Vrije Universiteit Brussel

Faculty of Science
Department of Computer Science

A Metamodel and Prototype for Fluid Cross-Media Document Formats

Graduation thesis submitted in partial fulfillment of the
requirements for the degree of Master in Computer Science

Ahmed A. O. Tayeh

Promoter: Prof. Dr. Beat Signer
Advisor: Dr. Bruno Dumas

Academic year 2011-2012

Acknowledgment

First and foremost, I offer my sincerest gratitude to my promoter Prof. Dr. Beat Signer and to my supervisor Dr. Bruno Dumas, for their excellent guidance and patience and for providing me with an excellent atmosphere for doing research.

I am deeply glad to convey my warmest regards to those who supported me during my whole life in my studies and in achieving this work, my precious parents, lovely brothers, sisters and friends.

Abstract

Alongside with the transformation of computing from personal computers to the Internet, document formats have changed over the years. Future document formats are likely going to adapt to the coming age of ubiquitous computing, where information processing has been embedded into everyday activities and objects. While document formats have originally been created as a digital emulation of paper documents, they have been slowly enriched with additional digital features over time. These features were mainly incorporated to take advantage of new capabilities offered by the devices from which these documents are accessed. With the advent of ubiquitous computing, document formats seem to be facing a new evolutionary step. They will have to adapt to novel mobile devices, innovative interaction modalities, the distribution over multiple machines as well as heterogeneous input sources. This adaptation to the age of ubiquitous computing asks for a number of new document features. These features have been selected based on their link to the challenges that document formats will face in the near future. We present a review of a set of representative document formats in the light of these features, including multidirectional linking, versioning, content reusability, user rights management and content adaptation. Based on the results and findings of this review, we outline a roadmap towards future document representations that are adequate for the upcoming age of ubiquitous information environments.

Towards this future representation of document formats, a *Fluid Cross-Media Document Format Metamodel* has been developed. This metamodel has been built by extending the resource-selector-link (RSL) metamodel, which is a general hypermedia metamodel for managing aspects related to distribution, user rights management and content adaptation.

Finally, to validate the features of our new Cross-Media Document Format Metamodel, an online text editor which allows a user to create documents has been implemented. All features have been gained by tightly mapping all elements of the documents generated by this text editor to the elements in the fluid cross-media metamodel.

Contents

List of Figures

Listings

List of Tables

Chapter 1

Introduction

1.1 Context

Over the last decade, we have witnessed an explosion of the number of devices connected to the Internet. While they were originally offering some basic communication functionality, nowadays mobile phones and smartphones are providing more and more functionality that was until recently only offered by personal computers. Furthermore, as part of the Web 2.0 movement, each user and device becomes a producer as well as a consumer of information and users further exploit information through mashups. With each user becoming a producer of information, every device ends up as a potential information server. This trend is likely going to increase in the coming years and we are at the brink of the age of ubiquitous computing, where any object is going to be a computer, each sensor a server and each human being at the very core of a cloud of devices.

Document formats have accompanied this transformation of the once "personal" computing towards ubiquitous computing. However, quite often the recent evolutions of computing have only been endured rather than followed by the majority of document formats. A remarkable example is how document formats had to adapt to the small screen size of smartphones. Even though current smartphones are equipped with high resolution screens, the actual size of the device forced content providers to find ways to adapt the content of documents originally intended to be printed on A4 or letter-size paper. Even natively reflowable document formats such as HTML had to pass through some adjustment phase, for example because of HTML generators originally relying on fixed-size table elements or frames for the layout of webpages.

It is however probable that the range of devices on which documents are supposed to be displayed will become much more diversified in the future. Beyond the screen size, other features including significant variations in memory capacity or available bandwidth will have to be taken into account as well. Also, documents might be distributed over multiple devices and servers, flowing from one to the other in a fluid manner, potentially without any human intervention.

To be able to address the multitude of challenges introduced by the coming age of ubiquitous computing, document formats will have to take into consideration a range of digital features which are going to be presented in this thesis.

1.2 Research Objectives

The first objective for this thesis is to provide a deep analysis and review for a set of representative document formats in the light of five digital features. These features, the dimensions of the review, have been selected based on their linkage to the challenges that document formats will encounter in the near future, such as distribution, reusability of evolving documents, adaptation of content to context or copyright issues. The five dimensions are as follows:

1. *Advanced support for content linking*, since content is going to be distributed and split across different machines. In fact, the unidirectional linking as we know it from the World Wide Web possibly has to evolve into bidirectional or even multidirectional linking.

2. *Support for transclusion and content reusability*, first in the limited acceptance of allowing a document to include other documents in a similar way as an image can be embedded in a webpage. However, once again with the potential scattering of resources with multiple versions and identifiers, a more adequate support for full transclusion, allowing reuse of content portions, might eventually be needed.

3. *Versioning* focusses on providing a full list of modifications applied to a document, as well as the user or users who made the modification. Linked with the multidirectional links and transclusion, it will provide access to the complete history of changes and versions of a given document.

4. *Digital rights management* is already a hot topic and will continue to be so, especially if features like transclusion and the related copyrights management are better supported in the near future. Eventually, inclusion of user rights management in the general sense at the document level or even a lower level will enable to address authorisation in a deeply interconnected digital world.

5. Finally, *adaptation* has already been mentioned as an example above. However, beyond adaptation of the representation of a document, that is output content adaptation, adaptation of input will also gain importance. Specifically, knowing how to interact with some content might become as important as knowing how to present it.

The second objective of the thesis is to come up with new ideas for future document formats. Towards the "perfect" document representation, a new document format metamodel has been built by extending the resource-selector-link (RSL) metamodel. The RSL metamodel is a very clean metamodel that is responsible for managing aspects related to distribution, versioning, user rights managements and content adaptation. The proposed metamodel is called *Fluid Cross-Media Document Format (FCMD) Metamodel*.

Finally, as a proof of concept, a prototype of the proposed FCMD metamodel in addition with an online text editor has been implemented. Documents generated using this text editor are mapped to the elements defined in the FCMD metamodel in order to enrich them with the digital features supported by the metamodel.

1.3 Thesis Structure

This thesis is organised as follows:

- **Chapter 2: Background.** The purpose of this chapter is to set the fundamental background about the five dimensions that are going to be used in the review. The RSL metamodel which forms the basis for the future document metamodel is also introduced in this chapter. Finally, some key concepts about document formats will be given.
- **Chapter 3: Review of Existing Document Formats.** A review of the most representative document formats in the light of the essential features for the upcoming age of ubiquitous computing is presented. This is followed by an outline of the future fluid document representations.
- **Chapter 4: Fluid Cross-Media Document Format Metamodel.** In this chapter, we present the essential metamodel required for fluid cross-media document formats.
- **Chapter 5:Implementation.** This chapter provides an overview of the implementation of the prototype of the RSL-based metamodel for fluid cross-media documents.
- **Chapter 6: Conclusions and Future Work.** Finally, we conclude this thesis and highlight some potential future work.

Chapter 2

Background

In this chapter, a brief introduction about the essential features for the upcoming age of ubiquitous computing is given and their importance is highlighted. Then, the innovative resource-selector-link metamodel will be introduced in a nutshell. Finally, some general concepts about documents that are essential for our research will be introduced.

2.1 Features for the Ubiquitous Computing Age

Over sixty years ago, hypermedia pioneers felt the immense need for inclusion of some digital features in document models. Therefore, early hypermedia systems introduced some of these features, even though some of these systems were only visionary and have never been implemented. Nowadays, document formats are obliged or condemned not to pay too much attention to these features in order to have a simple presentation. However as we will see later, some formats did pay attention to a number of these features.

Digital features are the dimensions for our document formats review. Some of them had been introduced before in the early hypermedia systems, some of them never. But all of them are quite important for document formats to overcome the future challenges. Therefore, in the following subsections we highlight these features and show their linkage and importance to address future challenges.

2.1.1 Linking

In 1945, Vannevar Bush introduced the concept of the Memex [14], which is often credited as being the origin of hypermedia systems. The motivation behind the Memex was to change the hierarchical classification for accessing information. Bush came with the idea that the best thing to do is to mimic the human brain.

> *"When data of any sorts are placed in storage, they are filed alphabetically, and information is found (when it is) by tracing it down from subclass to subclass. [...] The human mind does not work that way. It operates by association. With one item it*

grasp, it snaps instantly to the next that is suggested by the association of thoughts, in accordance with some intricate web of trails carried by cells of the brain" [14].

The Memex looks like a desktop as shown in Figure 2.1. It has at the bottom front side drawers where users can put persistent storage in the form of microfilms. It has two displays at the top in the center, each having the size of an A4 paper. Those two displays are used to view the information in the form of pages, stored on the microfilms which will be accessed by the Memex and projected on the two displays. To make a link between the two pages being displayed, a trail between them has to be registered. Each trail has a unique identification code and it is stored permanently. The link can also be annotated with a pen with some handwritten comments and drawings. Later the user can reach the trail by entering its code and the Memex will automatically display both linked pages.

Figure 2.1: The Memex is often credited as being the origin of hypermedia systems

The idea of trails and associative links in the Memex has been taken into account in later hypermedia models, especially on the World Wide Web. These systems introduce what is called unidirectional links, meaning that the link can be followed only from the source to the target. These links can possibly be augmented with some metadata and a user's annotations. Some models such as Xanadu [42] proposed to use special links called bidirectional links, in which the link can be traced from the both sides, but unfortunately bidirectional links have not been realised so far on the Web or in any daily life digital application. Bidirectional links are useful. For example, in the scenario of the unidirectional links, imagine a webpage that has a link targeting another webpage. If the target page has been deleted, then the link in the first page will become broken. But this is not the case with bidirectional links, if one tries to delete the target page, they will be informed that this page is linked by the source page and the deletion of the target page will be prevented or it will be deleted and a notification will be sent to the source page forcing it for a link deletion. Also bidirectional links are very useful in the scientific domain. Imagine that a publication system uses such links, then an author of a scientific publication will know who, and which scientific work refers to their work. In such areas, computing the impact factor of a journal will be instant, instead of wasting couple of hours or days using complex computing algorithms.

In the context of ubiquitous computing, linking also plays an important role, due to the fact that it allows for the distribution of documents over multiple devices. Snippets of the same document can therefore "live" on different machines and can still be presented to the user as one complete document. Many-to-one links make it possible to have multiple redundant document snippet versions stored on different servers. The best accessible server could then be used during the document retrieval process.

2.1.2 Transclusion and Content Reusability

The Memex trails linked entire pages and not parts of pages, such as pieces of text or figures. If such facility were to be offered, it would be handy, especially in the case of composing a document out of some other document parts. The newly composed document can then have some links targeting the intended parts and the underlying system can easily render and embed the parts in that document only when it is visualised. That means that there are no copy-paste operations, but only rendering of the link targets when the document is displayed. This idea of content reuse is called *transclusion* and has been introduced in Ted Nelson's Xanadu [42] document model. Note that Ted Nelson has been influenced by the ideas of Vannevar Bush. The principal idea of his Xanadu project was a networked system that would store and index all the world's literature and other public and private information with transclusion as the key feature of that system.

> *"The central idea has always been what I now call transclusion or reuse with original context available, through embedded shared instances (rather than duplicate byte)"* [41].

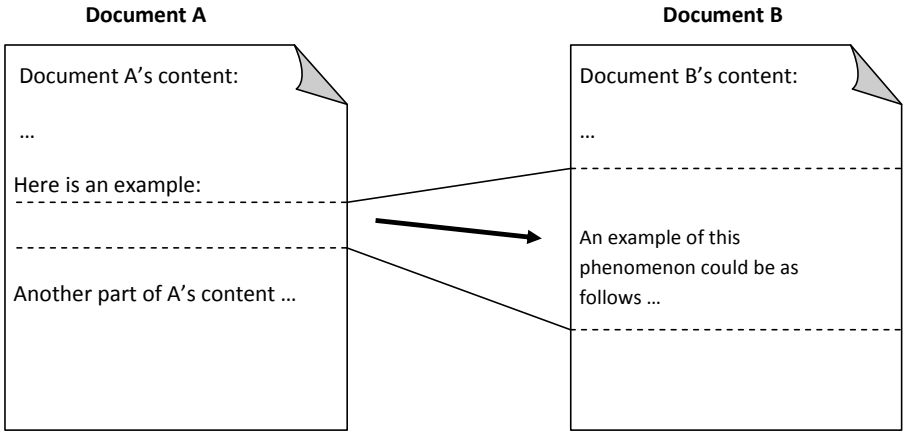


Figure 2.2: Instead of data replication, document A transcludes parts of document B

Transclusion is the idea of a so-called deep document, where snippets of information can be reused in higher-level document structures [54]. With transclusion, it becomes possible to

store information about the original document, as well as the exact position where the quote must be rendered. The exact position of the quote is the position where the link is defined. Also the transclusion idea as defined by Ted Nelson forces the original document to know which documents have quoted its parts in order to prevent broken links inside the newly composed document. Therefore, the tranclusion has to be implemented using bidirectional links.

To illustrate the idea of transclusions, suppose that while an author is editing a document `A`, they have to give an example about some physical phenomenon, but they realise that a document `B` contains the same example, even with more explanations, as the document `B` is mainly targeting this phenomenon. Then it would be much better for the author to transclude that part from document `B`. Figure 2.2 shows that document `A` links and transcludes a part of document `B`. Supposing that after some time, the transcluded part in document `B` is updated and some new information is added or deleted, then document `A` will also be updated since the links are to be only rendered in the visualisation step of the document and the transcluded part will always be rendered from its origin.

Transclusion has quite important advantages. First, it protects the intellectual property, since using some text via copy and paste removes not just the context of the quote, but also other metadata (e.g. the original document, its authors etc.) [36]. Second, as mentioned in the example above the propagation of the update operation will be solved. Third, the two way reading, knowing the original context of the quote is an added value for the reader and it is also of interest to the authors to know who uses parts of their articles. Of course this can only be achieved if the transclusion mechanism is built using bidirectional links. Last but not least, content reusability saves a lot of disk space.

2.1.3 Versioning

Keeping track of the version history for a given document allows the system to relate the different versions implicitly as different facets of a same document, rather than saving multiple versions of the same document which have to be related explicitly (for example through the same file name with a version number at the end). This in turn allows to relate meta-information to the set of documents instead of its individual versions. Such meta-information can be the creator, bidirectional links pointing to the set of documents or the full list of modification dates.

Saving disk space as with transclusion could also be achieved by versioning. Some decades ago, storage space was an issue because of its costs. Nowadays, disk space of several terabytes is not an issue neither in terms of availability nor in term of costs. Hence, one might argue that saving disk space is not a problem anymore. Actually this is not true. Imagine a large multimedia document where you want to add minor changes to each version. Saving each version as a standalone document will become an issue after several changes. Therefore, transclusion, versioning or both should be used. Figure 2.3 illustrates an example of using both features in the same document model. The document has been modified by rephrasing only one paragraph. The new version of the document will then contain every non-updated object from the previous versions as transcluded objects, in addition to the newly rephrased paragraph. In our metamodel for fluid cross-media formats, we will use a more mature and innovative way of implementing the versioning mechanism as discussed in Section 4.5.

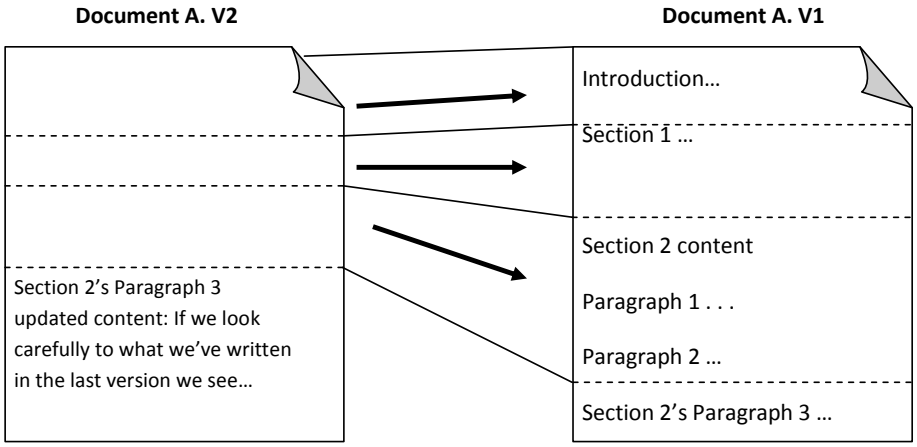


Figure 2.3: The new version of document A contains every non-updated content as transcluded objects, in addition to the newly updated paragraph

Currently, versioning is prominently supported at the application level. As the closest example to the operating systems, Apple included for example the *Versions* utility in its MacOS X operating system in 2011. However, applications have to specifically support the *Versions* functionality in order to profit from them. Authors such as Müller et al. [40] proposed to include a middleware layer between the lower file system and the desktop GUI to introduce format-independent versioning. Third-party solutions like DropBox[1] allow also server-based versioning of a document by keeping each new version of the document on the server. Finally, revision control software such as CVS [59], Subversion [19] or Git [39] are particularity effective with text-focused documents such as source code files. However, all these approaches are extensions on top of the operating system, meaning that support for versioning is dropped as soon as a document is migrated to a device on which the versioning system is not installed.

2.1.4 User Rights Management

In the Xanadu project, Ted Nelson had the belief that a good document model must support the digital copyrights management. Thus Xanadu automatically bills users with some micro payments for the delivery of copyrighted material and rewards the author with that money. Besides copyrights management, supporting the user access rights to the document has many advantages. For example, one could write a document containing questions and their optimal answers for an exam. Students are given access rights to only the questions, while the teacher has access to both the questions and their answers. By doing so, some of the disk space is saved and semantic linking between the questions and answers is explicit, rather than storing them in two different documents.

[1] http://www.dropbox.com (accessed July 14th, 2012)

Also, user rights management is an integral part of a successful integration of document formats in tomorrow's deeply ubiquitous environment. Indeed, as documents will be exchanged from server to server with potentially no human intervention, keeping track of author and source information will become delicate without embedding it into the document.

2.1.5 Adaptation

In the human computer interaction field, the main focus is on the usability aspect. Usability as defined in ISO standard is :

> *"The extent to which a product can be used by specified users to achieve specified goals with effectiveness, efficiency and satisfaction in a specified context of use"* [33].

Hence, some important usability considerations are: who are the users of the intended system, and in which context are they working. Considering the context helps to adapt the output of a document or an application according to it. Considering the users and their preferences will also help to adapt the document or the application according to their preferences. Therefore, in the web engineering field, we can see some web design methodologies that pay attention to such important aspects, thus they are called "Audience and User Driven Web Design Methodologies". The *Web Semantic Design Method (WSDM)* [21] is one example, that has been developed by some members of the WISE lab at the VUB.

With the recent advent of smartphones, documents also have to be adapted to the pocket-size screens. Document formats which were designed for A4 or letter-size printing have a hard time being displayed in a readable manner on smartphones. To counter this issue, multiple solutions have been explored. A first solution is to provide different versions of the same document, each one of them targeted at a given class of devices. This was the recommendation in the "W3C Web Content Accessibility Guidelines"[2] and was done for example for HTML 4, with web developers encouraged to create "mobile" versions. Pinkney et al. [46] also explored a similar track, but for print-oriented documents. A second solution is to reprocess automatically an existing document in order to make it compatible with different classes of devices. Cesar et al. [17] as well as Kumar et al. [37] have for example followed this track. However, none of these solutions have proven to be satisfactory enough. Automatic processing of the document is prone to errors. As for creating multiple versions of the same document, this solution is time- and resource-consuming, while introducing the challenges in keeping all versions of a same document up to date. Those are challenges when supporting adaptation of content representation based on the device capabilities. However, adaptation of input can also be considered. Indeed, novel devices now offer a full range of interaction modalities such as multi-touch screens, speech recognition or mid-air gesture interaction. How documents should be interacted with based on the available input modalities is a challenge that has to be investigated.

[2] http://www.w3.org/TR/WCAG10 (accessed July 14th, 2012)

2.2 The Resource-Selector-Link Cross-Media Metamodel

Many hypermedia models have been proposed in order to implement extra navigational functionality in hypermedia systems. Spatial and adaptive hypermedia models are examples. Few models have been implemented based on metamodeling principles akin to the database and modeling tools. Aside from the non-separation between technical and conceptual issues in these models, these models lose the generality and uniformity across systems.

The resource-selector-link (RSL) metamodel [55] has been developed to be general and flexible enough in order to be used for evolving hypermedia systems. RSL is based on the concept of linking arbitrary resources. A cross-media information platform called *iServer* [53] has been implemented based on the RSL metamodel. iServer supports various categories of hypermedia systems through the generality and extensibility of the mentioned metamodel. iServer has, for example, also been used to build a semantic file system to overcome the classical hierarchical way of managing files [16, 54]. It has also been used over many years in a variety of projects for physical-digital information integration and in particular for the implementation of the iPaper framework [44] for interactive paper. Therefore, we have a strong motivation to use the RSL as the basis for building the future document formats metamodel. In the following subsections, we briefly introduce the different RSL components.

2.2.1 RSL Core Components

The RSL metamodel was expressed using the semantic, object-oriented data model (OM) [43]. OM integrates concepts from both object-oriented data models and the entity relationship model. A collection of object instances (classification) are grouped in rectangles. The name of the collection is given in the unshaded rectangle, while the name of its associated type is given in the shaded part. The shaded ovals represent associations between object collections. Figure 2.4 shows the core of the RSL metamodel. In this core, we can explore six types of collections. We will start with the `Entities` collection.

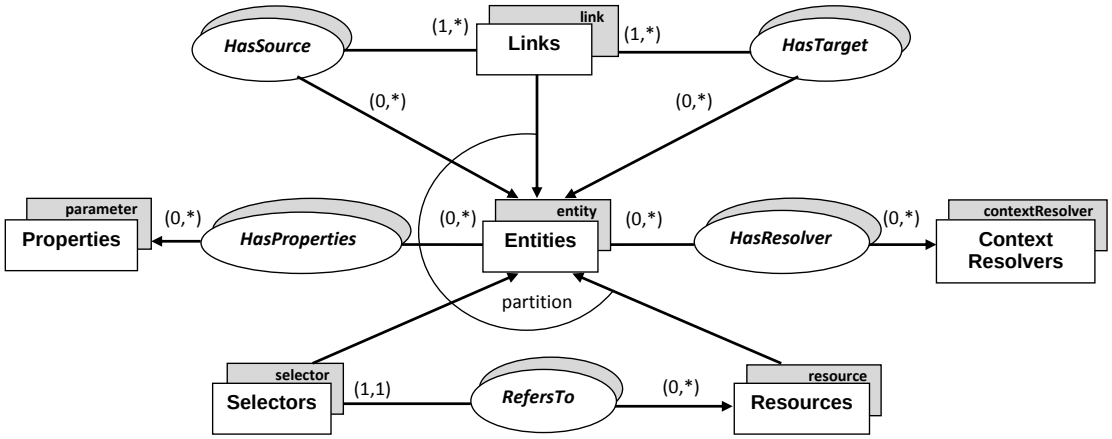


Figure 2.4: RSL core components, based on [55]

An entity from the entities collection is an abstract representation of any object that exists in the hypermedia system. Therefore, the entities are extended by three different subtypes: the `resource`, `selector` and `link` types.

The resource type is the simplest subtype of an entity. It is an abstract concept representing any resource type that exists in the hypermedia system, for example an image, a video or a text block. The resource type must be extended to address the concrete types of media that exist in hypermedia systems.

To support transclusion and links that address part of a resource, the selector type is introduced. A selector type is an abstract concept representing parts of resources in the hypermedia system. The selector type has to be extended to support the concrete resources in the hypermedia system (e.g. image selector if there is an image resource). The association `RefersTo` represents the fact that a selector is always associated with exactly one resource, while a resource can have zero or more selectors. These two constraints are shown in the RSL with `(1,1)` constraint on the selector side and `(0,*)` on the resource side.

The link type has the purpose of linking entities. For example, we can link a resource with a selector, a resource with a resource, a resource with a link, a selector with a link or a link with a link. Links in RSL are directed and lead from one or multiple source entities to one or more target entities. The association `HasSource` enforces that a link must have at least one source entity and possibly many. The association `HasTarget` enforces that a link must have at least one target entity and possibly many. By forcing the constraint that a link must have at least one target and one resource, the underlying hypermedia system will never have any broken link (dangling link).

Two points are worth mentioning here. First, the idea of having more than one resource for a link has not been introduced in most of the hypermedia models. The authors of RSL argued that the concept is very powerful in the context of integrating information across different digital and physical information spaces. For example, if the same information is published on different output channels (e.g. a webpage and an interactive paper document) and this information contains a link, the resolution of the link at all the output channels will be the same, meaning that it has different sources (the output channels). Second, the flexibility to have a link as a source or as a target. That means that we can annotate a link between entities with another link, it is a high level of data association and linking.

`Context resolvers` are associated with each entity. They are complex objects or functions that returns boolean values. An entity can be accessed when all its context resolvers are evaluated to true, otherwise it will not be accessed. An example of the usage of context resolvers is a resource that is linked to multiple targets. By defining a context resolver on each target, we can define in which context the link will be visible or not. This means that the link can adapt itself to a specific target in a specific context. Each entity can have multiple context resolvers as indicated by the `HasResolver` association.

The last concept in the core of the RSL metamodel is the `property`. A property is a key/value tuple. The properties can be individually defined to customise an entity's behaviour for a specific application. Each entity can be associated with a set of properties. RSL makes it flexible by not predefining a set of properties but rather introducing them as an abstract concept which can be extended for specific domains.

2.2.2 RSL Structural Links

As mentioned in the previous section, entities are standalone objects defined in the hypermedia system. Hence, one might ask, how could it be possible to compose a document from these entities? To answer that, the authors of RSL distinguish between: `navigational` and `structural` links as shown in Figure 2.5. Navigational Links have the property of linking between entities like the WWW links, but with more features as mentioned before (e.g. multi-targeted). Structural links are links which are used to compose a new resource, for example a document, out of one or more entities. Structural links are inspired by Ted Nelson's transclusion concept. In order to know the order of these entities composing a document, the `HasChild` ordered association is introduced.

