



Vrije Universiteit Brussel

Faculty of Engineering

An Investigation of Adapting Document Structures

Thesis submitted in partial fulfillment of the requirements for
the degree of Master of Science in Applied Science and Engineering: Applied Computer Science

Serafeim Kourlos

Promoter: Prof. Dr. Beat Signer

Advisor: Dr. Bruno Dumas

2013-2014



Abstract

Traditional computer applications normally expect a static execution environment. However, over the last decade this has constantly changed with the widespread availability of mobile systems. Thus, it is necessary for several applications to be adapted on different contexts. A lot of researchers tried to investigate new models and platforms to support mobile applications. However adaptation is a very broad domain. Adaptation can be divided into several parts such as session continuity, user management, application migration or dynamic adaptation.

In our work we focus on adaptation in terms of how we can construct documents. We provide a conceptual model which someone can use to build different structures of the same document in order to adapt to different devices. We believe that the structure of a document plays an important role in the adaptation procedure. Even more, structures can be reused by other domains for several applications.

Another issue that is investigated are the benefits that an application can get when we use a hypermedia approach such as the RSL (Resource-Selector-Link) metamodel. We prove that using a hypermedia approach for adapting content to different contexts is more flexible and can be used to solve complex problems. We will extend our model using a component-based approach and showing how each component can adapt itself independently.

In order to prove the benefits of our approach we provide a use case of a web page that can be adapted to different devices. In this use case an XML document changes structures over the different devices to achieve better results in terms of adaptation, meaning that we send our document to our client device and the device can display the results of our document with no extra processing. Furthermore in this document each of our components are treated differently from our component based approach for its presentation.

Acknowledgements

The road towards this final thesis was very productive. I have gained hands-on experience in Java, XML and Android applications.

First of all, I would like to gratitude Dr. Bruno Dumas for being my advisor, spending a lot of time in meetings and giving me feedback during this academic year. I would like also to express my gratitude to Prof. Dr. Beat Signer for giving me the chance to do my thesis in the WISE lab.

Finally, I would like to thank my family for giving me support during this year. Their support was very important for me to achieve this difficult task.

Thank You

Serafeim Kourlos

Contents

1	Introduction	1
1.1	Research Context	1
1.2	Goal of the Thesis	2
1.3	Research Questions	2
1.4	Methodology	3
1.5	Structural Overview of the Thesis	4
2	Background	5
2.1	Hypermedia	5
2.2	Context Awareness	6
2.3	Cross-Media	8
3	Related Work	9
3.1	Media Adaptation	9
3.2	Structure UI for Web Applications	10
3.2.1	User Contribution	10
3.2.2	User Profile	12
3.2.3	Device-Centric Approach	13
3.2.4	Structure UI based on the Cultural Background	15
3.2.5	Device and User Preferences	16
3.3	Available Frameworks	18
3.3.1	TERESA	18
3.3.2	MARIA	19
3.3.3	USIXML	20
3.3.4	CHISEL	20
3.3.5	MADAM	21
3.4	Summary	21
3.4.1	Comparison Table	22

3.4.2	Analysis of the Table	23
3.4.3	Conclusion	24
4	Modelling with RSL	26
4.1	The Resource-Selector-Link (RSL) Metamodel	26
4.2	Entities, Resources, Selectors and Links	27
4.2.1	Entities	27
4.2.2	Resources	28
4.2.3	Selectors	28
4.2.4	Links	29
4.2.5	Layers	29
4.2.6	Structure	30
4.3	Elements in RSL	31
4.3.1	Granularity in RSL	31
4.3.2	An Example with Different Granularity Levels	33
4.4	Document Component Link	34
4.4.1	Document Component Link Approach	34
4.4.2	Reasons to Modify the Logical Structure of a Document	35
4.5	An Example of Document Component Link	36
4.5.1	How We Store the Elements	38
4.5.2	How We Connect Elements with Document Links	38
4.6	The General HTML Table Model	40
4.6.1	Elements of an HTML Table	40
4.6.2	Ordering of the HTML Table Elements	43
4.7	The Importance of Crosslets	44
4.7.1	Dynamic Crosslets	44
4.7.2	Functionality	45
4.8	Conclusion	45
5	Conceptual Model	46
5.1	Elements in our Conceptual Model	46
5.2	Components	46
5.2.1	HTML Table Component	47
5.2.2	Paragraph Component	50
5.2.3	Image Component	53
5.2.4	Other Components	53
5.3	Crosslets	54
5.3.1	Static Crosslets	54
5.4	Structures	57
5.4.1	How to Create Structures	58
5.4.2	Elements in Structure	60

5.4.3	An Example of Our Approach	61
5.5	Conclusion	63
6	Implementation	65
6.1	Use Case	65
6.1.1	Motivation	65
6.1.2	Infrastructure	66
6.1.3	Scenario	66
6.1.4	Tablet Device	68
6.1.5	Smartphone	69
6.1.6	Possible Structures	71
6.2	Conclusion	73
7	Conclusion and Future Work	74
7.1	Objectives of the Research	74
7.2	Contributions	75
7.3	Limitations	76
7.3.1	Order of the Structures	76
7.3.2	Future Work	76

List of Figures

3.1	An example of pruning the logical structure of a document [Ghani et al., 2010]	11
3.2	The adaptation algorithm [Paternò and Zichittella, 2010]	12
3.3	Support of user constraints by concatenation [Lemlouma and Layaïda, 2003]	13
3.4	The PDA interface of the shopping application [Paternò et al., 2010]	14
3.5	The DTV application interface [Paternò et al., 2010]	14
3.6	A sample English UI and the localised RTL version (in Arabic Language) [Khaddam and Vanderdonckt, 2010]	16
3.7	A dialysis patient record on a standard PC [Chaari et al., 2007]	17
3.8	The same dialysis patient record on a smartphone [Chaari et al., 2007]	17
3.9	Table of Comparison	22
4.1	Core link metamodel [Signer and Norrie, 2007]	28
4.2	Layers [Signer and Norrie, 2007]	29
4.3	Structural and navigational links [Signer and Norrie, 2007]	31
4.4	Elements [Weibel et al., 2007]	32
4.5	An example which illustrates three different granularity levels	33
4.6	PDF version of the Wikipedia page of the VUB	37
4.7	Elements granularity stored in RSL	39
4.8	XML description of vub PDF file	40
4.9	Source and target entities in our <i>document-component-links</i>	41
4.10	A simple table example	42
5.1	The HTML table from wikipedia page in 5.1a and the corresponding HTML code in 5.1b	48
5.2	The resulting table of the example	49

5.3	A paragraph example from Wikipedia VUB website with its corresponding HTML description	50
5.4	A paragraph example from Wikipedia VUB website with a URL link	50
5.5	The association between a paragraph component and a URL selector	52
5.6	Elements that inherit from <i>Paragraph</i> component without any modification	52
5.7	Elements that inherit from <i>Component</i> without any modification .	54
5.8	Component in our model inherits from the RSL <i>Resources</i> without any modification	54
5.9	Conceptual model: Component and Crosslets	55
5.10	An example of a paragraph crosslet	56
5.11	Our component resources along with their corresponding crosslets	56
5.12	The constructor of a paragraph crosslet accepts an instance of paragraph component	57
5.13	An example an HTML table (a) after we apply crosslets in our elements (b)	58
5.14	Different steps to modify the logical structure of a document . . .	59
5.15	Some of the <i>Substructure</i> specialisations	60
5.16	Conceptual model: Structure	61
5.17	The XML logical structure of an HTML document	61
5.18	The resulting structure after apply a <i>WithSelectors</i> structure to the logical structure	62
5.19	The resulting structure after apply a <i>ReferenceResource</i> structure to the previously modified structure	63
5.20	The resulting structure after apply a <i>ReverseOrder</i> structure to the previously modified structure	64
6.1	Tools and Platform	66
6.2	Steps in adaptation process	67
6.3	Device profile	68
6.4	Main page until <i>table of contents</i> element	69
6.5	The remaining VUB page	69
6.6	Section “Organisation” of the VUB web page	70
6.7	Main VUB page until <i>table of contents</i> element	71
6.8	The remaining VUB page with URLs	71
6.9	Campus and Facilities section	72
6.10	Basic principles section	72
6.11	Sections from the VUB web page in reverse manner	73
6.12	The VUB web page with no links	73

Chapter 1

Introduction

1.1 Research Context

Nowadays a wide variety of devices are available to a user in order to run their applications in a comfortable manner. This introduces the challenge for the developers to implement so called context-aware applications which best suit the user needs. The need for adaptable user interfaces became necessary in order to support different devices and different content. The term of plasticity [Thevenin and Coutaz, 1999] in applications was introduced for that reason. A so called ‘plastic’ application can run in different environments such as a workstation or a PDA device without requiring a complete system redesign and re-implementation. Over the last decades the computer science community tried to tackle the problems regarding adaptivity issues. However adaptivity is a general concept which is very broad, thus there is no unique solution for adaptable user interfaces. The researchers tried to address problems in different areas such as session continuity [Shacham et al., 2007], migratory user interfaces [Bandelloni and Paternò, 2004], context-aware platforms for mobile data management [Norrie et al., 2007], web content adaptation for mobile devices [Zhang, 2007] or self-adaptive systems [Salehie and Tahvildari, 2009].

1.2 Goal of the Thesis

In this thesis we will focus on adaptable user interfaces for mobile devices. More specifically, we show how to represent different information (such as images, audio, text e.t.c.) using a hypermedia approach and how we can change the structure over these elements depending on the device capabilities. We will use the concept of crosslets to show how different resources can be represented in a way that will be comfortable for the mobile user. We are convinced that in order to achieve better results in terms of presentation we need to program atomic crosslets for each resource. The resource will have its own presentation on a variety of mobile devices. Furthermore we will introduce a new subtype of link in order to change structures over a document. While a lot of researchers take into consideration how to adapt the different components of a document for mobile devices, they do not give much attention to change the structure of the document. A good example is the Amazon website which has different versions of the same webpage. In order to maintain these different web pages they need different document structures. They probably even need different persons to maintain the variety of their versions. The logical structure of the document can describe the position where each element is placed inside the document and the way that they are assembled each other. We believe that by using the logical structure of the document we can modify it in such ways in order to create different documents with different structures that can be used for different devices. This could be for instance a good solution for the Amazon website. It could be the case that they could have the logical structure of one generic document and by using different structures to modify it and take a new document for every device. Furthermore we believe that links can be very flexible way of constructing documents. By using links one can easily add and delete individual elements from the structure of the document. We argue that also the structure of the document plays an important role in the adaptation procedure.

