Deriving a Comprehensive Document from a Concise Document

Document Engineering in Scheme

Kurt Nørmark
Department of Computer Science, Aalborg University, Denmark
normark@cs.aau.dk

Abstract

In this paper we analyze and discuss the problem of deriving a comprehensive and coherent document from a concise starting point. In more concrete terms we discuss how to grow and consolidate a slide presentation to a complete and self-contained document. Throughout the paper we are concerned with avoiding unnecessary duplication of document fragments and authoring efforts. Our experience is based on the Scheme-based, XML-constrained authoring system called LENO, which we use for creation of annotated slide presentations in programming courses. We explain how a LENO slide presentation can be extended to a document in the style of a textbook. The textbook and the slide presentation may share both structure and contents. From an authoring perspective we discuss the process of dealing with both a primary source and a derived source. From a technical perspective we discuss the ideas of single sourcing, derivation of a secondary source from a primary source, and the use of Scheme for document engineering purposes.

1. Introduction

In this paper I will discuss the use of Scheme as a document description language. This will involve document engineering issues related to Scheme. The work in the paper describes a niche of the work on LAML (Nørmark 2005).

In almost a decade I have made extensive use of Scheme as a textual markup language. The authored materials include slides, annotated slides, and textbook materials in the area of programming and programming languages. In this paper I will focus on issues that bridge the gap between authoring of slide materials and textbook materials. The use of Scheme as a document description language will be illustrated throughout the paper.

The starting point of this paper is best described by the following scenario:

> We have developed a concise slide material that covers some topic, such as a lecture in a university course or a professional area presented in a seminar. We need to make an extended version of the material, which essentially includes the slide elements as a subset. The extended version is supposed to be coherent and comprehensive, as a contrast to the slide material, which typically is itemized and concise. It means that more content items need to be filled in. To obtain an optimal exposition of the material a few structural changes need to be accommodated. Due to media concerns, a few dynamic elements need to be replaced by static counterparts. The extended document should be printable.

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. To copy otherwise, to republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee.

2007 Workshop on Scheme and Functional Programming September 30, 2007, Freiburg, Germany

Copyright © 2007 ACM [to be supplied]. . . $5.00.

Slide presentations are popular and widespread in most university courses. Many teaching materials are first created as slide presentations and later developed to more complete and self-contained contributions (notes and textbooks). The problem of deriving a more complete and comprehensive version of the slide material is therefore a well-known and commonly occurring problem for many teachers and authors. We hypothesize that the ideas and the solutions to this concrete problem can be generalized to other contexts, which demand a derivation of a coherent and comprehensive material from a concise starting point.

The paper also relates to the authoring process of the material. Our work supports a process where the key elements, in terms of brief and essential statements, are formulated first. In addition, the overall structure of the material, as needed for presentation purposes, is also determined early in the process. The remaining parts of the full document are built as layers around these key elements. Overall, the structure of the concise document is supposed to be preserved in the comprehensive document. However, the approach, which will be detailed below, is flexible enough to accommodate almost arbitrary structural differences between the concise document and the comprehensive document.

Our work is based on a Scheme representation of both the slides, possible annotations of the slides, and the derived comprehensive document. The target formats are HTML and SVG[1]. The backbone of the Scheme source representation is defined and constrained by an XML Document Type Definition (DTD). With this positioning we are able to benefit from the power of a flexible, general purpose programming language in the document source text. We are also in tune with the communities that make use of XML for document representation purposes. The differences between XML fragments and the S-expression counterparts in Scheme are small, and lexical transliteration can, in principle, be used for transformations in between them.

The rest of this paper is structured as follows. First, in section 2, we discuss the advantages of avoiding duplication of document elements. In section 3 we discuss a particular document model based on a primary source of the concise document and a derived, secondary source of the comprehensive document. The concrete tool, which implements the document model, is discussed in section 4. The tool is implemented in Scheme. The use of Scheme for document engineering purposes is summed up in Section 5. In

[1] SVG (W3C 2003) is an XML language for describing two-dimensional graphics.

section 6 we discuss the current work relative to similar work. The conclusions are drawn in section 7.

2. Authoring without duplication

The handling of a non-trivial teaching material, and the management process involved during the lifetime of such a material, typically involves a variety of duplicated elements and efforts. In this section we will discuss these problems at a general level. Consider the following examples of duplications:

1. Document fragments which both exist inside and outside the material (such as external images, tables, and computer source programs).
2. Elements fragments which occur at two or more different places in the material.
3. Full and abridged versions of the material (such as a coverage of a curriculum in many or just a few lectures).
4. A number of different, but temporally identical editions of the material (such as a web version, a CD version, a version that can be downloaded, and a printable version).
5. A number of temporally different versions of the material (such as versions related to each year a given course is offered).

My personal experience stems from authoring of slides, notes, and textbook materials about computer programming. A material about Functional Programming in Scheme may serve as an example (Nørmark 2003c). The most recent example is a teaching material about object-oriented programming in C# (Nørmark 2007b), which currently is in active development. I have found it attractive to include program source files by *transclusion* (Kolbitsch and Maurer 2006), superficially following the ideas of Ted Nelson (Nelson 1995). With this solution the teaching material is updated[2] whenever the source programs are updated. Transclusion should be seen as an alternative to manual pasting of program copies into the document.

Some key parts of the material may appear more than once in the material. If these parts are duplicated at the source level, it is difficult to keep the material consistent and up-to-date during the lifetime of the document. On several occasions I had to give short versions of the courses, where only a subset of the material was directly and explicitly exposed. In the starting point it is easy to make a copy and hereby to edit an abridged version, but it is very difficult to keep both the full and short version consistent with each other subsequently.

Overall, the tendency of avoiding both small-scale and large-scale content overlaps has been crucial for the quality of the material. The amount of inconsistencies can be reduced if a single source is responsible for multiple appearances of some detail. If, for instance, the teaching material exposes a C source program it is in most situations[3] essential that the latest and most up-to-date version of the program is being exposed. In addition, avoiding content overlaps in the document source has a positive impact on the efficiency of the authoring process.

