Snippets and Component-Based Authoring Tools for Reusing and Connecting Documents

Laurent Kirsch, Jean Botev, Steffen Rothkugel
Faculty of Science, Technology and Communication
University of Luxembourg
Luxembourg
{laurent.kirsch, jean.botev, steffen.rothkugel}@uni.lu

Journal of Digital
Information Management

ABSTRACT: *This paper presents the Snippet System, a new operating system environment that aims at providing enhanced document management facilities. For this, the proposed system utilizes a novel document model based on finer-grained entities, so-called Snippets. These support Relations, which capture the context of individual document excerpts. Snippets furthermore enable a flexible reuse of documents, i.e., user-defined excerpts can be included in several other documents with only selected properties remaining synchronized between different instances. Moreover, dedicated mechanisms allow for the efficient retrieval of these instances and thus support the user in keeping track of reused excerpts and synced properties.*

The Snippet System relies on a component-based authoring tool architecture that enables different authoring tools to collaborate for handling documents. This allows for combining the contents created by various tools in a single document, independently of the type of content and tools involved, thus making possible the reuse of documents and the definition of Relations across tool boundaries.

Categories and Subject Descriptors:
I.7.1 [Document and Text Editing]; Document management; I.7.5 [Document Capture] Document analysis

General Terms:
Document Processing, Text Analysis, Authoring Tools

Keywords: Applied computing, Document management and text processing, Document management, Human-centered computing, Authoring Tools

Received: 22 July 2012, Revised 26 September 2012, Accepted 29 September 2012

1. Introduction

Creating high quality documents is a labor-intensive and time-consuming process. Often it is desirable to reuse slightly adapted excerpts of existing documents for creating a new document. However, today's operating sys tems do not provide adequate support for this kind of reuse, especially when different applications manage the involved excerpts. For example, it is usually not possible to insert an editable, synchronized instance of a vector graphics object into a text document. A wide-spread solution consists in including a non-editable, exported version. The main drawback here is the lack of a connection between the original object and its exported version, which renders a synchronization between them impossible.

State-of-the-art operating systems do not provide native support for capturing any kind of connections between document excerpts, especially when they are managed by different authoring tools [4] [5]. For example, it normally is not possible to define a contextual link between a lecture slide, a task from an exercise sheet and an exam question which all treat the same problem. Such connections between individual excerpts are, however, critical to both authors and readers when dealing with a document [3]. While creating or adapting documents, authors can utilize these links to conveniently access the context of a specific excerpt also across different applications. These connections serve the author as cognitive artifacts [18] or augmentation [14], which allow for being transported back to the original creation process of a given excerpt, thereby alleviating some of the problems users experience when returning to long-lasting tasks [7]. Readers, in turn, tend to perceive not only linearly, but mostly laterally [21] by using multiple, complementary sources. Such links - predefined by the author - allow the reader to better grasp the context of specific document excerpts, preventing also possible wrong associations.

The aim of the presented *Snippet System* is to provide new facilities for the creation and management of documents and in particular to overcome said limitations of existing systems [12]. Most importantly, it allows for the fine-granular reuse of documents, i.e., user-defined document excerpts can be included in several other documents with different instances being completely or partially synchronized among each other. In the latter case synchronization is limited to selected properties. For example, when creating a new presentation, existing slides can be reused, and leveraging partial synchronization, it can be ensured that only specific attributes remain in-sync among the new and the original instance of the slides.

However, users work with many documents and typically cannot keep track of all reused excerpts and synchronized properties themselves. Therefore, dedicated mechanisms within the Snippet System allow for checking which excerpts of a document have also been included in other documents. These can furthermore be retrieved for assessing the impact of intended modifications. Moreover, for enabling easier access to the context of a document, the Snippet System supports *Relations*, i.e., collections of related document excerpts, which can be defined and browsed at any moment while editing or reading a document.

The two main architectural components of the Snippet System - which set it apart from existing systems - are a component-based architecture for authoring tools and a novel model for the representation of documents. The architecture specifically ensures that all tools support Relations by allowing excerpts of their documents to be connected with other related excerpts that can also be managed by different tools. Furthermore, it also guarantees that every tool is able to insert document excerpts managed by other tools into its own documents. This allows for combining the reduced feature sets of several tools to create more complex compound documents, i.e., documents consisting of many smaller documents that are handled by the same or separate tools.

For the representation of such documents, an appropriate model which has been specifically designed for the requirements of the Snippet System, is utilized. The essence of this model is that each document is mapped onto a persistent graph of interrelated entities, so-called *Snippets*. An individual Snippet within the graph can represent any structural element of the corresponding document: an entire table or a single cell in case of a spreadsheet, a node in case of a mind map, or a class, method or property in case of a UML class diagram.

This finer granularity of Snippets immediately raises scalability concerns. However, by examining the core operations' complexity, it is shown that the approach is scalable. Actually, the fine-granular nature of Snippets offers some advantages. Snippets can, for example, be shared in several document graphs, thereby synchronizing the data in these Snippets - and only these data - between the involved documents.

While the Snippet System's concepts extend even into the application and user interface layers [10], this article focuses on the underlying document model and component-based tool architecture. The different Snippet types and their relationships are presented in the following section. Section 3 then introduces complete and partial synchronization of documents or excerpts and explains how different instances can be retrieved efficiently. The concept of Relations is detailed in Section 4. The design of the component-based tool architecture is presented in Section 5, whereas its implementation is described in Section 6. Finally, before summarizing the contributions, indicating limitations and future work in Section 8, related approaches are covered in Section 7 with a particular focus on XML documents.

2. The Snippet Model

The UML diagram in Figure 1 depicts the Snippet model with its various Snippet types and their relationships. Every document's Snippet graph contains a *Document-Snippet*. It constitutes the root of the graph and stores meta-information, such as the document's name or authoring information. Moreover, Document-Snippets are also used for representing *Relations*, i.e., collections of related document excerpts, which are discussed in Section 4.

Element-Snippets represent structural elements of a document. In a presentation example, an Element-Snippet may figure as an entire slide or an individual element on a slide, such as a text box, a chart or a single bullet. The mapping of a document's content onto Element-Snippets is created and adapted automatically by the authoring tool in response to the user's actions. Element-Snippets are also used to insert a document into other documents and therefore enable the Snippet model to natively support the representation of compound documents.