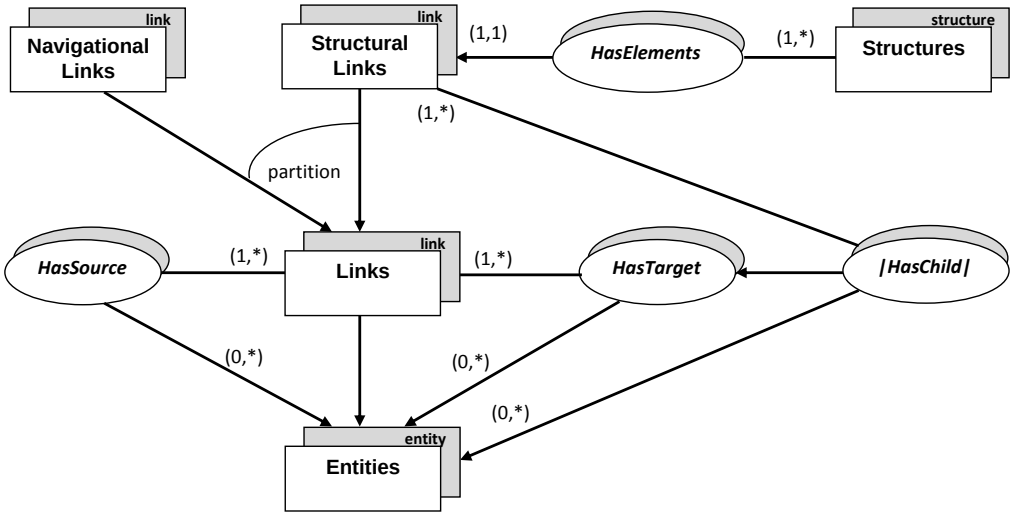


Figure 2.5: RSL links have two subtypes, the structural links and the navigational links, based on [55]

To illustrate this by an example, suppose a hypermedia system contains a text block resource collection and an image resource collection with its corresponds selectors. In order to compose a document containing a resource `A` from the text block collection, a selector `B` of an image and another resource `C` from the text block collection, we can define the document as a structural link with three targets. In other words, the structural link, which is the document in our example, is an entity and it is the source of a link targeting three other entities: text block `A`, the selector of image `B`, and the other text block `C` respectively, as illustrated in Figure 2.6.

The fact that the structural links are still entities makes it possible to compose a new entity out of the already existing ones in the system and the newly composed ones. For example, one can easily compose an article from the already composed sections of the entities defined in the system. Therefore, RSL stores the newly composed structures inside the structure collection. This also helps in entity reusability, as there is no obstacles if more than one structure contains the same resource, selector, link or even another structure.

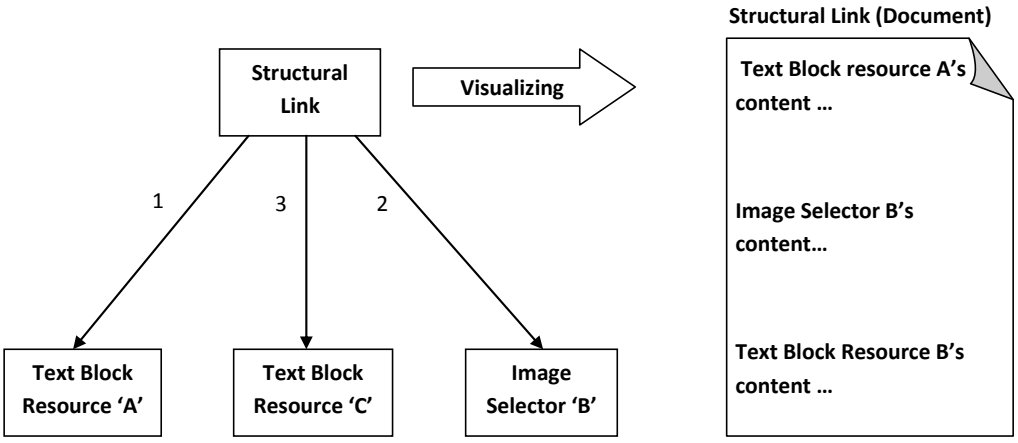


Figure 2.6: Composing a document with RSL structural links

2.2.3 RSL Users

Because of the importance of user rights management, RSL authors incorporated the users in the the metamodel. In RSL, the access rights are defined at the entity level, meaning that individual permissions for links, resources and selectors can be defined. The representation of the user management component in RSL is illustrated in Figure 2.7.

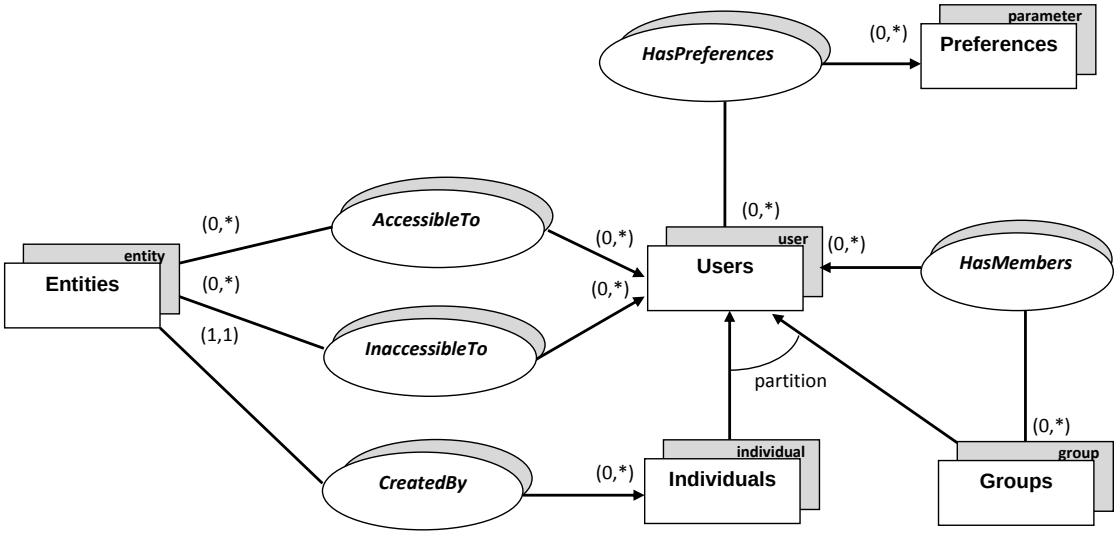


Figure 2.7: The user management component in RSL, based on [55]

A user is either an individual or a group. A group can contain individuals or other groups. Each entity is created by only one individual. The users access policies for each entity are defined using the associations `AccessibleTo` and `InaccessibleTo`. Each user can have some

preferences stored in the preferences collection. User preferences can be exploited for usability purposes, by adapting the presentation and visualisation of the entities based on them.

2.2.4 RSL Layers

So far we have introduced the concept of selecting and addressing parts of a resource using RSL selectors. However, the selector concept is not enough when parts of a resource are defined by selectors that overlap. For example, one can define two selectors, the first specifies a paragraph in a document, while the other specifies one line in the same paragraph. This generates *a link resolution problem* in terms of not knowing which link to activate when the line is selected. Therefore, RSL deals with this by introducing the concept of layers. Figure 2.8 shows the layers component of the RSL model.

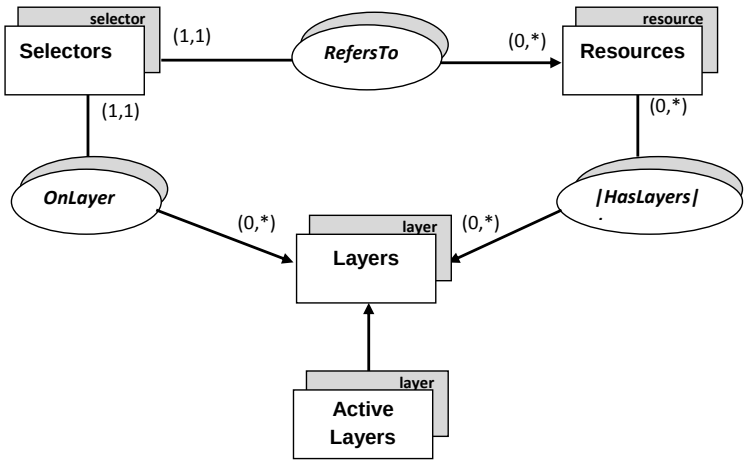


Figure 2.8: The layers component in RSL, each selector is only on one layer, based on [55]

Two concepts in the Layers component deal with that problem. First, `Layers` collection that maintains the different overlapping selectors on a same resource, by defining a layer for each. Therefore, every selector is on a different layer. This is reflected by the constraint `(1,1)` on the selectors side of the `OnLayer` association. The `HasLayers` between resources and layers is an ordered association and means that a resource can have multiple ordered layers and possibly none. Second, the `Active Layers` is a sub-collection of the layers collection. It maintains the active layers in the layers collection. Layers can be activated or deactivated depending on which selector we want to view and in which context we are.

2.3 Document Understanding

Document production systems started in the 1960s. The first systems were low-level formatters [7]. These formatters, like RUNOFF [51], were defined in terms of the physical characteristics of the printed document [27]. Their role was very simple and essentially it was to construct

lines of equal length and to produce justified pages on the basis of a ribbon of text. To control the page layout, a few commands had to be introduced. These were elementary commands, very close to those understood by the printer (e.g. space, line feeds or form feeds). The commands of RUNOFF were enhanced to take greater advantage of possibilities provided by new printers or even photo typesetters [7]. The enhanced commands were used to create macro-instructions, making it possible to adapt the formatter and to provide the user with higher-level commands as in TROFF [34]. It was also at this time that Donald Knuth defined TEX [35]. TEX's principal concern was typographic quality obtained at the time by means of electrostatic printers and to provide a system that would give exactly the same result on all computers.

At the end of the 70s, a revolutionary approach was invented by B. Reid, who regarded the language of formatters as high-level language called Scribe [49], which describes a document in logical terms rather than as a function of the desired presentation. Scribe brought *Generic Markup* (also called *Generic Coding*) to the attention of the academic community and provided a practical demonstration that separation of the document's content from its appearance improved the portability and reuseability of the document markup [27]. In such markup languages, the author of a document specifies the document components, for example a book as a sequence of chapters, each chapter as a sequence of sections, each section as a sequence of subsections and so on until the document's content is completely specified. Examples of such markup languages are the Generalized Markup Language (GML) [29] from IBM, the Standard Generalized Markup Language (SGML) [30] as well as for Hypertext Markup Language (HTML) [18].

What You See Is What You Get (WYSIWYG) is another approach for document production systems. The document content is displayed during the editing in a form closely corresponding to its appearance when printed or displayed as a finished product. The document may be modeled as a sequence of paragraphs, where each paragraph is a sequence of characters [26] or it can also be represented as monolithic blocks of linear content [54], without any semantic interpretation [8]. The first system was Bravo [60] developed for the Xerox Alto workstation[3]. PDF and Word documents are examples of WYSIWYG documents.

2.3.1 Document Production

A printed document is the result of a multi-step production process called document production [8] or document processing [26]. Shaw [52] modeled these processes or activities in a simple but useful model, which is highlighted in Figure 2.9. Three representations of the document are identified in this model: the document expressed in terms of an abstract model, the document expressed in terms of its concrete appearance and the concrete representation of the document projected onto a display medium. An example may clarify the distinctions between these three representations of the document. Suppose that a person wants to write a book. First, the content of the document is edited, leading to a logical document (`Document Model`), which consists

[3]The Alto is a personal workstation that was developed in 1973, incorporating an 8.5 by 11 inch bit-mapped display with a resolution of about 70 pixels per inch, a typewriter keyboard and an attached pointing device called the mouse.

of a sequence of chapters. Each chapter consists of a sequence of sections and continuing in this fashion until the document is described in terms of its basic component parts (such as individual characters). Then, various layout rules are applied on the edited content resulting in a formatted document called the physical document (`Output Model`). In our example, the physical appearance will describe the book in terms of that two-dimensional page space; for example the position of elements within that page space will be represented. A physical appearance of a string of text will also include specification of the line breaks, page breaks, and hyphenations that are associated with the string of text as well as the fonts and the sizes that are associated with the individual characters. Finally, the physical document is rendered to a final fixed-layout document called the printed document or the (`display`) of that document. The display of a document can be in one of the following three forms: hard copy, bitmap or electronic. The model of the document production activities also names the transformations between different representations. `Formatting` transforms the document model into an output model representation. `Viewing` transforms the output model into a display representation.

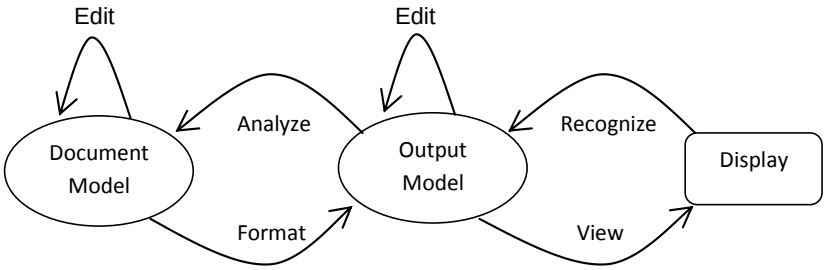


Figure 2.9: The document processing activities as modeled by Shaw.

Depending on the editor or the document production system, the content editing and formatting may be separated or mixed together. For instance, LaTeX and Scribe enforce the author to separate the content editing from the formatting. Indeed, these documents are based on the idea that the authors should be able to focus on the content of what they are writing without being distracted by the output appearance. In preparing such documents, the author specifies the logical structure and lets the document preparation system worry about the physical representation. In the other way, which is the interactive methodology that serves the WYSIWYG approach, the editing functions have been merged with the formatting functions into one unified function. In the interactive editor/formatter systems, documents are created, viewed and revised without leaving the editor/formatter. Standard word processors such as Microsoft Word allow the author to edit the document content together with its formatting, which might lead to inconsistent layouts.

2.3.2 Logical Document

The logical structure of a document reflects the way information is organised in terms of logical objects, including chapters, sections or paragraphs. The logical document structure can be defined as the division of the document into smaller parts and the relations between these parts. At the lowest level of this structure, the actual content (atomic objects) to be found.

Some aspects play a role in differentiating between different document models. First, what is the lowest level of atomic objects within a model. The atomic object may be a text string representing a paragraph, part of a paragraph, a character or even a sub-part of a character. Second, is it allowed for some objects to be a result of relations between other objects and how strongly are these relations constrained. As an example, consider a document model in which atomic objects are combined into higher level objects. This combination process continues, resulting eventually in the formation of the object representing the document as a whole.

Listing 2.1: Logical structure of an article formalized in XML

```
<Article>
    <Header>
        <Title>What is Wrong With Digital Documents?</Title>
        <Author>Beat Signer</Author>
        <Affiliation>
            <Address>Brussels-Belgium</Address>
            <Email>bsigner@vub.ac.be</Email>
        </Affiliation>
    </Header>

    <Abstract>Many of today's ...</Abstract>

    <Body>
      <Section> ... </Section>
      <Section>
          <Title>Background </Title>
          <Paragraph>As denoted in the previous section... </Paragraph>
      </Section>
    </Body>

    < Bibliography>
      <Entry> ... </Entry>
      <Entry> ... </Entry>
    </ Bibliography>
</Article>
```

The logical structure of the document reflects the allowed relationships among objects. The relationship may be specified to result in a linear document model adopted by Bravo system, a tree model or for example an acyclic model. The tree model is dominant in most document models. Figure 2.10 expresses the logical structure of the article document represented in Listing 2.1.

The primary logical structure of the document can be enriched with a secondary one, in order to express relationships that cannot be expressed within the primary itself [15]. The secondary structure is mainly defined by three constructs: attributes, floating objects and cross-references. Attributes are used to denote semantic information that is not provided by the primary structure. A floating object is an object which is allowed to appear at a more or less arbitrary place in part or all of the document. The reference construct allows objects to refer to other objects.

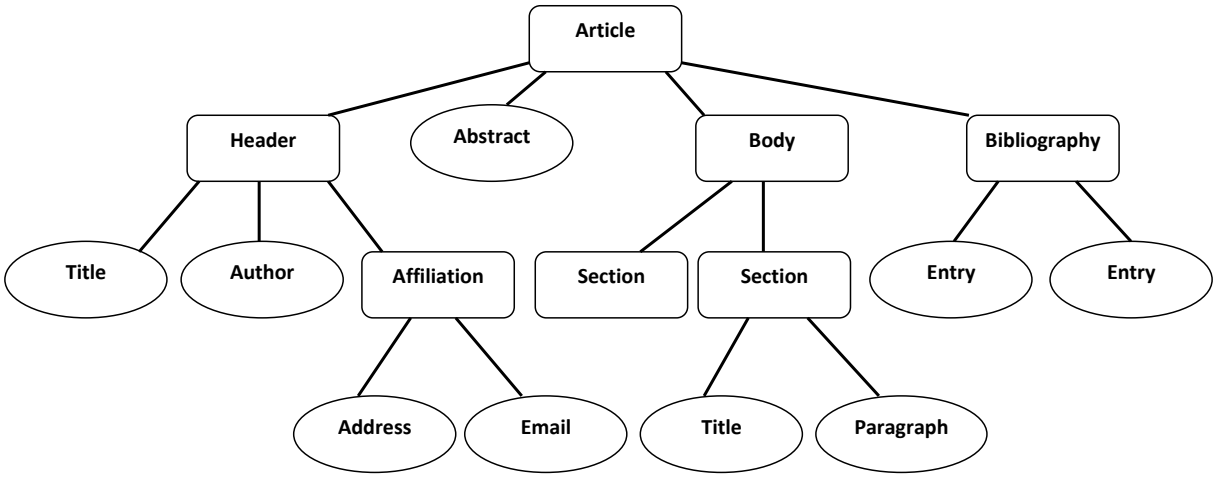


Figure 2.10: The logical tree structure of the article presented in Listing 2.1

References may be defined among the objects in the same document or among objects in different documents. Hypertext documents, like HTML, use the references to define the links. Figure 2.11 shows the overall logical document structure reflected in both the primary and the secondary logical structures.

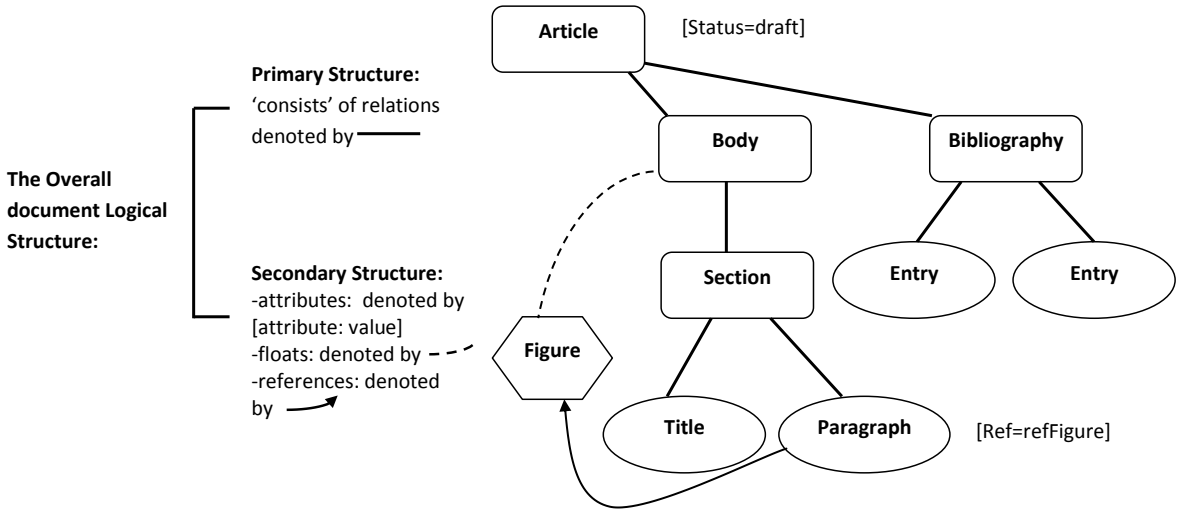


Figure 2.11: The overall logical document structure has a primary structure and is possibly augmented and enriched by a secondary structure

A class of documents is the set of documents that can be described with the same logical formalism [8]. Consider a journal that contains several articles. One article may have four sections, another article may have five sections and also subsections, but in general all articles contain a sequence of one or more sections and their subsections. Thus, the *generic structure* of

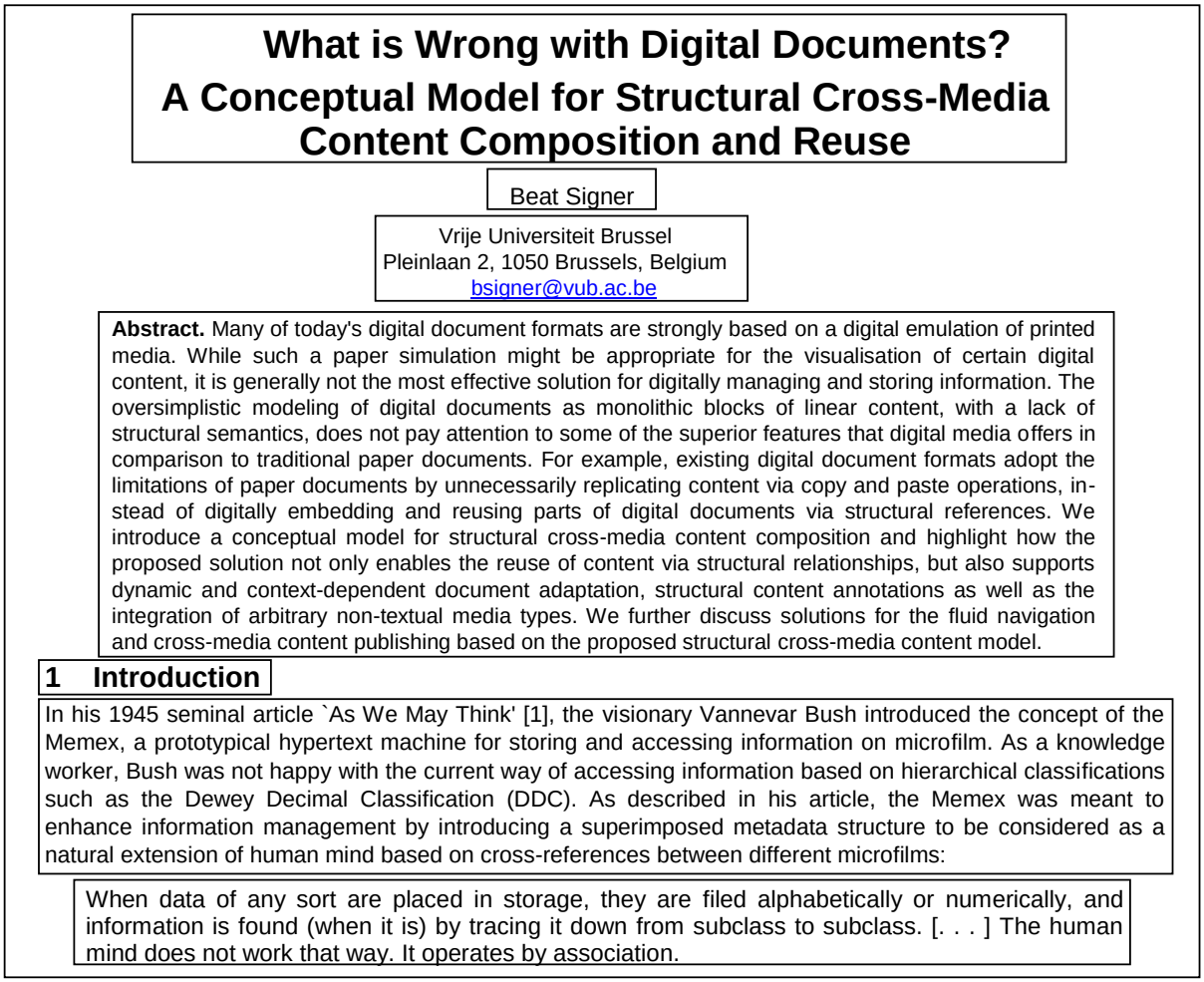
these articles can be defined by a title, followed by a sequence of one or more authors, followed by an abstract, followed by a sequence of one or more sections and optionally subsections. The generic structure is to be called the *Document Type Definition (DTD)*, which contains the object definitions, allowed relations between them and constraints.

The availability of a generic structure together with documents that conform to this structure has many advantages and can be exploited in many applications. A *generic layout* can be defined for all documents conforming to that generic structure. The LaTeX document format that has been used to write this report defines such a layout for different document classes. An application can be built to store the title and authors in a database. Using the generic structure, the application has knowledge of where it can find a specific kind of information in a document.

2.3.3 Physical Document

In the document production process, the formatting activity transforms the logical structure into the physical document. On the one hand, poor editing software that mixes editing and formatting may lead to non-homogeneous formatting styles (e.g. Microsoft Word). On the other hand, structure-oriented document preparation systems enable a single document to have multiple formatting styles or multiple documents to have a similar presentation by providing a standalone formatting style-sheet.

The visual appearance of a printed document reflects its physical structure, without any semantics. The physical structure of a document corresponds to the organisation of the page in terms of regions delimited by images and text blocks that can be further split into text lines, words and characters [8]. The physical structure is often represented as a tree structure in order to transcribe the relationships between the various physical objects. Figure 2.12 shows a possible view of the physical structure of a conference paper.



What is Wrong with Digital Documents?
A Conceptual Model for Structural Cross-Media Content Composition and Reuse

Beat Signer

Vrije Universiteit Brussel
Pleinlaan 2, 1050 Brussels, Belgium
bsigner@vub.ac.be

Abstract. Many of today's digital document formats are strongly based on a digital emulation of printed media. While such a paper simulation might be appropriate for the visualisation of certain digital content, it is generally not the most effective solution for digitally managing and storing information. The oversimplistic modeling of digital documents as monolithic blocks of linear content, with a lack of structural semantics, does not pay attention to some of the superior features that digital media offers in comparison to traditional paper documents. For example, existing digital document formats adopt the limitations of paper documents by unnecessarily replicating content via copy and paste operations, instead of digitally embedding and reusing parts of digital documents via structural references. We introduce a conceptual model for structural cross-media content composition and highlight how the proposed solution not only enables the reuse of content via structural relationships, but also supports dynamic and context-dependent document adaptation, structural content annotations as well as the integration of arbitrary non-textual media types. We further discuss solutions for the fluid navigation and cross-media content publishing based on the proposed structural cross-media content model.