1.3 Research Questions

As the goal and the context of the thesis has been clarified we can summarise our research questions as follows:

- **Question 1:** *How can we define structure over a document?*
- **Question 2:** *How can we represent information in different contexts?*

1.4 Methodology

In our work we use the *Resource-Selector-Link* (RSL) metamodel. The RSL uses a hypermedia approach and it can be used for a variety of applications such as personal information management (PIM) or content management systems (CMS). Since the model we use (RSL) considers links as first class objects, our contribution will be to use links to describe structural components and relationships among different resources. Thus we place structure on the same level as resources. Note that we will not give priority to structure over data as sometimes proposed by structural computing [Nürnberg and Schraefel, 2003] but rather consider them to be on the same level. More details about the RSL metamodel are discussed in Chapter 4.

We did an analysis over the different approaches that researchers address the structure of their documents. We found that there is not so much attention on how they construct their documents. There is no clear conceptual model that someone could use in order to create structure. By using the RSL approach we can address structure via structural links. A structural link can help us to create structure of a document and we can easily modify it by adding or deleting individual elements in the document. Adding and deleting individual elements is very easy for the RSL model since it offers element granularity, meaning that each entity that is stored in the model is unique and can be managed separately. We have implemented a model that someone can use and easily modify the logical structure of a document. In this way we will show how this could be beneficial to construct several documents consisting by different types of information.

Furthermore from the analysis that we did we are convinced that most approaches use general XML files in order to describe their elements, meaning that all their resources are managed all together. We will show how to represent each resource via the concept of crosslets. We will use crosslets for different types of media (images, audio) to show how a resource can be represented in different ways. It will be shown how the same information can be represented on different devices and how this is beneficial for the user experience. We will use a Wikipedia page as a use case to demonstrate our approach. We chose this web page because it has a variety of different resources such as audio, images and a tremendous amount of links.

1.5 Structural Overview of the Thesis

The **first chapter** discusses the research context. Afterwards, the goal of the thesis and the research questions are proposed. Then the methodology that is used and finally the structural overview of the thesis are presented.

The **second chapter** provides a background overview about general concepts that related to this thesis. These concepts are divided in three categories. First the hypermedia approach is discussed. Second the context-aware applications and finally the cross-media approach.

The **third chapter** discusses the state of the art and the related work that is done so far. Furthermore an analysis of the related work and a conclusion with our statements are presented.

Chapter four contains information about the RSL model. More specifically this chapter explains in detail our model and how we can create a link in order to change structures in a document.

The architecture and the implementation of our proposed solution is described in **chapter five**.

A discussion about possible issues that may arise are included in **chapter six** along with future work.

Chapter seven provides a summary of the thesis and our contributions.

Chapter 2

Background

2.1 Hypermedia

In order to better understand hypermedia systems, it is worth to give a simple definition of what hypermedia is.

“Hypermedia is a nonlinear medium of information which includes graphics, audio, video, plain text and hyperlinks”¹.

Ted Nelson [Nelson, 1965] first used the term of hypermedia in his paper in 1965. He was inspired by Vannevar Bush’s memex [Nyce and Kahn, 1991] and introduced a file structure that can be used to escape from the limited concept of “What You See Is What You Get”. He argues that this concept is based on printing a document out which limits the use of the digital documents. Nelson started the project Xanadu in 1960 [Nelson, 1990]. This was the first hypertext project. In this project he discusses concepts like non-sequential writing, embedding parts of a document in another document (transclusion), bidirectional links, version and rights management.

In August 1991 the first public release of the World Wide Web [Berners-Lee and Fischetti, 2000] took place. This is the most well-known hypermedia application nowadays. The following years there was an explosion of new concepts and

¹<http://en.wikipedia.org/wiki/Hypermedia>

technologies that were introduced for the World Wide Web. Afterwards, the mobile web created the need for new requirements and functionalities such as location based-services. Furthermore, HTML was not sufficient anymore and other markup languages were introduced (e.g WML). The Web 2.0 gave the user the opportunity to become an author and share their information by tagging or social networking. The Semantic Web with the use of ontologies added explicit semantics to the web resources.

Since mobile devices have different capabilities, the need for context aware applications became necessary. Moreover, individuals that use these applications are characterised by different needs. The problem with the static hypermedia systems was that they provide the same content to all users. In order to overcome the static nature of the traditional hypermedia applications, adaptive hypermedia [Brusilovsky, 2001] was introduced to serve the user needs and preferences.

Our approach is based on a hypermedia model (RSL) proposed by Signer and Norrie [Signer and Norrie, 2007]. We use the RSL model to manage links and to describe relationships among different resources. The iServer platform and the RSL approach are discussed in detail in chapter 4.

2.2 Context Awareness

Context can provide developers the necessary information to describe a certain situation of an application. According to the information that they get they build context-aware applications. A simple definition is given by [Abowd et al., 1999]:

“Context is any information that can be used to characterize the situation of an entity. An entity is a person, place, or object that is considered relevant to the interaction between a user and an application, including the user and the application themselves”.

In his paper he refers to context as any piece of information which is used to describe a situation of an individual in an interaction. As an example he uses location-based scenarios when users are located in different countries. This situation will affect for instance a sum of weights which will have to be computed in pounds or kilograms. He divides context into four categories, which are *location*, *identity*, *activity* and *time*.

It is worth to mention also other points of views to the notion of “context”. For instance Schmidt in his work “There is more to Context than Location” [Schmidt et al., 1999] mentions that context-aware research lacks a general conceptual model. In his paper he introduces the importance of sensors in context aware applications. He mentions that an abstract context can be provided via sensors such as temperature and a very specific context can be provided by a user’s attention level.

Another categorisation can be found in [Schilit et al., 1994]. There are four categories such as *proximate selection* which is a user interface technique where the objects located next to each other are grouped together or emphasised. The other one is *Automatic contextual reconfiguration* which shows how components are treated. Components can be added, deleted or even change connections between each other due to context changes. Next is *Contextual information and commands* which can give different outcomes according to context. Finally *Context triggered actions* are based on if-then rules to specify the adaptation.

In general the computer science community refers to the context awareness concept when computers can sense and react based on their environment. This means that an application follows some rules and reacts accordingly under different circumstances. These rules refer mostly to device capabilities and modalities that may be supported and to the user interaction with the application. Thus, an application is a context aware application if it can make assumptions about the current user’s situation and the environmental state and react accordingly. Context-aware applications are used in ubiquitous or hybrid environments in order to design innovative user interfaces to facilitate the user experience.

Over the last decades context aware applications have been widely used by the mobile developers community. Nowadays there is a tremendous number of mobile devices available with diverse capabilities. Thus, context-aware-applications address the problem of how we can use content in different contexts.

2.3 Cross-Media

A simple definition can be found in wikipedia in order to understand the term cross-media.

“Cross media is a media property, service, story or experience distributed across media platforms using a variety of media forms.”².

Media includes the Internet, video and film, broadcast and cable TV, mobile devices, DVD, print and radio. Cross-media addresses the problem of how to represent information in different media and how we can use the power of the digital media. In the WISE department there are a lot of research projects for cross-media linking including the interactive paper solution. For the interested reader more information about interactive paper can be found in [Signer, 2005]. Furthermore PaperPoint [Signer et al., 2007] is a general prototyping tool for interactive paper applications. The approach consists of having a simple paper and a digital pen in order to make slide-show presentations. This project reveals how someone can use simple paper as an interactive medium which links to digital information.

In fact the RSL model that is used for this project is also a model for structural cross-media content composition and reuse as pointed out in [Signer, 2010]. Furthermore in this project we use crosslets as a powerful concept. Crosslets have the ability to bind a variety of multimodal interactions to different resources. Each resource may have different interactions and different representations. Crosslets decide which presentation matches the best depending on the device capabilities. In that way we treat each resource independently from each other. We then use the cross-media linking concept of the RSL model to construct our single document.

²<http://en.wikipedia.org/wiki/Crossmedia>

Chapter 3

Related Work

In this chapter we introduce some related work. First we analyse different approaches of adaptation procedures. Then we compare all the papers in a table and discuss an analysis it. Afterwards we analyse some of the frameworks that are available and their possible usage. Finally we conclude the chapter with our proposal.

3.1 Media Adaptation

Many researchers tried to address the problem of presentation continuity. Cesar [Cesar et al., 2008] present the benefits of using structured multimedia documents to adapt content in different contexts with session continuity. They propose an approach in which a user can render a video on a high definition screen and at some point can switch the video on a mobile device and continue with the same session. The adaptation is performed almost automatically by the system since it takes into account the characteristics of the various devices that surround the user. In order to describe different structured multimedia formats they use the SMIL [Bulterman and Rutledge, 2004] standard. Furthermore for each device they also take into consideration the different modalities that the device supports as output. Their attention focusses on the visual and acoustic modalities. This work attracted our attention since they use structured documents to describe different media elements. However, all these elements tightly belong to an XML file description. In our work we propose structural links which can treat each element separately

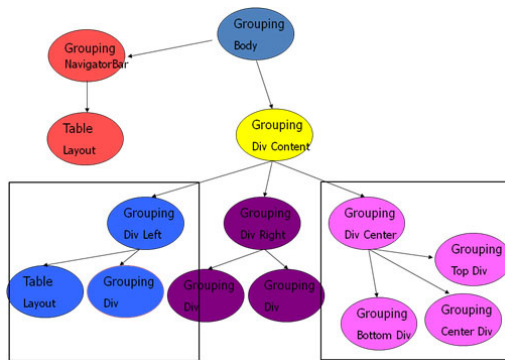
and independently of each other.