The primary challenge discussed in this paper is to manage a concise (slide) version and a more comprehensive (textbook) version of the same material in relation to each other. The amount of overlap between these versions is massive, and therefore it is crucial to avoid unnecessary duplication of document elements and authoring efforts. However, the potential difference between the concise and the comprehensive version is large enough to challenge or disable traditional single sourcing approaches (see section 6). In the next section we will explain the document model we have developed, and in section 4 we will discuss our concrete solutions in Scheme with the LENO system.

[2] Depending on the propagation strategy, the actual updating may be instantaneous, or it may depend on some action (processing) to be initiated by the author.

[3] In rare situations, where we for instance address an early version of a program, we may want to prevent propagation of the most recent version.

3. Primary and secondary sources

Our work is based on a two-level document model called the PriSec model (the primary/secondary source model). The PriSec model has been created as part of our work on the LENO system. In the model, the concise document, corresponding to the slide presentation, is represented by a *primary source*. The primary source contains those elements, which are relevant and necessary at the slide level. In addition, the primary source may hold annotations which are tightly connected to the individual elements. The primary source is structured according to the needs and use of the concise documents (typically a number of individual pages, each with a limited amount of information).

In the PriSec model, the comprehensive document is represented by a *secondary source*. The comprehensive document may, for instance, be structured as a textbook, in terms of of chapters, sections, and subsections with use of figures, tables, etc. In order to minimize the amount of duplication, as discussed in section 2, those elements of the concise document that also appear in the comprehensive document are represented as references from the comprehensive document to the concise document. Additional content elements can be added to the secondary source. This organization of the two documents relative to each other is illustrated in Figure 1.

In the figure, there are 12 primary elements pe_1, pe_2, ... pe_{12} structured in four sequential units (slide pages). There are 13 secondary elements structured in two sequential units (article or book chapters). Of the 13 secondary elements 8 are references to the primary elements, and 5 are contents elements that contribute only to the comprehensive document. Some primary elements (pe_4, pe_6, pe_8, pe_{10}, and pe_{11}) are not part of the comprehensive document. A single primary element (pe_5) appears twice in the comprehensive document. The ordering of the primary elements in the comprehensive document is pe_1, pe_2, pe_3, pe_7, pe_5, pe_9, pe_{12}, and pe_5.

With this organization of the primary and secondary sources, the freedom of composing the comprehensive document by means of references to the elements of the primary source is evident. The existing elements from the primary source may be thought of as building blocks of the secondary source. The order of appearance of the elements in the comprehensive document can be controlled freely. Some elements from the primary source can be eliminated, and others may be duplicated if needed.

The consistency between the primary source and the secondary source is seen as the major challenge in the application of the PriSec model. In the starting point, it is necessary to produce the secondary source, with a potentially large amount of references to elements in the primary source. During the authoring process both the primary and secondary sources evolve. If a new element is added to the primary source it may be appropriate to update the secondary source accordingly. If an existing element is deleted from the primary source, and if this element is referenced from the secondary source, measures need to be taken. If the primary source is reorganized care must be taken to keep the references between the two sources intact. In section 4 we will discuss our solutions to these problems in the context of the LENO system.

4. The LENO solution

LENO is a tool for authoring of LEcture NOtes (Nørmark 2003b,a), primarily in the area of computer science, and with special empha-

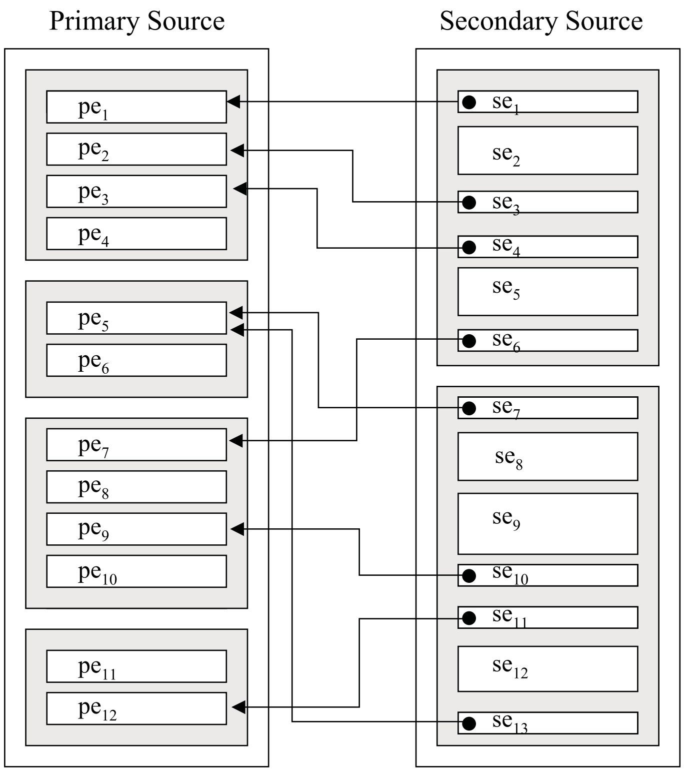

Figure 1: *The primary and secondary sources together with the references from the secondary source to the primary source.*

sis on the need in programming courses. In a LENO context, the concept of *lecture notes* covers the spectrum from a naked set of slides, via text/voice annotated slides, to more complete teaching materials in the style of a textbook. As a matter of LENO terminology, a material in the style of a textbook is represented as a sequence of *themes* at the authoring level, and presented as a sequence of *chapters* at the level of the end-user.

LENO is based on two XML languages both of which are defined by XML Document Type Definitions, DTDs (W3C 1998). We have used an authoring approach where the XML documents are actually written as expressions in the programming language Scheme (Kelsey et al. 1998). The connection between the XML level and the Scheme level is defined via the LAML system (Nørmark 2007d). LAML, and the mirroring of XML in Scheme, is well-documented in another paper (Nørmark 2005). The LENO author writes the documents in a text editor. In case Emacs is used, the author is supported by a number of editing commands and templates that can be accessed via the Emacs menu system. The authored text must be a sequence of Scheme expressions. When the Scheme expressions are evaluated the target format of the material is produced.

We use the term *programmatic authoring* for document authoring in the context of a programming language (Nørmark 2002). Programmatic authoring is a powerful approach. The main reason is that many instances of document complexities can be encapsulated in functional or procedural abstractions. In addition, many tedious authoring tasks can be dealt programmatically in the document source text. Due to "clutter" and involved syntax, programmatic authoring is not within reach in mainstream programming languages (such as languages with syntax derived from C).