1 Introduction

In his 1945 seminal article `As We May Think' [1], the visionary Vannevar Bush introduced the concept of the Memex, a prototypical hypertext machine for storing and accessing information on microfilm. As a knowledge worker, Bush was not happy with the current way of accessing information based on hierarchical classifications such as the Dewey Decimal Classification (DDC). As described in his article, the Memex was meant to enhance information management by introducing a superimposed metadata structure to be considered as a natural extension of human mind based on cross-references between different microfilms:

> When data of any sort are placed in storage, they are filed alphabetically or numerically, and information is found (when it is) by tracing it down from subclass to subclass. [. . .] The human mind does not work that way. It operates by association.

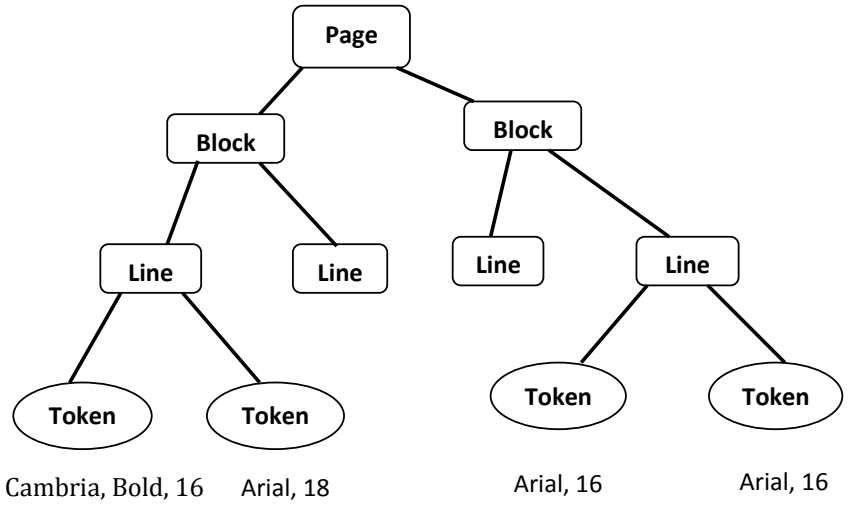


Figure 2.12: The physical structure of a conference paper is described in terms of hierarchy regions delimited by text blocks that can be split into text lines, words, and characters.

Chapter 3

Review of Existing Document Formats

In this chapter, a number of existing document formats will be introduced. We then present a review of the presented document formats in the light of the five features introduced in the previous chapter. Finally, we outline a roadmap towards future fluid document representations for the upcoming age of ubiquitous information environments.

3.1 Document Formats

Thousands of document formats exist nowadays. Hence, reviewing all of them is not practical and almost impossible. Therefore, the first step we had to do was to categorise document formats in families. Then, from each family we have selected the most prominent and popular document formats. One has to note that we did not consider document formats that are domain- or application-specific, for example database document formats or programming languages document formats.

The first document family is the *meta-languages family*. Other document formats can be defined using this family of formats. The SGML and XML document formats have been selected as representatives for this family. A second family is the *document preparation family*. This family of document formats uses the "generic coding" for marking up the documents. From this family, Scribe, GML, LaTeX and DocBook have been chosen. A third family is the *print-oriented family*, with PDF, OpenDocument and Open Office XML as representatives. This family of document formats have been primarily targeted towards WYSIWYG editing and printing. A fourth family is the *World Wide Web family*, with the different versions of HTML and XHTML as representatives. A fifth family format have been selected, which tries to integrate the flexibility of the document preparation family with the naturalness of the print-oriented family. We called this family the *not-exact representation family*. The TNT document model was chosen to be the representative of this family. A sixth family is the *document image analysis family*, with XCDF and OCD as representatives. This family of formats tries to recover the logical and physical structures for scanned documents after applying some image analysis methods. Finally, two of the presented formats are not classified in any family, since they are unique for their purposes. The first format is EPUB which has been created as a digital device-oriented reflowable document

format. The second format is the Open Document Architecture (ODA) which is a standard format to facilitate the interchange of documents.

The following subsections present the reviewed document formats in ascending order according to their development year.

3.1.1 Scribe Document Model

Scribe [49], developed by Brian Reid and described in his doctoral thesis in 1980, was the first language that made the separation between the logical document structure and presentation. Scribe was revolutionary by introducing the idea of styles separated from the marked up document [64]. Scribe is often accredited as being the origin of the later markup languages [64, 27].

Device portability and *discrimination of content and format* were the two most important requirements for the Scribe language [48]. The intent of the device portability is to remove any device dependencies and easy reuse with other printing devices or displaying devices. The intent of the content separation is for easier reuse of the same content in a different format or context. The third requirement was the motivation for creating the Scribe commands (markups) or the named environments. Scribe commands are not commands in the ordinary sense of the word since they do not directly command anything, rather than to semantically describe the content by a name (e.g. section, chapter, table).

Scribe introduced named environments which had the role of containers (e.g. ordered lists, tables) [62]. Each environment is considered to represent a logical part of the document, like a section or quotation [15]. Environments could be nested and any kind of hierarchical structure can be defined through relationships between environments [62, 26]. Some environments, such as the one defining the chapter's title, cannot be nested and nesting is not used for identifying higher-level document structuring such as the division of a document into chapters or sections. With the limitation that some environments cannot be nested, relationships between environments are not constrained [26].

The commands used to define the content can be used in two ways, either with begin/end illustrated in Listing 3.1 or by the abbreviated syntax shown in Listing 3.2.

Listing 3.1: Scribe commands with begin/end form

```
@begin (quotation)
Scribe document model is often accredited as being the origin
of the later markup languages
@end(quotation)
```

Listing 3.2: Scribe commands in abbreviated form

```
@Quotation [Scribe document model is often accredited as being
the origin of the later markup languages]
```

Semantically, the commands in the Scribe specification language are labels marking text in specific formatting environments. Each environment places certain requirements on the appearance of its text in the final document, without giving specific details. For example, an *Italics*

environment `@I` requests that its text should be set in an italic font appropriate to the surrounding text or be underlined if there is no such font.

3.1.2 Generalized Markup Language

GML [29] is a markup language that defines tags for the IBM text formatter, SCRIPT/VS [3]. GML was developed by Charles Goldfarb, Edward Mosher and Raymond Lorie (whose surname initials were used by Charles Goldfarb to make up the term GML [28].

The idea behind GML is the same as in the Scribe document model: to describe what something is rather than what it looks like on the page and to let the text-processing program take care of what it looks like on the page. That is the reason behind its first name "Text Description Language" [28]. Using GML, a document is marked up with tags that define what the text is, in terms of paragraphs, headers, lists, tables and so forth. The physical appearance of the document can differ from one device to another by specifying a distinct profile for each device.

Listing 3.3 shows a simple document written with GML. The tags `h1`, `p`, `ol` and `li` define heading, paragraph, ordered list and a list item respectively. Besides describing the elements in the document, GML tags are used to describe the overall document structure. According to GML, the overall document structure contains four parts [1]:

1. Front matter: The front matter contains the title page (`titlep` tag), abstract (`abstract` tag), preface (`preface` tag), table of contents (`toc` tag), list of figures (`figlist` tag) and list of tables (`tlist` tag).

2. Body: The body of the document is the main portion of the document (`body` tag).

3. Appendices: The appendix part follows the body and contains information supplemental to the material in the body of the document (`appendix` tag).

4. Back matter: The back matter contains the glossary (`glossary` tag) and the index (`index` tag).

Listing 3.3: Example of a simple document written with GML

```
:h1. Generalized Markup Language
:p. The idea behind the GML is to describe what something is,
rather than  what it looks like on the page ...
:p. GML is the ancestor of later markup languages like:
:ol
:li. SGML
:li. SCribe
:eol.
:p The GML was a revolution in text editing and document engineering
 communities.
```

3.1.3 LaTeX

LaTeX [38] is a typesetting system that is very suitable for producing scientific and mathematical documents of high typographical quality. LaTeX uses the TeX formatter as its typesetter. With LaTeX, not only scientific papers can prepared, but also excellent letters, presentations and much more. The last revision of LaTeX is LaTeX 2_ε which has more support for font types, hyperlinks and other features.

The input for LaTeX is a plain text file. It contains the text of the document as well as the commands that tell LaTeX how to typeset the text. LaTeX commands are case sensitive and take one of the following two formats:

1. They start with a backslash \ followed with a name consisting of letters only. Command names are terminated by a space, a number or any other 'non-letter', for example the command `\LaTeX` is used to print LaTeX.

2. They consist of a backslash and exactly one non-letter, for example the command `\#` is used to print the character #.

Some commands require some parameters, which have to be given between curly braces { } after the command name. Some commands take optional parameters, which are inserted after the command name in square brackets [ ].

Listing 3.4 shows the content of a minimal LaTeX file. The document class is specified to be of a type 'article' with the optional parameter '11pt' for the font size. The author and the title are specified by the `\author` and `\title` commands respectively. LaTeX is asked to show the title, which contains the author, title and the date, using the `\maketitle` command. The article contains only two sections, each section starts with the `\section` command.

Listing 3.4: Example of an article written in LaTeX

```
\documentclass[11pt]{article}
\author{Beat Signer}
\title{ What is Wrong With Digital Documents ?}
\begin{document}
\maketitle
\section{Introduction}
The over-simplistic modeling of digital documents as monolithic
 blocks of linear content, with a lack of structural semantics ...
\section{Conclusion}
There is a need for structural semantics for the documents ...
\end{document}
```

3.1.4 Standard Generalized Markup Language

SGML [30] descended from GML. SGML defines a syntax for including the markup in documents, as well as one for separately describing which tags are allowed, where and which attributes are allowed for each tag (in a form of a DTD). This helps authors to freely define tags

that are most suitable for them in their own language. Therefore, SGML is a meta-language and many markup languages derived from it [64]. XML and HTML are examples of markup languages that are applications of SGML. SGML frees documents from hostage relationships to processing softwares and enables the sharing of machine-readable large documents. It has been adopted by some governments, aerospace, airlines, etc. [31].

SGML documents are tree structures with additional connections between the nodes. This feature makes SGML able to represent documents with arbitrary structures, because most of the conventional documents are in fact tree structures. Except for the terminal nodes representing the *data*, each node in an SGML document tree is the root of a subtree, called an *element*. The descendants of a node are the *content* of that element. The document as a whole is called the *document element*. The structure of the document element consists of one or more hierarchies (tree structures) each conforming to a separate DTD. Arbitrary complex structures, such as hypertext, are supported by reference attributes that represent various types of relationships among nodes and among documents.

3.1.5 TNT

Richard Furuta tried to merge the flexibility found in the abstract object-oriented approach, which represents the document as abstract objects like Scribe, with the naturalness of document manipulation provided by WYSIWYG editors. A tree-based document model called TNT [24] that allows a variety of document objects as leaves (e.g. text, tables and mathematical equations) has been defined.

The TNT document model consists of abstract objects, hierarchically related to each other. For example, one document can have chapters. Each one has multiple sections, sections have subsections and so on. A wide variety of leaf object types are desirable at the lowest level of the hierarchy, for example, objects containing textual material, line drawings or scanned images. Additionally, these lowest-level objects may interact with each other. For example, tabular objects may contain textual or mathematical objects within the individual entries of the table.

Furuta believed that the tree structure is an adequate representation between objects that are not at the leaves level, but not adequate for modeling the wide variety of structures found in the leaves. For example, tables are not naturally tree structured because the entries in the table have multiple parents (row and column). For this reason, a hybrid and heterogeneous structure is used to model the document. The highest level structure of the TNT is an ordered tree. The objects found on this higher level structure are said to be in the *strict tree* portion of the TNT. Many different kinds of structures are defined to represent the leaves, which are called *free structure, not strict*, portions of TNT. The terminating nodes of any particular free structure may be defined to either atoms or transition nodes. Atoms are the actual terminating points in the TNT. A transition node is a terminating point for the structure that contains it and also a root for an enclosed strict tree structure (see Figure 3.1). In traversing a TNT from the root downward, one encounters alternating structures- first the strict tree, then a tree block enclosing a free structure and then perhaps a transition node that leads to another strict tree. This alternation is the reason for the choice of the term “TNT”, which stands for *strict tree - not strict tree* [26].

A prototype system, called the *pedTNT* has been developed to edit TNT [25]. The intention of the prototypical user interface was not to provide an "exact" representation of the document as it would appear on paper but a "sufficient representation" that gives a good intuitive feeling for what the document elements but that does not necessarily indicate what the exact details of the placement of these elements will be in other representations. The approach taken in the prototypical user interface is to provide a template-driven system for the creation and modification of a document, directed and constrained by Context Free Grammar. Manipulation involves a family of editors. One is called the generalised manipulator which is used for altering the strict tree portion of the TNT. A separate specialised editor is provided for each of the classes of user-defined terminals, that is the free structures.

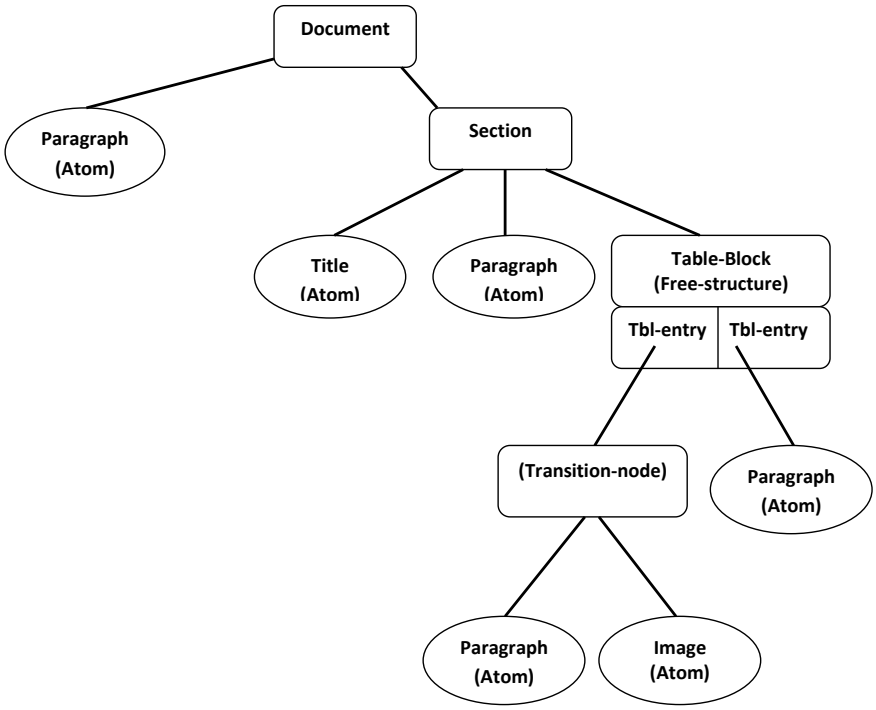


Figure 3.1: The logical structure of the TNT is a strict tree represented as an ordered tree, and free structures representing the leaves.

3.1.6 Open Document Architecture

The Open Document Architecture and Interchange Format (informally just ODA) [4] is an international standard document format. Its purpose is to facilitate the interchange of documents in a manner such as:

1. Different types of content can coexist within a document.
2. The transmission of the intentions of the document originator with respect to the logical and layout structure to the document recipient.

The ODA emphasised the document processing model modeled by Shaw, shown in Figure 2.9, but enriched it with the "interchange" activity. According to ODA standard the interchange activity is the process of providing a document to a receiving person or device, by means of data communication or by exchange of storage media.

The ODA document structure has a tree form. The only classification for the objects in the logical structure is that the object is either the root of the document, a basic logical object or a composite logical object. Logical object categories such as section, chapter, paragraph are application-dependent and can be defined by a document application profile using the object class mechanisms. The basic elements of the content of the document are called content portions or content elements. Characters are the content elements in the text content while picture elements are the content elements in images or graphics, etc.

ODA is considered as the first document format that explicitly defines the objects in the layout structure of the document. The following types of layout objects are defined in the ODA:

1. *block*: A basic layout object, corresponding to a rectangular area within the document physical structure. It contains a portion of the document content.

2. *frame*: A composite layout object corresponding to a rectangular area within the document physical structure. It contains either one or more blocks or one or more frames.

3. *page*: A basic or composite layout object corresponding to a rectangular area within the document physical structure. It is a basic object when it contains one or more content portions of the document content, while it is a composite object when it contains one or more frames or one or more blocks.

4. *page set*: A set of one or more pages and/or page sets.

5. *document layout root*: The highest level object in the hierarchy of the specific layout structure.

Figure 3.2 illustrates the relationships between the logical objects, layout objects and the content portions in the ODA. A basic logical object is associated with one or more content portions. A basic layout object is associated with one or more content portions. Any logical or layout object (basic or composite) is associated with zero or more content portions. In general, there is no one-to-one correspondence between logical objects and layout objects.

3.1.7 HyperText Markup Language

HTML [18] is a markup language derived from SGML and primarily used for webpages. It provides means to create structured documents by denoting structural semantics for text such as headings, paragraphs, lists, etc. Currently, HTML is the predominant markup language for webpages. In its latest revisions HTML 4 and HTML 5, content and layout are supposed to be separated. Unfortunately common webpages tend to mix both of them.

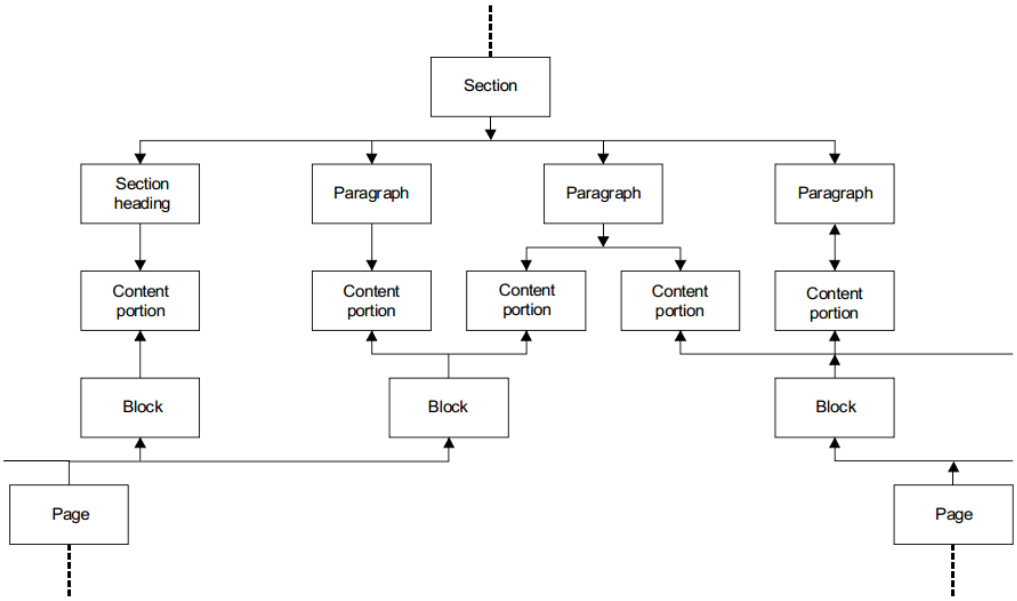


Figure 3.2: The relationships between the logical and layout objects in the ODA, based on [4]. No one-to-one correspondence exists between them.

3.1.8 Portable Document Format

Portable Document Format (PDF) [5] is a file format invented in 1993 by Adobe Systems for document exchange. PDF is used for representing two-dimensional documents in a manner independent of the application software, hardware and operating system. Each PDF file encapsulates a complete description of a fixed-layout 2D document that includes text, fonts, images and 2D vector graphics which compose the document. PDF has been widely adopted for long term storage and archiving.

PDF can be generated from any document processing software in order to get an accurate and fixed-layout representation of an original document. The fixed-layout representation or the electronic document is the result of the last process in the document production activities (see Section 2.3.1). Therefore, PDF focuses on the preservation of the visual appearance of a document and does not ensure the preservation of its physical and logical structures. To reveal and discover the logical and physical structures of the document, Document Image Analysis (DIA), a reverse engineering process, has to be applied on it (see Section 3.1.13).

3.1.9 Extensible Markup Language (XML)

The Extensible Markup Language (XML) [13] is a markup language for creating special-purpose markup languages. Hence, XML is a meta-language, capable of describing any kind of data [8]. The primary objective is to facilitate the sharing of both the format and data across different systems, in particular systems connected via the Internet. XML design purposes emphasise simplicity and generality over the Internet.

XML is a simplified variant of SGML. XML documents do not require an explicit DTD or a schema. Therefore, they are easy to code. There are several schema systems designed to assist in the definition of XML-based languages. DTD with its limited capability can be used to define XML-based languages. Two more expressive schema languages, in more widespread use, are XML Schema[1] and Relax NG[2]. According to Wikipedia[3]:" *As of 2009, hundreds of XML-based languages have been developed, including RSS, Atom, SOAP, and XHTML. XML has become the default file format for most office-productivity tools, including Microsoft Office, OpenOffice.org*" [65]. An XML document is valid if it has an associated document type declaration, either DTD or a schema. An XML document is well formed if it meets all the well-formedness constraints given in its specification [13].

XML has many strengths. First, it is a platform-independent format. Second, it is a human and machine-readable format. Third, it is self descriptive. Fourth, its syntax is simple and strict. Last but not least, it is a standard file format. However, XML suffers also from major weaknesses: its syntax is fairly verbose and partially redundant. Therefore, XML is rarely used to represent heavy file formats such as image, music or movie formats.

It is worth noting here that XHTML [45], which appeared in 2000, was essentially an XML-based rewriting of HTML, mapping all of HTML 4 features. It is a way to benefit HTML from XML with minimal effort, instead of replacing HTML with a new language for the World Wide Web. XHTML documents need to be well-formed in order to be parsed using standard XML parsers.

3.1.10 Open Document Format for Office Applications

The Open Document Format for Office Applications [63] (also known as OpenDocument or ODF) defines an XML schema for Office documents. Office documents include text documents, spreadsheets, charts and graphical-like drawings or presentation. The XML schema for OpenDocument is designed so that documents valid to it can be transformed using XSLT[4] and processed with XML-based tools.

OpenDocument defines two methods of document representation:

1. A single XML document that has `<office:document>` as its root element. It contains content, styles, metadata and settings in one single file.

2. A collection of files within a package. The content, styles, metadata and settings are stored in separated files.

When OpenDocument is represented as a package it takes the format of a ZIP compressed archive files and directories. These can contain binary content and benefit from ZIP's lossless compres-

[1]XML Schema : http://www.w3.org/XML/Schema (accessed August 12th, 2012)

[2]Regular Language for XML Next Generation: http://www.relaxng.org (accessed August 12th, 2012)

[3]http://www.wikipedia.org/ (accessed August 12th, 2012)

[4]XSLT is a language for transforming XML documents into other XML documents, http://www.w3.org/TR/xslt/ (accessed August 12th, 2012)

sion to reduce file size. OpenDocument benefits from separation of concerns by separating the content, styles, metadata and application settings into four separate XML files or different elements in the same document.

3.1.11 DocBook Document Format

DocBook [61] is a semantic markup language for writing structured documents using XML. It was originally intended for writing technical documents related to hardware and software, though it is by no means limited to them. Because its main structures correspond to the general notion of what constitutes a book and most of the document classes like Articles, Reports, etc. can be written using the book document class. DocBook roots are in SGML, where it was defined with a DTD. It was released as both an SGML and an XML vocabulary starting with V4.1. The V4.x versions of DocBook, like the versions that came before them, were also defined with a DTD. Starting with DocBook V5.0, DocBook is exclusively an XML vocabulary defined with RELAX NG and Schematron[5]. DocBook has been adopted by a large and growing community of authors. Hundreds (perhaps thousands) of organisations use DocBook for millions of pages of documentation in a wide variety of print and on-line formats.

DocBook elements can be divided logically into these categories:

1. *Sets*: Contain two or more books. It is the hierarchical top of DocBook. It is used for a series of books on a single subject.

2. *Books*: The most common top-level element in a document. The DocBook definition of a book is very loose and general. It consists of a mixture of the following elements:

   (a) *Dedication*: Always occurs at the front of a book.

   (b) *Navigational Components*: There are a couple of component-level elements designed for navigation: `toc` for tables of contents and `index` for indexes.

   (c) *Divisions*: Are the first hierarchical level below book. Divisions contain parts and references. A part contains components. Books can contain components directly and are not required to contain divisions.

   (d) *Components*: These are the chapter-like elements of a book.

3. *Components*: Are the chapter-like elements of a book or part: preface, chapter, appendix, glossary and bibliography. An article can also occur at the component level. Components generally contain block elements and/or sections.

4. *Sections*: They subdivide components.

[5]Schematron is a language for making assertions about the presence or absence of patterns in XML documents, http://www.schematron.com/ (accessed August 12th, 2012)

5. *Meta-Information Elements*: All of the elements at the section level and above include a wrapper for meta-information about the content. That element is named 'info'. The meta-information wrapper is designed to contain bibliographic information about the content (author, title, publisher and so on) as well as other meta-information such as revision histories, keyword sets and index terms.

6. *Block Elements*: They occur immediately below the component and sectioning elements. These are the (roughly) paragraph-level elements in DocBook. They can be divided into a number of categories: lists, admonitions[6], tables, figures, etc.

Listing 3.5 shows the structure of a typical book written in DocBook markup language. Additional content is required where the ellipses occur.