While an Element-Snippet represents a structural element of a document, up to two *Data-Snippets* store - if existing - the actual data, whose format is arbitrary and defined by the managing tool. For supporting partial synchronization, the data is divided into two parts: a synchronized part and a non-synchronized part, both stored in dedicated Data-Snippets. The Element-Snippet itself establishes the link between these Data-Snippets. Partial synchronization is discussed in detail in Section 3.2.

Tools typically require the content of their documents to be presented in a particular order. The *head* and *next* relationships therefore allow for the Element-Snippets of a document to be organized as an ordered tree.

Finally, the *linked from* and *source document* relationships are intended for the efficient retrieval of all synchronized document instances, discussed further in Section 3.3.

The most important concepts are illustrated by the concrete example in Figure 2. For better clarity, elements of the Snippet model that are not relevant to the following discussion, such as Data-Snippets, are not depicted. The given example involves two compound documents: a text and a vector graphics object. Both documents contain another text document with formulas.

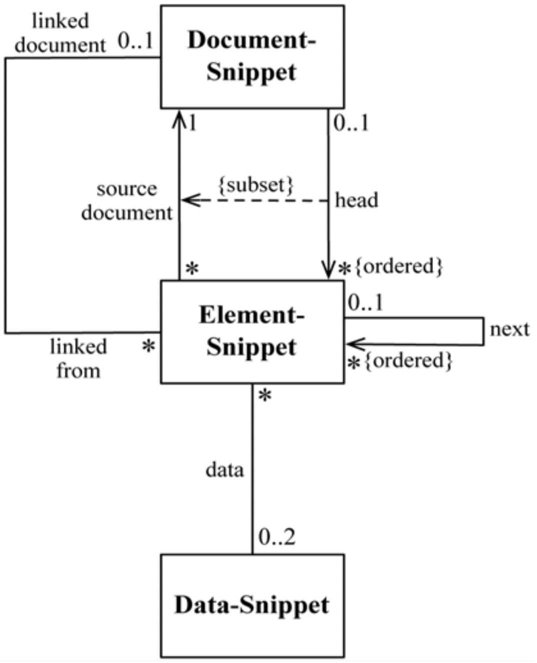

Figure 1. The Snippet Model - Snippet Types and Relationships

Each shape of the vector graphics object is considered to be a structural element and is therefore represented by an Element-Snippet. In case of the two text documents, the entire content is represented by a single Element-Snippet. In both the representation of the *Vector Graphics Object* and the *Text*, there is also an Element-Snippet which comprises the document containing the formulas. These two intermediate Element-Snippets and their linked Data-Snippets are used to store information on the exact position and size of the formulas within the *Vector Graphics Object* and the *Text*.

Figure 2 also shows that a reused excerpt, here the formulas, is always represented like an individual document, i.e., by a Snippet graph with a Document-Snippet at its root. As a consequence of a reuse decision, the Snippet structure of a document may need to be adapted. The complexity of such a rearrangement operation is *O(n)*, where *n* is the number of Element-Snippets that need to be added to the new Snippet graph representing the reused excerpt.

3. Synchronization and Retrieval

The different mechanisms for document instance synchronization are an important aspect of the Snippet model. Furthermore, the efficient retrieval of such instances is considered by a series of special relationships. As reused document excerpts are equal to individual documents in terms of representation, the following discussion of the various approaches and their peculiarities also covers the synchronization and retrieval of such excerpts' instances.

3.1 Complete Synchronization

Complete synchronization of document instances ensures that any modification to the document propagates to all involved instances which then remain entirely in-sync.

The two instances of the formulas in Figure 2 are completely synchronized. Both the *Vector Graphics Object* and the *Text* contain an Element-Snippet referencing the corresponding Document-Snippet, which includes the formulas in the two documents. Furthermore, as exactly the same Snippet graph is referenced in both cases, modifications to the formulas made either from within the *Vector Graphics Object* or the *Text* automatically propagate to the other document.

In general, when a new, completely-synchronized instance of a document A shall be inserted into a document B, only a new Element-Snippet referencing A's Document-Snippet needs to be created and inserted into B's representation. This operation is of complexity $O(1)$.

3.2 Partial Synchronization

If only the modifications to selected properties of the document shall propagate to all involved instances, whereas other changes shall affect solely the edited instance, partial synchronization can be utilized.

Figure 3 illustrates how instances can be partially synchronized. Again elements of the Snippet model that are not relevant for the discussion, such as some of the Element- and Data-Snippets, are not depicted. The documents are the same as for Figure 2, but in the current example the two instances of the formulas are partially synchronized. The font attributes evolve independently for each instance, whereas all other properties remain in-sync.

When a partially-synchronized instance is created, the document's Snippet graph is partly copied: the Document-Snippet, all Element-Snippets and all non-synchronized data parts are duplicated. However, each of the copied Element-Snippets references the same synchronized data part as its original counterpart. This ensures that the non-synchronized data parts - in the given example the font attributes - evolve independently for the newly created instance, whereas the synchronized data parts remain in-sync with other instances. Creating a partial copy is of complexity $O(n)$, where n is the number of Element-Snippets used in the document's Snippet graph.

3.3 Retrieval of Synchronized Instances

The Snippet System offers a high degree of flexibility with respect to synchronizing documents or excerpts. From the user's point of view, keeping track of synchronized instances therefore requires system support.