To keep the discussion at a concrete and tangible level we have written a primary source of a few demo slides. In addition, we have written a secondary source of a slightly more comprehensive version of the demo material. The Scheme documents, which represent the primary and secondary sources, are shown in Appendix A. The generated materials, as well as the document sources, can be accessed from an accompanying web page (Nørmark 2007c).[4] It may be instructive to compare the sources in Appendix A with the generated HTML pages. Throughout the rest of this section we will illustrate our points with excerpts from the documents in Appendix A.

4.1 The LENO source forms

The primary source of a simple slide presentation is shown in Appendix A.1. As it appears, a LENO presentation is structured as a `leno-front-matters` clause with lots of attributes, followed by a number of `note-pages` surrounded by `begin-notes` and `end-notes`. Each `note-page` contains elements such as `point`, `items`, `concept-list`, etc. LENO supports 38 different kinds of immediate constituents of note pages. When the document is processed by the LENO tool, a large number of interlinked HTML and SVG pages are generated.

[4] Use the URL `http://www.cs.aau.dk/~normark/cc.html` to access the accompanying web resources of this paper.

The following shows a document fragment that represents a single note page in the primary source:

```
(note-page 'id "intro"
   (title (main-text "Introduction"))
   (point 'id "pt1"
    (main-text
     "This paper is about..."))
   (items 'id "it1"
    (item (main-text "Outline:")
     (items (item (main-text "Model"))
            (item (main-text "The LENO system"))
            (item (main-text "Conclusions"))))
   )
)
```

Note pages appear in the context of a given lecture, which has assigned a unique lecture id. A note page has an id of its own, which must be unique within a lecture. In addition, most note page subelements have ids that are unique within a single note page.

When the primary source is processed by LENO, the tool is able to automatically *derive* an initial, secondary source from the primary source. The secondary source represents the LENO themes. In the starting point each theme corresponds to a subsequence of slides, which are separated by `section-title` elements in the primary source. In the primary source document shown in Appendix A.1 there are three note pages with `section-title`s, and therefore there are three themes in the initial derived document, as shown in Appendix A.2.

The derivation of the secondary source is controlled by the front matters attribute `theme-source` in the primary source. (If the value of this attribute is `"new"` a new secondary source is derived and written to a new file in a template directory. The value `"delta"` is used for derivation of a delta source, see Section 4.3). The substance of the secondary document is made up by `leno-element` references to note page constituent elements. Almost all kinds of note page constituents (such as itemized lists, points, images, source program listings, concept lists, and syntax diagrams) can be addressed from the secondary source. In the comprehensive material we use itemized list (from the concise document) for overview and summary purposes. The following is a typical excerpt of the derived secondary source.

```
(theme 'id "intro-sec"
  (leno-element 'lecture-id "demo" 'page-id "intro-sec"
    'element-type "section-title" 'element-number "1" )

  (leno-element 'lecture-id "demo" 'page-id "intro"
    'element-type "title" 'element-number "1" )

  (leno-element 'lecture-id "demo" 'page-id "intro"
    'element-type "point" 'element-id "pt1" )

  (leno-element 'lecture-id "demo" 'page-id "leno-prim"
    'element-type "items" 'element-id "it1" )

  ; ...
)
```

The first `leno-element` shown above initiates a new chapter, and the second initiates a new section of the chapter. The third and fourth `leno-element` contribute to the contents of the new section. The `leno-elements` address elements from the primary source with use of `lecture-id` and `page-id` attributes. Some LENO elements use in addition the `element-id` and, redundantly, the `element-type` attributes. Others use the `element-number` attribute instead of `element-id` in the meaning of element number n of type `"title"`, for instance. (Notice in this context that only one `title` element is allowed in each note page. Therefore there is no need to have a unique id of the note page `title` element).

Using the automatically derived secondary source as a starting point, the author of the comprehensive material is assumed to add more contents. This is typically done by adding `theme-text` clauses to a theme, as siblings to the `leno-element` clauses, and with the purpose of adding "raw text" to the secondary source. In addition, the author may reorganize the `leno-elements` arbitrarily. Also, new themes may be added and existing themes may be deleted. The ordering of `leno-elements` within a `theme` clause may be changed, some `leno-elements` may be eliminated, and others may be duplicated. Elimination of a `leno-element` from the secondary source can in principle be done by deleting it. However, for the sake of consistency management (see below) elimination should be done by adding a `drop` attribute to the `leno-element` with the value `"true"`.

4.2 Cross-references

As part of `theme-text` clauses it is often relevant to make a *cross-reference* to another location in the comprehensive document. If we, for instance, want to include a reference to a presentation of a source program in the comprehensive document, we may aggregate an identification of the target in terms of the name of the theme source file name, the id of the `theme` clause in the secondary file, the ids of the primary source lecture and note page which contain the `source-program` element, the type of the target element (here a `"source-program"`), and the unique id of the element within the note page. The aggregation is represented as a Scheme expression[5], which generates an HTML `a` anchor element with a suitable `href` value. The following is an example of a reference clause in a LENO secondary source:

```
(ref "structures_themes-linked-sec" "structures"
     "list-fu" "source-program" "sp1")
```

It is error prone to type this clause directly, as text. Therefore, there are two alternative ways to create the `ref` form shown above. If a LENO theme is generated in a special author mode, the `ref` clause of each element can be reached from the browser. (The `ref` clauses are represented together, in an internal HTML page). As another possibility, if Emacs is used as the authoring tool, the editor can aggregate a `ref` clause from the similar target `leno-element` clause, using the `make-theme-ref` editor command. Subsequent use of another editor command, `insert-theme-ref`, inserts the aggregated `ref` clause.

The comprehensive document can be presented either as hypertext (HTML) or as text suitable for printing (PDF). In the hypertext version the cross-references as well as references to other parts of the LENO material are rendered as anchored links. In the printable version the cross-references are rendered as numbered entities, such as "Chapter 3", "Section 3.1", and "Figure 3.2". Other kinds of document cross-references are eliminated in the printable version.