Listing 3.5: Structure of a typical book written in DocBook markup language

```
<book>
   <info>
      <title>Digital Media</title>
      <author>
         <firstname>Jone</firstname>
         <surname>Jack</surname>
      </author>
      <copyright>
        <year>2010</year>
        <holder>O'Rielly</holder>
      </copyright>
  </info>
  <preface><title>Foreword</title> ... </preface>
  <chapter> ... </chapter>
  <chapter> ... </chapter>
  <chapter> ... </chapter>
  <appendix> ... </appendix>
  <index> ... </index>
</book>
```

3.1.12 Office Open XML

Office Open XML, informally known as OOXML or Open XML, is an XML-based file format for representing word processing, spreadsheets, charts and presentations. On the one hand, the goal of the OOXML standard is to be capable of representing the pre-existing corpus of word-processing documents, spreadsheets and presentations that had been produced by the Microsoft

[6]There are five types of admonitions in DocBook: caution, important, note, tip and warning

Office applications. On the other hand, the goal is to facilitate extensibility and interoperability by enabling implementations by multiple vendors and on multiple platforms. OOXML and OpenDocument formats are two different standards, but they have a lot of things in common, since they are targeting office documents produced by WYSIWYG office applications.

3.1.13 Document Image Analysis Formats

Since the beginning of the computer era, electronic documents have gained an increasing role in humans' life more than the paper documents. Therefore, the existing paper documents are more and more replaced by their electronic form. With Document Image Analysis process, computers try to understand and recover the logical and physical structure of the paper documents and transform them into electronic versions. This process could drastically improve indexing and information reuse.

Document Image Analysis methods have traditionally been developed for paper documents acquired by scanners or cameras. However, structural document analysis became more and more useful when dealing with documents that already exist in electronic form. In practice, it is of highest importance to know the electronic document structure to ensure document access to screen readers and other adaptive equipments used by disabled people, such as synthetic speech or Braille output systems. Also it allows documents to be reedited, restyled or re-flowed. Thus research on PDF structure recognition, and more generally, electronic document recognition, has greatly inherited from Document Image Analysis and uses most of its methodologies [8].

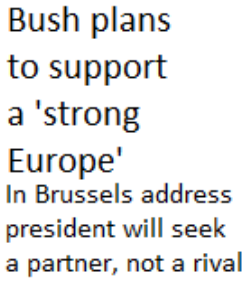


Figure 3.3: Paragraph of an electronic newspaper, taken from [10]

Two recent Document Image Analysis formats have been studied: A canonical and Structured Document Format (XCDF) [10] and the Optimized and Canonical Document Format (OCD) [9]. Both document formats are structured XML-based formats representing the logical and physical structures determined from the reverse engineering of the PDF documents using Document Image Analysis methods. It is however to be noted that it is not guaranteed that the logical structure will be determined for the documents. The logical structure could be revealed if the scanned or the electronic document has a well defined generic document structure and the image analysis methods applied are trained before with some documents from the same document class. To illustrate, using XCDF format, Listing 3.6 represent the result of document image analysis for a

paragraph in an electronic newspaper shown in Figure 3.3. You can see that XCDF could succeed in recovering the physical structure, while it gives no clue about the logical structure.

Listing 3.6: XCDF representation of a paragraph in an electronic newspaper, taken from [10]

```
<textblock x="81" y="374" w="145" h="137">
    <textline x="81" y="374.85" w="145" h="32">
        <token size="32" content="Bush"/>
        <token size="32" content=" "/>
        <token size="32" content="plans"/>
   </textline>
  <textline x="81" y="409" w="140" h="32">
       <token size="32" content="to"/>
       <token size="32" content=" "/>
       <token size="32" content="support"/>
   </textline>
   <textline x="81" y="444" w="116" h="32">
     <token size="32" content="a"/>
     <token size="32" content=" "/>
     <token size="32" content="  "/>
     <token size="32" content="strong"/>
   </textline>
   <textline x="81" y="479" w="105" h="32">
     <token size="32" content="Europe"/>
     <token size="32" content="  "/>
   </textline>
 </textblock>
```

3.1.14 EPUB

The EPUB [20] (short for Electronic PUBlication) standard format is an open eBook format developed and recommended by The International Digital Publishing Forum[7]. EPUB 3 is the current version of the EPUB standard. The EPUB is a reflowable document format. It also supports fixed-layout content. Cascading Style Sheets can be defined for different layout representations.

Every EPUB publication is a zipped file containing some XHTML or SVG documents. XHTML or SVG documents describe the readable content of the EPUB document and reference associated media resources (e.g. images, audio and video clips). XHTML content documents are defined by a specific profile of HTML 5 that requires the use of XML serialization in order to ensure that the content can be reliably manipulated and rendered. This HTML 5-specific profile has some EPUB-specific language constructs.

[7] http://idpf.org/ (accessed July 15th, 2012)

3.2 Document Format Analysis

This section is devoted to the review of the document formats presented in the previous section in the context of the five dimensions introduced earlier. One has to note that the XCDF and OCD document formats from the "document image analysis" family of formats are low-level document formats. They merely could success in recovering the scanned and electronic document structure and therefore they have never paid attention to any digital feature. Furthermore, the intention in modeling TNT was to integrate the flexibility found in the "generic coding" methodology and naturalness found in WYSIWYG. Therefore, Furuta did only pay attention to the linking feature. Last but not least, ODA only pays attention to the linking feature.

3.2.1 Linking

In the Scribe, GML, TNT and ODA document formats there is not much support for linking. Cross references inside the same document are possible to headings, list items, figures, tables and footnotes. However, references to content outside the current document are not possible.

SGML introduced a number of new concepts in particular linking functionality by extending GML. In SGML, arbitrary complex structures, such as hypertext, are supported via attributes representing various types of relationships between nodes and among documents. Listing 3.7 shows an example of how links are defined in SGML. A unique identifier is declared for the element to be pointed to, which in this example happens to be a figure. Then the hyperlink is declared with help of the `IDREF` keyword. The reference then points to the unique identifier declared for the figure element. The link is therefore a separate element which can be potentially enriched with additional meta-information. Links in SGML are unidirectional and furthermore an element has to be specifically declared with an IDREF attribute in order to support linking in SGML.

Listing 3.7: Example link in SGML

```
<!-- Element content --!>
<! ELEMENT figure (figbody, figcap) >
<!-- ELEMENT NAME VALUE DEFAULT -->
<ATTLIST figure id ID #IMPLIED >

<!-- ELEMENT CONTENT -->
< ELEMENT figref EMPTY >
<!-- ELEMENT NAME VALUE DEFAULT -->
<! ATTLIST figref refid IDREF #IMPLIED>
```

XML is, as its predecessor SGML, a meta-language. The XML meta-language offers no direct mechanism to create links. In 2001, the XML Linking Language (XLink) [23] was introduced. XLink offers sophisticated link support. Aside from so-called simple unidirectional links, XLink supports what is called *extended links*. The XML link shown in Listing 3.8 establishes a simple link from the `lab` element to the formal webpage of the WISE lab of the VUB. Browsers

are advised to interpret this link as HTML links when `xlink:show` attribute is omitted or it is set to be `new` or `replace`.

Listing 3.8: Simple XML link using XLink syntax

```
<lab xmlns:xlink= "http://www.w3.org/1999/xlink"
       xlink:type= "simple"
       xlink:href= "http://wise.vub.ac.be/">
   <title>Web & Information Systems Engineering (WISE) Lab</title>
   <professor>Olga De Troyer</professor>
   <professor>Beat Signer</professor>
</lab>
```

The XML extended link describes a collection of resources and paths between those resources. Each path connects exactly two resources. Any individual resource may be connected to another resource, two other resources, no resources, all other resources or any subset of other resources in the collection. It may even connect back to itself. Extended links are directed, labeled graph in which the paths are arcs, the documents are vertices and the labels are URIs. This mechanism allows for bi- and multi-directional links. The XML link shown in Listing 3.9 establishes an extended link with three vertices (locators) and four arcs. Figure 3.4 displays this extended link. Resources (locators) are represented as paper icons and the arcs are represented by arrows. However, it is an abstract way for understanding this type of links and it does not tell us anything about how a browser might present links to users and how users might choose which link to follow. One naive interpretation could be nothing more than the order on how to print these publications.

Listing 3.9: Extended XML link using XLink syntax

```
<!-- Web series publications, cs.xml document -->
<web>
  <series xlink:type= "extended"
            xmlns:xlink= "http://www.w3.org/1999/xlink">
  <author>Beat Signer</author>
<!-- locator elements -->
  <publication xlink:type="locator" xlink:label= "p1"
             xlink:href="http://dl.acm.org/citation.cfm?id=1784522"
    <title>As We May Link: A general Metamodel for Hypermedia Systems
    </title>
    <year>2007</year>
  </publication>
  <publication xlink:type="locator" xlink:label= "p2"
           xlink:href="http://dl.acm.org/citation.cfm?id=1929795"
    <title>What is Wrong with Digital Documents?</title>
    <year>2010</year>
  </publication>
  <publication xlink:type="locator" xlink:label= "p3"
          xlink:href="http://dl.acm.org/citation.cfm?id=1692464"
```

```
    <title>An Architecture for Open Cross-Media Annotation Services
    </title>
    <year> 2009</year>
  </publication>

<!-- arcs -->
  <next xlink:type="arc" xlink:from="p1" xlink:to="p2" />
  <next xlink:type="arc" xlink:from="p2" xlink:to="p3" />

  <previous xlink:type="arc" xlink:from="p2" xlink:to="p1" />
  <previous xlink:type="arc" xlink:from="p3" xlink:to="p2" />

</series>
</web>
```

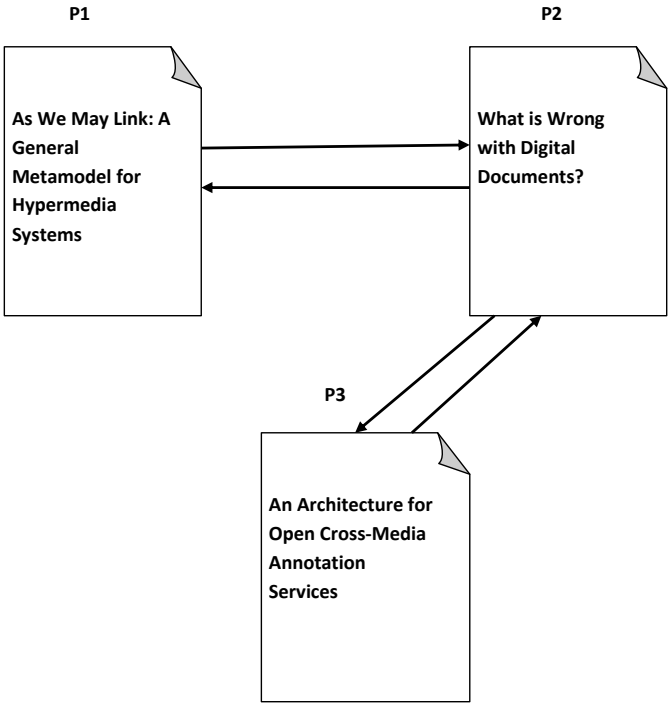


Figure 3.4: Example of an XML extended linking

The advent of XPointer [22] and XPath [66] made it possible to address specific points or ranges inside XML documents. Listing 3.10 shows a simple XML link pointing to the first `publication` child element of the `series` child element of the `web` root element of the cs.xml document shown in Listing 3.9

HTML was first designed as an application of SGML. As such, the definition of hyperlinks in HTML is technically limited to the capabilities of SGML. HTML introduced however a substantial difference to SGML in considering hyperlinks as basic building blocks of the language

and integrating them in its core. Interestingly, HTML has support for typed links through the <link> and <a> tags, which support a rel attribute. In the latest HTML version (HTML 5) the inclusion of RDFa [57] for metadata in the rel attribute was introduced, but the links are still unidirectional. XHTML has the same features as HTML 4 and therefore it supports only the unidirectional links. Support for metadata through RDFa was introduced in 2004.

Listing 3.10: Simple XML link with Xpointer and Xpath expressions

```
<thesis xlink:type="simple"
xlink:href="cs.xml#xpointer(/web/series/publication[position()=1])">
Fluid Cross Media Document Formats
</thesis>
```

The original LaTeX specification has no specific support for links. However, LaTeX has been extended over the years via various packages. One of these packages called hyperref, provides LaTeX with the ability to create hyperlinks within the document, thus providing the possibility to include interactive external links and turning internal references into hyperlinks. The links provided by this package are simple unidirectional links.

Links in office-like document formats have evolved along hypertext. In the case of OpenDocument, the <text:a> element is used to represent the links. This is an inherited feature from XLink, however OpenDocument kept only the "simple" specification of XLink links, and therefore only unidirectional links are supported. Furthermore, in OOXML hyperlinks are also available, however only in their unidirectional form. The target can either be an anchor inside the document or an URL. One thing worth noting is that a package in OOXML can also have a hyperlink attribute as a whole.

In DocBook, links can take four different shapes. Not specifically a link, an *anchor* is identifying a single spot in the content. It may serve as a target for cross references, for example, from a link element. The anchor element may occur almost anywhere. Anchors have no content and generally produce no output, as they are link targets. *Link* is a general type of hypertext element which references either an anchor or external resource. The XLink:href attribute has to be used with the links in the case of the link to external resources. The *XRef* element is used to cross-reference parts of the document. Xref is used for content reusability, unlike the link with anchor, Xref transcludes the referenced part. Finally, the *olink* element provides a mechanism for establishing links across documents, where ID/IDREF linking is not possible and URI-based linking may be inappropriate. Therefore, olink is application-specific and for example, connection to an Oracle DB can be done via an olink.

In PDF, links can either reference intra-document anchors or be a URL pointing to external resources. As such, links in PDF documents are typically unidirectional.

Besides standard, HTML-like links, EPUB 3 introduced a novel concept to reference arbitrary content within an EPUB document through the use of fragment identifiers. This mechanism, called EPUB Canonical Fragment Identifier (epubcfi) [56] allows to reference any position in an EPUB document, in a way similar to XPointer. Although it is unidirectional by nature, this mechanism is important in the context of transclusion and reusability, as shown in the next section.

3.2.2 Transclusion and Content Reusability

Transclusion as originally defined by Ted Nelson had little support until now [54]. None of the reviewed document formats supports transclusion as originally envisioned by Nelson. We will however consider two weaker variants, named *restricted* and *full* transclusion, which both support parts of the original transclusion concept.

The restricted form of transclusion, as it was defined by Krottmaier et al. [36], can be considered as a weaker variant of full transclusion, which itself is a weaker variant of transclusion as defined by Nelson. On the one hand, *full transclusion* allows a document to embed a subpart of another document in itself. As an example, consider a webpage which would include only a subpart of an image. On the other hand, *restricted transclusion* can be described as the inclusion of only the complete document into another document. For example, an image included in an HTML webpage is a case of restricted transclusion. Note however that this definition of transclusion still differs from the original definition of Ted Nelson, which establishes a bidirectional link between the documents allowing to give knowledge to the transcluded document that it is referenced by others.

Scribe has more than one form of the restricted transclusion. Using the `@Picture` command an image can be transcluded. If the text content is frequently used, then Scribe advises to write it once using the command `@String` and to give it a unique name. The text content can be reused by referring to its name. Finally, the document can be divided into smaller parts. Different parts can be referenced from a single file and will be embedded and rendered in the visualisation of the document.

In GML, transclusion can be defined in its restricted form. Indeed, GML has a Page Segments mechanisms, which allows to embed page segments in a given document page (which may be a picture or logo) by using the `(.si)` control word and referencing the name of the file containing the segment. Also, GML allows to embed entire files, like Scribe parts, with the `(.im)` control word. This control word causes the file whose file-identifier is given as the parameter to be processed as though the current file and the embedded file were a single file. Many different files can be embedded to form a single document.

SGML also supports a restricted form of transclusion through an entity reference feature in which a short name is given to text sections, so as to be able to refer to them when needed. Also parts of the documents that are stored in separate files can be embedded by using the same mechanism. Finally, the result of dynamically executed processing instructions, such as the instructions to retrieve the current date, can be embedded as parts of the documents.

XML inherits from these SGML features, but adds to them capabilities introduced by XLink, XPointer and XPath. The XLink behaviour attribute, `show`, is used to embed a referenced resource into another document, when it has the `embed` value. Also, if the target document has a mime type supported by XPointer, then addressing and embedding part of this document is possible. Therefore, XML supports the full transclusion.

In HTML, transclusion has always played a major role, through the inclusion of images, audio or other HTML documents in webpages. This in turn allowed the development of plug-ins which enriched HTML documents, as was done with flash plug-ins. However this form of transclusion still is a restricted transclusion. XHTML allows basically the same restricted form

of transclusion that HTML 4 does. It appears that HTML 5 is going to slightly enhance the current situation by allowing users to specify more precise information on the class of elements included in HTML 5 documents. Also, it has explicit support for embedding multimedia documents through `<audio>` and `<video>` tags. But still there is no steps taken towards the full transclusion feature.

LaTeX supports also a restricted form of transclusion, through the use of various packages. Including images for example uses the `graphics` package. Including multimedia files makes use of the `movie15` package. The `pdfpages` package allows to include PDF documents in the output. Finally, the package `attachfile` enables to attach external documents to the generated PDF file, similar to how documents can be attached to an email. Furthermore, a LaTeX document can be divided in multiple individual files, like Scribe and GML, with help of `include` and `input` commands.

In OpenDocument, full transclusion is supported through XLink. Indeed, the XLink behaviour attribute `show` is used to embed a referenced resource into another document, as it is presented above, is also supported by the OpenDocument specification.

OOXML supports restricted transclusion with inclusion of images or multimedia documents. One has to note however that it is advised to include embedded documents such as pictures into the package of a document. This mechanism limits however the potential for transclusion, as only a local copy of the embedded file is considered. Please note that OOXML supports a case of full transclusion, but restricted only to other OOXML documents. Parts of a spreadsheet can for example be included in another document. Also the reference can either be internal or external to the package.

EPUB 3 supports restricted transclusion through HTML 5's `<audio>` and `<video>` tags, as well as inclusion of SVG documents and bitmap images. However, as for OOXML, the underlying philosophy of providing EPUB 3 documents in the form of a self-contained package contradicts with the ultimate goal of transclusion.

PDF introduced the concept of a collection of file attachments whose intent is "*to present, sort, and search collections of related documents, such as email archives, photo collections, and engineering bid sets*" [5]. However, these documents are collected within the PDF file, with no possibility to reference external documents as part of such collection. In general, PDF documents are supposed to be self-contained units, with reference to outside elements only through unidirectional links. PDF therefore only supports the restricted form of transclusion.

DocBook supports three forms of restricted transclusion. *Physical divisions* allow to divide a document into separate files. On the other hand, the *XRef* element allows embedding a text block by referring to it. Finally, the *olink* element provides DocBook with the ability to retrieve information from an external source, for example a database.

3.2.3 Versioning

Few of the introduced document formats offer support for versioning. HTML, XHTML, Scribe, GML and PDF have no support for versioning. SGML is a peculiar case, as versioning is not supported explicitly. However, by using *marked sections*, one can achieve a similar result, as is illustrated in Listing 3.11. In this example, a document contains elements related to two different

versions. The different versions are included in the same document through marked sections, which are denoted by identifiers selected by the user. Then, using the keywords `INCLUDE` and `IGNORE`, a specific version of the document can be displayed. The output of Listing 3.11 would look like:

> Text for version 1
> Common text for both versions
> More text for version 1

Listing 3.11: Versioning in SGML

```
<![%v1; [text for version 1]]>
<![%v2; [text for version 2]]>
Common text for both versions
<![%v1; [more text for version 1]]>
<![%v2; [more text for version 2]]>

<!ENTITY %v1 "INCLUDE">
<!ENTITY %v2 "IGNORE">
```

Surprisingly, XML dropped this functionality of SGML and marked sections as in SGML are not allowed in the document itself. Only the DTD can contain such marked sections. As no support for versioning is integrated in the meta-language itself, XML versioning is considered as a complex problem, solved by using external frameworks as proposed by Thao et al. [58].

In EPUB, the only support for versioning is offered through "publication identifiers". There are two types of publication identifiers: *unique identifiers* and *package identifiers*. The unique identifiers are defined as to be used to identify a document, while package identifiers allow a finer grained identification by allowing an identification number to refer to a newer version of the same document, while retaining the same unique identifier. However this only adds the functionality for versioning external to the document and intra-document versioning is not supported.

LaTeX has no integrated support for versioning. However, since LaTeX documents are plain text files, LaTeX authors can (and do) use versioning control software such as the ones described in Section 2.1.3.

OOXML supports versioning with the *track changes* mechanism of the Microsoft Office suite in mind. Thus for WordProcessingML and SpreadSheetML types of documents, revision-specific elements are declared. It is however to be noted that these revision elements are rather heterogeneous from one type of document to the other, with revision elements declared at the paragraph level in WordprocessingML and the workbook level in SpreadsheetML.

In OpenDocument, versioning is supported through the `<text:tracked-changes>` (intended for text processing) and `<table:tracked-changes>` (intended for spreadsheets) elements. The same as OOXML, the underlying philosophy is inspired by the "track changes" mechanism of the Microsoft Office suite in mind. The two elements provide information about change tracking for insertion, deletion and format change. They also provide information about the creator of the change, when it was changed as well as other information such as the deleted content. They also mark the start and the end of the changes that have occurred.

DocBook is able to keep a *revision history* of all the revisions which occurred in a specific document through the use of the <revhistory> element. However, this element just provides metadata information about the different revisions that a document went through, but does not provide explicit information about which parts of the document were changed and how. As such, versioning as defined in Section 2.1.3 is not supported by DocBook.

3.2.4 User Rights Management

Scribe has no support for this feature. <meta name="author" content="..."/> is used in the different HTML versions as well as XHTML to support the specification of the author of the document. However, user rights management as we consider it goes beyond than merely describing the author of a document. First and foremost, user rights management should describe who has reading/editing rights on a document, as well as specify maybe explicitly who has not these rights. Second, user rights management should help track the history of modifications, in conjunction with versioning.

GML, SGML and XML are meta-languages, and as such, can be used to model user information. For example, XML User Profile (XUP)[8] is based on XML, as its name implies. However, as languages, they have no built-in support of user rights management.

DocBook does provide <author> and <authorgroup> elements for specifying the authors of a document, akin to HTML's meta author tag. LaTeX provides a similar mechanism using a \author command. Once again though, it does not qualifies as user rights management.

EPUB 3 documents allow to encapsulate in the package a <rights.xml> file containing information about digital rights management. However, this is an optional requirement in the specification, and there is no specific *Digital Rights Management (DRM)* system is provided. It is also good to mention that an optional metadata.xml file can be provided as well in an EPUB 3 document, although no specification is provided for this file either. Finally, EPUB 3 allows the specification of Dublin Core metadata[9] terms such as <dc:creator> or <dc:contributor>. All in all, EPUB 3 is a format which leaves a lot of open space for providing user rights management.

In OpenDocument, the author of a document is defined either through the Dublin Core metadata or through <text:author-name>. Other than that, OpenDocument can include XML digital signatures as defined by the W3C[10].

For specifying author information, OOXML recommends following the Dublin Core metadata format. Author information can be specified at a fine-grained level, linked to the "track changes" mechanism described in the previous section. OOXML also allows the specification of access rights using <w:documentProtection>, however these access rights are not specific to a user or group of users, but just stipulate that a document is for example "read only". OOXML format however defines a set of extended properties, among which the possibility to

[8] http://xprofile.berlios.de/ (accessed July 16th, 2012)

[9] http://dublincore.org/ (accessed July 23rd, 2012)

[10] http://www.w3.org/TR/xmldsig-core/ (accessed August 9th, 2012)

attach XML digital signatures as well as the indication of a security level (with the interesting distinction between "recommended to be opened as read only" and "enforced to be opened as read only").

Metadata in PDF files allow to specify a range of information, among which authors and creators. However, the real strength in user rights management in the PDF format lies in its support for digital signatures. Digital signatures in PDF allow to ensure that a document cannot be read and/or modified. PDF supports a range of different encryption algorithms, among which RSA and DSA. Permissions to create, delete, modify, copy, import and export can then be assigned.

As can be seen, none of the reviewed formats contains any form of integrated user rights management. At best, digital rights management and digital signatures are supported. EPUB 3 is an exception, as it made provisions for the future inclusion of extended user rights management in its specification. For now, the specification of EPUB 3 just provides these facilities as an optional requirement, which is already better than all the other reviewed formats. However, any form of fine-grained authorisation rights will need to be addressed, either in the document format itself or as meta-information attached to it.

3.2.5 Adaptation

HTML 4 and XHTML are the typical cases of reflowable document formats, able to adapt their presentation to the layout constraints enforced by a device. Furthermore, the `<style>` tag allows to specify a `media` attribute, with the following list of official values: `screen, tty, tv, projection, handheld, print, braille, aural, all`. When linked to Cascading Style Sheets (CSS)[11], the specification of a media output allows to link one style sheet per media type, effectively adapting the appearance of the document to the output. In HTML 5, the `media` attribute has been extended to be applied also to the `<a>` and `link` tags, in order to be able to specify the media type of a linked resource.

SGML, GML and XML as meta-languages can be used as a basis for content adaptation. However none of these languages has support for content adaptation directly embedded into the language description. This means that content adaptation implemented based on one of these meta-languages has to be declared explicitly by the developer.

In LaTeX, as content and output are clearly separated, one can specify a different stylesheet for different devices or contexts.

In PDF, the only notable steps towards content adaptation are some features introduced in the document format towards the good rendering of a document through text-to-speech or braille. Pinkney et al. [46] proposed to pre-generate PDF documents at different screen sizes as a way to adapt to different sizes. This approach needs however to know in advance the classes of devices for which a document has to be generated.

Scribe, OOXML and DocBook have no capabilities for content adaptation. OpenDocument has some restricted capabilities for content adaptation through a `<text:hidden-text>` and a `<text:hidden-paragraph>` elements, which will show or hide a paragraph based on a

[11] http://www.w3.org/TR/CSS2/media.html (accessed July 17th, 2012)

<text:condition>. However, this only applies to text processing documents. Nothing is provided for spreadsheets, presentations or graphics.

In EPUB 3, adaptation to context (called "content accessibility" in the specification) is supported in a number of ways. First, EPUB supports WAI-ARIA[12] for accessibility. WAI-ARIA stands for 'Web Accessibility Initiative - Accessible Rich Internet Applications'. It is a technical specification published by W3C that specifies how to increase the accessibility of webpage, in particular, dynamic content and user interface components that were developed using the last web technologies recommended by the same party. Second, EPUB 3 documents are centered around dynamic layout. The content is supposed to be formatted on the fly, depending of the device capabilities. Furthermore, EPUB 3 provides fall-back capabilities if a device is not able to render a document in its native format. Finally, text-to-speech is also supported.