The Snippet model allows for efficiently retrieving all

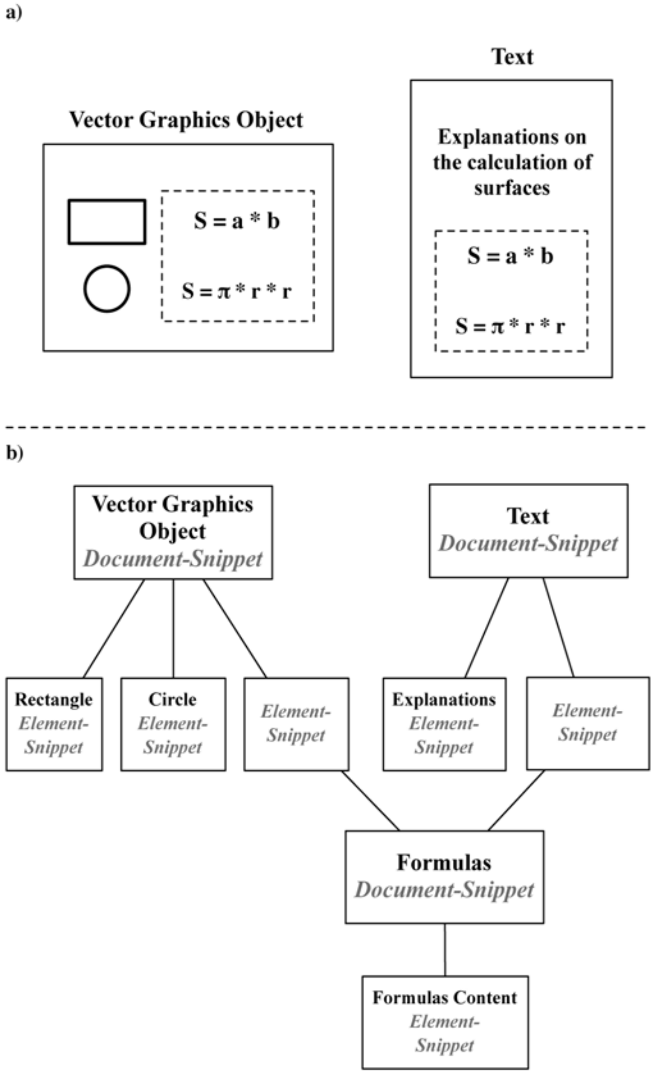
a)
Text
Vector Graphics Object
Explanations on the calculation of surfaces
S = a * b
S = π * r * r
S = a * b
S = π * r * r
b)
Vector Graphics Object Document-Snippet
Text Document-Snippet
Rectangle Element-Snippet
Circle Element-Snippet
Element-Snippet
Explanations Element-Snippet
Element-Snippet
Formulas Document-Snippet
Formulas Content Element-Snippet

Figure 2. Two Exemplary Compound Documents (a) and Their Snippet Graphs (b)

instances of a given document (or reused excerpt) via dedicated mechanisms, which are discussed based on Figure 4. The given example contains the documents of Figure 3 and an additional mind map with a completely-synchronized instance of the vector graphics object. There are furthermore two partially-synchronized versions of the formulas. The *Bold Formulas* are included in a vector graphics object, which itself is part of a mind map. The *Italic Formulas*, in turn, have been inserted into a text document.

Generally, there can be several partially-synchronized instances of a document, each with its own completely-synchronized instances comprised in other documents. Therefore, retrieving all synchronized instances is equivalent to accessing the Document-Snippets of partially-synchronized instances and then searching all of the documents which include them in their own representation.

The first step is efficient, because a Relation - managed by the Snippet System - maintains a link between the Document-Snippets of partially-synchronized instances. The different Relation types are detailed in Section 4.

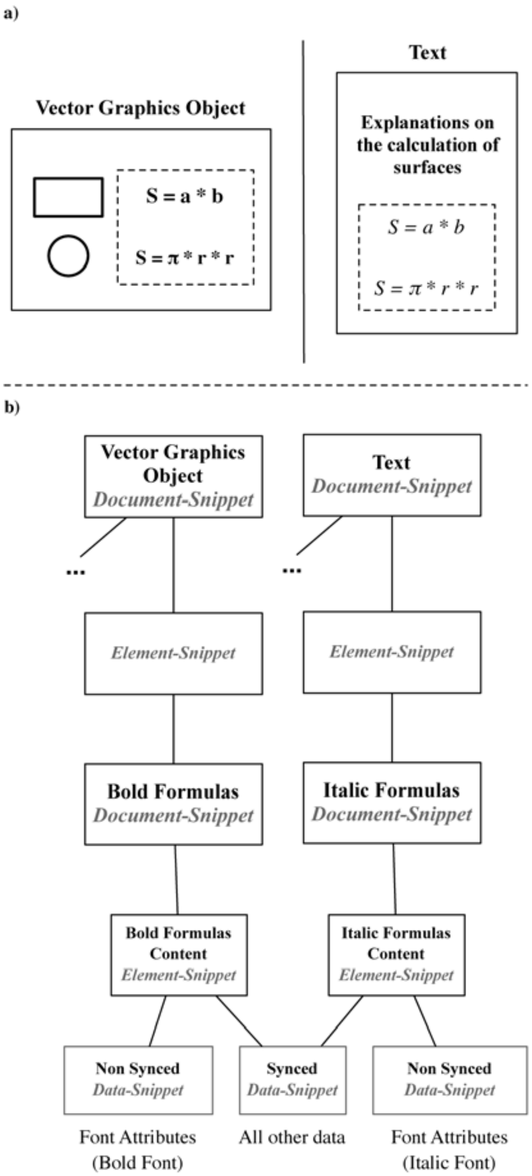
a)
Text
Vector Graphics Object
Explanations on the calculation of surfaces
S = a * b
S = π * r * r
S = a * b
S = π * r * r
b)
Vector Graphics Object Document-Snippet
Text Document-Snippet
...
...
Element-Snippet
Element-Snippet
Bold Formulas Document-Snippet
Italic Formulas Document-Snippet
Bold Formulas Content Element-Snippet
Italic Formulas Content Element-Snippet
Non Synced Data-Snippet
Synced Data-Snippet
Non Synced Data-Snippet
Font Attributes (Bold Font)
All other data
Font Attributes (Italic Font)

Figure 3. Two Compound Documents with Partially-Synchronized Formulas (a) and Their Snippet Graphs (b)

The second step, in turn, is efficient because the *source document linked from* relationships are integrated into the Snippet model (Figure 2). The *source document* relationship links every single Element-Snippet to its document's Document-Snippet, whereas the *linked from* relationship links a Document-Snippet to other Element-Snippets that reference it. When completely-synchronized instances need to be retrieved, for example for the *Bold Formulas*, these two relationships are navigated recursively: Navigating them once yields the Document-Snippets of documents in which the *Bold Formulas* have been linked directly. The recursive step ensures that the documents in which they have been linked indirectly via other intermediate documents - here the mind map - are also retrieved.