3.2 Structure UI for Web Applications

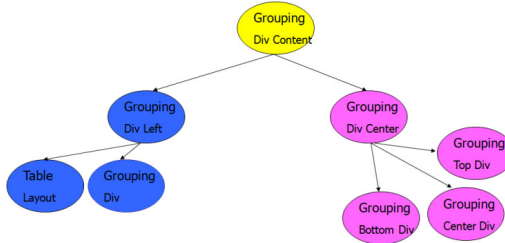
In this section we analyse some approaches that are used to structure HTML documents for web applications.

3.2.1 User Contribution

[Ghiani et al., 2010] try to give to the user the opportunity to interact more with the application. In their approach the user can access a web application on a desktop system in order to perform some interaction, and then, when they have to move, they can migrate the application to a mobile device in which they can continue the task from the point they left off. Before moving the application to the mobile device the user has the opportunity to select only parts of the application to migrate to the mobile device. Thus the outcome is only a light version of the application. The migration server that they use are scripts which are dynamically inserted in the original web application. The application can transfer the state of the Javascript variables, cookies, interactive forms and other features, so the target device takes all the necessary information to continue the session at any point. For instance all the fields that a user has filled in a form will be migrated with the same values on the mobile device. In this approach they use Amazon web page as an example to construct their document. They analyse web page elements (such as div, tables, forms etc.) in order to get the logical structure of the page. The resulting logical structure follows a tree structure manner beginning with a root element. Then root element has two children, the top part implemented via a layout table and the content part defined by a main div. Figure 3.1 shows an example of how the algorithm prunes the logical structure of the elements, 3.1a shows the structure with two groups of elements selected by the user and 3.1b shows the resulting logical structure. There are three steps to get the resulting structure of the final tree. First they prune all the parts of the tree that are connected with parts of UI that have not been selected (*interactor pruning step*). Then, the connections to other pages defined by UI elements that no longer belong to this new partial CUI are deleted (*connection pruning step*). Finally there is a *tree reduction step* in order to discard any useless redundancy in the logical structure of the tree. This last step is performed in order to avoid nodes having only one child.



(a) The logical structure of the page considered in the example (at the beginning, before a partial migration)



(b) The resulting logical structure of the UI, after the tree pruning and reduction due to a partial migration request

Figure 3.1: An example of pruning the logical structure of a document [Ghiani et al., 2010]

In a recent implementation, [Paternò and Zichittella, 2010] proposed a tool for desktop to mobile adaptation. The solution also supports end-user contribution to customise multi-device ubiquitous user interfaces. They exploit logical user interface descriptions able to capture interaction semantic information indicating the purpose of the interface elements. In their approach the user can specify values such as maximum font size or how to treat radio buttons of a web application that he/she uses.

The system runs an algorithm which calculates the different costs for each element and groups of elements. Figure 3.2 shows the adaptation algorithm. The cost is calculated recursively and if the cost remains high then it splits the original page into multiple mobile screens. The tool is based on an automatic re-authoring solution and it uses a proxy to exploit the logical descriptions. The proxy server contains a semantic redesign module that creates an interface suitable for a platform different than the desktop.

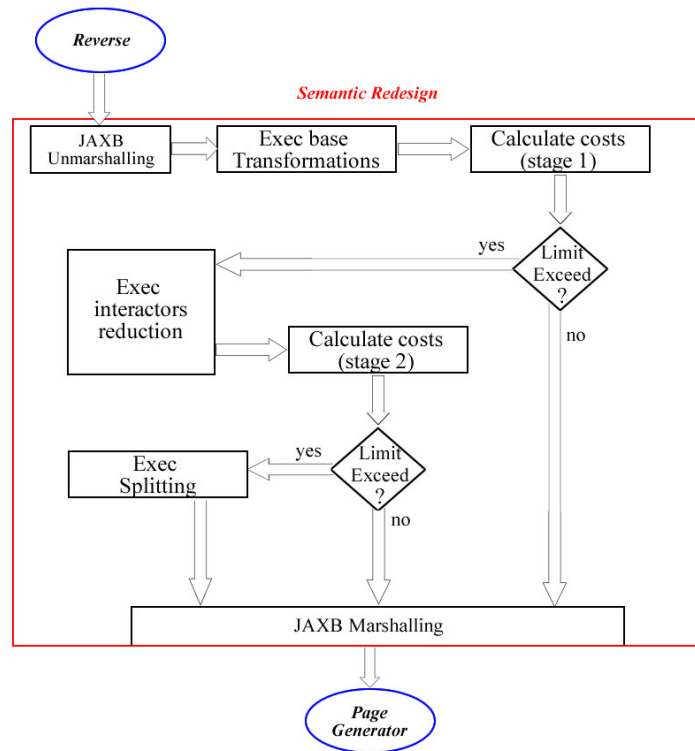


Figure 3.2: The adaptation algorithm [Paternò and Zichittella, 2010]

3.2.2 User Profile

Another interesting approach is the framework that is proposed in [Lemlouma and Layaïda, 2003]. The interesting feature that they propose is that they use XSLT [Clark, 1999] for structural transformation of documents and resource-aware transcoders for the media adaptation, thus they can tackle the frequent changes of the different environments. The framework is based on the Universal Profiling Schema UPS [Lemlouma and Layada, 2002] for describing the environment characteristics and on a profile exchange protocol. One of the main advantages of this framework is that the server receives only the necessary information when needed, for instance the server receives only the network profile when it is needed, thus avoiding redundant information. This framework is used in the OPERA project [Lemlouma and Layaïda, 2002]. Despite the fact that XSLT can be applied to change a document structure, XSLT cannot ensure advanced adaptations such as those depending for instance on the client screen size. Thus, the structure of the document does not take into account the capabilities of the target device. Another drawback is that in practice it is complicated to use XSLT templates for every different client profile, thus in their approach they use only

a generic XSLT that handles the constraints and the original document in order to take the final document structure. Figure 3.3 shows the generic transformation that is applied.

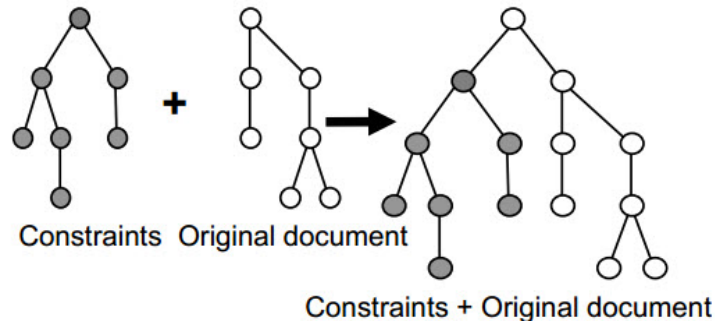


Figure 3.3: Support of user constraints by concatenation [Lemlouma and Layaïda, 2003]

As mentioned above in order to tackle some limitations of XSLT they use resource aware transcoders for media adaptation such as images and videos. We consider this also a drawback since in our approach adaptations are performed on any kind of resources via our crosslet approach. Since this paper was published eleven years ago, we thought that it is worth to give it some attention for its innovative approach.

3.2.3 Device-Centric Approach

Migratory user interfaces proposed by [Bandelloni and Paternò, 2004] offer a good solution in terms of session continuity and adaptation for different devices such as PDAs. In their solution they propose an implementation where the user can interact with a mobile banking application on a desktop device and then the user can switch to a PDA device and continue the navigation from the device. At some point the user can switch back to the desktop by continuing with the same session. They use a mapping algorithm in order to adapt the web page to the requirements on the target platform. The resulting web page is structured in a manner that is more comfortable for the user to interact with. In terms of adaptation it is worth to mention that different tasks are treated differently for each device. For instance if the user wants to reserve a ticket before he/she goes to the bank from the desktop application, the “ShowRealTimestate” task is composed by different objects than in the PDA application. However they do not use the granularity effect to treat the different objects while in our proposal we want to treat

each object separately through crosslets and then use hyper-links to construct our document.

An application that supports task continuity, adaptation across multiple devices and implementation languages is proposed by Paternò [Paternò et al., 2010]. Since users nowadays are surrounded by several devices and appliances, there is a need for session continuity across these devices. The problem is that not all the devices support the same implementation languages and we need also several user interfaces across devices. In this paper they address the problem of session continu-



Figure 3.4: The PDA interface of the shopping application [Paternò et al., 2010]

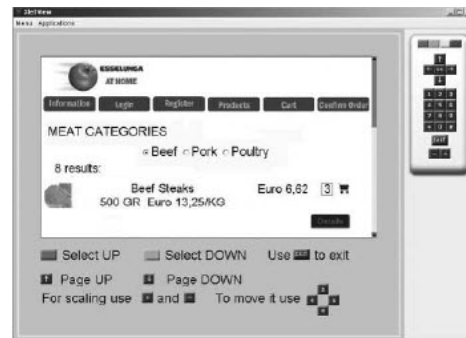


Figure 3.5: The DTV application interface [Paternò et al., 2010]

ity and how to generate dynamically the user interface for various target devices without the need to start the application session from scratch. They use mobile devices and digital TV's in order to show that their approach supports different implementation languages. The client devices subscribe to the migration service by running a migration client agent that provides information regarding the device characteristics. The devices access Web applications through the migration server, which includes proxy functionalities. Migration can be triggered by the user or some specific external event (e.g. very low battery). They analyse the device char-

acteristics through an intelligent agent and according to some rules they generate the most suitable user interface. These rules consider the device descriptions that are sent through an XML file. This file includes information about the state of the device, if the device is personal or can be used by a group, the type of the device, the id of the device, the operating system that is running on the device, etc. Then they build logical descriptions for the user interface and from these logical descriptions through semantic redesign techniques they build the corresponding UI for the target devices. At this point it is worth to mention that the logical description operates at a concrete level and for instance if they have a textual label it is not yet specified if it will be implemented in Java or XHTML. This information is defined in the final level of the UI, which is its implementation. As a use case they illustrate a scenario where a user can make online shopping via his/her mobile device and when he/she comes home can continue his/her shopping through a digital TV (DTV) or a desktop computer with the state updated to the point that was left off in the previous device. The user interface in different devices changes drastically for every circumstance along with the diversity of functionalities that the user may have across platforms. For instance some tasks are not supported on the mobile device while in the DTV they are supported. Figures 3.4 and 3.5 show the different user interfaces. Another difference that can be seen on the representation level is that a combo box (displays only one choice at a time) for selection is provided for a mobile device and a radio button (displays all the choices at the same time) in DTV.