3.3 Towards the Ubiquitous Computing Age

Table 3.1 summarises the analysis of the different document formats presented in the previous sections. Note that the mark (✔) means that the intended document format has a limited support for the feature. The table shows a remarkable support for unidirectional linking as well as the restricted version of transclusion. However, support for more advanced features is limited or non-existent for input-side content adaptation or user rights managements. In more details, bi- and multi-directional links have been sparsely explored by XLink. Full transclusion has also been explored by XML technologies, namely the conjunction between XLink, XPointer and XPath. As OpenDocument supports these standards, it can potentially benefit from full transclusion. Versioning is supported in OpenDocument and Office Open XML using the *track changes* mechanism introduced by the Microsoft's Office suite, which forced both formats to include fine-grained modification tracking. SGML contains a set of elements and attributes which also enables it to support a top-down or version ID centered form of versioning, in contrast to the bottom-up or atomic change centered approach offered by OpenDocument and Office Open XML. User rights management has been approached only from the digital rights management side by EPUB 3 and PDF. Even if this is a form of authorisation granting, no facilities for user profile management are provided. Finally, for the content adaptation, we can differentiate between output and input-side adaptation. On output-side adaptation, HTML 4 and HTML 5 as well as XHTML documents have the potential to adapt their layout to different devices, especially when linked with Cascading Style Sheets. Finally, EPUB 3 provides numerous capabilities for content adaptation.

Members of the "Meta-languages documents family" support fundamental features such as versioning (for SGML), multi-directional linking and transclusion (for XML). Support for the other features either have not been considered or can be accessed through an application of the meta-language (as for example XUP for user profiles). However, with the coming age of ubiquitous computing, features which could until now be considered as relevant only in certain

[12] http://www.w3.org/TR/2010/WD-wai-aria-20100916 (accessed July 17th, 2012)

Format	Links		Transclusion		Versioning	Users Management	Adaptation	
	Uni	Bi	Rest	Full			Output	Input
Scribe	✔	✗	✔	✗	✗	✗	✗	✗
GML	✔	✗	✔	✗	✗	✗	✗	✗
LaTeX	✔	✗	✔	✗	✗	✗	(✔)	✗
SGML	✔	✗	✔	✗	(✔)	✗	✗	✗
TNT	✔	✗	✗	✗	✗	✗	✗	✗
ODA	✔	✗	✗	✗	✗	✗	✗	✗
HTML 4	✔	✗	✔	✗	✗	✗	✔	✗
HTML5	✔	✗	✔	✗	✗	✗	✔	✗
XHTML	✔	✗	✔	✗	✗	✗	✔	✗
PDF	✔	✗	✔	✗	✗	(✔)	(✔)	✗
XML	✔	✔	✔	✔	✗	✗	✗	✗
OpenDocument	✔	✗	✔	✔	✔	✗	(✔)	✗
DocBook	✔	✗	✔	✗	✗	✗	✗	✗
OOXML	✔	✗	✔	(✔)	✔	✗	✗	✗
DIA formats	✗	✗	✗	✗	✗	✗	✗	✗
EPUB	✔	✗	✔	✗	(✔)	(✔)	✔	✗

Table 3.1: Summary of investigated document formats

application domains will need to be supported in the core of the language.

On the opposite side compared to meta-languages, "print-oriented documents family" have very pragmatic roots, evolving along with the needs of their clients. The support of digital rights management is a striking example of this pragmatic approach. However, these languages suffer from their pragmatism in adding layer upon layer of complexity, when support for new features is needed. The 5585 pages long specification [6] of Office Open XML is a testament to the never-ending growth of complexity of this family of formats.

The strength of decoupling content and presentation in the "document preparation family" introduced a major weakness in this family of formats. Indeed, multiple features considered in this review ask for a tight link between content and presentation. Therefore, introducing such features in formats such as LaTeX or DocBook might be challenging. The non-standard support of simple unidirectional links in LaTeX is a good illustration of this problem.

Interestingly, the "world wide web family" shares strengths and weaknesses with the three other families presented above. It brings its own strengths too, in particular in the field of output-wise content adaptation.

Finally, EPUB 3 is the most interesting format from the point of view of our analysis. It supports unidirectional links as well as a XPointer-like fragment addressing mechanism. It is the only format to provide the basic elements for user rights management. It uses the most advanced output-wise content adaptation of all the considered formats. Very basic versioning-friendly information is available. Incidentally, EPUB has been created as a digital device-oriented re-

flowable document format. Print is considered as one potential output, among others. Hence, *EPUB can be considered as one of the first formats of the coming age of ubiquitous computing*.

However, EPUB 3 is still far from being the "perfect" document format for the coming age of ubiquitous computing, as shown in Table 3.1. Towards this "perfect" document representation, multiple tracks can be followed. A first track is to create a brand new document format considering a full set of requirements dedicated to ubiquitous computing. Even if this track first appears as the most evident one, it is also the most prone to failure. Indeed, introducing a brand new document format without clear motivations and strong support for it would almost certainly end up in poor acceptance. Most "new" formats are in fact extensions of one or multiple standards to guarantee easier acceptance. This means they also inherit from the weaknesses of their ancestor formats. The track followed by the new formats such as EPUB, OpenDocument or Office Open XML has been to package multiple sub-documents in already well established formats. A second track which could be followed is to extend existing document formats with the features needed for ubiquitous computing. In fact, this track will almost certainly be followed by a number of the formats considered in this article in the coming years. Obviously, this will primarily worsen the complexity of these formats, while answering the needed features one step at a time.

We believe there exists a third track. This track would consist in adding a new layer of complexity not on top of existing document formats, but below them, in an intermediate layer. Specifically, as document formats will adapt to the coming age of ubiquitous computing, documents will more and more be stored on servers and database, in place of local file systems. This will in turn allow to access these documents remotely through RESTful interfaces or web services. The ubiquitous computing and Internet of Things communities are currently examining architectures for middlewares allowing storage, localisation and addressing of data in heavily distributed and heterogeneous environments [32, 47]. Documents will in turn be part of this Internet of Things either as complete entities, or decomposed and distributed over multiple locations. For example, different snippets of a same document can be distributed over multiple locations. Document formats such as EPUB, OpenDocument or Office Open XML are already composed of a package that can be decomposed into multiple sub-documents.

Based on some of the concepts presented above, Boyer et al. [12] already demonstrated the interest of decomposing OpenDocument documents into interactive web documents which can then be addressed through a REST interface. Krottmaier et al. [36] also proposed a similar server-based approach to implement transclusion. We, too, consider that such a middleware-based approach should be further explored and exploited. Indeed, by embedding meta-information which would be common to information pieces, advanced features such as the ones analysed in the previous sections could be supported. Bidirectional and multidirectional links should be first-class citizens and would be used to glue together subparts of a document. They would also be used for transclusion. As each information piece would be identified by a unique "fingerprint" identifier generated based on its content, transclusion as originally defined by Ted Nelson could be supported. Müller et al. [40] already demonstrated how integrating versioning as a layer between the file system and the documents offers multiple advantages. User profiles and rights could also be embedded at this level. Finally, as documents would consist of a collection of snippets of information instead of one bulky file, adaptation of content could also be supported.

This middleware should itself be built on a very clean metamodel. The metamodel would

be responsible for managing aspects related to distribution, versioning, user rights managements and possibly content adaption-related information. When the metamodel would not be able to provide enough information explicitly, it should provide hints implicitly that the document formats could follow. The different document formats would in fact be modeled on top of the metamodel, at different levels of granularity. For example, a document format tightly linked to the metamodel would map its elements to individual components of the metamodel. With such an approach, all features supported by the metamodel would also be supported by the document format. For document formats which would not directly support the metamodel, a document would be embedded into one component of the metamodel, still allowing it to support some of the features offered by the metamodel. Finally, the metamodel information could be stored either in the middleware itself, or as a separate metadata information next to the document itself, when the document would not be recorded in the middleware.

This metamodel in the middleware could be based on the RSL metamodel. As mentioned in Section 2.2, RSL is centered around the *resource* describing content such as a piece of text, the *link* which can be a one-to-one, one-to-many or many-to-many bidirectional link, and the *selector* attached to a resource and allowing to refer to a part of it. Aspects like distribution, user management and adaptation are supported by the RSL. Therefore, in the following chapter we introduce the metamodel for future fluid cross-media document formats as an extension of the RSL metamodel.

Chapter 4

Fluid Cross-Media Document Format Metamodel

In this chapter, we introduce the *Fluid Cross-Media Document Format* (FCMD) metamodel, which is based on the RSL metamodel. The purpose of modelling FCMD is to provide a metamodel that is general enough for all document models, and therefore any document format can be modelled on top of it by tightly or partially mapping its elements to the metamodel elements.

We will discuss how the FCMD metamodel accommodates to various logical document models. Then, how FCMD metamodel will support the digital features introduced earlier will be discussed. Interestingly, the RSL metamodel has support for aspects related to distribution, linking, user rights management and content adaptation. Therefore, the FCMD metamodel will inherit all of them and the main concern will be on how to support the other features.

4.1 Logical Document Objects

Most of the document formats and document models provide support for objects such as images and text blocks. However, few of them support extra objects. For example, PDF supports 2D vector graphics, HTML5 supports the inclusion of video and audio objects instead of using a browser plug-ins mechanism. The LaTeX and TNT document model in addition to other document formats support the inclusion of mathematical equations. Nevertheless, for ubiquitous computing, all the aforementioned objects are not enough. The document metaphor has to be extended to apply to dynamic objects derived from the enclosing environment, for example, information obtained from a database or information from sensors (e.g. temperature or accelerometer data) [27].

In the RSL metamodel, any object can be addressed via the abstract *entity* type, therefore an extension with three subtypes was introduced, as seen in Section 2.2.1. The *resource* subtype has a great relevance in the topic addressed here. A resource is an abstract concept representing any resource type that exists in hypermedia systems. Some of the potential object types in the document formats are text, images or crosslets for dynamic objects [2]. In the FCMD metamodel, the resource concept has been extended in order to cover potential object types, as shown in Figure 4.1.

The object types introduced in the metamodel are meant to be general enough. Therefore, specific document formats are enabled to specialise them according to the granularity level allowed. For example, a `text block` can be a `paragraph` in a specific document format, a `sentence` or even a single `character` in others.

As shown in Section 3.2, none of the reviewed document formats supports transclusion as originally envisioned by Nelson. The weaker forms of transclusion we considered, the restricted and full transclusion are supported in some document formats. Therefore, as part of the goals of this master thesis, we propose a model that supports transclusion as originally defined by Ted Nelson. The original transclusion was based on the following concepts:

1. The hypermedia model must have the support for addressing parts in each resource.

2. Bidirectional links have to be supported.

As a first step towards Nelson's transclusion, the FCMD metamodel inherits the *selector* type of RSL metamodel. The selector concept has been introduced in the RSL as the second subtype of the entity. The selector is an abstract concept representing a part of a resource. The selector in FCMD metamodel has been extended in order to support selectors for each type of resources. In the next section, the second step towards Nelson's transclusion is discussed.

The concrete selector subclasses are meant for specific object types (i.e. subtypes of the resource type in the FCMD metamodel). The selector contains information about the selected part of its related object type resource. For example, an *image selector* can have the coordinates of the selected part of the image, a *text block selector* can have the start and end indices of the selected part, while a *video selector* may define the selector based on a time span or a shape within the video. Figure 4.2 shows how the FCMD metamodel extends the selector subtype in order to accommodate for different object type resources defined in the same metamodel. A video selector is defined for the video object type resource, a text block selector is defined for the text block, etc.

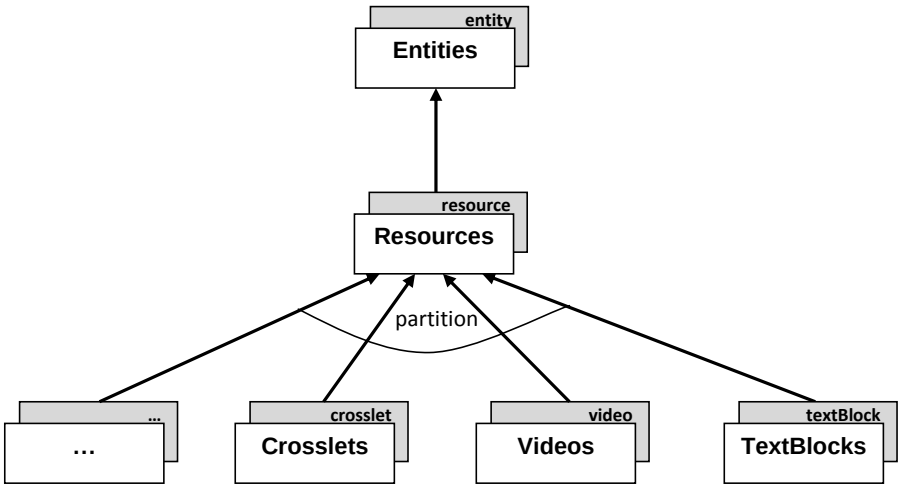


Figure 4.1: Concrete resources (object types) in the FCMD metamodel

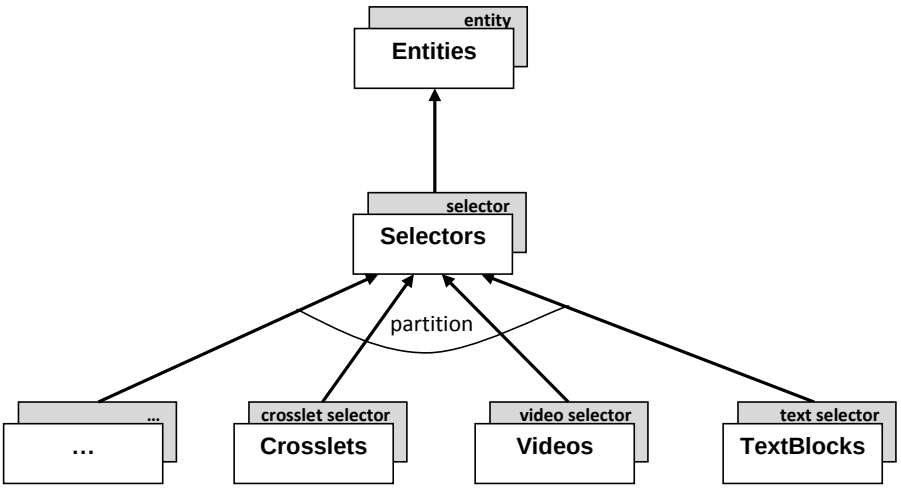


Figure 4.2: Concrete object types' selectors in the FCMD metamodel

4.2 Links

In the document formats reviewed in the previous chapter, the support for advanced linking features was limited. Bi- and multi-directional links have been sparsely explored by XLink. Surprisingly, OOXML and OpenDocument document formats degrade the linking to simple and unidirectional links even though XLink is used.

Document formats have to support advanced linking features in order to face the challenges of the ubiquitous computing environments. Without many-to-one links, the distribution of documents over multiple devices as well as the distribution of information over different media types will not be possible. One-to-many links have a quite important role for adaptation. Without bidirectional links, content reuse and transclusion will not be possible. Therefore, the intention of the FCMD metamodel is to support all advanced linking features required for the upcoming age of ubiquitous computing.

As presented in Section 2.2.1, RSL introduced the links as the third subtype of the entities collection. Links are treated as first-class objects [55], which means that they are not treated as simple metadata but can be used for other purposes like structural links. *A link* is a concept that logically connects entities, and is directed (bidirectional). Therefore, the links collection was extended by two subtypes: structural and navigational links. Structural links in the context of the FCMD metamodel are discussed in the next section. Navigational links are the same as WWW links, but with extended capabilities. They can have multiple targets and multiple sources. Hence, all the needed linking features for the ubiquitous computing are supported. Thus the FCMD metamodel inherits the navigational links from RSL without any modification, as shown in Figure 4.3. This also implies that transclusion as defined by Nelson is fully supported in RSL, and therefore it is also supported by the FCMD metamodel.

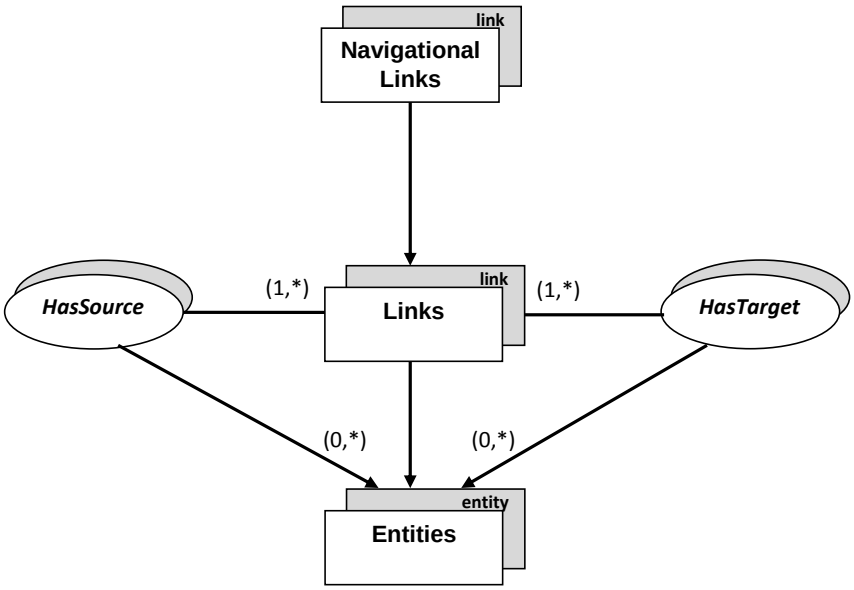


Figure 4.3: Navigational links in the FCMD metamodel are the inherited from the RSL without any modification.

4.3 Metadata

Attributes (properties) are one of the main constructs in the secondary logical structure in any document format. Properties are used to denote semantic information that is not provided by the primary logical structure. In the FCMD metamodel, each entity can have many attributes, thanks to the use of the *properties* collection introduced in RSL. For example, a text object type can define properties such as font-size, emphasised and font-type with integer, string and boolean values respectively.

Metadata has multiple advantages aside from the naive description and semantic information about the entity that we consider. First of all, search functionality can drastically be enhanced. To illustrate this, imagine a teacher who would like to upload a video on *Youtube*[1]. In their video, three "Human Computer Interaction" topics are discussed: usability, localisation, and accessibility. Using Youtube tagging functionality, which semantically describe the content, they entered three tags with the three topic names. Later, one of the teacher's student wants to watch what their teacher uploaded on Youtube about localisation. Thus, they search using the keyword "localisation" using video search functionality provided by Youtube. The first item in the result is their teacher's video. In order to watch only that specific part about localisation, they must waste some time searching for it inside the video. This is not the case with the FCMD metamodel based documents, thanks to the selectors in the metamodel. One can define three different selectors inside the video with the start and end time span for each topic. Each selector has a property

[1] http://www.youtube.com/ (accessed July 26th, 2012)

called “name” with the name of the topic as a value. Once you search for a specific topic, the underlying system will automatically lead you to the beginning of that topic.

Another advantage for the metadata is the typing of links and semantics. Semantics can be attached to any link by adding some metadata. A semantic link has the purpose to link data in order to semantically unify, organise and view it in different aspects. For example, one easily can define a type of a link as `alternate`, `bookmark` or any other type found in the attribute `rel`[2] of HTML5. Semantic links are quite important for the reader to know the relation between the linked objects. Moreover, document production systems can be adapted differently for each type of a link. For example, if a link is of type ‘author’ then one possibility is to present the name of the author and their photo when the user hovers the mouse over it. Aside from the importance of semantic links to the reader, the authors for such a link can benefit from other resources and information that is already written and explained about specific topics. We can imagine an author is writing a chapter about ‘time complexity’ in order to make a comparison of the time complexity for data structures in a book called ‘Data Structures and Algorithms’. It would be easier for the author if they can just suggest and provide the user with some mathematical resources that explain the linear equations, logarithmic equations and other topics instead of writing some sections to explain them. In such a way, their work will have more acceptance, since it is linked to books or articles specialised in mathematics. Besides that, the readers will refer to the link targets when needed.

4.4 Logical Document Structure

Document formats introduced in the previous chapter have different logical structures. As shown in Section 2.3.2, aside from the different levels of granularity for the atomic objects within these models, there is a difference in the relations allowed among those objects. The object types resources introduced in the FCMD metamodel were general enough to overcome the first difference by allowing the various document formats to specialize them to any level of granularity. To overcome the second difference, besides the deep understanding of different document models, we have to understand the generic structures found in the most popular document classes, like report, book, etc.

4.4.1 Document Classes

The physical representation of document classes are different, even some chapters or sections in the same document class can have different representations. It is however to be noted that this is not a problem and it is not related at all to the logical structure of a document. *“Thanks to the universal physical structure formalisms, whereas all the document class’s physical structure can*

[2] `Rel` in HTML5 is attached to the `link` tag. It specifies the relationship between the current document and the linked document. the user can choose its value out of predefined ones. The predefined values are: `alternate`, `archives`, `author`, `bookmark`, `help`, `external`, `first`, `icon`, `last`, `license`, `next`, `nofollow`, `no-referrer`, `pingback`, `prefetch`, `prev`, `search`, `sidebar`, `stylesheet`, `tag` and `up`.

be described" [8]. Therefore, the physical representation is (and should be) totally independent from the logical structure. It has its own representations, either with *a command-stream image description language* or with *page description language (PDL)* [50].

The generic logical structure of a *report* is divided into numbered chapters, sections, subsections, table of content, list of figures and list of tables. A *manual's* generic structure extends the report's generic structure with an index. An *article's* generic structure is a sectioned document, without chapters. *Sections* are the highest level of sectioning in the articles. A *book's* generic structure is almost equivalent to the report's generic structure excluding some extra elements, such as preface, colophon, index, etc. Besides these four document classes, *letters*, *slides*, and *newspapers* are considered.

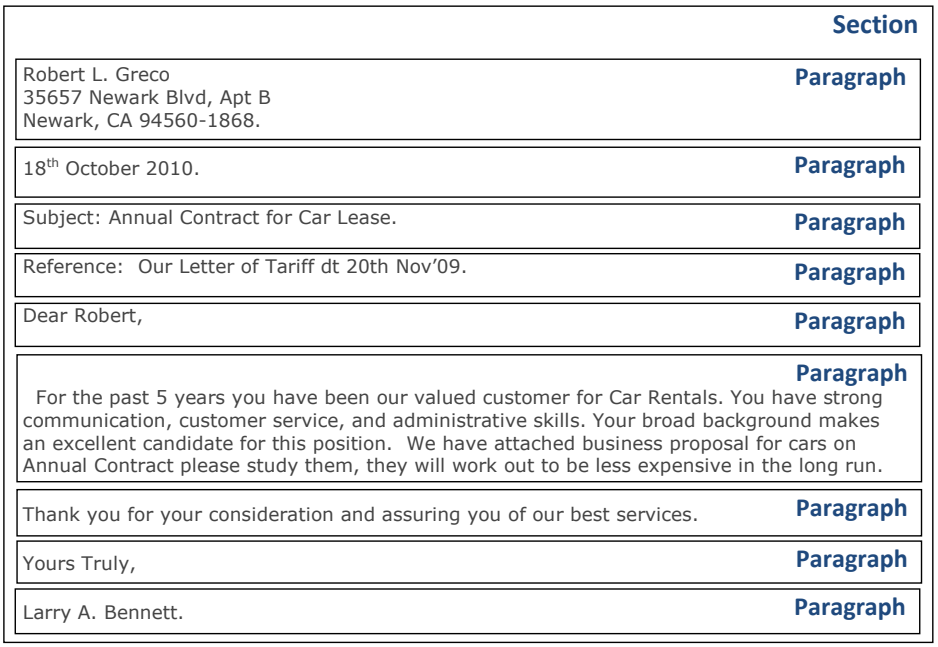


Figure 4.4: A section representation is one possible view of the letter document class.

The generic structure of a *letter* can be viewed as a section, containing multiple paragraphs (e.g. sender's address paragraph, receiver's address paragraph, date paragraph, main body paragraph, etc.). Figure 4.4 shows a business letter represented as a section. One can see the limitation of such a view, since it represents the content as a sequence of paragraphs and no explicit relationship between them is specified. It would be better if the logical letter structure shows the relation between various parts of the letter by grouping the header paragraphs, body paragraphs and the tail paragraphs. Therefore, a more suitable view of the letter is as a chapter containing more than one section. Each section represents a part of the letter. Figure 4.5 shows the same business letter, but represented as a chapter.

Manual, Report and Book contain chapters and specialised parts such as appendices, those are chapter-like elements, since they contain sections and subsections. Prefaces, acknowledgments and dedications can also be considered "section-less" chapters. It is however to be noted that

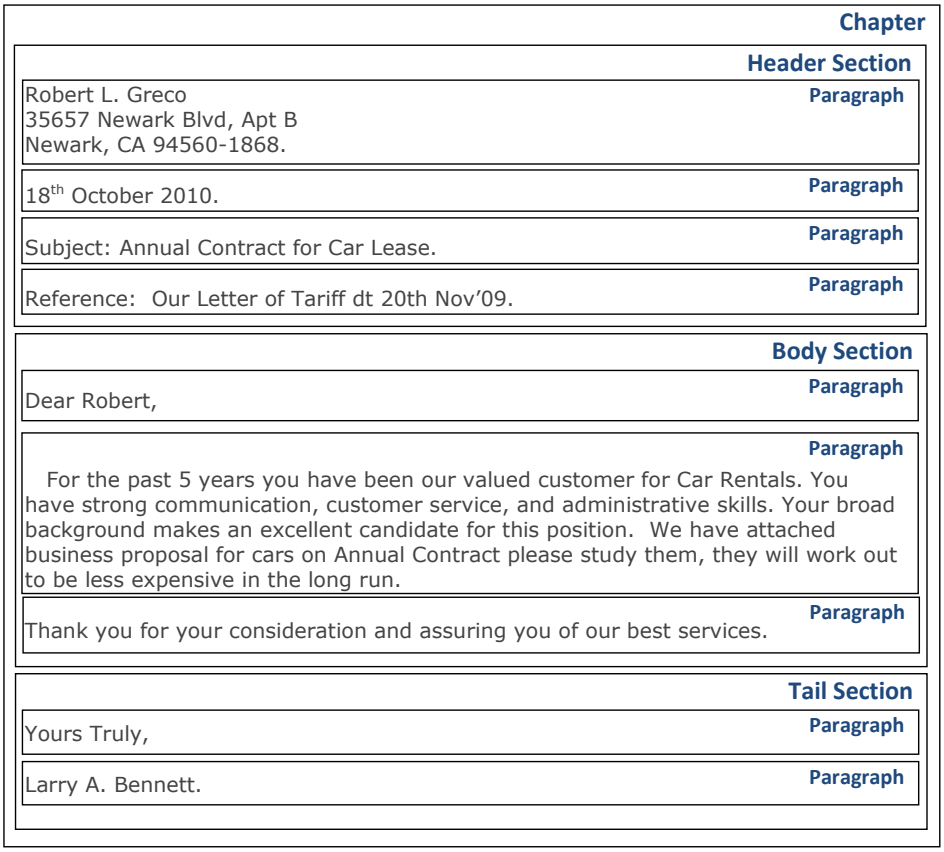


Figure 4.5: A chapter representation of the letter document class is more suitable than the section representation.

the differences between chapters and chapter-like elements are at the physical representation and not on the logical structure level. Therefore both can be considered the same from the logical-structure point of view. As a conclusion, we could say that the three document classes contain classic chapters and chapter-like elements and a representation of a collection of chapters is enough to represent them.