The complexity of retrieving all synchronized instances of a document is $O(p*d*n)$, where p is the number of partially-synchronized instances, d is the depth of the

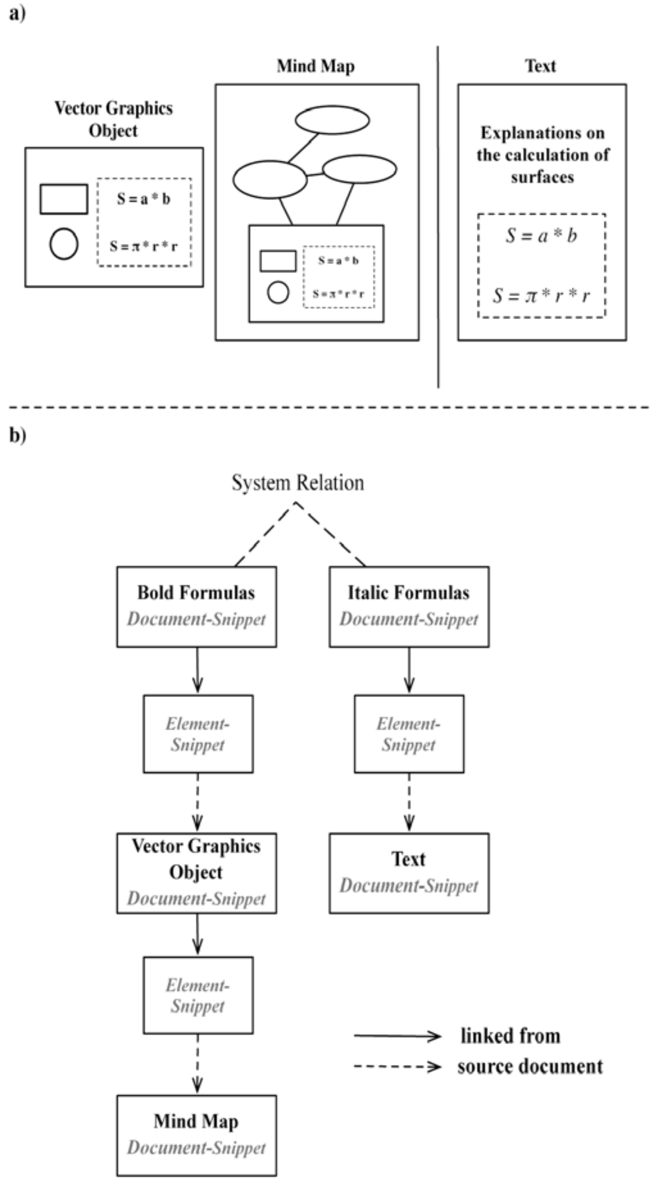

Figure 4. Synchronized Instances of Formulas (a) and Interrelationships in the Snippet Model (b)

document hierarchy and n is the number of documents retrieved at each recursive step.

4. Relations

Relations are system- or user-defined collections of documents or excerpts. System-defined Relations are not editable by the user and maintain a link between partially-synchronized instances of a document or reused excerpt, which, as described in Section 3.3, is necessary for their efficient retrieval.

User-defined Relations maintain a contextual link between selected document excerpts. In Figure 5a parts of lecture materials, all treating the same problem, are interconnected. An example covered during the lecture, a related exercise and a question of an exam are grouped via the custom *Same Problem* Relation. It allows the teacher to easily access related lecture materials while editing, for instance, the lecture slide, and assess whether related materials need to be adapted as well.

Figure 5b depicts the given Relation's representation by Snippets. A Document-Snippet constitutes the root of the corresponding graph. This ensures that documents and Relations are equal in terms of Snippet representations, i.e., the same mechanisms of the Snippet model can be utilized for structuring both documents and Relations. In the current example, there are three Element-Snippets, each referencing the Snippet graph of a related excerpt. As for their reuse, excerpts need to be represented like individual documents before they can be added to a Relation.

The Data-Snippets attached to the intermediate Element-Snippets are not depicted here for better clarity. They store information on how each of the involved excerpts shall be presented to the user when the *Same Problem* Relation is accessed. The way a specific excerpt is displayed can vary for different Relations, that is contexts, it is part of.

5. Architecture

While the Snippet model allows for efficiently representing compound documents, component-based authoring tools that collaborate with each other are required for creating and managing these documents. This section details how the involved communication is supported by the tools' architecture. Furthermore, it discusses how authoring tools and their internal components are instantiated at run-time.

For efficiently supporting the complete and partial synchronization of documents, the Snippet System adopts a document-centric approach to the instantiation of authoring tools. Instead of a single instance per tool with open documents, the Snippet System runs as many instances of a given tool as there are different, open documents created with it. A single instance manages all open, synchronized instances of one such document. This simplifies communication channels between internal tool components and is consistent with the document-centric approach at the user interface level described in [10].

The UML diagram in Figure 6 depicts the internal architecture of authoring tools: Every instance of the *View & UI* component manages one document view and handles user interaction for that view. Therefore, at run-time there are as many instances per authoring tool instance as there are open, synchronized instances of the involved document. The View & UI component ensures in particular, that its document view allows for embedding the document views of other authoring tools. This is essential for combining contents managed by different tools and, thus, for supporting compound documents. Moreover, this component enables the definition of Relations by allowing excerpts of the document view's content to be connected to other excerpts. Inspectors that allow the user to see and modify properties of the document are also an integral part of this component.

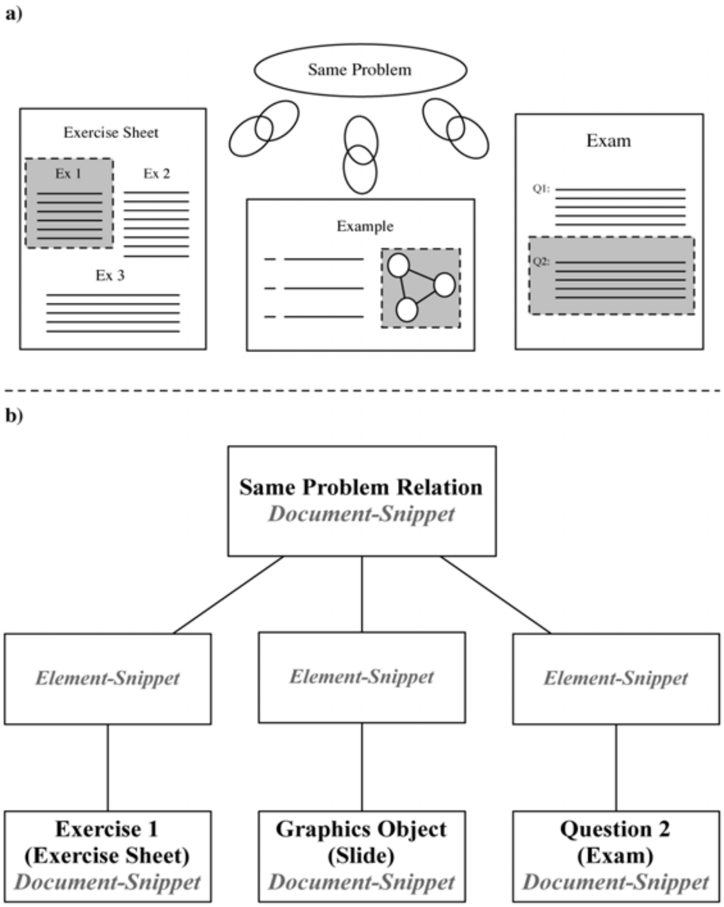

Figure 5. Relations Between Excerpts of Lecture Materials (a) and Representation in the Snippet Model