3.2.4 Structure UI based on the Cultural Background

In terms of structuring a user interface Khaddam [Khaddam and Vanderdonck, 2010] proposes a solution where the user interface (a web page or an application) can change structure depending on the cultural background of the user. The paper addresses the RTL (Right To Left) and LTR (Left To Right) problem and shows how the user interface can be structured in a manner that serves both cases without losing functionality and maintaining the usability requirements. They use the USIXML [Limbourg et al., 2005] framework in order to construct the concrete user interface (CUI).

In their implementation they rely on some values that they define on an XML file such as “normal” or “reverse” in order to define if the objects (radio-buttons, list boxes etc.) will be presented in a RTL or LTR manner. The most attractive part of this work is that each element is treated differently, so if for an element the RTL presentation is not supported then a LTR presentation is applied. Figure 3.6 shows

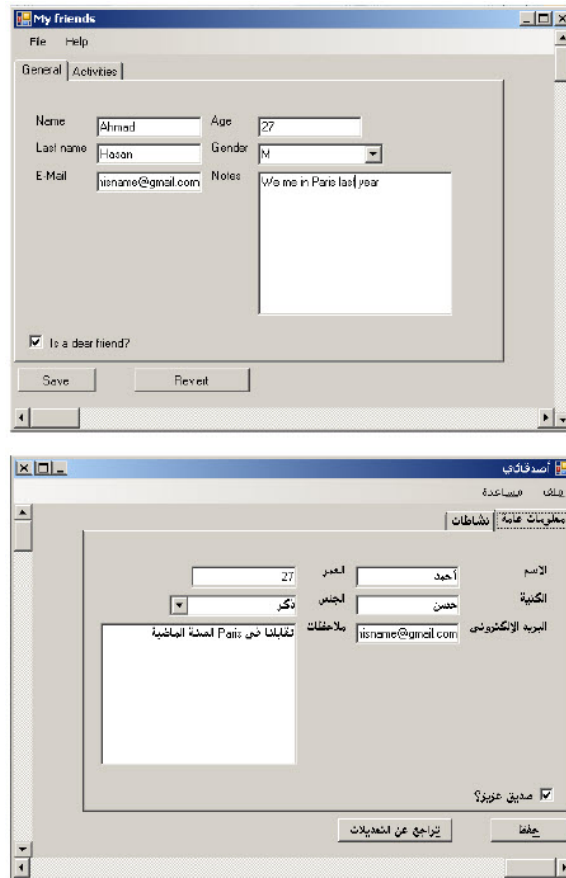


Figure 3.6: A sample English UI and the localised RTL version (in Arabic Language) [Khaddam and Vanderdonckt, 2010]

such a UI and as it can be seen the email address follows an LTR manner while the whole other UI follows a RTL manner. However this sometimes may mislead RTL users and lacks in concreteness of the web page or application since parts of the application will have different orientations. For instance from Figure 3.6 it can be seen that the word *Paris* follows a LTR orientation. Furthermore they do not address mobile adaptation case so they do not show how they construct their files according to different device capabilities.

3.2.5 Device and User Preferences

An innovative approach for context-aware systems is proposed in the SECAS project [Chaari et al., 2007]. In this paper each element presentation depends on the

device capabilities and user preferences.

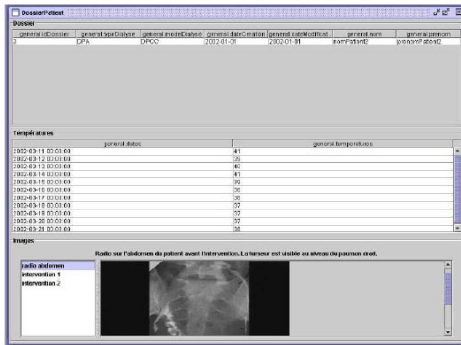


Figure 3.7: A dialysis patient record on a standard PC [Chari et al., 2007]

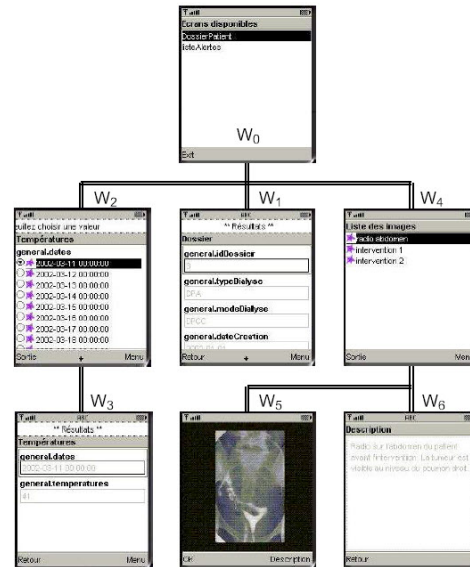


Figure 3.8: The same dialysis patient record on a smartphone [Chari et al., 2007]

All this information is received in the server in a XML file. First they construct the generic abstract components (*SecasTextComponent*, *SecasImageComponent* etc). This defines a generic logical model of the window. This model is independent from the target device. The second step consists in selecting the concrete visual components and their layout on the screen to provide the final physical model of the window. This selection is based on a specific user interface vocabulary which describes the available user interface components on the target device, their behaviour (instantiation and methods) and their layout on the target screen. Figures 3.7 and 3.8 show the same dialysis patient record on a standard PC and a smartphone. This approach is close to our vision of modelling since also in RSL approach first we define the abstract components and the logical structure of the document and then through the crosslets we define the presentation of each element. However in our approach we treat each component as an independent entity which can be linked to other entities or even other documents while in their approach all the elements are parts of the same entity. As an example with RSL we would use the *temperatures* component as a unique entity and then we would put a link to that entity on a standard PC presentation or a link to the same entity to the smartphone presentation.

3.3 Available Frameworks

In this section we illustrate some of the available frameworks that are used by developers for context-aware applications.

3.3.1 TERESA

The increasing availability of the new hand-held devices forced the researchers over the last decade to investigate new frameworks to support multi-platform applications. One valuable work that was done in 2004 was TERESA (Transformation Environment for inteRactivE Systems representAtions) [Berti et al., 2004]. This framework is a model-based authoring environment that provides support when developing and designing interfaces for a variety of devices. They use a top-down approach to obtain interfaces for different types of devices via logical descriptions. In the first step they use the ConcurTaskTrees notation [Paternò, 1999] for the high-level task modelling of a multi context application. The second step consists of developing the task model considering the diversity of platforms. Here the designer has to filter the task model according to the target device. The third step is to define the abstract user interface which means the set of presentations which are identified by an analysis among the task relationships. Finally the user interface generation takes place. This phase is completely platform dependant and has to consider specific properties of the target device. Each abstraction level can be described in TERESA using an XML-based language. TERESA offers the flexibility of different entry points for the abstraction levels. For example a designer may want to start with a logical interface description and not with the task model. Furthermore TERESA offers an environment for generating XHTML interfaces for desktop, mobile phones and VoiceXML user interfaces. However TERESA framework also has its own limitations. One of them is that it does not support complex dialogs and parallel inputs. Another one is that the transformations in order to generate the corresponding implementations are hard coded in the tool and are not specified externally which would allow for customisation without changing the tool implementation. Events that may have different behaviour depending on the type of object that these events are associated with is not supported. These issues led the authors to investigate a new framework that overcomes all these limitations.

3.3.2 MARIA

As mentioned above TERESA framework has some limitations on its functionalities and this led the authors to create a new framework to overcome these issues. MARIA (Modelbased lLanguage foR Interactive Applications) [Paternò et al., 2009] was proposed by Paternò as a multiple abstraction language for service oriented applications in ubiquitous environments. MARIA XML inherits the modular approach of TERESA XML and introduces some new models. They introduce a data model which is described in the XSD [Gao et al., 2009] type definition language. An event model supporting property change events to change the status of some UI properties and activation events to activate some application functionality (e.g. access to a database) were introduced. Furthermore they added support of Ajax scripts for continuous updates of fields. Another feature is the introduction of a dynamic set of user interface elements which gives the flexibility to change only a part of the UI. In their architecture they have an abstract interface description which proved beneficial for the designers since they do not have to learn all the details of the many possible implementation languages supported by the various devices. Thus, the design phase is not tied to a particular platform, modality or implementation language. As a consequence we can have one data model and one or more presentations. There is also a dialog model which is responsible for the available events and interactions at a given time.

The concrete description provides a platform dependent description of the user interface but not implementation language dependent. As an example of the concrete description they can define general events like changing the orientation of the screen as well as some gesture events that are common along different devices such as *touchStart* or *touchEnd*.

They also provide functionality for web services. MARIA supports user interface annotations. The annotation's goal is to provide hints for creating the user interface to access a web service. In a web service-based application they also support UI composition. The main goal of UI composition is that designers and developers often have to compose all the existing functionalities and corresponding user interface specifications. These can be identified via the annotations discussed before.

Finally there is an authoring tool available which supports the user interface specification. There is a general transformation that supports the analysis of the document structure and definition of mappings. In addition there is also a document instance transformation when the user needs to change a transformation rule for a small part of a specific document.

3.3.3 USIXML

Multi-device user interfaces are discussed also in USIXML [Limboung et al., 2005]. USIXML stands for USer Interface eXtensible Markup Language and allows designers to apply a multi path development of user interfaces. They define four levels of abstraction, Task and Concepts, Abstract User Interface, Concrete User Interface and Final User Interface. The development process can be initiated from any level of abstraction and proceeds towards obtaining one or many final user interfaces for various contexts of use at other levels of abstraction. USIXML can be used by designers, analysts or even novice developers. It consists of a User Interface Description Language (UIDL), that is a declarative language capturing the essence of what a UI is or should be independently of physical characteristics. USIXML supports device and platform independence, which means that it remains autonomous with respect to the devices used in the interactions such as mouse, keyboards and other devices. Furthermore with respect to the various computing platforms such as mobile phones or tablets. USIXML is used in a lot of research projects since it offers a flexible way to design a UI supporting multiple devices.

