.* **199(1–2)**, pp. 231–247, 1998.

[15] E. S. Raymond, "*Homesteading the noosphere*". URL http://www.catb.org/~esr/writings/cathedral-bazaar/homesteading/, 2000.