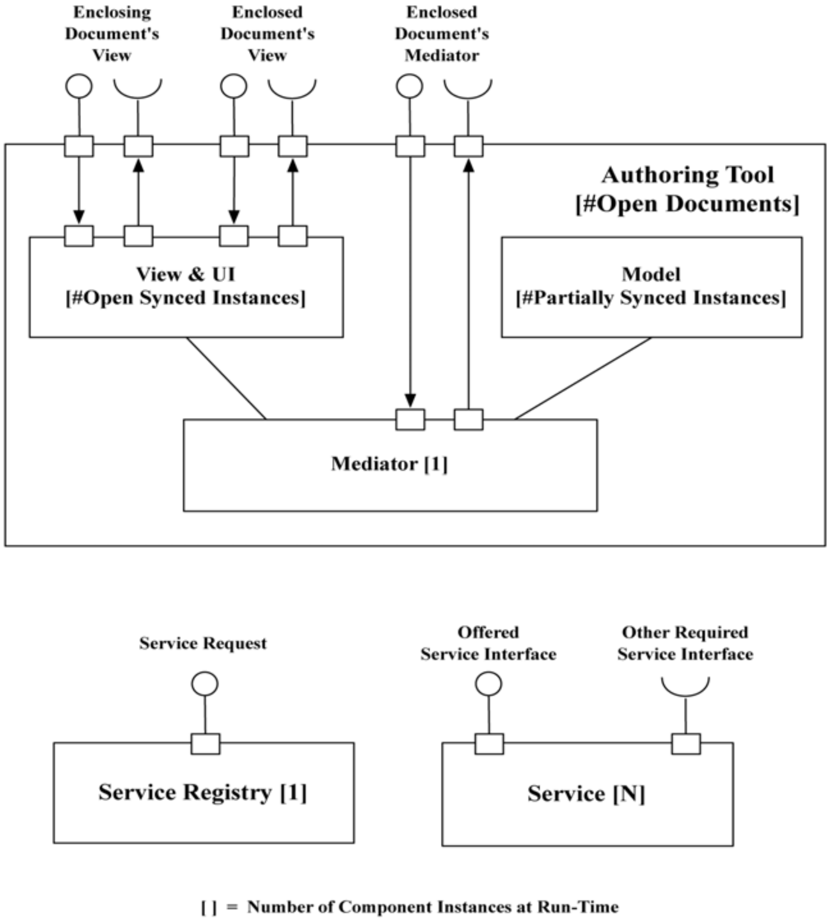

Figure 6. Authoring Tool and Service Architecture

The *Model* component manages Snippet structures and ensures that the modifications to a document at the View & UI level are made persistent. It also determines how exactly document contents are mapped onto Element- and Data-Snippets. The internal design of the Model component is such that one of its instances handles one partial document copy. Thus, at run-time there are as many Model instances per authoring tool instance as there are different, partially-synchronized versions of the managed document.

The *Mediator* component, in turn, coordinates the communication between View & UI and Model instances. It ensures, for example, that modifications made to a specific document instance are propagated to the right Model instance and that all View & UI instances which are affected by the change refresh their document view.

The *Enclosing/Enclosed Document's View* and the *Enclosed Document's Mediator* interfaces allow different authoring tool instances to communicate either at the level of their Mediator or at the level of their View & UI components. Information on a specific document instance is transferred at the level of View & UI components, whereas all other data is transmitted at the level of Mediator components. On the one hand, all authoring tools provide these three interfaces, which allows them to receive information by other authoring tools. On the other hand, they also require these interfaces from other tools for sending information of their own to them. Communication at the level the of View & UI components is bidirectional, because a document view has to communicate with its enclosed views as well as with its enclosing view, while communication at the level of Mediator components is currently unidirectional.