3.3.4 CHISEL

A policy driven, context-aware, dynamic adaptation framework is Chisel [Vanderdonckt, 2005]. In this paper it is discussed how software should adapt itself to changing requirements and changing content. Chisel is an open framework for dynamic adaptation of services using reflection in a policy driven manner. The system is based on decomposing the aspects of the service objects into possible non-functional behaviors. These behaviors are meta types that can be statically or dynamically associated/disassociated with the service object. The service object will be adapted to use different behaviors, driven by a human-readable declarative adaptation policy script. They achieved that by implementing the meta types with Iguana/J [Redmond and Cahill, 2000], which supports non-invasive dynamic associations of meta types to service objects without any requirement to interrupt, change or access the objects source code. Absolutely no changes were required to the application code and all adaptations can occur without stopping the client or server application. In order for an adaptation to occur, the context changes that may trigger some adaptation must be monitored. The context manager should then leverage all available context knowledge and intelligence to determine if some adaptation is required. A separate adaptation mechanism, controlled by an adaptation manager can then perform this triggered adaptation as a response to an

adaptation request. The goal of Chisel is to use unanticipated context. However this does not seem possible since it is the policy that drives the meta-level adaptation manager, thus it has to obey to some strict rules.

3.3.5 MADAM

Another available framework for context-awareness is MADAM [Geihs et al., 2009]. The adaptation in MADAM happens without user intervention and follows a model driven approach. They implemented abstract adaptation models in order to follow an unanticipated adaptation for a variety of devices and they have a middleware that supports the dynamic adaptation of component-based applications. The adaptation policy should be expressed in utility functions that are specified by the developer. In order to represent a component we may have different variants for each type of context, user requirements or application properties. These variants are defined by the utility functions. In MADAM there are atomic and composite components. Composite components follow a tree structure approach which ends up into leafs and each leaf then is an atomic component. Since MADAM is a generic framework it provides the developer with abstract concepts like “*ContextEntity*” and then its up to the developer to define his own entities. During application runtime the MADAM adaptation middleware monitors the context and adapts the application in response to context changes that may occur according to the structure and adaptation rules reflected by the platform-specific adaptation model by finding alternative applications or service implementations. However since the user is not involved at all in the adaptation procedure it may be an issue of how often an adaptation occur without being an annoyance to the user. This is an issue that is discussed also in the paper along with the question of how we can validate a self-adaptive software system that is able to handle unanticipated adaptation at runtime.

3.4 Summary

In this section we discuss of the related work that is done so far by the community including a table 3.9 of comparison. We highlight some of the issues that we address for each approach. Then an analysis of the table in order to motivate our work is discussed. Finally a conclusion of our research is given.

3.4.1 Comparison Table

An explanation of every header that we chose and the rating values that correspond to each column is provided as follows:

- **Model** defines if there is any specific framework which is used for the implementation of application.

Paper	Model	Layer Adaptation			User Influence @runtime	Adaptation Granularity	Dynamic Modification @runtime
		Device Characteristics	User Preferences	Structure			
<i>Multimedia Adaptation in Ubiquitous Environments.</i>	None	Partial	✓	Limited	✓	✗	✓
<i>Adapted Content Delivery for Different Contexts.</i>	None	Limited	✗	Limited	✗	✗	✗
<i>On Demand Cross-Device Interface Components Migration</i>	MARIA	Limited	✓	Limited	✗	✗	✓
<i>Desktop to Mobile Web Adaptation through Customizable Two Dimensional Semantic Redesign</i>	MARIA	Limited	✓	Partial	✗	✗	✗
<i>Migratory User Interfaces Able to Adapt Various Interaction Platforms</i>	TERESA	Limited	✗	Limited	✓	✗	✓
<i>Adapting UsiXML User Interfaces to Cultural Background</i>	UsiXML	✗	✓	Limited	✗	✗	✗
<i>Ambient Intelligence for Supporting Task Continuity across Multiple Device and Implementation Languages</i>	TERESA	Partial	✓	Partial	✓	✗	✓
<i>The SECAS Project</i>	None	Partial	✓	Partial	✓	✗	✓

Figure 3.9: Table of Comparison

- **Device** stands for the capabilities of the device. We consider limited support if the application takes into account only basic characteristics of the device such as screen size. Partial support is if the application uses also some of the modalities of the device but not all. Finally we consider full support if all the device characteristics are taken into account. Note that we consider modalities as output resources of the device.
- **User Preferences** is the column where we consider whether the user may enter some pre-defined preferences before the adaptation occurs. Here the

application takes into account the user preferences in order to perform the adaptation. The rating values are expressed in a boolean manner with *yes* and *no*.

- **Structure** is the way that the application constructs a document. By structure we mean the way that each element of a document is assembled with other elements. One common structure of a document is its logical structure. The logical structure of a document can give us the order of the elements that compose the entire document. We consider limited support if the initial logical structure of the document does not change, which means that since we have the logical structure of the document this structure is followed across all the devices. Partial support if the logical structure changes across different devices by adding or deleting nodes. Finally full support means a changing structure for even atomic elements.
- **User Influence @runtime** considers the user influence in the adaptation procedure during the execution of the application. For instance if the user can dynamically choose the platform that he wants the application to be adapted to, this is considered as a dynamic influence of the user during the procedure. Another example is if the user can give specific values for the representation of components (e.g. font-size). We need also here a boolean *yes* or *no* to express our rating values.
- **Adaptation Granularity** defines how the user interface elements are treated. Elements can be seen as a concrete group of elements or we can have atomic elements that we manipulate in order to make a user interface. Thus, here we examine if the application produces an output for every element or one output for all the elements.
- **Dynamic Modification @runtime** indicates if there is the possibility for the application to perform adaptation without the user influence, meaning that the application takes the initiative for the adaptation. This modification may occur on server or client side. Again a boolean value will be used.

3.4.2 Analysis of the Table

Table 3.9 shows different works which are used in order to tackle adaptation procedures. As it can be seen most of the approaches (five out of eight) are using frameworks to build their applications. These frameworks are discussed in detail

in 3.3. In general these frameworks give the flexibility to build general models which are independent from specific programming languages and are used in a variety of projects. Another thing that was our intention to investigate is how different works handle mobile device capabilities. Almost all (except one) of our approaches consider the device capabilities to build their applications. Even if some of the papers are flagged as having *limited* support, we believe that this is enough to achieve a good adaptation on the devices. User preferences and user influence at runtime are also important in the adaptation procedure. Most of the approaches (six out of eight) are taking into consideration the user preferences. Half of the approaches are giving to the user the possibility to influence the adaptation at runtime. However sometimes it is not clear enough how the user can achieve a good adaptation like in [Bandelloni and Paternò, 2004] where the user can give specific values in text fields, but it is not clear how the user knows prior which size of text is suitable for his device. Moreover we have seen that half of the papers are able to do adaptations at runtime which means after the initial adaptation of the content, the application can adapt this content again dynamically.

The granularity effect of the elements is one of our core concepts in our research. As it can be seen none of the research projects consider the element granularity effect. Each adaptation is considered as a single unit. We believe that this is a burden that we try to overcome via the granularity effect that our RSL model offers. The RSL metamodel uses a hypermedia approach to manage different elements. Each element can be stored in the model as an atomic entity and then we can manipulate each entity separately, change its representation, combine elements together e.t.c. Structure also is an important factor in our research. Changing structures involves a radical change in the way that elements of a document are assembled together. Even if applications change structures during the adaptation procedure, there is lack of a conceptual model which may lead in different interpretations. Structure in these papers seems to be more a domain specific issue rather than an important factor which can lead to more flexible solutions. We argue that structure is a crucial factor in the adaptation procedure and we propose a conceptual model able to build complex structures.

3.4.3 Conclusion

We provided some related work on the adaptation techniques and context-aware applications. From the state-of-the-art analysis we concluded that there is not so much attention to the structure of the documents and how this could affect the adaptation procedure. Furthermore we see that the granularity effect is not

considered at all. We argue that granularity in such applications is important since components can be treated separately and can be reused. In the next chapter we aim to illustrate our vision of how we can construct a document by using the granularity effect from RSL and how different structures can be beneficial to the presentation of an application depending on different device capabilities.

Chapter 4

Modelling with RSL

Most of the existing applications do not give priority on content processing and content management. For instance in a pdf file most applications simulate a printable version of a document, which means that the content of the document does not have any semantic information. This approach has one main drawback, the applications are limited to a simulation of the physical space rather than the digital space. This means that we do not use the power of the digital information but we make a digital simulator to represent a physical environment. In this chapter we present the *Resource-Selector-Link* (RSL) metamodel which is used for this thesis. We introduce the powerful concepts of such a model in detail.

4.1 The Resource-Selector-Link (RSL) Metamodel

The core concepts of the RSL metamodel is divided in three parts, namely: resources, selectors and links. These three concepts result in the name “*Resource-Selector-Link* (RSL) metamodel”. The RSL metamodel uses a hypermedia approach. The essential concept of RSL is linking resources together as proposed by Vannevar Bush in his Memex approach. A lot of hypermedia systems have been developed during the last decade [Millard et al., 2000] [Halasz and Schwartz, 1994]. However as pointed out by the authors of [Signer and Norrie, 2007] the existing approaches are not general and flexible enough to support a wide range of applications. The RSL metamodel has been used for projects like iPaper [Norrie, 2006]. The iPaper framework allowed the creation of PaperPoint [Signer et al.,

2007] (a paper-based presentation and interactive paper prototyping tool). A system to support the reading of publications called Print-n-Link [Norrie et al., 2006] that also developed with the RSL notion. In addition a cross-media information platform was created which has the RSL model as basis. This platform is called iServer [Signer, 2005] and it is able to support different categories of hypermedia systems due to its generality. Furthermore iServer via plug-in mechanisms has been proven an extensible solution for a variety of applications. In this thesis we use this extensibility through plug-ins in order to implement our application.

4.2 Entities, Resources, Selectors and Links

It is crucial to mention that RSL uses the semantic of the object-oriented data model OM. The interested reader can find more details about OM in [Norrie, 1994]. OM is used in order to exploit powerful features such as multiple classifications and order collections in the metamodelling process. OM provides in its core functionality some distinguishing features such as *typing* and *classification*. While typing deals with representation of entities as objects, classification deals with the semantic roles of the entities. With OM we can define collections for semantic grouping of entities. Furthermore we can define constraints over the collections. For instance one can constrain relationships among collections depending on the membership type. OM defines a fully operational model over objects, collections, associations and constructs for their definition. This expressiveness allows us to capture the semantics of the application domain by using a simple set of constructs. The main advantage of this model is the direct representation and manipulation of associations which is really useful for link management systems with hypermedia functionality.