Articles can be considered as a single chapter (as a chapter would be in a book) with some sections. This chapter representing an article has authors, title, date and some other meta-information. Thus, a representation of a chapter is enough to represent the generic structure of an article.

A slide of a presentation having more than one topic, with a title for each one of them, can be viewed as a chapter with more than one section. However, in a collection of slides, such a view is not appropriate, because some of the slides are the continuation of another. Isolating each of them — by representing each slide as a sole chapter — is not suitable. Therefore, it would be better to consider a collection of slides as a book, containing chapters, where each chapter has more than one section. Each chapter could be seen as being a physical representation of one slide or more and each section can be related to the physical representation of one or multiple slides.

The best view for a newspaper is as a book containing chapters, each chapter — containing sections — is related to some topic (i.e. sport, news, politics, etc.). In the physical representation, it is not a problem if one section is being visualised in different pages, for example one part in the main page and the other part in another page at the end. In fact, it is not a problem either if one topic is divided and distributed into multiple pages. The logical structure would not be changed.

The previous analysis shows that the basic unit in any document class is the *section*. *Basic unit* means that the section is atomic and no further sectioning is contained inside. Therefore, if a section contains subsections, then the atomic units are the subsections. In order to constitute a logical structure of a document class with a bottom-up approach, many relations have to be realised. First of all, one or more atomic sections can be composed to represent a higher level of sectioning (a section). Secondly, the newly composed sections in turn participate in relations with each other and/or with atomic sections in order to represent chapters or chapter-like elements. Finally, the relations between chapters and/or chapter like elements represent the intended document classes. The previous analysis is akin to the one suggested by DocBook presented in Section 3.1.11.

4.4.2 Logical Document Structure

Structural Links, introduced in the RSL metamodel, are responsible for composing a new resource. Thus, a composed resource is a structure built from the object type resources already existing in the model and/or from the already composed resources. In the context of the document formats, the newly composed structure could be a section, a chapter, a book, etc.

The flexibility that a new structure can be composed from already built structures, in addition to other object type resources, gives us the intuition to explicitly model the lowest and atomic structure in any document class (a section). In the FCMD metamodel, in order to avoid any ambiguity, this atomic structure is called *Component*, since a communication letter could be the atomic structure as mentioned in the previous section. An atomic structure can be specified to be a communication letter by adding a property `type` with a value 'communicationletter' or specified to be a classical section by 'section' value. It is up to the application to specify its domain specific properties for each structure. This mechanism had also been suggested by ODA as discussed earlier in Section 3.1.6.

Figure 4.6 shows the FCMD metamodel extension to RSL structural links. To illustrate how a document class can be generated using the metamodel we propose, atomic structures can be composed from the object type resources shown in Figure 4.1 and object type selectors shown in Figure 4.2 in addition to links. To compose higher level parts, a new structure can be built with more than one link, some of them targeting already composed atomic components and some others targeting object type resources, selectors or links. This is possible since the built structures are stored in the *structures* collection. The same mechanism is used to build higher level components such as chapters. Finally a structure is composed by targeting all the highest level components that have already been built. Figure 4.7 shows an example of how a document class of type `book` is composed. Figure 4.7 (a) shows how the atomic sections, `component1` and `component2` are composed out of text block object type resources, image object type resources

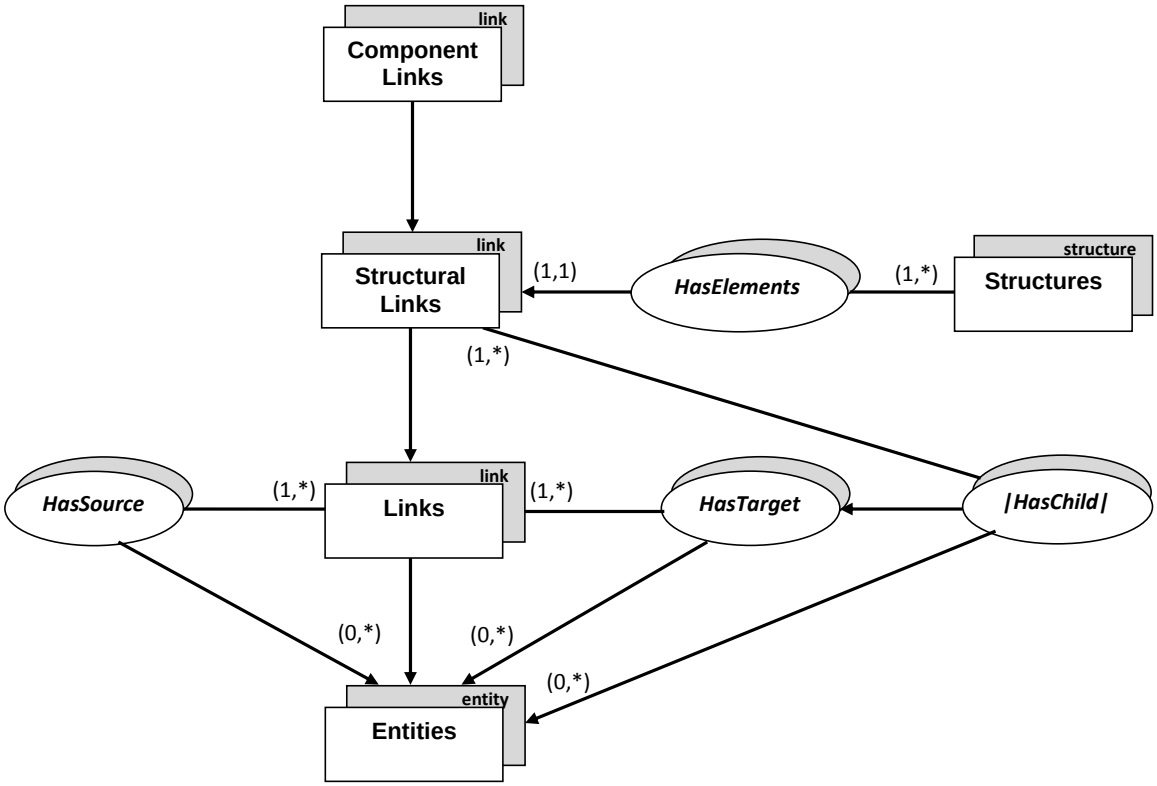


Figure 4.6: Component structural links in the FCMD metamodel. A component can be a section, a chapter, a book, etc.

and image selectors. Then, Figure 4.7 (b) shows higher level components, `component3` and `component4`, that are composed using the atomic components and another text block object type resource. Finally, Figure 4.7 (c) shows the book represented by `component5` which is composed of the lower level components, `component3` and `component4`. Note that the order of the elements comprising a `Component` is shown with a number on the arrows. In the visualisation process of the book, the underlying document production system will ask each component to draw itself by presenting each entity in the correct order. This example shows not only how a document logical structure can be constituted, but also shows the power of transclusion, as seen with `component3` and `component4` transcluding the same object type, text object type resource `G`.

Note that the component structural links in the FCMD metamodel are capable to generate any document model. Aside from the tree document model generated in Figure 4.7, linear, acyclic and constrained tree document models can be generated. A constrained tree model is generated in the same way as a tree model, but the underlying system has to enforce some constraints. These constraints are enforced for example in the sectioning levels or the object types allowed for a specific type of components. The acyclic document model, which has never been realized in any document model, is also allowed and can be generated. Figure 4.8 shows an acyclic

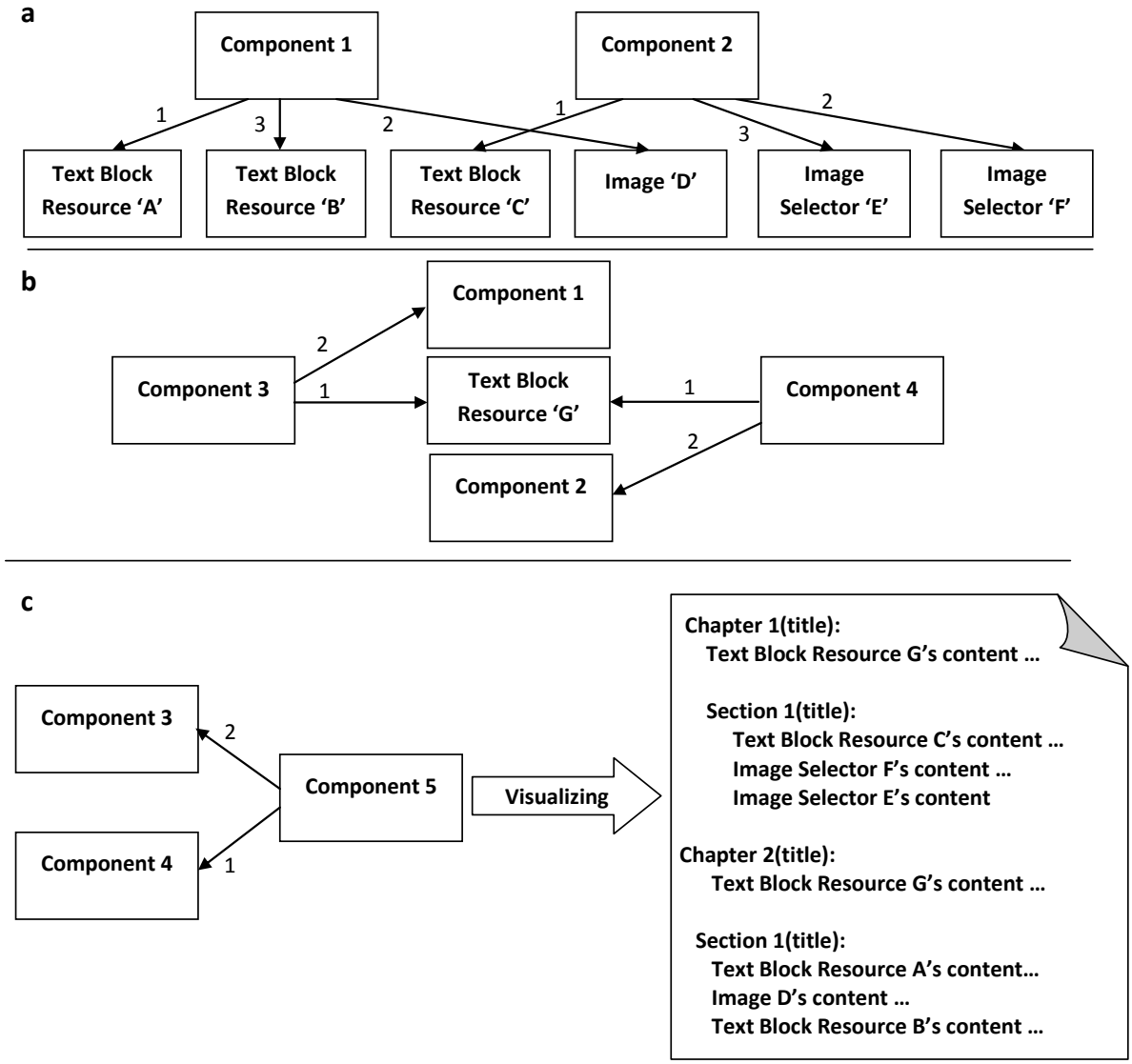


Figure 4.7: Constitute a book using the FCMD metamodel

document model. With acyclic document models, higher level components can be referred by lesser level components, in other words, child components can contain their parents, by referring to them. As shown with `component1` which refers to and includes its parent `component3`, the same for `component2` and `component4`. Actually, an ambiguity is post by the model, one cannot differentiate between the higher and lesser level components. The visualisation of such a model is difficult. However, this feature is very beneficial for *learning adaptive environments*. Suppose that Figure 4.8 is modeling a learning document material, which is used in such domain. If a user successfully grasps the concepts in `component1`, then the system will lead him to `component3` (the higher one). Later, if they successfully grasp the concepts in

`component3` then a step forward to `component5` will be achieved, otherwise a step backward to `component1` will be decided by the system.

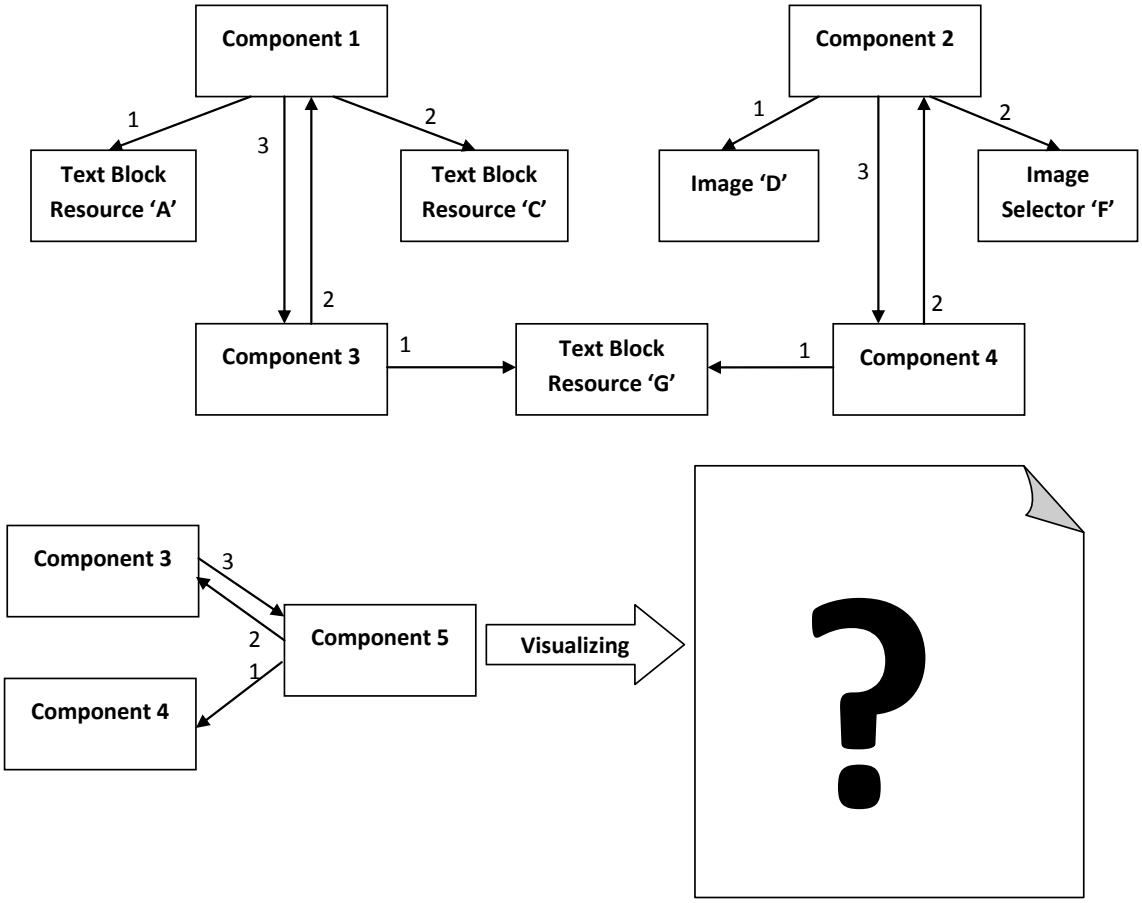


Figure 4.8: Acyclic document models can be generated using the FCMD metamodel. It is to be noted that the visualisation activity will not be trivial for them

4.5 Versioning

As shown in the previous chapter, Versioning in a limited form is supported by a few document formats. It has been supported either by bottom-up or top-down approaches. The former track changes in some atomic objects as in OOXML. While the second keeps the version ID for the whole document or a part of a document as in SGML. In the FCMD metamodel, versioning is supported in all document levels: the atomic level, parts of the logical structure and the document level thanks to the RSL links.

One important axiom which was taken into consideration in the process of modelling the versioning component in the FCMD metamodel is to enable any *document component* to tran-

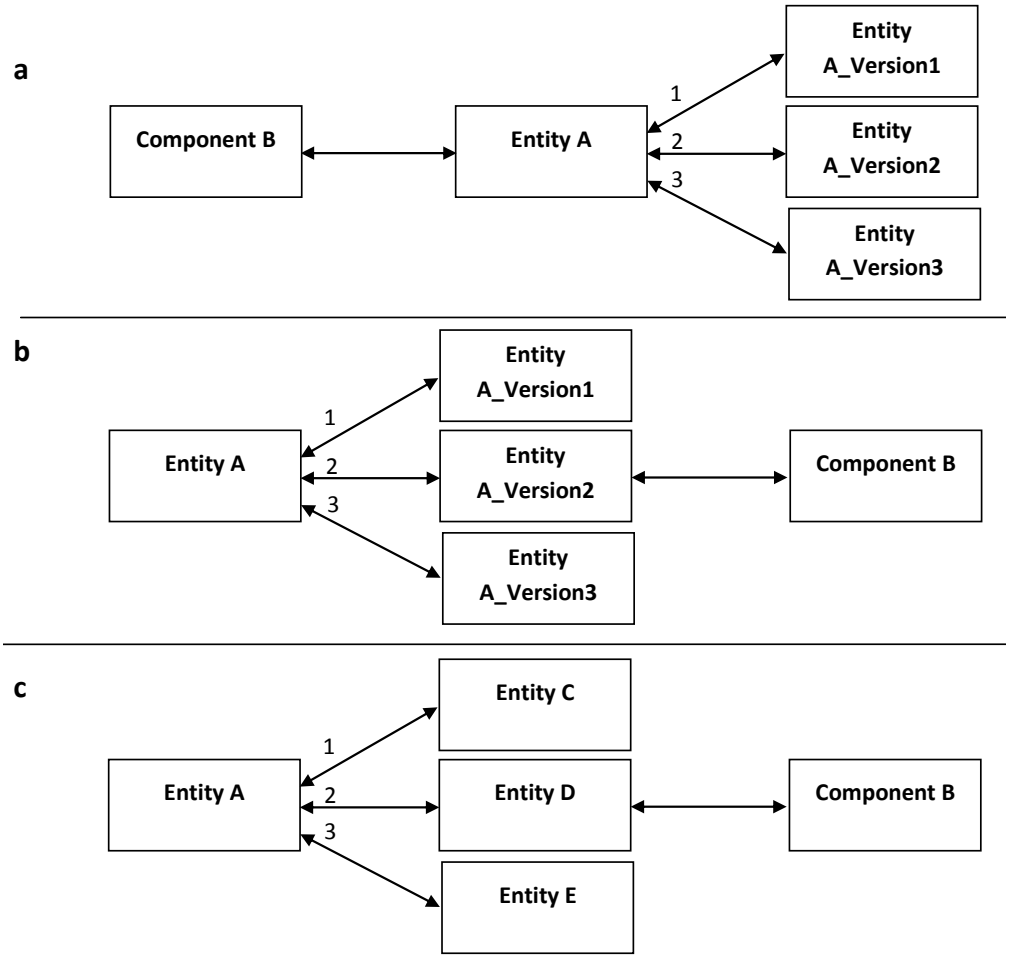


Figure 4.9: Document components should be linked to the entity's versions and not the entity itself

sclude and link to any specific entity's version. For instance, imagine an entity A that has three versions and a document component B which wants to transclude the second version of A, then the metamodel should allow the component B to establish a link to the second version of A but not to A itself. The reason can be clear with the Figure 4.9. In Figure 4.9 (a), B establishes a link to A, even though it wants to transclude the second version of A. The bidirectional link between A and its second version cannot solve such ambiguity. The reason is that A is linked to more than one entity, therefore, the underlying system will never know that the intended entity is the second version of A. While in Figure 4.9 (b), B establishes a link to the second version of A. The sole bidirectional link in the second version of A makes it clear that B is linked to the second version of an entity called A.

It is clear now that the versioning mechanism should be established using one-to-many links. The sole link's source is the entity that has been created for the first time, while the targets are the versions of that source entity. The entity has at least one version entity, which is the first

version when it was created. Any update operation carried out on that entity must enforce the creation of a new `version` entity and must be added to the list of the version link's targets of the source entity. It is worth mentioning that the version of an entity is an entity and has its own unique "fingerprint" identifier. Thus, using the combination of the original entity name plus the version number in the naming does not make any sense. In order to know the version number of it, the version link's targets must be an ordered list akin to the document component links. Figure 4.9 (c) shows the same example shown in Figure 4.9 (b) but each version entity has its own "fingerprint" identifier, hence, the linkage between the source entity and its version entities is explicit by using the version links without any naming conventions.

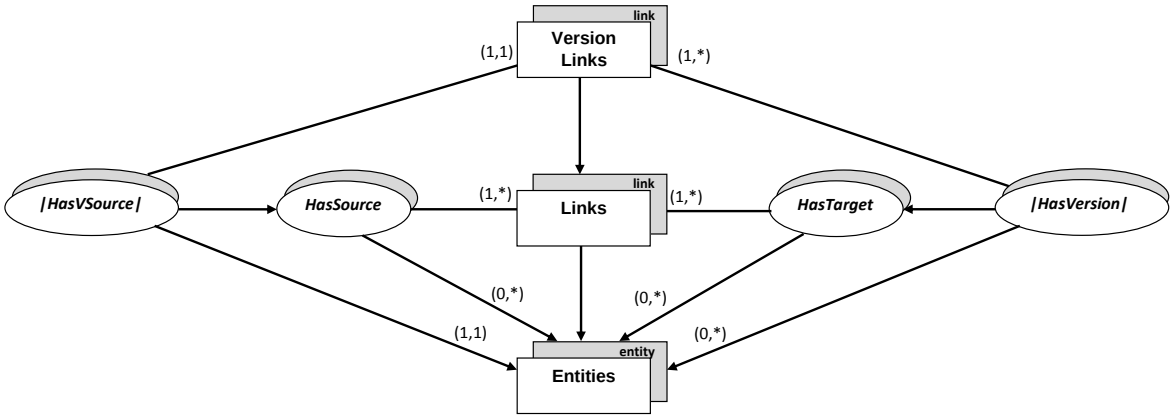


Figure 4.10: Versioning component in the FCMD metamodel

Figure 4.10 shows the FCMD metamodel versioning component. The *Version links* are a new subtype of the *links* type. The `HasVSource` is a subassociation of the `HasSource` association. It enforces that each entity has at least and at most one version link. It also enforces that the version links have at most one source entity. While the order of the version entities is introduced by the ordered `HasVersion` subassociation of the `HasTarget` association.

Actually, some points are worth mentioning here. First, the version links are a good solution for offering the versioning feature in the document formats. Nevertheless, they can create a notable overhead, especially that the metamodel suggests a creation of a new version entity whenever an update operation is carried out, so they should be used carefully. In order to reduce such overhead, one possible solution is to let the user decide when such a versioned entity should be created. For example, by providing an icon that allows the user to click whenever they want to create a new entity version. Second, the version links are on the entity level, meaning that a new version entity is created for the entity and not for its parts that have been updated. For example, if a text block entity has been updated by adding or deleting one word, then a new version entity should be created for the whole text block. So this approach also creates a notable overhead as well. Therefore, a research should be conducted to investigate how the FCMD metamodel would support the incremental version control for parts and selectors inside the entity. One possibility can be to exploit the concept of the *selectors* and to use them as the targets of the version links. Last but not least, document production systems implementing the FCMD metamodel should be

aware of the different document components targeting an entity. An option should be provided to such components to see if they are targeting a specific version of that entity or they are asking all the time for the most recent version of that entity. If they are asking for the most recent version, then in the case of creation of a new version of the entity the document component link shall be updated and linked to the last version of the entity.

4.6 User Management

As seen in Section 3.2.4, one form of authorisation granting, digital rights management, has been realised in EPUB, PDF, OpenDocument and OOXML. Unfortunately, none of the document formats have any support for user access management. Therefore, the FCMD metamodel has to inherit the user management component in RSL in order to profit from digital rights management and user access management features.

In the FCMD metamodel, as in RSL, one individual is the creator of the entity having the full control over its contents. However, it is worth mentioning that this will not prevent the "collaborative authoring" of entities, since the creator can define write and read access rights for other users, groups or/and individuals. The set of individuals having the access permission to a specific entity can be computed as defined in Equation 4.1, that is, the set of groups and individuals associated by `AccessibleTo` minus the set of groups and individuals defined through the `InaccessibleTo` association. The FCMD metamodel enforces the constraint that the individual access rights have priority over access rights defined for a group. This allows complex access rights to be defined, for example access rights like "the entity should not be accessed for one particular individual, while it should be accessed by all other members of his group" can be defined.

$$\begin{aligned} PrivilegedIndividuals = (PrivilegedGroups + PrivilegedIndividuals) - \\ (UnprivilegedGroups + UnprivilegedIndividuals) \end{aligned} \tag{4.1}$$

4.7 Adaptation

The *context resolver* concept introduced in the RSL model is used to adapt the entities and its content depending on the situation or the preferences. Therefore, each entity can have a set of context resolvers which are then be used to compute the visibility of an entity. If a single entity has multiple context resolvers, then it will only be visible if all of them return a positive feedback. For example, an audio object type resource can define a context resolver which returns a positive feedback only when the privileged user is blind. One possible way to implement such context resolvers is to use the XML configuration file shown in Listing 4.1. The `resolver` element includes a `user property` with key `status` and `blind` value. The context resolvers are very powerful for adaptive hypermedia systems, and can be used to define the adaption rules for the acyclic document model.