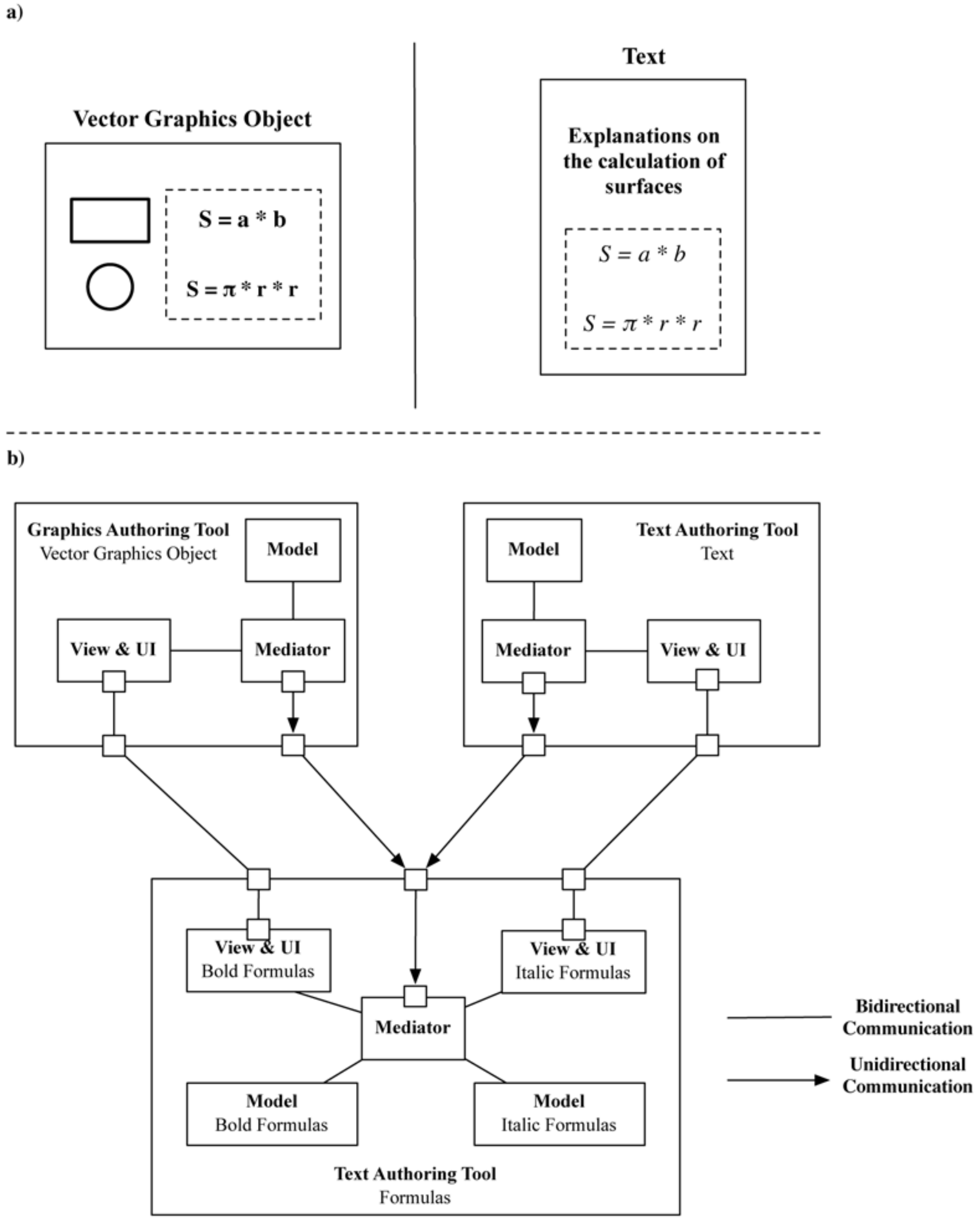

Figure 7. Two Compound Documents with Partially-Synchronized Formulas (a) and the Tool Components Managing Them (b)

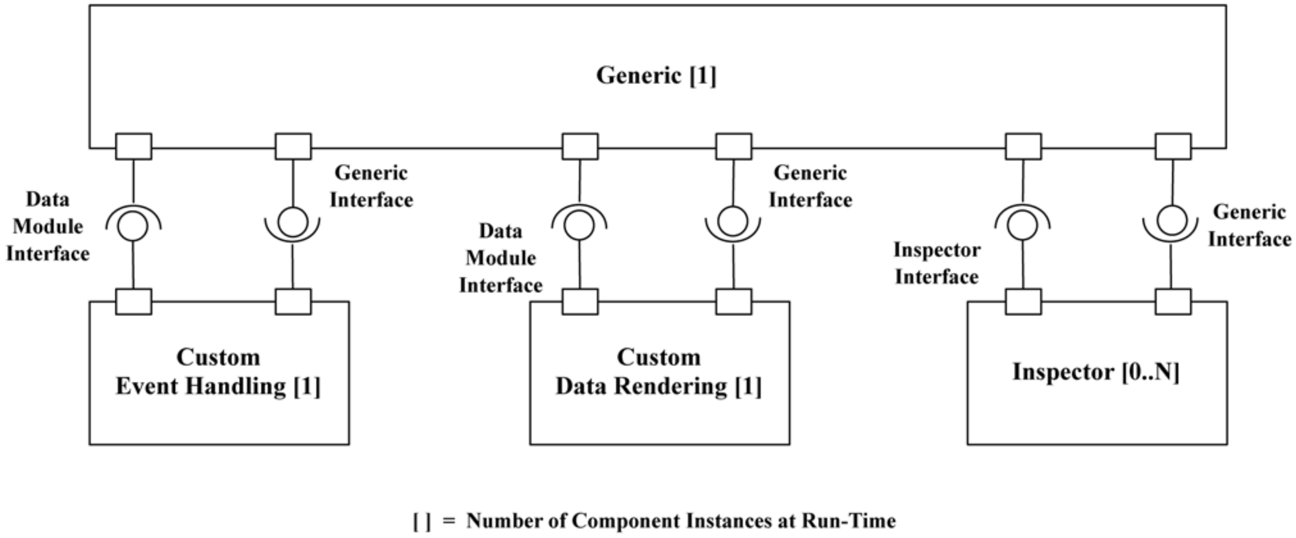

Figure 8. Internal Design of the View & UI component

Authoring tools provide some functionalities that are non-critical to their normal operation as *Services*, which are separate components. This prevents the three major components from getting bloated with rarely used functions and allows for reducing their memory footprint. For example, the export functionality provided by every authoring tool, for converting the Snippet representation of its documents into legacy file formats, such as. html or .docx, is realized by Service components. A more detailed description of authoring tools' export functionality is given in [11].

Services can be requested by any of the major tool components, but also by other Services. For this, they send a message to a *Service Registry*, which then instantiates the requested Service component. Each component provides an interface, depicted as *Offered Service Interface*, which allows the requester to invoke it. As mentioned previously, a Service may also need the assistance from other Service components for offering its functionality. Figure 7:

Figure 7 shows the two compound documents of Figure 3 with partially-synchronized formulas, as well as the tool components that are needed to manage them. Each of the two text documents, the *Text*, but also the *Formulas* [1], is managed by its own instance of a *Text Authoring Tool*. There is also an instance of a *Graphics Authoring Tool* to manage the *Vector Graphics Object.* For the *Formulas*, there are two open, synchronized instances and consequently two View & UI components are instantiated. Furthermore, there are two partially-synchronized versions of the *Formulas*. Thus, two instances of the Model component are required. Finally, the interconnections between View & UI components and those between Mediator components reflect the relationships between the individual documents that the two compound documents are composed of and allow the involved authoring tools to communicate and collaborate. To streamline the illustration the two communication interfaces between View & UI components have been combined.

6. Prototype

An emulation of the Snippet System has been developed on top of Mac OS X frameworks. It serves as a proof of concept and realizes all aspects described thus far, i.e., Relations, the fine-granular reuse of documents, the retrieval of synchronized instances, but also the component-based tool and Service architecture. The Snippet model itself has been implemented using Apple's Core Data framework, which offers functionalities of object databases.

A framework built on top of the Cocoa framework allows for developing authoring tools consistent with the architecture requirements. The framework ensures that each of the major tool components, i.e., the View & UI, the Model, and the Mediator component, is realized by a single generic subcomponent, which encapsulates basic functionalities that are required by all tools, and a set of custom subcomponents that implement the tool-specific behavior.

Figure 8 depicts the internal realization of the View & UI component. Here, especially the *Generic* subcomponent ensures that a document view is able to embed other document views. It also guarantees that the facilities provided to the user for defining Relations, or for reusing and synchronizing excerpts, are similar across all authoring tools, which enforces a consistent user experience throughout the entire system. User interaction facilities are detailed in [11].