4.2.1 Entities

The RSL model introduces its most general concept as an *entity*. Entities can be classified and grouped by collection entities. Entities can have properties that are assigned in a form of key-value pairs. Resources, selectors and links are sub-types of the entity concept. Figure 4.1 shows the core of the RSL model.

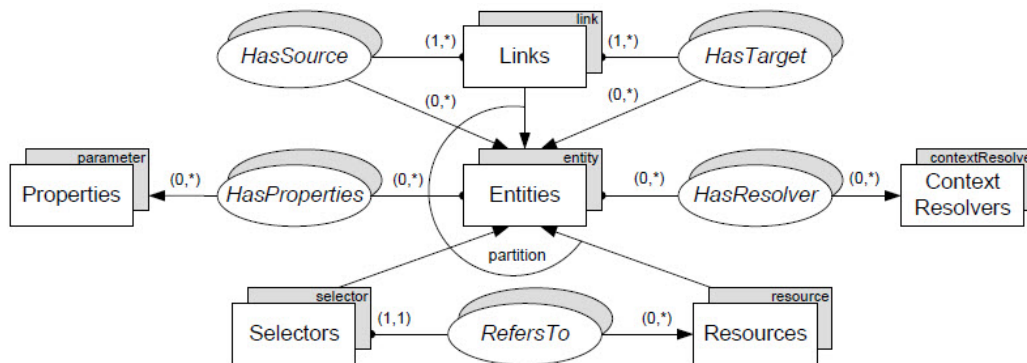


Figure 4.1: Core link metamodel [Signer and Norrie, 2007]

4.2.2 Resources

The first type of an entity is a resource which represents an entire information unit. Of course the resource is still an abstract concept for the iServer implementation and we need to implement specific resource types for a specific application domain. Thus in our implementation we implement a variety of resource types and we use them as a plug-in mechanism for the iServer. Since we are focussing on a web page application we define resources such as *paragraph*, *table*, *bulleted list*, *image* etc. Note that other resources that can describe the physical space of an application can be covered by RSL (for instance interactive paper). As it can be already seen iServer offers flexibility via this abstract concept to cover a variety of application domains.

4.2.3 Selectors

The second main type in RSL is the selector concept. Often we want to define links between not only the entire resources but also specific parts of resources. An anchor link on a web page is an example. Through the *href* tag we can address only a specific part of the HTML document. In our implementation we also address specific parts of a web page via a *url-selector* that we implemented in order to take the external links that we may have in our HTML document. With respect to the RSL model cardinality constraints each resource may have more than one referencing selector while a selector can always be associated to exactly one resource. Specific details of the implementation of such a selector will be given in chapter 5.

4.2.4 Links

Finally a link within the RSL hypermedia metamodel is always directed and leads from one or more sources to one or more targets. A source can be an entire resource or parts of a resource addressed by a selector. Once again respecting the cardinality constraints of our model each link must have at least one and possibly many sources and targets. In the next section we describe a general *document-link* which is implemented by Tayeh [Tayeh, 2012].

4.2.5 Layers

As already explained in the previous section, selectors are used to address parts of a resource as a link source or target entity. However, sometimes we have to deal with overlapping selectors which is a so-called nested links problem. For instance we may have a selector that specifies a word in an XML document while another selector specifies a character in that word. The problem arises when we need to select the character selector we have to distinguish the different layers that exist in order to not select the word selector by mistake. Even if some hypermedia approaches support the nested links they do not support any functionality to control their behaviour. The notion of layers that is shown in Figure 4.2 was introduced

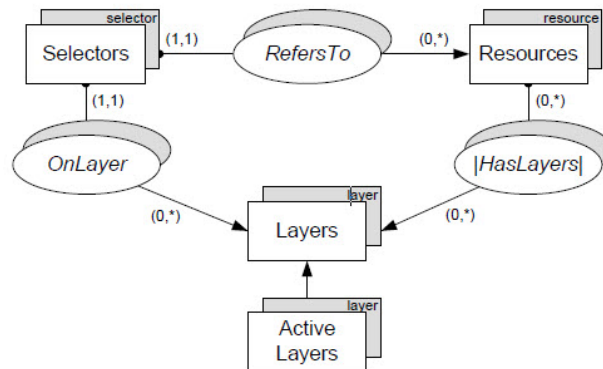


Figure 4.2: Layers [Signer and Norrie, 2007]

in RSL in order to give some flexibility with such issues and keep the semantics of nested link source and target anchors.

Layers allow us to associate each selector with exactly one layer. Thus, we are forcing our overlapping selectors to be defined in separate layers. The problem of multiple occurrences of elements and ordering is solved by the association

|*HasLayers*| with the vertical bars. This indicates that we have explicit ordering of the layers in our model. Furthermore we may activate or deactivate layers by adding or removing them from our *ActiveLayers* collection.

4.2.6 Structure

Structure is a topic that we give much attention in this thesis. Since structure plays an important role in our model, it is worth to give a simple general definition from Wikipedia¹ that reflects the semantics of a structure.

The structure of a thing is how parts of it relate to each other, how it is “assembled”.

The word “assembled” is translated by our model through links. Links are responsible of how to connect parts of information in order to build a “thing”.

In RSL as links are first class objects, we place structure on the same level as resources and navigational links. We illustrate an extension of our model in Figure 4.3. As it can be seen, modelling structural links as a subcollection of regular links gives us the flexibility to define structure over arbitrary entities such as resources, selectors or even links. A single structure from the *Structures* collection is related to its structural links by the *HasElement* association. Note that parts of structures might be reused by other structures. If we want to model the structure of a document we need to know the specific order of its contents. We achieve that by using the sub-association |*HasChild*| of the association *HasTarget*.

The first type of structure is to define structural relationships between resources (*structure over data*). It is up to the domain specific application how it defines different structures over resources which can be chapters or sections in a document. We may even use the concept of transclusion that was suggested by Nelson [Nelson, 1992] and reuse the same resource for different structures.

Structural links may also define *structure over structures*. Each structural link within a structure can define a substructure which contains the structural link’s source elements and all of its children. In order to address a substructure we do not define a structural link to structure elements but rather the corresponding

¹[http://en.wikipedia.org/wiki/Structure_\(disambiguation\)](http://en.wikipedia.org/wiki/Structure_(disambiguation))

structure link defines the substructure. An example could be a chapter that could be referenced by other documents.

Lastly we may define *structure over links*. Thus, we can use navigational links in relation to each other. For instance we may use a tree-like structure with nodes of navigational links defined by a single structural link. Then it is up to specific application domains how they treat such a structure.

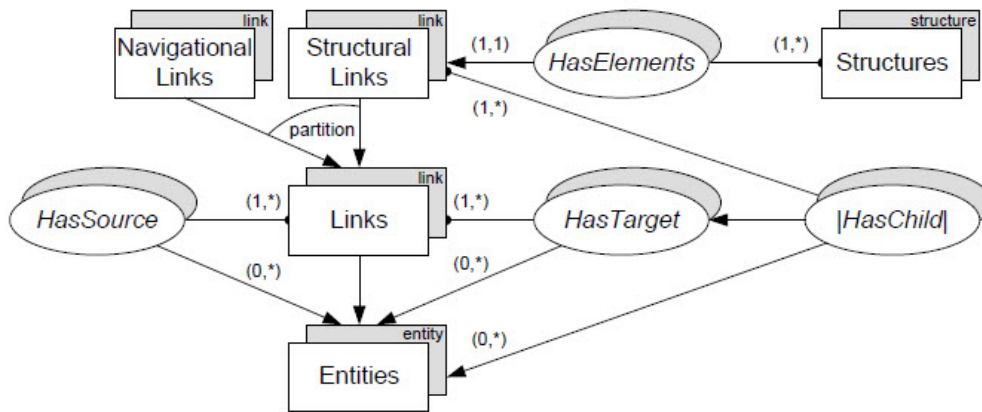


Figure 4.3: Structural and navigational links [Signer and Norrie, 2007]

4.3 Elements in RSL

In this section we demonstrate how the different entities can be stored in our model in terms of granularity level. First we present the architecture that is behind our model and then we illustrate how granularity can be achieved via a simple example.

4.3.1 Granularity in RSL

The highest level of abstraction in a logical structure of a document is the document itself. Thus, the logical structure is a collection of atomic and higher level elements combined together to define the highest level of abstraction as mentioned in [Weibel et al., 2007]. Some of the existing logical models [Reid, 1981] [Dori et al., 1995] tried to address the problem of defining different hierarchies of objects by using tree structures. One of the most interesting models is *tnt* [Furuta

et al., 1989] which uses a forest of ordered trees to represent the different document parts. In this model the atomic values are defined in a heterogeneous way at a higher level of granularity. Instead of storing single characters as leaves of a tree, the atomic values might be represented as a whole text string. By encapsulating non-tree structures into the leaves, the primary logical structure remains relatively simple and easy to manipulate, while the storage of other structures might be addressed at another level in an easier way. This model resembles a lot our RSL model

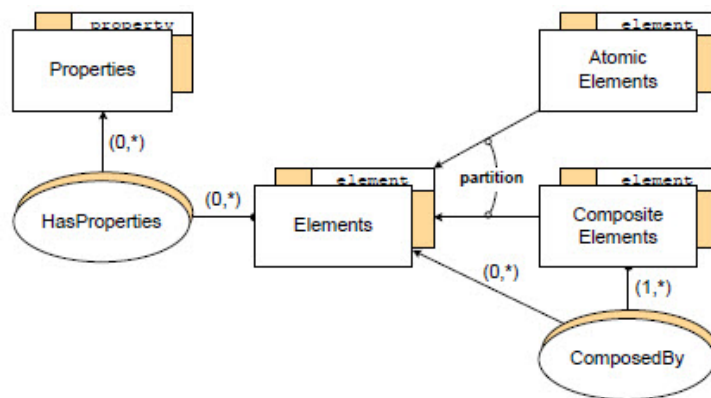


Figure 4.4: Elements [Weibel et al., 2007]

However RSL goes one step further by resulting in a recursive element hierarchy. The composite pattern is used in order to represent this behaviour. Figure 4.4 illustrates that elements may belong either to the *AtomicElements* collection or to the *CompositeElements* collection. An element that belongs to *AtomicElements* means that this element does not contain any lower level elements. A partition constraint is defined in order to avoid the fact that an element belongs to both collections. In order to build the hierarchical structure, the *ComposedBy* association is defined.