4.3 Consistency issues

As already discussed in Section 3, the major challenge of dealing with both a primary and a secondary document source is the problem of keeping them mutually consistent during the life time of the material. As a typical scenario, the concise slide document, represented by the primary source, is updated through a multitude of modifications. Without specialized tool support it is very difficult to manage the corresponding updates of the comprehensive, textbook document. This is especially the case if the updating of the

[5] Alternatively we could have extended the LENO XML theme language to accommodate cross-references. However, the keyword style of XML parameter passing is more bulky than native positional parameter passing of Scheme functions. This is the reason behind the design of the `ref` form.

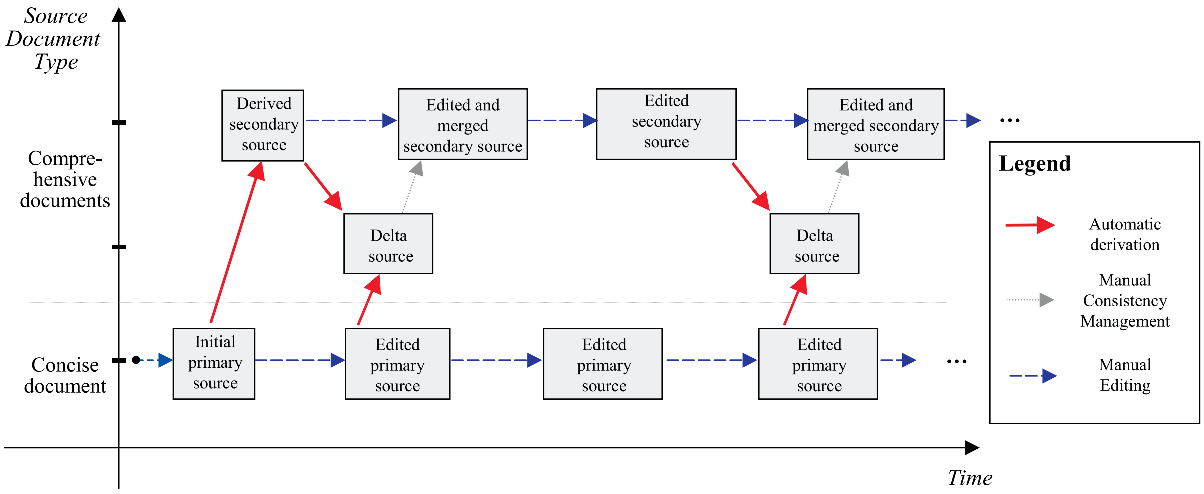

Figure 2: *An overview of the editing and derivation of primary and secondary sources.*

comprehensive document is done days or weeks after the updating of the concise document. Figure 2 shows a scenario of the temporal development of both sources. In the figure we distinguish between manual editing, manual consistency management, and automatic derivations (see the legend of the figure).

It is not realistic to go for an automatic updating of the secondary source, because only the author can sort out the implications of the changes to the concise document. The solution in LENO is to automatically derive a new secondary *delta source*, similar to the original derivation, in which the new elements in the concise document are clearly marked in the secondary source. Technically, this is done by identifying those elements of the primary source which are not represented somewhere as `leno-elements` in the secondary source. It should be noticed that `leno-elements` that are dropped in the secondary source will not be rediscovered, and not marked as new, relative to the primary source. (In contrast, `leno-elements` that are physically deleted from the secondary source will be marked as new elements in subsequent delta sources). The author of the secondary source can now manually merge the existing secondary source and the secondary delta source, and hereby effectuate the updating of the comprehensive document. Typically, this updating process also affects the existing `theme-text` elements, and it may call for adding new `theme-text` elements as well.

Reorganizations of the primary source do not harm the secondary source, as long as the identities of lectures, note pages, and note page subelements are not affected.[6] Deletions of note pages or note page constituents from the primary source should be done by marking these as deleted (lazy deletion, by use of `drop` attributes). Actual deletions of note pages, or constituents of note pages, may cause dangling references in the secondary source. A lazily deleted page or a lazily delete page constituent is still internally available, and it can be addressed from the secondary source.

It is sometimes necessary to modify the wording of the print version in relation to the wording of the hypertext version. Such variations are accomplished by use of conditionals (`if` or `cond`) of the Scheme programming language.

[6] Unfortunately, a reorganization that moves a primary element from one `note-page` to another does affect the identity of the element, as addressed from the secondary source. In the current version of the system it is therefore tedious to deal with such reorganizations.

Taken all together, the LENO solution to the consistency problem requires a great deal of work. We find, however, that this work is unavoidable due to the individual sequencing of the concise document and the comprehensive document. The LENO solution helps the author to keep an overview of the changes, and it makes it realistic to update the secondary source with use of fragments from the secondary delta source.

The granularity of reuse and consistency management has been chosen as that of the note page constituents of the primary source. Recall that a typical note page constituent is an itemized list, a point in terms of a single emphasized statement, an image, a source program, or an exercise. The arguments behind this choice are the following:

- The individual note page constituents make up the structural units, which reflects a natural and conceptual decomposition of the primary source.
- The individual note page constituents are identifiable and addressable, and therefore easy and attractive to reuse.
- An individual note page constituent is conceived as a unit, which typically should appear in its totality.

In principle, the granularity could have been either larger or smaller. In one extreme, the secondary document could be created as a copy of the primary document, extended with additional contents, and subsequently managed by general version control and diff tools in relation to the primary document. This would be an unstructured and low-level approach. In the other extreme, individual atoms (such as textual characters) of the primary document could be reused from the secondary document. It would, however, be difficult and convoluted to provide for addressing of such small units.

4.4 Post processing

We post process the print version of the HTML document in an interactive word processor (Microsoft Word). The most important concern in this process is page breaking (which is tricky and difficult to deal with in batch processing mode). In addition we add details such as page numbers. Finally, a PDF version of the Word document is generated using a PDF generator, such as the tool called PDF Creator. As an alternative to interactive use of an interactive word processor, we could have used XSL-FO (W3C 2000) for a description of the printable version. With this approach, an XSL batch processing tool (such as the Apache FOP processor) could be use to obtain a PDF document. It is a long term goal of the work with LENO to support PDF creation via XSL-FO.