The three other components are custom for every authoring tool: the *Custom Event Handling* component allows a tool to respond to mouse clicks, key strokes, but also multitouch gestures. The *Custom Data Rendering* component is responsible for correctly rendering a document's content in the document view. Finally, there may be a set of *Inspector* components which allow the user to see and edit document properties - such as font attributes in case of a text, or shape colors and sizes in case of a vector graphics object - or to navigate between slides in case of a presentation.

For better interchangeability of components, all communication between the custom and generic components is handled via dedicated interfaces.

Three prototypical authoring tools have been built on top of the framework: a text editor, a presentation tool, and a vector graphics tool. They allow for testing the Snippet System's most important concepts. More information on the prototype and videos showing how some of the concepts work in practice, can be found at http://mocca.uni.lu/snippets/.

7. Related Work

Enabling the fine-granular reuse of documents and connections between individual document excerpts has been subject of research in various areas. Some of the most important and closely related approaches and methodologies with regards to the Snippet System are described in the following.

7.1 XML

Compound XML documents are generally enabled by namespaces. Furthermore, in combination with XLink, XPointer and style sheets, XML allows for the reuse of specific document excerpts, the complete and partial synchronization of their different instances as well as for contextual connections. However, when using XML the efficiency of document access falls back to the efficiency of the XML parsing.

For this, several different approaches exist, each with specific advantages and disadvantages [13]. DOM-style parsing needs the entire XML document to be loaded into memory, which is a serious drawback for large files, especially if only a small fraction of the data needs to be accessed. SAX- or StAX-style parsing requires less memory, but displays poor performance under random access patterns. Finally, VTD-style parsing which is based on integer arrays also has performance issues when a document is updated frequently. This is due to the fact that the arrays need to be reconstructed every time.

There are also approaches to enable the parallel parsing of XML [16]. The core idea is to split the XML data into chunks that are then parsed concurrently. The main issue here is that the divisions between chunks must occur at well-defined points in the XML grammar. Therefore, an initial pre-parsing pass, analyzing the tree structure of the given XML with the aim of subsequently decomposing it according to the results, is required. Although this initial pass can also be parallelized [15] [20], it nevertheless introduces a certain overhead and thus still impairs performance.

The validation of compound XML documents itself is a non-trivial problem. DTD schemas do not support namespaces and therefore cannot be used to validate compound XML documents. XSD or Relax NG schemas do support namespaces, but they usually cannot be combined easily in order to define larger vocabularies. Standalone language schemas typically do not have the right level of modularity and abstraction which is needed for their seamless integration [17]. The purpose of NVDL, the Namespace-based Validation Dispatching Language, is to establish a connection between individual schemas. An NVDL schema enables the validator to use several schemas for validating an XML document. Essentially, NVDL in an initial step divides the compound XML document into parts containing elements from a single namespace. Every part is then validated using the adequate schema.

When relying on a representation by Snippets, accessing documents and specific parts is both more efficient and natural. The reason is that, by definition, each Element-Snippet in a given document's Snippet graph represents an individual structural element of the document. Different content types and, thus, elements likely to be represented in different namespaces, are separated by construction. Therefore, any steps to determine a document's structure can be skipped. When used for representing XML documents, Snippets have the potential to render their parsing and validation more efficient.

7.2 OpenDoc

Developed by Apple for Mac OS, the discontinued compound document system OpenDoc [1] [2] was a set of shared libraries for the creation of component-based software that was able to handle compound documents. Its major drawback was the drastically increased memory footprint when compared to equivalent standalone applications. But also the fact that it was hard to develop for the system and that OpenDoc was not integrated into the operating system itself so that developers were not able to rely on an existing installation, ultimately led to its demise. Furthermore, OpenDoc did not rely on an optimized document model, and therefore could provide only limited support for synchronization and relationships between documents. Partial synchronization and an equivalent for Relations, for instance, were not supported. Still, and despite its limited capabilities, OpenDoc showcased some interesting concepts such as the possibility to combine the functionality provided by several tools to create more complex compound documents.

In order to overcome said issues caused by the lack of integration into the underlying system, the Snippet System's most important concepts extend into the operating system itself, which can therefore be tailored for efficiently supporting Snippets and component-based

[1] The *Formulas* are represented like a separate document (see Figure 3), and therefore they are also managed alike

authoring tools. For instance, memory management strategies can be optimized for this scenario, thereby reducing the memory footprint of individual tools. Furthermore, as discussed in Section 5, non critical tool functionalities are implemented as Services, which are instantiated only when the corresponding functionalities are needed. This further reduces the tools' memory footprint.

7.3 Object Linking and Embedding

OLE, Object Linking and Embedding (Brockschmidt 1995), aims at allowing host applications to integrate data from other applications, i.e., data that the application is usually not able to create itself, into their documents. OLE is an integral part of the Microsoft Windows operating systems. Therefore, the OLE functionality can be incorporated into any Windows application. The main issue is that the synchronization of linked objects is not bidirectional. When linking a document *A* from within another document *B*, subsequent modifications to the original document *A* propagate to the linked instance in *B*. However, modifications to the instance linked in *B* do not propagate to *A*. The link between both instances is however maintained, despite the fact that they are no longer synchronized. Therefore, if afterwards *A* is edited, the linked instance in *B* is simply replaced by a new instance of *A*, and all previous modifications which have never been propagated to *A* are lost.

7.4 ArCoMo

The *ArCoMo* system [8] [9] allows users to create annotations across multiple documents which may even have been created with different applications. In order to do so, ArCoMo uses so-called *Artefacts* and *Anchor Points*. Anchor Points define excerpts of documents. Artefacts, in turn, create a link between related Anchor Points and store the annotation itself. The annotation is not limited to text, but may also contain other data. However, ArCoMo requires plugins to be developed from scratch for every application that must support Anchor Points, whereas in the Snippet System the generic subcomponents discussed in Section 6 provide most of the functionality required for defining Relations. Still, ArCoMo constitutes an early approach for the interlinking of related document excerpts.

8. Conclusion

This article introduced the *Snippet System*, an operating system environment which provides new, improved ways for managing documents. Most importantly, it allows for their flexible reuse, i.e., selected excerpts can be included in several other documents with different instances of such an excerpt remaining completely or partially synchronized. Dedicated mechanisms assist the user in keeping track of reused excerpts and synced properties, by allowing for the efficient retrieval of synchronized instances. *Relations,* which capture the context of individual document excerpts, additionally support the retrieval of related documents.