The key step is the analysis of the source document based on the logical model. Through the analysis of the source document we can easily identify the highest level objects in our hierarchy. These objects can be chapters, paragraphs or even sections of a document. In RSL we define elements at the highest possible level of granularity and store them as atomic elements within the model. The main advantage of this approach is that we drastically reduce the number of objects that we initially have to store in our model. However RSL is intended to be used for any level of granularity. Often low level objects are not in practice required. For example in a paragraph we do not need a character level of granularity in most of the cases. The idea of RSL is to store the elements at the highest possible level of

granularity and then define composite elements if needed.

4.3.2 An Example with Different Granularity Levels

Let us assume that we have a document that contains only one paragraph and examine how many levels of granularity we can reach. Note that the red rectangles of Figure 4.4 represent an atomic element instances that are stored in our model while the green rectangles represent composite element instances.

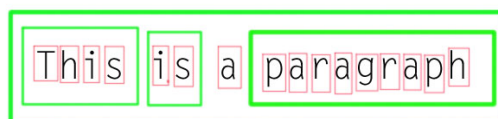
As it can be seen from Figure 4.5a in our model initially we identify a paragraph as an atomic element and we store it in our model. The first step of the process stores only one atomic object in our model, then if needed we can reach lower levels of granularity. The words that are contained in that paragraph can be further used



(a) An atomic paragraph element



(b) A composite paragraph element contains atomic word elements



(c) The composite word elements contain atomic character elements

Figure 4.5: An example which illustrates three different granularity levels

to create a composite element paragraph that contains atomic word elements. In Figure 4.5b our paragraph object switched from atomic element to composite. This composite element now contains atomic word elements. Then if we need

lower levels of granularity we can further switch the word elements from atomic to composite and by using the characters of the word elements to create atomic character elements, Figure 4.5c illustrates that scenario. Note that the word “a” is already an atomic element that cannot be switched to composite because it contains only one character element, thus the “a” word remains as it is.

We consider this a flexible way of storing our objects since we only create new objects if needed. Thus, in RSL we do not need to analyse at first all the details of the document in terms of granularity levels which would be really a challenge, but we just need to identify the highest possible element abstractions and store them in our model with their metadata. Note that this simple example cannot reveal all the power and functionality of our RSL model since this particular paragraph used in the example just contains text elements. There are much more complicated elements that we discuss in section 5 that give the general idea behind our powerful RSL model.

4.4 Document Component Link

In this section we present the abstract concept of *document-component-link* the model implemented by Tayeh [Tayeh, 2012] in his master thesis. After the analysis of the *document-component-link* approach we show its functionality with an illustrated example.

4.4.1 Document Component Link Approach

Since we already illustrated how we instantiate and store our elements in our model in section 4.3.2. Let us discuss how all these pieces of information are structured together. In order to do that we use the concept of links of our model. Since the link concept is already discussed in 4.2.4 we jump directly to explaining the *document-component-link* approach that is implemented by Tayeh [Tayeh, 2012] in his master thesis. A simple explanation of a *document-component-link* is that this link is responsible to connect our different entities inside a document and it is able to give us from that connections the logical structure of the document.

In general this link provides functionality to interpret the entire logical structure of a document. Everytime that we instantiate such a link we can define a source and a target and of course the creator of the link. In the instantiation phase our

link may have no targets so the target attribute may be *null*. The document link is responsible to maintain associations between resources, selectors or other links. At runtime we may add one or more targets to our *document-link*. These targets may be other resources, selectors or even other links.

This link provides methods to get the target entities that we added in the order that we added them to its target collection. This gives us the flexibility when we construct our document to “target” our entities in the manner that we prefer and get them back in the same order. In practice when we add a target to the *document-link* we define ‘where’ this target ‘stands’ in our structure. Note that “stands” does not mean that our target entity is committed to one structure. For example let us assume that we have a *document-component-link* that has as source the paragraph from Figure 4.5b from the example in section 4.3.2. The atomic element “this” can “stand” as a target for this paragraph, but also can “stand” as a target to any other paragraph in a document. This reveals also the power of the RSL concept since as it can be seen by putting links between different entities we may easily construct documents without changing the entities themselves.

One basic concept of the *document-component-link* is that we can easily navigate through our entities. By getting the order target of the paragraph in 4.5b we get as a first entity the atomic entity “this”, as second the entity “is”, etc.

Furthermore the RSL metamodel offers the functionality to take any link from a source or from a target, so we may for instance take a *document-component-link* from a section (source) and then navigate through the rest of the target entities of that section.

Since our abstract entity concept covers resources, selectors and links, we may have targets that are other *document-component-links*. This reveals also the flexibility of modelling links as subtypes of the entity concept. So for instance we may have a *document-component-link* which has a source of a paragraph that has a target of another *document-component-link* which has a source of an audio file, etc.

4.4.2 Reasons to Modify the Logical Structure of a Document

In this section 4.4.1 we presented the *document-component-link* approach that was defined by Tayeh [Tayeh, 2012] in his master thesis. His contribution was very important and we realise that the logical structure of a document is a fundamental concept. Indeed having the logical structure of a document makes everything more

flexible in terms of modelling and implementation. Our intention is to extend that link in order to get also a variety of different structures of documents. We use the word *extend* since we realise that we cannot escape from the advantages that the *document-component-link* offers. Any attempt to create from scratch different structures of documents will be risky and there is always a danger to lose the semantics of a document.

One of the reasons that we need to escape from the logical structure of a document is the variety of different applications and the variety of different devices that these applications run. For instance in a PDF document it may be the case that a chapter of the document is structurally referenced by other documents in one device and in another device has to be structurally presented inside that document. Another example could be to define a structure that reuses other resources via transclusion (e.g. RSS feed news reuse articles from different web sites) in one device and in another device it may be the case that we use navigational links to retrieve that information. These structural differences in the document are difficult to be addressed by using only the logical structure of the document.

Furthermore the diversity of devices forces us to adapt our documents to specific requirements. Sometimes keeping the logical structure in a document makes the adaptation procedure more complex since the developer has to find algorithms in order to find how all the resources (e.g chapters, paragraphs) of a document will be adapted to different capabilities of device (e.g in a small screen). We argue that keeping the logical structure of a document in the adaptation procedure converts the adaptation problem to a calculation problem. One common example is that some web sites in order to fit their web site on small mobile screens reduce the font size of the text. This is totally frustrating for the user and sometimes the user cannot even read the text of the web site. However over the decades more advanced techniques have been developed as discussed in chapter 3 to tackle such issues. We think that all these issues become easier if we escape from the logical structure of the original document and define new structures for it in order to gain flexibility. Then since we have the structure of our document the only thing that we need to adapt is each resource in our document for the different devices in order to get a nice representation for each device.

4.5 An Example of Document Component Link

Before we explain the importance of our structural links and how we can make new structures we would like in this chapter to illustrate in detail how *document-*

component-link works.

In the same manner that we previously explained the element granularity (Section 4.3.2) we will give an example of how we can create a *document-component-link* in order to connect different elements in a document. This time we take a much more complex example rather than the simple paragraph that is used in 4.3.2.

Academic Profiles

The university is consistently ranked as one of the top universities in Belgium. It is also included in major world university rankings such as Times Higher Education World University Rankings, QS World University Rankings and Academic Ranking of World Universities.

Rankings


ARWU (2013, national)	20-7
ARWU (2013, world)	301-400
QS (2013/14, national)	5
QS (2013/14, world)	172
THE (2013/14, national)	6
THE (2013/14, world)	251-275

Student life

Every student that enrolls at the VUB, automatically becomes a member of the Brussels Studentengenootschap (BSG – *Brussels Student Society*), unless they refuse to be. This means that every student has the right to vote and participate in the annual elections for the BSG committee. The BSG is the umbrella organisation for all other student organizations and acts as the defender of the moral interests of the students. Together with their French-speaking counterparts ACE at the ULB, they organise the annual St V memorial.

These are some of the student organizations at the VUB:

- Studiekring vrij onderzoek: a collective of students from various faculties, promoting free inquiry through the organisation of debates, lectures and more
- Letteren-en Wijsbegeertekring (LWK) and Perskring (Pers): for students studying at the Arts and Philosophy faculty
- Geneeskundige Kring (GK) and Farmaceutische Kring (FK): for students studying at the Medicine and Pharmacy faculty



A traditional *klak* or *penne*

Figure 4.6: PDF version of the Wikipedia page of the VUB

Figure 4.6 illustrates a small part of the english VUB web page on Wikipedia ². The page is formatted in a PDF file. The reason that we illustrate the PDF version of the page is that since we have not talked yet about selectors in detail we want to avoid any distraction from the goal of the example which is to show how we

²http://en.wikipedia.org/wiki/Vrije_Universiteit_Brussel

can connect entities in RSL model. In addition we keep our granularity level of the elements in sections, paragraphs, tables, bulleted lists and images. With RSL we can go even to lower level of granularity elements like the cells of the table but this is not the intention of the example since we have not talked yet of how we model such complex elements.

4.5.1 How We Store the Elements

As it can be seen our document contains two sections “Academic Profiles” and “Student life”. The academic profiles section contains one paragraph and one table. The student life section contains two paragraphs, one image and one bulleted list. Figure 4.7 illustrates how our model stores the different elements in terms of granularity. We illustrate two different levels of granularity by colouring higher elements with red and lower elements with blue color. Each element is instantiated as an atomic element. We use then *document-component-link* functionality in order to compose our elements.