5. Document Engineering in Scheme

In the work presented in this paper the Scheme programming language serves several different purposes:

1. Scheme is used as a text processing language.
2. Scheme is used as an image processing language.
3. Scheme is used as a host of several XML languages, most notable the primary and secondary source languages of LENO.
4. Scheme is used for abstraction of document details which we want to encapsulate and hide in the document sources.
5. Scheme is used as the implementation language of the tool that transforms from the primary and secondary sources to the target formats.

We will now discuss each of the purposes in turn.

Ad 1. Scheme as a text processing language. First and foremost, it may be asked if it is reasonable to write large amounts of text—with markup—as Scheme expressions. In such text, unbroken pieces of textual contents are represented as text string literals.

Based on my experience, it *is* reasonable and profitable to use Scheme for text processing purposes. The primary key to success seems to be good editor support for embedding of textual selections in Scheme forms, splitting of a string in substrings, opening of a new form with initial empty content, nesting of forms in other forms, and cleaning up of messy markup. Over the years we have developed very helpful Emacs editor commands for these purposes. These commands are applicable in LENO and in other contexts where Scheme is used for programmatic authoring.

Ad 2. Scheme as an image processing language. Teaching materials contain both text and images. It is disruptive for the author to create text and images in two different editors. Therefore we create most graphical illustrations in SVG, on a textual basis.[7] Like the LENO XML languages and XHTML, SVG is mirrored in Scheme. As a consequence, SVG images can be authored as Scheme expressions, and they can be inlined in the primary LENO source document. It is our experience that most of our illustrations are graph structures (in terms of nodes and edges) occasionally in some relaxed meaning. We have created a graph library extension of SVG (Nørmark 2007a), which we use for a teaching material about object-oriented programming in C# (Nørmark 2007b). In the target documents we most often transform SVG to a more accessible format, such as PNG (by use of the Apache Batik SVG toolkit).

Ad 3. Scheme as a host of XML. There exist several different ways to deal with XML in Scheme: SXML (Kiselyov 2002), WebIt (Bender), Scribe/Skribe (Serrano and Gallesio 2002; Serrano 2006), and LAML (Nørmark 2005). In LAML each XML element is represented by a named *mirror function*[8]. An expression with mirror functions generates an internal syntax tree. The mirror functions have exact knowledge of possible attributes and possible document constituents. As a consequence, a Scheme expression that activates several mirror functions validates the corresponding XML document (relative to the XML DTD) when the expression is evaluated. Relative and absolute links (URLs) that appear in the expression can also be checked.

It is very important to provide for a clean and smooth XML notation in Scheme. A source form in LENO/LAML is an expression that activates named mirror functions (see Appendix A for concrete examples). Mirror functions pass and interpret the actual parameters in a special and *liberal* way (see Section 3 of (Nørmark 2005) for details). As an alternative to the LAML approach, the document source could have been a list expression, which reveals the internal AST representation. Such a representation, either dominated by list functions or quasiquotations, is at a lower level of abstraction, and therefore it typically appears to be polluted with disturbing details.

In LAML/LENO a list of XML attributes, a list of element content items, or a mixed list of attributes and element content items is automatically and recursively spliced into its context. The following three XHTML mirror expressions are equivalent:

```
(a 'href "URL" "Anchor text")
(a "Anchor text" (list 'href "URL"))
(a (list "Anchor" (list "text"))
   (list (list 'href "URL")))
```

In real-life situations, `list` would stand for specific list-valued functions. The automatic and recursive splicing means that a part a document easily can be abstracted by a list-valued function. Without systematic splicing of lists it would be necessary for the document author to deal with list-flattening. This would disturb the cleanliness of the source.

At the most detailed level, the handling of white spacing matters a lot. In a LENO/LAML source document there are white space in between element content items, unless explicitly suppressed. With this decision, it is not necessary to have annoying prefix or suffix white spacing in literal text strings (such as `" text"` or `"text "`). A white space suppress value, bound to an underscore, is used if two content items next to each other should appear without spread. With this convention, the two expressions `(span "The end.")` and `(span "The" "end" _ ".")` are equivalent. The already mentioned editing commands insert most underscore symbols automatically if a string is splitted in the neighborhood of punctuation characters.

Ad 4. Scheme for ad hoc document abstraction. When working with textual markup, there is frequently a need for introducing ad hoc abstractions beyond the abstractions of the markup language. (In TeX and LaTeX this need can, to some degree, be remedied by TeX macros). When a document is authored in Scheme it is straightforward to write a number of auxiliary document abstraction functions in Scheme. When textual contents or XML attributes appear as parameters to such functions, it is our experience that the functions should accept parameters in the same liberal way as the mirror functions of the XML language. The higher-order function `xml-in-laml-abstraction` generates such a function. As an example, the definition

```
(define f
  (xml-in-laml-abstraction
    (lambda (contents attributes)
       (list contents attributes))))
```

binds `f` to a function with liberal parameter passing. `(f "t" 'x 5 "s" 'y "6")` is evaluated to the list `(("t" #t "s") (x "5" y "6"))` in which the first element represents the contents and the last element represents the attributes. The attributes are represented as a property list. `#t` represents a forced white space value.

[7] It is a pain to switch between text and image editing environments. It may also be felt as a pain to create graphical illustrations via textual commands or markup. It would be possible to go for an integrated environment like MS Word or Powerpoint. I have used MS Powerpoint extensively through several years. The lack of abstraction mechanisms and the missing support of *authoring without duplications*, see Section 2, makes Powerpoint a poor solution to the encountered document engineering challenges.

[8] Each named Scheme function is seen as a mirror of the corresponding XML element. Hence, the name *mirror function*.

Mixed parameter passing, as in the definition

```
(define g
  (xml-in-laml-positional-abstraction 2 1
    (lambda (x y contents attributes z)
       (list x y z contents attributes))))
```

binds g to a function in which the two first parameters and the last parameter are positional. The actual parameters in between these are interpreted in the liberal way. `(g 1 2 "t" _ "s" 'y "6" 'x 5 3)` is evaluated to `(1 2 3 ("t" #f "s") (y "6" x "5"))`. `#f` represents a white space suppress value.

Ad 5. Scheme as an implementation language. The LENO system is implemented in Scheme. Currently, the size of the LENO system is approximately 15.000 lines of code on top of the LAML libraries. The day-to-day needs of features in the teaching materials, through almost a decade, have led to a messy implementation, and to a complex web of Scheme source files behind the LENO tool. The flat name space of R5RS Scheme and the lack of modular encapsulation mechanisms have made it difficult to keep high standards in the underlying LENO implementation. If timed allowed, a complete rewrite of the system would be desirable.