The Snippet System relies on a component-based tool architecture as well as a novel document model. The architecture enables many authoring tools to collaborate for managing a single document, which is essential for enabling the reuse of documents and the definition of Relations across tool boundaries.

The document model, in turn, provides a fine-granular document representation by means of persistent *Snippet* graphs. An inherent advantage is that these graphs can share *some* Snippets, thereby synchronizing only the corresponding data between the documents represented by the graphs involved. Additionally, this enables the creation of dedicated Snippet graphs representing Relations. A Relation's graph interconnects subgraphs of Snippets representing excerpts of documents, which are shared with the graphs representing the original documents.

An emulation of the Snippet System has been implemented on top of existing Mac OS X frameworks. It serves as a proof of concept and realizes all aspects described in this article.

A current limitation is that the properties, which remain in-sync between partially-synchronized instances of a reused document excerpt, have to be the same for all instances involved. Future work therefore is dedicated to enabling groups, each syncing a different property set between its instances. Furthermore, as discussed in Section 7.1, it remains to be analyzed in more detail whether a representation of XML documents by Snippets renders their parsing and validation more efficient.

References

[1] Apple Computer Inc., Apple Computer Inc. Staff (1995). OpenDoc Programmer's Guide for the MAC OS. Boston, MA: Addison-Wesley.

[2] Apple Inc. (1993). OpenDoc - Shaping Tomorrow's Software. White Paper.

[3] Blanc-Brude, T., Scapin, D. L. (2007). What do people recall about their documents? Implications for desktop search tools. *In:* Proc. of Intelligent User Interfaces (IUI '07), p. 102–111. ACM Press.

[4] Boardman, R., Spence, R., Sasse, M. (2003). Too many hierarchies ? The daily struggle for control of the workspace. *In: Proc. of HCI International 2003.* Citeseer.

[5] Bondarenko, O., Janssen, R. (2005). Documents at Hand: Learning from Paper to Improve Digital Technologies. *In:* Proceedings of the SIGCHI Conference on Human Factors in Computing Systems (CHI '05), p. 121-130. ACM.

[6] Brockschmidt, K. (1995). Inside OLE. Microsoft Press.

[7] Czerwinski, M., Horvitz, E., Wilhite, S. (2004). A diary study of task switching and interruptions. *In:* Proc. of the SIGCHI Conference on Human Factors in Computing Systems (CHI 2004), p. 175-182. ACM.

[8] Hoff, C., Wehling, U., Rothkugel, S. (2008). ArCoMo - An Artefact-Based, Collaborative Mobile Learning Environment. *In:* Proc. of the 6th IEEE International Conference on Pervasive Computing and Communications (PerCom 2008), p. 383-388. IEEE Computer Society,.

[9] Hoff, C., Wehling, U., Rothkugel, S. (2009). From Paper-and-Pen Annotations to Artefact-Based Mobile Learning. *Journal of Assisted Learning* 25 (3) 219-237.

[10] Kirsch, L., Botev, J., Rothkugel, S. (2012a). An Extensible Tool Set for Creating and Connecting Reusable Learning Resources. *In:* Proc. of World Conference on Educational Media, *Hypermedia and Telecommunications* (EdMedia 2012), p. 1434-1442. AACE.

[11] Kirsch, L., Botev, J., Rothkugel, S. (2012b). The Snippet System - Reusing and Connecting Documents. *In:* Proc. of the 7th International Conference on Digital Information Management (ICDIM 2012). IEEE.

[12] Kirsch, L., Esch, M., Rothkugel, S. (2011). The Snippet System - Fine-Granular Management of Documents and Their Relationships. *In:* Proceedings of the 6th *IASTED International Conference on Human-Computer Interaction* (*HCI 2011*)*,* Acta Press.

[13] Lam, T., Ding, J., Liu, J.-C. (2008). XML Document Parsing: Operational and Performance Characteristics. *Computer* 41 (9) 30-37. IEEE.

[14] Landauer, T. K. (1996). The trouble with Computers: Usefulness, Usability, and Productivity. MIT Press, Jun.

[15] Li, X., Wang, H., Liu, T., Li, W. (2009). Key Elements Tracing Method for Parallel XML Parsing in Multi-Core System. *In: Proc. of the 2009 International Conference on Parallel and Distributed Computing, Applications and Technologies,* p. 439-444. IEEE.

[16] Lu, W., Chiu, K., Pan, Y. (2006). A Parallel Approach to XML Parsing. *In:* Proc. of the 7th IEEE/ACM International Conference on Grid Computing, p. 223-230.

[17] Nalevka, P., Kosek, J. (2007). Advanced Approaches to XML Document Validation. *In:* Proc. of Extreme Markup Languages 2007. Mulberry Technologies, Inc.

[18] Norman, D. A. (1993). Things that make us smart: Defending human attributes in the Age of the Machine. Boston, MA: Addison-Wesley.

[19] Pan, Y., Lu, W., Zhang, Y., Chili, K. (2007). A Static Load-Balancing Scheme for Parallel XML Parsing on Multicore CPUs. *In:* Proc. of the Seventh IEEE International Symposium on Cluster Computing and the Grid (CCGRID 2007), p. 351-362, IEEE.

[20] Pan, Y., Zhang, Y., Chiu, K. (2008). Simultaneous Transducers for Data-Parallel XML Parsing. *In:* Proc. of the IEEE International Symposium on Parallel and Distributed Processing (IPDPS 2008), p. 1-12. IEEE.

[21] Waks, S. (1997). Lateral Thinking and Technology in Education. *Journal of Science Education and Technology,* 6 (4) 245-255. Springer.

Author Biographies

Laurent Kirsch received his Master degree in computer science from the University of Luxembourg in 2010. Since November 2010 he is a doctoral candidate and assistant at the University of Luxembourg. His research focuses on document engineering, interactive systems and personal information management.

Jean Botev is currently a postdoctoral researcher at the Computer Science and Communications research unit of the University of Luxembourg. He received his diploma degree in Computer Science and Media Studies from the University of Trier (Germany) in 2007, followed by a PhD in Computer Science from the University of Luxembourg in 2011. His research interests include complex networks, self-organization and collaborative systems.

Steffen Rothkugel received a Ph.D. in computer science from the University of Trier, Germany, in 2001. Since 2002 he is associate professor in the Computer Science and Communications research unit of the University of Luxembourg, where he is heading a small team. His work and teaching revolve primarily around the domains of system software and distributed systems. His current research focuses on interactive distributed systems and document engineering.