Listing 4.1: Context resolver for an audio object type resource

```
<contextresolver>
<resource>
    <type>audio</type>
    <name>only for blind audio</name>
</resource>

<resolver>
      <user:property>status:blind</user:property>
</resolver>
</contextresolver>
```

Adaptation and user management in the FCMD metamodel should work simultaneously. All the systems implementing the FCMD metamodel have to provide context resolvers for each entity to handle the access rights. Figure 4.11 shows the FCMD metamodel user management component and the context resolvers attached to each entity.

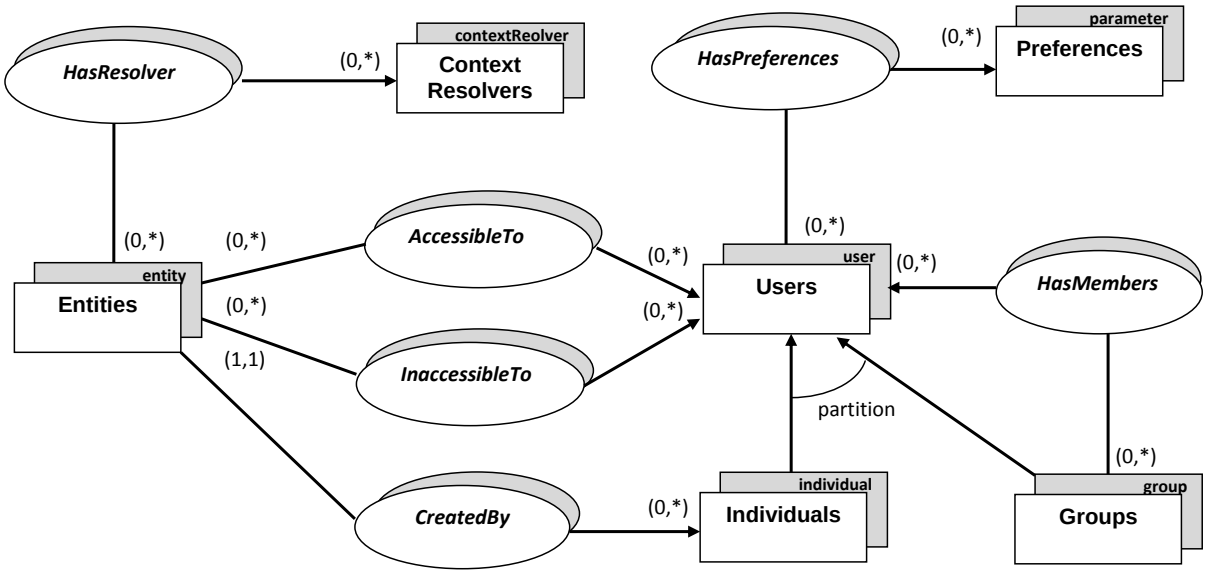


Figure 4.11: User rights management and adaptation components in the FCMD metamodel are inherited from RSL without any modification

4.8 Summary

In this chapter, the FCMD metamodel for document formats has been introduced. The FCMD metamodel was built as an extension of the RSL metamodel. Supporting the various digital features in the FCMD metamodel was an easy mission, because the RSL is responsible for managing

aspects related to transclusion, bidirectional links, adaptation and user management. Therefore, the main issue was on how to support the various logical document structures and how to support the versioning feature.

Table 4.1 presents the analysis of the different document formats presented in the previous chapter, also shown in Table 3.1, with addition to the FCMD metamodel based document formats. The FCMD metamodel supports all the digital features required for ubiquitous computing, except for the input adaptation. Supporting the input adaptation is one of the potential future work of this research. It needs a good understanding of the various interaction modalities such as multi-touch screens, speech recognition or mid-air gesture, as well as how these modalities relate to users, environments or device context.

The coming chapter will introduce the prototype implementation of the FCMD metamodel. We will be focusing on the prototype architecture, the used technologies and the gained digital features when using the proposed metamodel.

Format	Links		Transclusion		Versioning	Users Management	Adaptation	
	Uni	Bi	Rest.	Full			Output	Input
Scribe	✔	✗	✔	✗	✗	✗	✗	✗
GML	✔	✗	✔	✗	✗	✗	✗	✗
LaTeX	✔	✗	✔	✗	✗	✗	(✔)	✗
SGML	✔	✗	✔	✗	(✔)	✗	✗	✗
TNT	✔	✗	✗	✗	✗	✗	✗	✗
ODA	✔	✗	✗	✗	✗	✗	✗	✗
HTML 4	✔	✗	✔	✗	✗	✗	✔	✗
HTML5	✔	✗	✔	✗	✗	✗	✔	✗
XHTML	✔	✗	✔	✗	✗	✗	✔	✗
PDF	✔	✗	✔	✗	✗	(✔)	(✔)	✗
XML	✔	✔	✔	✔	✗	✗	✗	✗
OpenDocument	✔	✗	✔	✔	✔	✗	(✔)	✗
DocBook	✔	✗	✔	✗	✗	✗	✗	✗
OOXML	✔	✗	✔	(✔)	✔	✗	✗	✗
DIA formats	✗	✗	✗	✗	✗	✗	✗	✗
EPUB	✔	✗	✔	✗	(✔)	(✔)	✔	✗
FCMD format	✔	✔	✔	✔	✔	✔	✔	✗

Table 4.1: FCMD metamodel based document formats versus the existing document formats

Chapter 5

Implementation

In this chapter, a brief discussion of the implementation of a prototype implementing the FCMD model is presented. The prototype has been implemented using different programming languages and web technologies. We first discuss the objectives of implementing the prototype. Then, we look at the architecture of the prototype. Finally, this chapter introduces the digital features from the prototype.

5.1 Objectives of the Prototype Implementation

The main objective of implementing the prototype is to create a proof of concept for the FCMD format. We would like to prove that document formats modelled on top of the FCMD metamodel will be enriched with the digital features supported by the metmodel. Indeed this will help document formats to adapt to the upcoming age of ubiquitous computing.

Another objective is to provide a RESTful interface to facilitate accessing documents and parts of documents remotely. Indeed in the ubiquitous computing age, the document formats will be more and more stored on servers and databases in place of local file systems. The RESTful interface also offers a more general language independent programming interface for the FCMD model platform.

Finally, to show how documents can be enriched with the digital features, an online text editor will be implemented. It will use the RESTful interface to map the documents generated using it to the elements defined in the metamodel.

5.2 The Prototype Architecture

The FCMD model has been implemented using the Java programming language and an object oriented database *db4o*[1]. The resulting platform offers separate Java classes for all the FCMD metamodel concepts as well as an APIs that provide a set of functions to create, access, update

[1] http://www.db4o.com/ (accessed August 18th, 2012)

and delete information stored in the database. As shown in Figure 5.1 the FCMD platform is divided in five main parts: `Model`, `Data Storage Manager`, `FCMD API`, `Grizzly` and `RESTful Interface`.

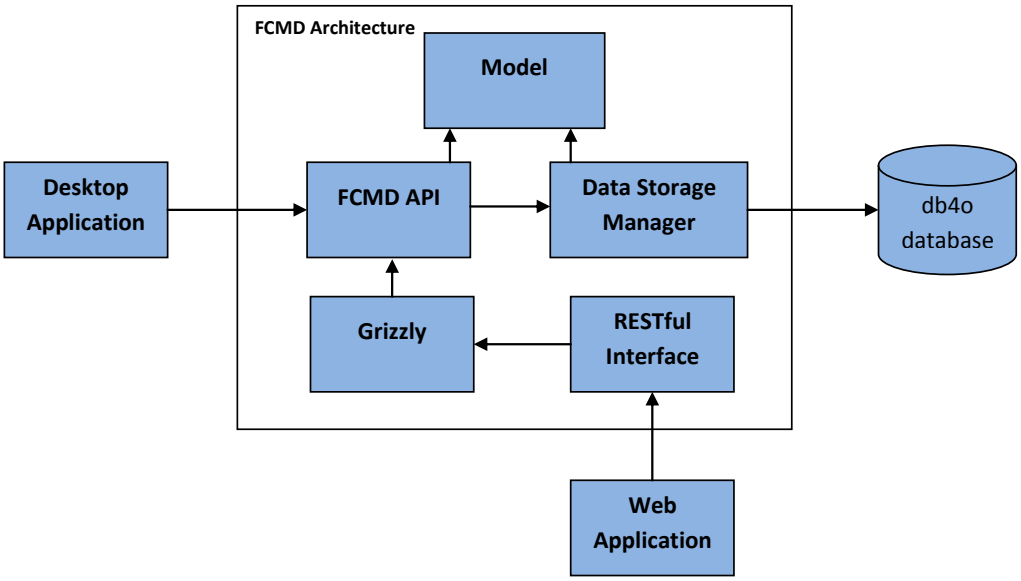


Figure 5.1: Architecture diagram of the prototype

The Model is where the FCMD metamodel is implemented. As such, it contains the definition of entities and links. Figure 5.2 shows the general structure of the FCMD metamodel implementation. Note that the goal of the implementation is to enrich documents with digital features rather than supporting multiple resource types, therefore we have only supported one resource (text resource) and its selector. The Data Storage Manager is responsible for saving and retrieving all objects and their relations (such as the links between resources) in the db4o database. The FCMD API is a Java API for accessing data as well as metadata and is the main entry point for applications when communicating with the system. Some important methods of the FCMD API are highlighted in Figure 5.3.

Since we cannot assume that any client application that wants to use the documents metadata managed by the platform has to be implemented in Java, we therefore offer a JSON[2] representation of all the data managed by the platform in addition to a Web Service (RESTful API) that provides the same functionality as offered by the Java FCMD API. Note that we do not primarily represent our data in JSON but only use JSON at the interface level to provide a language neutral interface for accessing the FCMD platform. In order to enable the implementation of the RESTful API, Grizzly[3] was used to build a scalable and robust webserver.

[2]http://www.json.org/ (accessed August 18th, 2012)
[3]http://grizzly.java.net/ (accessed August 18th, 2012)

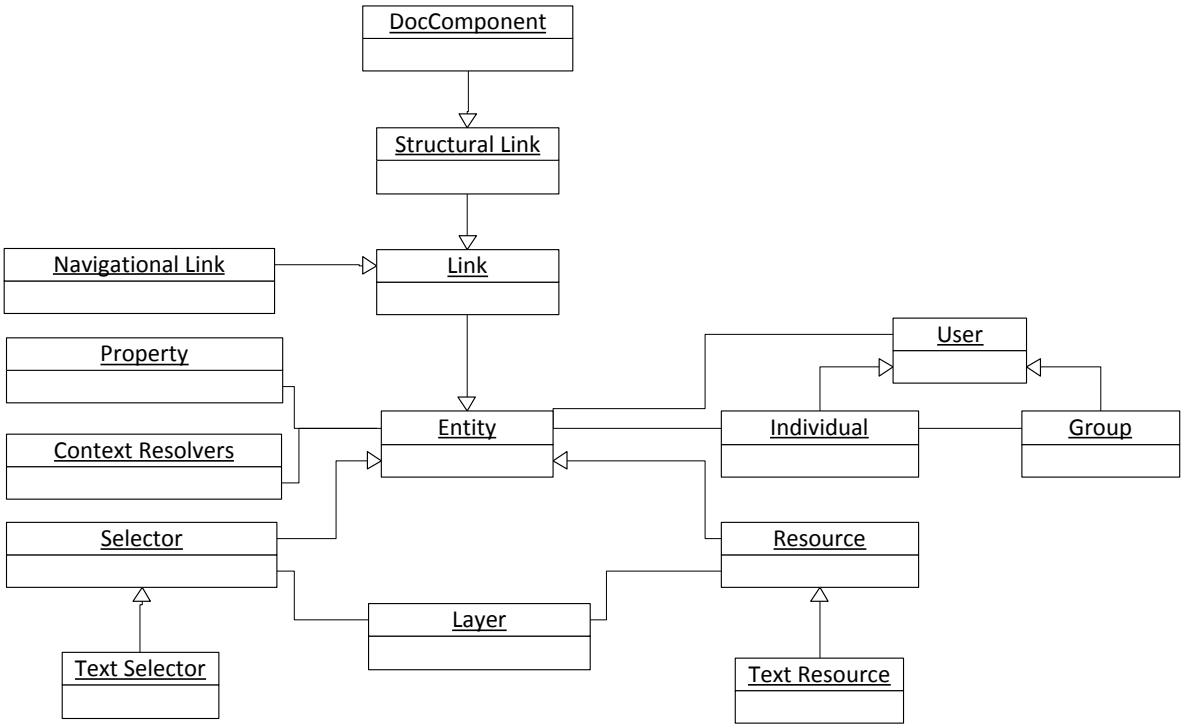


Figure 5.2: General organization of the FCMD metamodel implementation

FCMD API
+getEntity(name:String):Entity +addToResource(name:Resource) +getUser(name:String): User +createGroup(name:String, description:String): Group +deleteGroup(name:Group) +createIndividual(name:Individual): Individual +deleteIndividual(name:Individual) +getLink(name:String):Link +createDocComponentLink(name:String, source:Entity, target:Entity, creator:Individual):DocComponent +getDocComponent(name:String):DocComponent +createTextResource(name:String, content:String, creator:Individual):TextResource +createTextSelector(name:String,start:int, end:int, resource:TextResource, creatore:Individual):TextSelector **...**

Figure 5.3: Some methods supported by the FCMD API

REST Web Service design principal establishes a one-to-one mapping between create, read, update and delete (CRUD) operations and HTTP methods. According to this mapping create uses `POST`, read uses `GET`, update uses `PUT` and delete uses `DELETE` requests. Using Grizzly, annotations have to be defined to identify which operation the RESTful function wants

to define, the content type that produced and/or consumed, and finally the path of an URI that applications can use to call this function. For example to retrieve the individual users in the metamodel database one can define a RESTful function with `@GET` annotation, with `MediaType.APPLICATION_JSON` for the content type produced by the function and with a path `/individuals`, as shown in Listing 5.1. Listing 5.2 shows how applications can call the previous function to retrieve the individuals stored in the metamodel database, and the result of the function represented with JSON. Applications must call a URI which starts with the server port, followed by the name of the server and then the path defined in the RESTful intended function. Listing 5.3 shows the four operations supported with the REST Web Service, as you can see only `DELETE` operations do not produce or consume any content, the others must consume and/or produce content. Some of the the functions that have been supported with the RESTful interface are highlighted in Figure 5.4

Listing 5.1: Get stored individuals using a RESTful GET function

```
@GET
@Produces(MediaType.APPLICATION_JSON)
@Path("/individuals")
public static String collectionIndividuals() {
  Collection<AbstractRslElement>individuals=IServer.collectionIndividuals();
  JSONObject returnObject = new JSONObject();
   try{
   for(AbstractRslElement i : individuals){
   returnObject.append(i.getName(), new JSONObject(i));
       }
           }
        catch(JSONException e){
        logger.log(Level.WARNING, "Problem when creating result", e);
        }
   String returnValue = returnObject.toString();
   DatabaseManager.getCurrentDatabaseManager().commit();
   return returnValue;
 } // collectionIndividuals
```

Listing 5.2: A RESTful request and its result

```
(Request URI) : http://localhost:9998/iserver/individuals/
(Result) :
{
"bruno":[
{"labelExtension":"bruno, dumasbr","description":"bruno's account",
"name":"bruno","class":"class org.mobicrant.iserver.rsl.Individual",
"label":"bruno","login":"dumasbr","password":"test","preferences":[]}],

"ahmed":[
{"labelExtension":"ahmed, hamadatayeh","description":"ahmed's account",
"name":"ahmed","class":"class org.mobicrant.iserver.rsl.Individual",
"label":"ahmed","login":"hamada","password":"ahmed","preferences":[]}]
}
```

Listing 5.3: The operations supported by the RESTful interface

```
@GET
@Produces(MediaType.APPLICATION_JSON)
@Path("/path")
-----------------------------------------------------
@POST
@Consumes(MediaType.APPLICATION_JSON)
@Produces(MediaType.APPLICATION_JSON)
@Path("/path/{resourcename}")
-----------------------------------------------------
@PUT
@Consumes(MediaType.APPLICATION_JSON)
@Produces(MediaType.APPLICATION_JSON)
@Path("/path/{resourcename}")
-----------------------------------------------------
@DELETE
@Path("/path/{resourcename}")
```

5.3 db4o Database

All the metadata (resources, selectors, links, properties, etc.) are stored in a database called db4o. db4o stands for `database for objects`, and is an embeddable open source object database for Java and .NET. It is very powerful as it can store and retrieve the most complex object structures with only one line of code. There are many reasons behind the motivation to use such a database engine:

1. It is portable: aside from the possibility to carry it with the actual application, it also can be transferred and used in other applications since it is simply a file.

2. We can use the same programming language used to build our application (Java) instead of using a special query language like SQL.

3. We do not have to create a database beforehand with tables, nor deal with relations between tables. Db4o stores all objects in a file. The advantage is that if the implementation in the FCMD model changes, nothing has to be changed in the db4o.

4. We do not have to care about how objects are stored (including their relation with other objects). When querying the objects, related objects are automatically queried along.

5. Indexing of objects is possible in order to speed up queries.

RESTful Interface			
Function	**URI**	**Operation**	**Request JSON Syntax**
Create group	/groups/{groupname}	POST	`{"description" : ...}`
Delete group	/groups/{groupname}	Delete	---
Get a Link	/links/{linkname}	GET	---
Create docComponent	/doccomp/{linkname}	POST	`{"creator" : ... , "source" : ... , "targets" : [ ...] }`
Add authorized users for an entity	/entity/addau/{name}	PUT	`{"users" : [...] }`

Figure 5.4: Some methods supported by the RESTful interface

5.4 The Text Editor

A text editor, or document production system, has been implemented as a web application. It uses the RESTful API provided by the FCMD platform. Therefore, it does not have its own database store, because it makes use of the db4o database. In order to have a highly responsive software, JavaScript and AJAX have been used. Other web technologies have also been used such as jQuery[4] and CSS[5]. Note that jQuery is used to enable HTML selections, manipulations and for the dynamic Style Sheets.

This text editor allows users to create and view documents that have the same generic structure found in the article document class. It also provides them with the flexibility to add links and citations inside these documents using a click and drag functionality. Figure 5.5 shows a user citing a quotation from "FCMD Metamodel and Prototype" article with the click and drag functionality. When the user selects a text to cite, the system will display the text in the "Quoted Text" text area, and it will be added to his document after confirming the selection. It also has many other functionalities. In the following section, in the light of the text editor functionalities we will highlight the digital features which enrich its documents.

5.5 Gained Digital Features

In the light of the functionalities supported by the text editor, we present the digital features that we have seen in the documents generated using this text editor. Note that the text editor does not support the input adaptation and the versioning digital features as the current version of the FCMD platform does not support them.

[4]http://jquery.com/ (accessed August 18th, 2102)

[5]http://www.w3.org/TR/CSS/ (accessed August 18th, 2012)

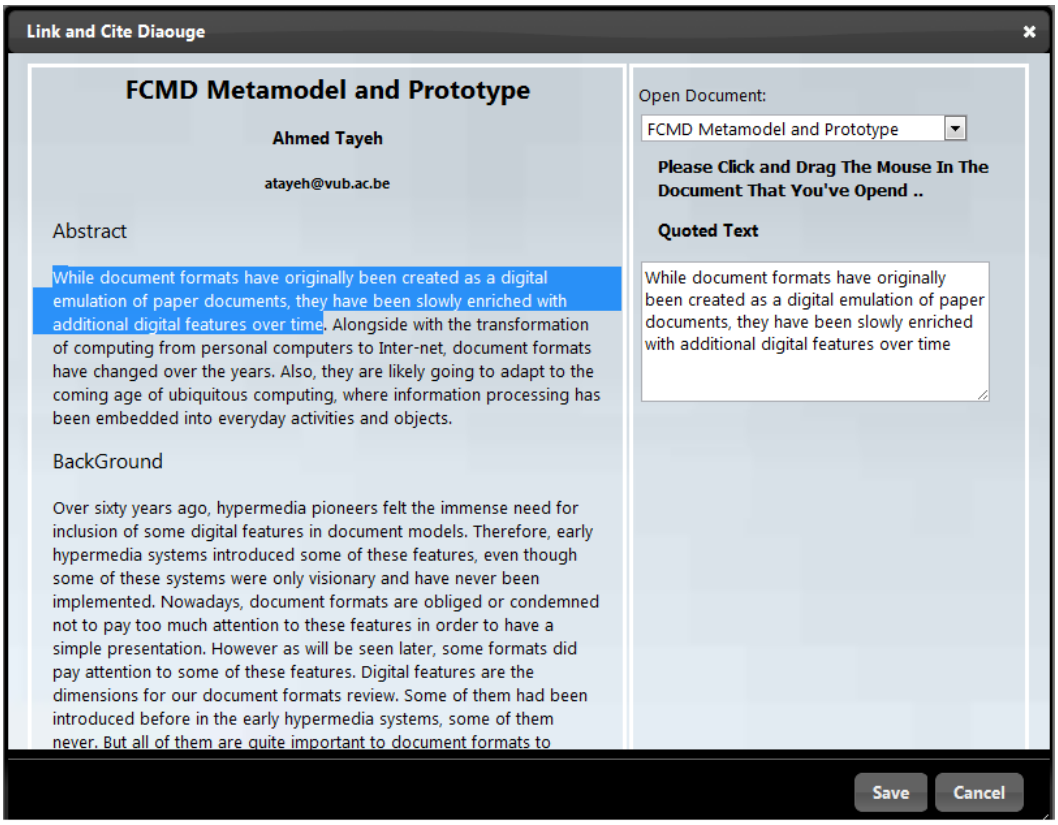


Figure 5.5: Create a citation using a click and drag functionality

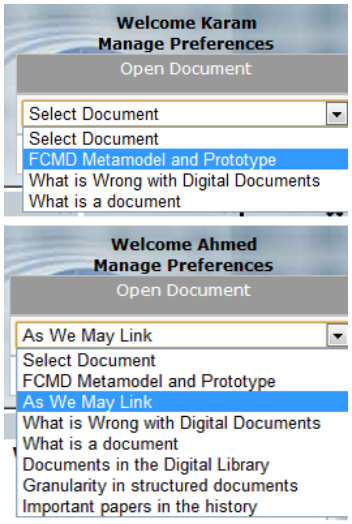


Figure 5.6: Each user has their own access permissions for the different documents

5.5.1 User Access Rights

The text editor allows the creator of a document to define access rights on the whole document level or parts of it. Using the defined access rights and a login functionality, the text editor

presents the right document or part of a document to the right user. Using the login functionality, the text editor allows individuals stored in the database to log into the system and to use its functionalities. After the user has logged into the system, they can only view documents that they have permission to access. An example is shown in Figure 5.6, with two different users having different lists of documents.

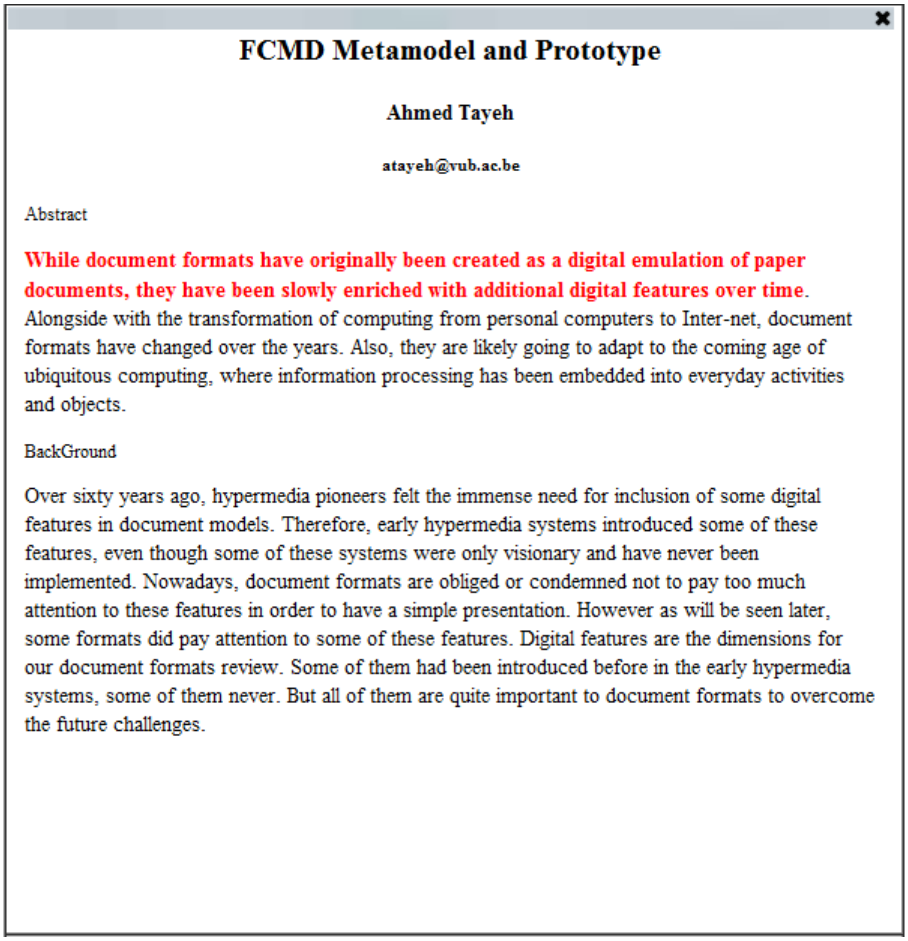


Figure 5.7: The text editor does not display the last section, because Karam has no access rights for it

Finally, when a user selects a document from the list to view, the text editor will visualise only the document parts that the user has privileged access to. To illustrate this with an example scenario, the user “Ahmed” created a document and has defined the last section of it to be inaccessible for the user “Karam”. When “Karam” asks for viewing that document, the editor will know which part “Karam” has no right to access. Therefore, as we can see in Figure 5.7 the last section is not visualised for the user “Karam” while the whole document has been visualised for its creator as shown in Figure 5.8.