6. Related Work

We are not aware of similar work, which directly shares the aims and the goals of the work described in this paper. In this section we will therefore describe related work, which positions the PriSec model and the LENO system with respect to similar issues in different contexts.

Slideshow (Findler and Flatt 2004; PLT) is similar to the primary LENO slide language, but without support of the secondary (textbook) language. Both Slideshow and LENO are based on functional programming, and both rely on Scheme. Slideshow provides a programmatic and functional approach to slide authoring, in particular with use of abstractions, and it is strong with respect to the handling of pictures. The Slideshow system comes with its own processor, which is embedded in the DrScheme environment. Slideshow produces slides via its own interpreter, and it is able to create a PDF file. In contrast, the distinctive characteristic of LENO is its orientation towards XML source formats (in the LENO XML languages) and HTML/SVG target formats.

The derivation of one source from another is known from the area of source-to-source transformations. Source-to-source transformation is used on programs and specifications for software engineering purposes (Baxter et al. 1994; Partsch and Steinbrüggen 1983). Source-to-source transformation is also used in the area of XML documents (Leinonen 2003; Krishnamurthi et al. 2000), as, for instance, supported via XSLT (W3C 1999). As a general rule, however, it is not the case that both the source and the transformed source are maintained. In other words, the document author does not actively edit both the source and the transformed source document. In comparison, our work on the PriSec model calls for maintenance of both the primary and the secondary sources.

The area of software documentation (Forward and Lethbridge 2002; Vestdam and Nørmark 2002) represents an example of two levels of source documents (source programs and internal program documentation, for instance) which are mutually dependent, and both of which are simultaneous maintained. What is worth noticing is that the program documentation is not derived from the source program, or vice versa. The program and its documentation are usually written in two different documents. The main challenge is to keep the documentation up-to-date when the program is modified. In that respect, we deal with the exact same challenges in LENO. However, our starting point is different, because one of the documents (the comprehensive document) is initially derived automatically from the other document (the concise document).

As already noticed earlier in this paper, the PriSec model and the LENO approach have much in common with the ideas of *single sourcing* (Rockley 2001; Fraley 2003; Kostur 2000). Single sourcing is defined as the use of a “single document source to generate multiple types of document outputs” and “workflows for creating multiple outputs from a document or database source” (STC). In the strict sense, PriSec is not a single sourcing model, but of a *double sourcing* approach. The reason is that two different sources are used to control the independent sequencing of elements in the two documents. In the starting point, however, the source of the concise document serves as “the single source”, used to derive the *initial source* of the comprehensive document.

Continuing our comparison with software documentation, it is interesting to notice that the program documentation approach known as Literate Programming (Knuth 1984) can be seen as a single sourcing program documentation approach, in which both the program and the documentation are authored in an extended, aggregated language. Similarly, API documentation in the style of Javadoc (Friendly 1995), Doxygen (van Heesch 2004), and SchemeDoc (Nørmark 2004) can be seen as single sourcing, because this kind of documentation is represented as specialized comments in the program source text.

The use of double sourcing (in terms of a primary and a secondary, derived source) can be seen as a reminiscence of the preferred authoring process. This particular authoring process first produces a slide material as the basis for an oral presentation, and later a written account in the style of a textbook. The reverse authoring process is also possible, and in fact quite well-known from the authoring of scientific papers and subsequent production of slides for oral presentation at conferences or workshops. We are, however, not aware of any attempt to derive slides (semi)automatically from the full paper.

7. Conclusions

As the main contribution of this paper we have developed a document model, called the PriSec model, which is based on a primary source and a secondary source. The primary source, which represents a concise document, can be used to derive an initial version of the secondary source of the comprehensive document. Once derived, the secondary source is assumed to be elaborated in various ways, leading to a situation where both the primary source and the secondary source need to be kept mutually up-to-date.

The PriSec model is implemented in the LENO system, and supported by commands in the Emacs text editor. Most interesting in the context of this paper, we have developed an approach where a secondary delta source can be re-derived from the primary source and the existing secondary source. This secondary delta source can manually be merged with the existing version of the secondary source, with the purpose of updating it relative to changes of the primary source.

The LENO system has been used since 1999 (by the author and a colleague) to produce collections of teaching materials for different programming-related computer science courses. The concise and comprehensive documents for these courses are available via (Nørmark 2007c).

The authoring process of a comprehensive material is interesting in its own right. As the first step, the essential key elements of the material are formulated, including elements such as important concepts, main points, itemized overviews, etc. In a LENO context, these key elements constitute the concise document—the slide presentation. As the second step, the key elements are plumbed together by adding additional contents, such as intro, outro, and explanations. For the materials mentioned above, we have found this two-step authoring process both interesting and rewarding. The process encourages the author to concentrate on the essentials in the first phase. The second phase is concentrated on consolidation and additional explanations, in order to make the material self-contained and approachable without an accompanying oral presentation. The comprehensive document, as delivered by the tool, is typically affected by the process through which is has been created. The itemized lists, which dominate most slide presentations, can either be used as introductory overviews or for summary purposes.

Our concrete experience with the PriSec model is gained within the area of computer science teaching materials. We hypothesize, however, that the model can be used in other situations where concise and comprehensive documents with massive overlaps are needed. We also hypothesize that the two-step authoring process, as discussed above, can be beneficial in these situations.

LENO is free software, bundled with LAML, and available from the LAML home page (Nørmark 2007d). The LENO home page (Nørmark 2003a) holds all available LENO resources, including a gentle introduction, examples, and a tutorial.

Acknowledgement

I wish to thank the anonymous reviewers for valuable input to the final version of the paper.

References

Ira D. Baxter, Christopher Pidgeon, and Michael Mehlich. DMS: Program transformations for practical scalable software evolution. In *The proceedings of the 26th international conference on software engineering*. IEEE Computer Society, 1994.

Jim Bender. WebIt! `http://celtic.benderweb.net/webit/`.

Robert Bruce Findler and Matthew Flatt. Slideshow: Functional presentations. In *Proceedings of the Ninth ACM SIGPLAN International Conference on Functional Programming*, pages 224–235. ACM Press, September 2004.

Andrew Forward and Timothy C. Lethbridge. The relevance of software documentation, tools and technologies: a survey. In *DocEng '02: Proceedings of the 2002 ACM symposium on Document engineering*, pages 26–33. ACM Press, 2002. ISBN 1-58113-594-7. URL `http://doi.acm.org/-10.1145/585058.585065`.

Liz Fraley. Beyond theory: Making single-sourcing actually work. In *Proceedings of the 21st annual international conference on Documentation*, pages 52–59. ACM Press, 2003.

Lisa Friendly. The design of distributed hyperlinked programming documentation. In Sylvain Frass, Franca Garzotto, Toms Isakowitz, Jocelyne Nanard, and Marc Nanard, editors, *Proceedings of the International Workshop on Hypermedia Design (IWHD'95), Montpellier, France*, 1995.

Richard Kelsey, William Clinger, and Jonathan Rees. Revised^5 report on the algorithmic language Scheme. *Higher-Order and Symbolic Computation*, 11(1):7–105, August 1998. URL `http://www.schemers.org/Documents/-Standards/R5RS/r5rs.pdf`.

Oleg Kiselyov. SXML, August 2002. URL `http://okmij.org/-ftp/Scheme/SXML.html`.

Donald E. Knuth. Literate programming. *The Computer Journal*, May 1984.

Josef Kolbitsch and Hermann Maurer. Transclusion in an HTML-based environment. *Journal of Computing and Information Technology*, 14(2):161–174, 2006.

Pamela Kostur. Information modeling for single sourcing. In *18th Annual Conference on Computer Documentation - IPCC, SIGDOC 2000*, pages 333–342. ACM and IEEE, 2000.

Shriram Krishnamurthi, Kathryn E. Gray, and Paul T. Graunke. Transformation-by-example for XML. In E. Pontelli and V. Santos Costa, editors, *PADL 2000*, LNCS 1753, pages 249–262. Springer Verlag, 2000.

Paula Leinonen. Automating XML document structure transformations. In *DocEng'03*, pages 26–28. ACM Press, November 2003.

Theodor Holm Nelson. The heart of connection: Hypermedia unified by transclusion. *Communication of the ACM*, 38(8):31–33, August 1995.

Kurt Nørmark. Programmatic WWW authoring using Scheme and LAML. In *The proceedings of the Eleventh International World Wide Web Conference - The web engineering track*, May 2002. URL `http://www2002.org/CDROM/alternate/296/`.

Kurt Nørmark. Web programming in Scheme with LAML. *Journal of Functional Programming*, 15(1):53–65, January 2005. URL `http://www.cs.aau.dk/~normark/laml/-papers/web-programming-laml.pdf`.

Kurt Nørmark. The LENO home page, 2003a. URL `http:-//www.cs.aau.dk/~normark/leno/`.

Kurt Nørmark. The why and wherefore of the LENO system, August 2003b. URL `http://www.cs.aau.dk/~normark/-laml/papers/leno/why-and-wherefore.pdf`.

Kurt Nørmark. Functional programming in Scheme - with web programming examples, 2003c. URL `http:-//www.cs.aau.dk/~normark/prog3-03/html/notes/-theme-index.html`.

Kurt Nørmark. Scheme program documentation tools. In Olin Shivers and Oscar Waddell, editors, *Proceedings of the Fifth Workshop on Scheme and Functional Programming*, pages 1–11. Department of Computer Science, Indiana University, September 2004. URL `http://www.cs.aau.dk/~normark/laml/-papers/documentation-tools.pdf`. Technical Report 600.

Kurt Nørmark. A graph library extension of SVG. In *Proceedings of SVG Open 2007, Tokyo, Japan*, September 2007a. URL `http://www.cs.aau.dk/~normark/laml/-papers/svg-open-2007/paper.html`.

Kurt Nørmark. Object-oriented programming in C# - for C programmers, 2007b. URL `http://www.cs.aau.-dk/~normark/oop-07/html/notes/theme-index.html`.

Kurt Nørmark. Web resources of the current paper, August 2007c. URL `http://www.cs.aau.dk/~normark/cc.html`.

Kurt Nørmark. The LAML home page, 2007d. URL `http://-www.cs.aau.dk/~normark/laml/`.

H. Partsch and R. Steinbrüggen. Program transformation systems. *ACM Computing Surveys*, 15(3):199–236, 1983.

PLT. PLT slideshow. URL `http://www.plt-scheme.org/-software/slideshow/`.

Ann Rockley. The impact of single sourcing technology. *Technical Communication*, 48(2):189–193, 2001.

Manuel Serrano. Skribe, 2006. URL `http://www-sop.inria.-fr/mimosa/fp/Skribe/`.

Manuel Serrano and Erick Gallesio. This is Scribe! In *Workshop on Scheme and Functional Programming (2002)*, October 2002. URL `http://www-sop.inria.fr/mimosa/Manuel.-Serrano/scribe/doc/scribe.html`.

STC. URL `http://www.stcsig.org/ss/`.

Dimitri van Heesch. Doxygen, 2004. URL `http://www.-doxygen.org`.

Thomas Vestdam and Kurt Nørmark. Aspects of internal program documentation - an elucidative perspective. In *10th International Workshop on Program Comprehension*. IEEE, June 2002. URL `http://dopu.cs.aau.dk/publications/-aspects-paper.pdf`.

W3C. Scalable vector graphics (svg) 1.1 specification, January 2003. URL `http://www.w3.org/TR/SVG11/`.

W3C. Extensible markup language (XML) 1.0, February 1998. URL `http://www.w3.org/TR/REC-xml`. `http://www.w3.-org/TR/REC-xml`.

W3C. Extensible stylesheet language (XSL) version 1.0. Technical report, W3C, November 2000. URL `http://www.w3.org/-TR/xsl/`.

W3C. XSL transformations (XSLT) version 1.0. W3C Recommendation, November 1999. URL `http://www.w3.org/TR/xslt`.

A. An example

In this appendix we show the primary and secondary sources of a simple LENO demo material. The Scheme sources, corresponding XML sources, and the resulting HTML documents are available as web resources of this paper (Nørmark 2007c). The web resources are located at `http://www.cs.aau.dk/~normark/cc.html`.