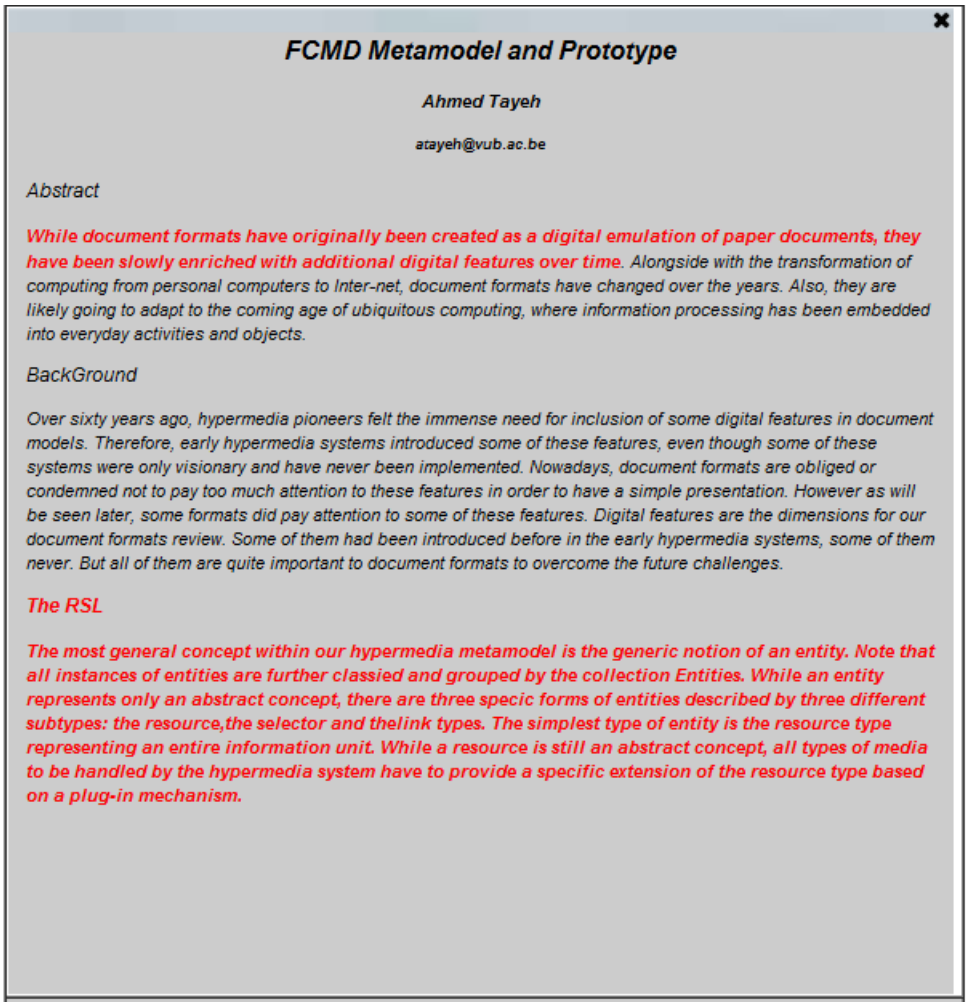


Figure 5.8: The user Ahmed has a full access to his document, because he is the creator of it

5.5.2 Linking and Transclusion

The fact that transclusion has to be implemented using bidirectional links motivated us to introduce both linking and transclusion in the same section. The text editor visualises the source and target of any link as links, therefore, a user can follow the links from both sides. While for transclusion, the quoted part has a link referring to its original document source, and in the original document source a hint is shown in parts that have been transcluded from other documents. To illustrate with an example scenario, Figure 5.9 shows two different documents. The user has opened the left document. While they are reading it, they saw a hint given in the “The RSL” section, showing that the section is transcluded from another document. In order to get more information and to see the original document, they followed the link provided on the section. The text editor then presented the “original document” in the same view, and it highlights the original quoted section.

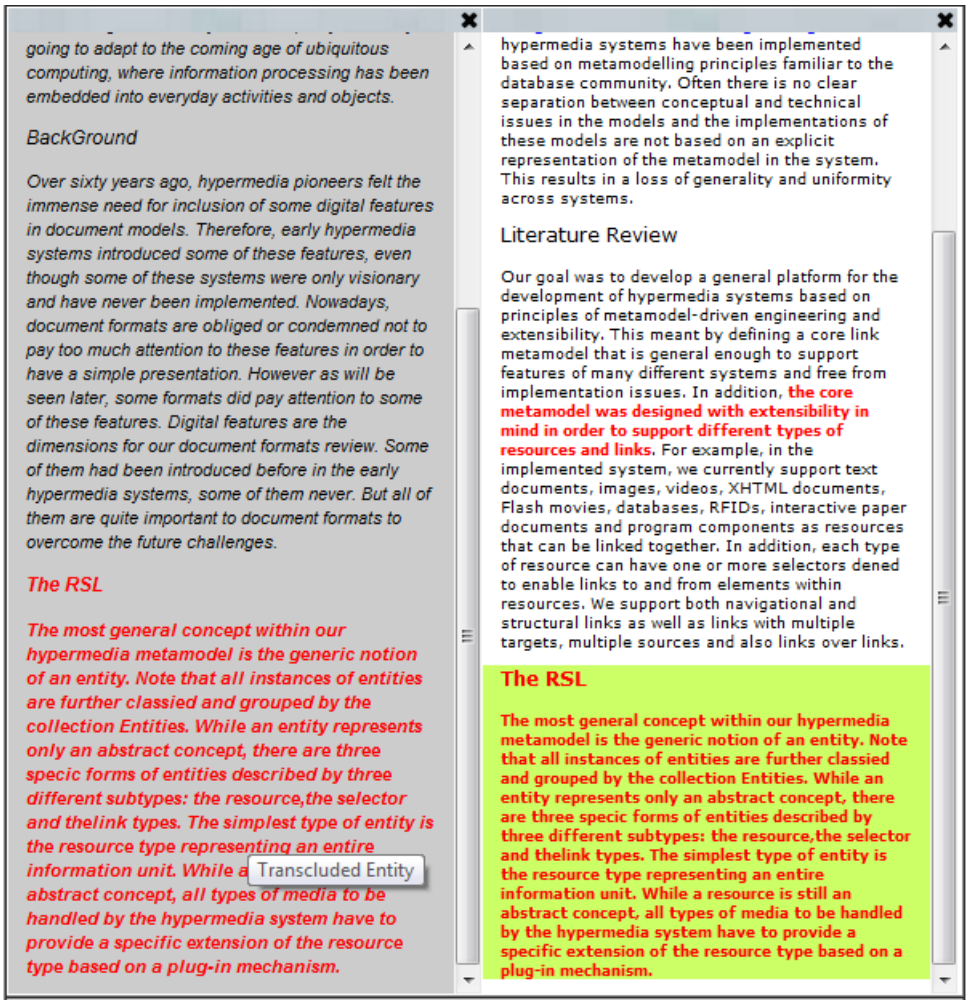


Figure 5.9: The text editor visualizes the link's sources and targets as links. It also gives a hint in parts that have been transcluded, and a link is provided referring to the original document source

5.5.3 Adaptation

A simple content adaptation has been embedded in the documents generated by the text editor by exploiting the user defined preferences. According to their preferences, the document content will be adapted. As we can see in Figure 5.8 and Figure 5.7 the same document is presented with two different layout styles. The reason is that the users "Ahmed" and "Karam" have different preferences. Figure 5.10 shows the defined preferences by the user Ahmed, while Figure 5.11 shows defined preferences for the user Karam.

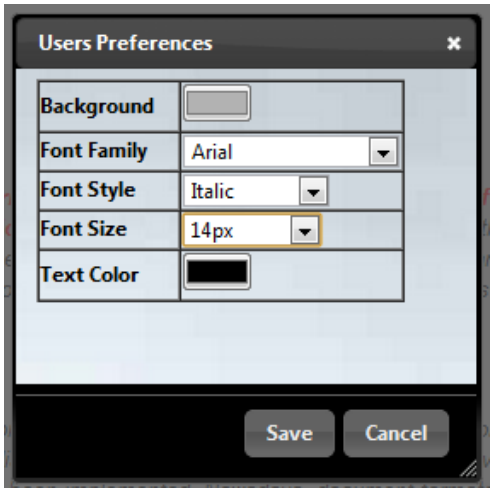


Figure 5.10: The preferences defined for the user Ahmed

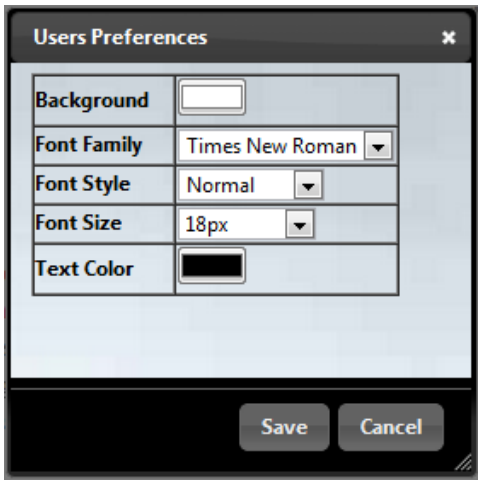


Figure 5.11: The preferences defined for the user Karam

Chapter 6

Conclusions and Future Work

6.1 Summary

How prepared are current document formats for the upcoming age of ubiquitous computing? In this thesis, we addressed this question by providing a review of a set of representative document formats including Scribe, GML, LaTeX, SGML, the TNT document model, ODA, HTML 4, HTML 5, XHTML, PDF, XML, OpenDocument (ODF), DocBook, Office Open XML, OCD, XCDF and EPUB 3. These document formats have been reviewed along five different dimensions which have been selected based on the challenges that they introduce for the integration of document formats with ubiquitous computing. For each of these dimensions, linking, transclusion/reusability, versioning, user rights management and content adaptation, we have provided a detailed discussion about their support in different document formats and provided a summarising overview in the form of a comparative table.

Our review of existing document formats reveals that most of them are not ready to adapt to the age of ubiquitous computing. With the exception of EPUB 3, which made provisions for a number of the surveyed features, existing document formats only support basic features or show only very limited support for advanced features. Based on these findings, we outlined a roadmap of potential tracks to be followed to offer better support and integration with ubiquitous computing. Three different tracks towards this goal have been discussed, with a description of their different strengths and weaknesses. As stated in our analysis, we believe that by addressing the challenges directly in the document formats will either end up in an increased complexity of current document formats or overlooked novel formats. We believe that a middleware-based approach with a carefully modelled metamodel, in combination with the body of work that is currently executed by the ubiquitous computing and the Internet of Things communities, will bring an elegant and flexible solution for the integration of documents in the coming age of ubiquitous computing.

The resource-selector-link (RSL) metamodel that has been developed for cross media systems was chosen as the basis for the metamodel mentioned above. Aside from its support for aspects like distribution, user rights management, transclusion and content adaptation, it is general and flexible enough. The newly created middleware metamodel was named *Fluid Cross-Media Doc-*

ument Format Metamodel. In order to enrich document formats with the features, they have to be modelled on top of the middleware. Document formats that are tightly linked to the metamodel can acquire all features by mapping their elements to individual components of the metamodel. Document formats which would not support the metamodel can also profit from some features by embedding them into one component of the metamodel.

Finally, a prototype of the Fluid Cross-Media Document Format metamodel has been implemented as a proof of concept. The prototype offers a RESTful API, which enabled us to implement an online text editor. The text editor is used to create and view documents. These documents are mapped to all elements in the metamodel and have been enriched with the digital features required for the upcoming age of ubiquitous computing.

Our work brings various contributions to the document engineering, ubiquitous computing and hypermedia communities. We can summarise our contributions as follows:

1. We addressed the ubiquitous computing challenges that existing document formats will face, by making a comprehensive and a profound review. Contradicting to earlier works [16, 36, 54] which only addressed HTML and XML document formats with only transclusion and bidirectional links, we have selected a representative set of document formats from all document format families and analysed them based on a set of features required for ubiquitous computing.

2. The results of our review show that there is a limited support for the digital features required for the upcoming age of ubiquitous computing in existing document formats. Therefore, we have proposed a metamodel approach for document formats. This metamodel is called the *Fluid Cross-Media Document Format Metamodel*, and it supports all the digital features required for ubiquitous computing. This metamodel is based on the resource-selector-link metamodel. In order to enrich document formats with the features supported by the proposed metamodel, they have to be modelled on top of the metamodel.

3. The fact that the proposed metamodel is based on the RSL metamodel also brings a contribution to the RSL metamodel and proves that RSL is very flexible, generic and suitable for evolving hypermedia systems.

4. Finally, A proof of concept prototype for the proposed metamodel has been implemented in the form of an online text editor which generates documents that are enriched with the digital features supported by the metamodel.

6.2 Future Work

This thesis suggests several potential future work ideas. First of all, further research has to be conducted on how to extend the document metaphor to deal with dynamic objects derived from the enclosing environments, for example information retrieved from sensors. Second, the FCMD metamodel still needs some enhancements and might need extensions to deal with incremental versioning and input adaptation. Last but not least, we have to investigate how to build open and

extensible authoring tools for cross-media documents. As we have witnessed, some cross-media information systems (e.g [11]) can be extended with new media and resource types on the model and data level. However, to the best of our knowledge there are no solutions that are extensible enough on the authoring and visualisation level. This implies that the introduction of a new media type requires substantial programming efforts due to the fact that the authoring and visualisation components have to be manually extended in order to support the linking to and from already implemented media types. To address the problem of limited scalability in existing cross-media information systems, the extensibility for new media types should not only be offered at the data level but also at the visualisation level via an extensible authoring tool for the next generation of open cross-media information spaces.

Bibliography

[1] GML Starter Set User's Guide. http://publibfp.boulder.ibm.com/cgi-bin/bookmgr/BOOKS/dsm04m00/CCONTENTS. [Online; accessed 20-September-2011].

[2] MobiCraNT: Second Generation Mobile Cross-media Applications: Scalability, Heterogeneity and Legacy. http://soft.vub.ac.be/mobicrant/. [Online; accessed 10-April-2011].

[3] SCRIPT/VS Language Reference. http://publibfp.dhe.ibm.com/cgi-bin/bookmgr/BOOKS/dsml7m00/CCONTENTS. [Online; accessed 16-August-2012].

[4] Information Technology- Open Document Architecture (ODA) and Interchange Format: Introduction and General Principles. Technical report, ITU, 1993.

[5] PDF Reference, Sixth Edition. Technical report, Adobe Systems Incorporated, November 2006.

[6] Standard ECMA-376: Office Open XML File Formats; 3rd Edition. Technical report, ECMA International, 2011.

[7] J. Andre, R. Furuta, and V. Quint. By Way of an Introduction. Structured Documents: What and Why? In *Structured Documents*, pages 1–7. Cambridge University Press, 1989.

[8] J.-L. Bloechle. *Physical and Logical Structure Recognition of PDF Documents*. PhD thesis, University of Fribourg, Switzerland, 2010.

[9] J.-L. Bloechle, D. Lalanne, and R. Ingold. OCD: An Optimized and Canonical Document Format. In *Proceedings of the 9th International Conference on Document Analysis and Recognition (ICDAR'09)*, pages 236–240, Barcelona, Spain, July 2009.

[10] J.-L. Bloechle, M. Rigamonti, K. Hadjar, D. Lalanne, and R. Ingold. XCDF: A Canonical and Structured Document Format. In *Proceedings of the 7th IAPR Workshop on Document Analysis Systems (DAS'06)*, pages 141–152, Nelson, New Zealand, February 2006.

[11] P. Bottoni, R. Civica, S. Levialdi, L. Orso, E. Panizzi, and R. Trinchese. MADCOW: A Multimedia Digital Annotation System. In *Proceedings of the International Working Conference on Advanced Visual Interfaces (AVI 2004)*, Galliopli, Italy, May 2004.

[12] J. M. Boyer, C. F. Wiecha, and R. P. Akolkar. A REST Protocol and Composite Format for Interactive Web Documents. In *Proceedings of the 9th ACM Symposium on Document Engineering (DocEng 2009)*, 2009.

[13] T. Bray, J. Paoli, C. Sperberg-McQueen, E. Maler, and F. Yergeau. Extensible Markup Language (XML). *World Wide Web Journal*, 2(4):27–66, 1997.

[14] V. Bush. As We May Think. *Atlantic Monthly*, 176(1):101–108, 1945.

[15] F. C. Heeman. Granularity in Structured Documents. *Electronic Publishing*, 5(3):143–155, 1992.

[16] G. Cardone. An RSL-based Associative Filesystem. Master's thesis, Vrije Universiteit Brussel, Brussels, Belgium, 2010.

[17] P. Cesar, I. Vaishnavi, R. Kernchen, S. Meissner, C. Hesselman, M. Boussard, A. Spedalieri, D. C. Bulterman, and B. Gao. Multimedia Adaptation in Ubiquitous Environments: Benefits of Structured Multimedia Documents. In *Proceedings of the 8th ACM Symposium on Document Engineering (DocEng 2008)*, pages 275–284, São Paulo, Brazil, September 2008.

[18] M. Chuck and K. Bill. *HTML & XHTML : The Definitive Guide*. O'Reilly, 2006.

[19] B. Collins-Sussman, B. Fitzpatrick, and C. Pilato. *Version Control with Subversion*. O'Reilly, 2004.

[20] G. Conboy, M. Garrish, M. Gylling, W. McCoy, M. Makoto, and D. Weck. EPUB 3 Overview, Recommended Specification, October 2011.

[21] O. De Troyer, S. Casteleyn, and P. Plessers. WSDM: Web Semantics Design Method. In *Web Engineering - Modelling and Implementing Web Applications*, volume 12 of *Human-Computer Interaction series*, pages 303–352. Springer, 2007.

[22] S. DeRose, E. Maler, and R. Daniel Jr. XML Pointer Language (XPointer) Version 1.0, January 2001.

[23] S. DeRose, E. Maler, D. Orchard, and B. Trafford. XML Linking Language (XLink) Version 1.0, June 2001.

[24] R. Furuta. *An Integrated, but not Exact-Representation, Editor/Formatter*. PhD thesis, University of Washington, Department of Computer Science, Seattle, WA, 1986.

[25] R. Furuta. Structured Document Models and Representations. In *Issues in Generalized Text Processing: Lecture notes of a Short Course held in association with PROTEXT IV, the fourth International Conference on Text Proces Systems*, pages 1–14. Boole Press, 1987.

[26] R. Furuta. Concepts and Models for Structured Documents. In *Structured Documents*, pages 7–39. Cambridge University Press, 1989.

[27] R. Furuta. Important Papers in the History of Document Preparation Systems: Basic Sources. *Electronic Publishing*, 5(1):19–44, 1992.

[28] C. F. Goldfarb. The Roots of SGML A Personal Recollection. http://www.sgmlsource.com/history/roots.htm. [Online; accessed 26-September-2011].

[29] C. F. Goldfarb. A Generalized Approach to Document Markup. In *Proceedings of the ACM SIGPLAN SIGOA Symposium on Text manipulation*, pages 68–73, Portland, Oregon, June 1981.

[30] C. F. Goldfarb. *The SGML HandBook*. Clarendon Press, 1990.

[31] C. F. Goldfarb and P. Paul. *The XML Handbook*. Prentice Hall PTR, 1998.

[32] D. Guinard, V. Trifa, F. Mattern, and E. Wilde. From the Internet of Things to the Web of Things: Resource-Oriented Architecture and Best Practices. In *Architecting the Internet of Things*. Springer, 2011.

[33] ISO 9241-11. *Ergonomic Requirements for Office Work with Visual Display Terminals (VDTs) – Part 11: Guidance on Usability*. International Organization for Standardization, Geneva, Switzerland, 1998.

[34] B. W. Kernighan, M. E. Lesk, and J. F. Ossanna. UNIX time-sharing system: Document preparation. *The Bell System Technical Journal*, 57(6):2115–2135, July-August 1978.

[35] D. E. Knuth. *The TEXbook*. Addison-Wesley, 1984.

[36] H. Krottmaier and H. Maurer. Transclusions in the 21st Century. *Universal Computer Science*, 7:1125–1136, 2001.

[37] R. Kumar, J. O. Talton, S. Ahmad, and S. R. Klemmer. Bricolage: Example-Based Retargeting for Web Design. In *Proceedings of the Annual Conference on Human Factors in Computing Systems (CHI 2011)*, pages 2197–2206, Vancouver, Canada, May 2011.

[38] L. Lamport. *LaTeX: A Document Preparation System , User's Guide and Reference Manual*. Addison-Wesley, 2004.

[39] J. Loeliger. *Version Control with Git*. O'Reilly, 2009.

[40] A. Müller, S. Rönnau, and U. M. Borghoff. A File-Type Sensitive, Auto-Versioning File System. In *Proceedings of the 10th ACM Symposium on Document Engineering (DocEng 2010)*, Manchester, UK, 2010.

[41] T. H. Nelsom. The Heart of Connection: Hypermedia Unified by Transclusion. *Communications of the ACM*, 38:31–33, August 1995.

[42] T. H. Nelson. *Literary Machines*. Mindful Press, 1982.

[43] M. C. Norrie. An Extended Entity-Relationship Approach to Data Management in Object-Oriented Systems. In *Proceedings of the 12th International Conference on the Entity-Relationship Approach (ER '93)*, pages 390–401, Arlington, USA, December 1993.

[44] M. C. Norrie, B. Signer, and N. Weibel. General Framework for the Rapid Development of Interactive Paper Applications. In *Proceedings of the 1st International Workshop on Collaborating over Paper and Digital Documents (CoPADD 2006)*, pages 9–12, Banf, Canada, November 2006.

[45] S. Pemberton, D. Austin, J. Axelsson, T. Celik, D. Dominiak, H. Elenbaas, B. Epperson, M. Ishikawa, S. Matsui, S. McCarron, A. Navarro, S. Peruvemba, R. Relyea, S. Schnitzenbaumer, and P. Stark. XHTML 1.0 The Extensible HyperText Markup Language (Second Edition): A Reformulation of HTML 4 in XML 1.0, August 2002.

[46] A. J. Pinkney, S. R. Bagley, and D. F. Brailsford. Reflowable Documents Composed from Pre-Rendered Atomic Components. In *Proceedings of the 11th ACM Symposium on Document Engineering (DocEng 2011)*, pages 163–166, Mountain View, USA, September 2011.

[47] M. Presser, P. Barnaghi, M. Eurich, and C. Villalonga. The SENSEI Project: Integrating the Physical World with the Digital World of the Network of the Future. *Communications Magazine, IEEE*, 47:1–4, 2009.

[48] B. K. Reid. A High-Level Approach to Computer Document Formatting. In *Proceedings of the 7th ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages*, pages 24–31, New York, USA, 1980.

[49] B. K. Reid. *Scribe: A Document Specification Language and its Compiler*. PhD thesis, Carnegie-Mellon University Computer Science Department, Pittsburgh, PA, october 1980.

[50] B. K. Reid. Text Processing and Document Manipulation. In *Proceedings of the International Conference*, pages 214–223, University of Nottingham, April 1986.

[51] J. Saltzer. Manuscript Typing and Editing: TYPESET, RUNOFF. In *The Compatible Time-Sharing System: A programmer guide*. MIT Press, 1965.

[52] A. C. Shaw. A Model for Document Preparation Systems. Technical Report 80-04-02, University of Washington, Department of Computer Science, Seattle, WA, April 1980.

[53] B. Signer. *Fundamental Concepts for Interactive Paper and Cross-Media Information Mangement*. PhD thesis, ETH Zurich, Switzerland, 2006.

[54] B. Signer. What is Wrong with Digital Documents? A Conceptual Model for Structural Cross Content Composition and Resuse. In *Proceedings of the 29th International Conference on Conceptual Modeling (ER 2010)*, pages 391–404, Vancouver, Canada, November 2010.

[55] B. Signer and M. C. Norrie. As We May Link: A General Metamodel for Hypermedia Systems. In *Proceedings of the 26th International Conference on Conceptual Modeling (ER 2007)*, pages 359–374, Auckland, New Zealand, November 2007.

[56] P. Sorotokin, G. Conboy, B. Duga, J. Rivlin, D. Beaver, K. Ballard, A. Fettes, and D. Weck. EPUB Canonical Fragment Identifier (epubcfi) Specification, Recommended Specification, October 2011.

[57] M. Sporny, S. McCarron, B. Adida, M. Birbeck, and S. Pemberton. HTML+RDFa 1.1: Support for RDFa in HTML4 and HTML5, March 2012.

[58] C. Thao and E. V. Munson. Version-Aware XML Documents. In *Proceedings of the 11th ACM Symposium on Document Engineering (DocEng 2011)*, pages 97–100, Mountain View, USA, September 2011.

[59] J. Vesperman. *Essential CVS*. O'Reilly, 2003.

[60] B. W. Lampson. Bravo Manual. In *Alto User's Handbook*, pages 26–60. Xerox Palo Alto Research Center, 1976.

[61] N. Walsb. *DocBook 5 The Definitive Guide*. O'REILLY, 2010.

[62] N. Weibel, M. Norrie C., and B. Signer. A Model for Mapping between Printed and Digital Document Instances. In *Proceedings of the 7th ACM Symposium on Document Engineering (DocEng 2007)*, pages 19–28, Winnipeg, Canada, August 2007.

[63] R. Weir, M. Brauer, and P. Durusau. Open Document Format for Office Applications (OpenDocument) Version 1.2. Technical report, Organization for the Advancement of Structured Information Standards (OASIS), march 2011.

[64] Wikipedia. Markup Language. http://en.wikipedia.org/wiki/Markup_language. [Online; accessed 10-September-2011].

[65] Wikipedia. XML. http://en.wikipedia.org/wiki/XML. [Online; accessed 12-November-2011].

[66] E. Wilde and D. Lowe. *XPath, XLink, XPointer, and XML: A Practical Guide to Web Hyperlinking and Transclusion*. Addison-Wesley Professional, 2003.