A.1 The primary source

The primary source of a simple slide presentation is shown first. The list of `leno-front-matters` attributes is slightly abbreviated.

```
(load (string-append laml-dir "laml.scm"))
(laml-style "xml-in-laml/lecture-notes/lecture-notes")

(leno-front-matters
  (front-title "Concise and Comprehensive Documents")
  (front-author "Kurt Normark")
  (front-affiliation "Aalborg University")
  (front-abstract
    "An ultra brief exposition of the relations between
     concise and comprehensive documents")

  'slide-view "true"  'annotated-slide-view "false"
  'aggregated-view "false" 'theme-view "true"
  'primary-view "slide-view" 'scheme-prefix "pre-notes.scm"
  'scheme-suffix "post-notes.scm"
  'css-prestylesheet "large-size"  'css-stylesheet "original"
  'theme-auto-process "false"  'theme-source "new"

  ; Some attributes have been elided in this version
)

(begin-notes)

(note-page 'id "intro-sec" (section-title "Introduction"))

(note-page 'id "intro"
  (title (main-text "Introduction"))
  (point 'id "pt1"
   (main-text
    "This paper is about derivation of a comprehensive document from a concise document"))
  (items 'id "it1"
   (item (main-text "Outline:")
    (items (item (main-text "Model"))
           (item (main-text "The LENO system"))
           (item (main-text "Conclusions"))))))

(note-page 'id "model-sec" (section-title "Model"))

(note-page 'id "primsec"
  (title (main-text "The Prisec model"))
  (concept-list 'id "con1" (concept 'concept-name "Prisec"
     (main-text "The Prisec model is a model with a primary and secondary source
                 of the concise and comprehensive documents resp." )))
  (items 'id "it1"
   (item (main-text "Issues:")
    (items (item (main-text "Derivation of the secondary source from the primary source"))
           (item (main-text "Consistency between the sources "))
           (item (main-text "An alternative to a single source model")))))
  (point 'id "pt1" (main-text "LENO implements the Primsec model")))


(note-page 'id "leno-sec" (section-title "LENO"))

(note-page 'id "leno-prim"
  (title (main-text "The primary LENO source"))
  (point 'id "pt1" (main-text "LENO is an XML-based presentation tool in the LAML family"))
  (cross-references 'id "cr1"
   (internet-reference 'href "http://www.cs.aau.dk/~normark/laml/" (main-text "LAML")))
  (items 'id "it1"
   (item (main-text "LENO primary source characteristics:")
    (items (item (main-text "Slide view, annotated slide view, and aggregated view"))
           (item (main-text "Aims at elimination of duplicated source elements"))
           (item (main-text "Structured as sectioned lectures and slide pages")))))
  (point 'id "pt2" (main-text "A secondary source can be derived from the primary source ")))
```

```
(note-page 'id "leno-seco"
  (title (main-text "The secondary LENO source"))
  (point 'id "pt1" (main-text "The secondary source contains lots of references to primary source elements"))
  (items 'id "it1"
   (item (main-text "LENO secondary source characteristics:")
    (items (item (main-text "Theme-text elements add to the comprehensiveness"))
           (item (main-text "Presented as traditional paper material"))
           (item (main-text "Structured as chapters and sections")))))
  (point 'id "pt2" (main-text "A PDF version can easily be provided for"))
)

(end-notes)
```

A.2 The derived secondary source

Below we show the secondary document source, as derived automatically from the primary source. As discussed in Section 4, it is intended that the author adds textual contents to this source in terms of `theme-text` elements. A version with added `theme-text` elements can be seen together with the accompanying web resources (Nørmark 2007c).

```
(load (string-append laml-dir "laml.scm"))

(laml-style "xml-in-laml/lecture-notes-themes/lecture-notes-themes")

(leno-themes-front-matters
  'scheme-prefix "pre-notes.scm"
  'scheme-suffix "post-notes.scm"
)

(begin-themes)

(theme 'id "intro-sec"
  (leno-element 'lecture-id "demo" 'page-id "intro-sec" 'element-type "section-title" 'element-number "1" )

  (leno-element 'lecture-id "demo" 'page-id "intro" 'element-type "title" 'element-number "1" )
  (leno-element 'lecture-id "demo" 'page-id "intro" 'element-type "point" 'element-id "pt1" )
  (leno-element 'lecture-id "demo" 'page-id "intro" 'element-type "items" 'element-id "it1" )
)

(theme 'id "model-sec"
  (leno-element 'lecture-id "demo" 'page-id "model-sec" 'element-type "section-title" 'element-number "1" )

  (leno-element 'lecture-id "demo" 'page-id "primsec" 'element-type "title" 'element-number "1" )
  (leno-element 'lecture-id "demo" 'page-id "primsec" 'element-type "concept-list" 'element-id "con1" )
  (leno-element 'lecture-id "demo" 'page-id "primsec" 'element-type "items" 'element-id "it1" )
  (leno-element 'lecture-id "demo" 'page-id "primsec" 'element-type "point" 'element-id "pt1" )
)

(theme 'id "leno-sec"
  (leno-element 'lecture-id "demo" 'page-id "leno-sec" 'element-type "section-title" 'element-number "1" )

  (leno-element 'lecture-id "demo" 'page-id "leno-prim" 'element-type "title" 'element-number "1" )
  (leno-element 'lecture-id "demo" 'page-id "leno-prim" 'element-type "point" 'element-id "pt1" )
  (leno-element 'lecture-id "demo" 'page-id "leno-prim" 'element-type "cross-references" 'element-id "cr1" )
  (leno-element 'lecture-id "demo" 'page-id "leno-prim" 'element-type "items" 'element-id "it1" )
  (leno-element 'lecture-id "demo" 'page-id "leno-prim" 'element-type "point" 'element-id "pt2" )

  (leno-element 'lecture-id "demo" 'page-id "leno-seco" 'element-type "title" 'element-number "1" )
  (leno-element 'lecture-id "demo" 'page-id "leno-seco" 'element-type "point" 'element-id "pt1" )
  (leno-element 'lecture-id "demo" 'page-id "leno-seco" 'element-type "items" 'element-id "it1" )
  (leno-element 'lecture-id "demo" 'page-id "leno-seco" 'element-type "point" 'element-id "pt2" )
)

(end-themes)
```