4.5.2 How We Connect Elements with Document Links

Since we currently have our elements stored in our model let us assume that the XML file of Figure 4.8 describes the elements of our document. Note that this XML file uses an hypothetical description for the purpose of the example. It is obvious that our highest element in terms of granularity level is the *pdf* element stored in our model. In order to connect this resource (*pdf* element) we need to instantiate a *document-component-link* and define that resource as a source for that link. Then we go to the inner elements of our XML file to reach lower level granularity elements. There are two *section* elements within our root element. After analysing the metadata of those elements we store them in our model. These elements are unique instances in our model. We can achieve that by giving them different names. The only thing that we need to do to take the logical structure of our *section* objects inside our *pdf* element is to add them as targets in our *document-component-link* in the same order as we parse them. Currently our *document-component-link* contains a source (which is the *pdf* element) and two targets (two *section* elements). We may take from that link the source to create our root element and then the ordered targets of the link (first and second *section* elements) to construct for instance the same XML file.

Academic Profiles

The university is consistently ranked as one of the top universities in Belgium. It is also included in major world university rankings such as Times Higher Education World University Rankings, QS World University Rankings and Academic Ranking of World Universities.

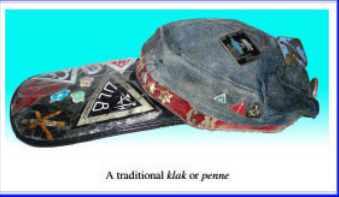
Rankings	
ARWU (2013, national)	20-7
ARWU (2013, world)	301-400
QS (2013/14, national)	5
QS (2013/14, world)	172
THE (2013/14, national)	6
THE (2013/14, world)	251-275

Student life

Every student that enrolls at the VUB, automatically becomes a member of the Brussels Studentengenootschap (BSG – *Brussels Student Society*), unless they refuse to be. This means that every student has the right to vote and participate in the annual elections for the BSG committee. The BSG is the umbrella organisation for all other student organizations and acts as the defender of the moral interests of the students. Together with their French-speaking counterparts ACE at the ULB, they organise the annual St V memorial.

These are some of the student organizations at the VUB:

- Studiekring vrij onderzoek: a collective of students from various faculties, promoting free inquiry through the organisation of debates, lectures and more
- Letteren-en Wijsbegeertekring (LWK) and Perskring (Pers): for students studying at the Arts and Philosophy faculty
- Geneeskundige Kring (GK) and Farmaceutische Kring (FK): for students studying at the Medicine and Pharmacy faculty



A traditional *blak* or *penne*

Figure 4.7: Elements granularity stored in RSL

If we need to reach lower levels of granularity, we need to create a *document-component-link* for each *section* element. In each link we define as source the corresponding *section* element. In the first *section* element we analyse and store a *p* and a *table* element and we add them as targets to our *section-component-link*. The same story stands also for the second *section* element with every element.

At this point we do not go further in terms of granularity levels, but the reader should already have an overview of how the *document-component-link* works and how flexible it is to connect our different resources. An illustration of what our links contain in terms of source and targets is shown in Figure 4.9.


```

<pdf name="vub">
  <section name="Academic Profiles">
    <p> The university is consistently ranked as....</p>
    <table> .....</table>
  </section>
  <section name="Student life">
    <p> Every student that enrolls at the VUB.....
    .....
    .....
    they organise the annual St V memorial</p>

    <p>These are some of the student organizations at the VUB:</p>
    <ul>
      <li> Studiekring vrij onderzoek.....</li>
      <li> Letteren-en Wijsbegeertekring.....</li>
      <li> Geneeskundige Kring.....</li>
      <li> Polytechnische Kring .....</li>
      <li> Psycho-Pedagogische Kring.....</li>
    </ul>
    </img>
  </section>
</pdf>

```

Figure 4.8: XML description of vub PDF file

4.6 The General HTML Table Model

In this section we introduce the general HTML table model. We do that because the table is one of our resources in our implementation and it is the most complicated. In Chapter 5 we discuss how we implemented the table element using the RSL metamodel. One of the reasons that the tables are a complicated concept is that the table structure depends on the content. Table headers may span across multiple rows or columns, table rows may have nested table header or table data elements, table data elements may have bulleted lists etc. We present in this section the general model of an HTML table. We first describe the elements that an HTML table may have. Secondly we give the order that these elements must follow to be a valid HTML table.

4.6.1 Elements of an HTML Table

The root element of a table is the *table* element. A table may contain a caption, a row group or a column group. It is obvious that the row group is composed by rows while the column group consists of columns. Rows and columns define the

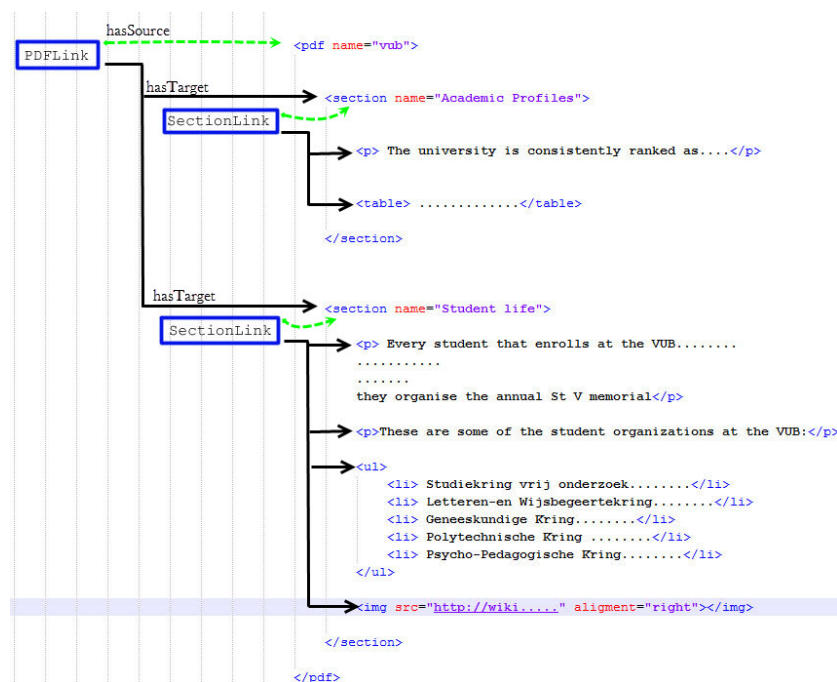


Figure 4.9: Source and target entities in our *document-component-links*

cells of a table. The table element generates an anonymous box that includes the caption box and the table box.

The *caption* element is bound outside of the table box but it is inside the anonymous box that is mentioned above. When the table is moved to another position, the anonymous box is then moved and enables the caption to follow the table. The position of the *caption* is specified by an attribute *caption-side*. This gives the developer the flexibility to put the caption at the top or at the bottom of the table. A table may not have a *caption* element. Furthermore there can be only one *caption* element inside a table.

After the *table* element we have three different section elements such as head (*thead* element), foot (*tfoot* element) or body (*tbody* element). Foot and head elements are used mostly for long tables when we need to scroll along the body sections and maintain the contents of the foot and head elements on the screen. Another reason is also when we need to print a long table and we need the foot and head elements to be printed on all pages. Naturally someone would assume that these three elements will follow the structure in the same order that the developer defines them. For instance if we have a *thead* element followed by a *tbody* element which is followed by a *tfoot* element, then the table would follow that structure.

Caption

Questionnaire Results

Question	Women		Men	
	Yes	No	Yes	No
Question 1	42%	58%	61%	39%
Question 2	53%	47%	69%	31%
Question 3	26%	74%	51%	49%
Average	40%	60%	60%	40%

Diagram labels: Row Group (bracketed on the left of the data rows), Row (bracketed on the right of a data row), Column (bracketed under the 'Yes' column), Column Group (bracketed under the 'Women' and 'Men' columns), Cell (pointing to a data cell).

Figure 4.10: A simple table example

However this is not the case for HTML, the correct order for the table sections is *thead* followed by *tfoot* followed by one or more *tbody* elements.

To identify individual table columns and apply styles with them the *col* element is used. This element is used mostly for style purposes. Unlike rows, columns do not contain any cells directly, they implicitly group adjacent cells between rows. Since there is no easy way for a table to define styles in different cells, the *col* element is used to simplify that process. The main idea is to define all our column elements with their corresponding style before we begin the process of writing our rows.

The *colgroup* element is a container for *col* elements. With this element we can apply attributes at a high level that can be applied to the lower level *col* child elements. Otherwise we should apply style attributes in each *col* element. Note that if we use one or more *colgroup* then all *cols* must be enclosed in one or more *colgroups*.

Rows are used in order to create a new row of data. The HTML table model is row-centric. Table rows are grouped inside the head, foot or body elements. Of course we may specify columns in markup via *col* or *colgroup* elements but cells are structurally nested within rows.

The most inner elements of a table are its cells. A cell can be a table header (*th* element) which is used for header information. Despite the fact that we use the term “head”, it does not mean that we use it for the table heading that appears at the top of the table. It may be the case that we apply the *th* element in any other

cell on any table. However in most cases we use the table header as the topmost cell in a table. Finally this element can span across multiple rows and columns.

A cell can also be a table data (*td*) element which can be used for any kind of data. Note that since table data is used for any kind of information it may contain other nested HTML elements such as images, bulleted lists, paragraphs etc. A *td* element can also span across multiple rows and columns as a *th* element does.

Each *td* or *th* element must be contained on a parent of a *tr* element.

4.6.2 Ordering of the HTML Table Elements

We would like to conclude with the structure of an HTML table model. Along with the order of the elements we give some cardinality constraints that are applied on a table to show if an element can be present more than one time or even if an element can be omitted by the structure.

We present the ordering of the elements structure beginning by the root element and ending the leaves which are the cells of the table as follows:

- *table* element is the root element for every table.
- *caption* element if it exists, must go first. There can be only one per table.
- *col* or *colgroup* element if it exists. There can be any number of *col* or *colgroup* elements, but not both. For a *colgroup* element we may define any number of inner *col* elements.
- *thead* element if it exists. It must contain at least one *tr* element. There can be only one per table.
- *tfoot* element if it exists. It must contain at least one *tr* element. There can be only one per table.
- *tr* or *tbody* element. There must be at least one of these elements in the table. If *tbody* is used there must be at least one *tr* element.
- *td* or *th* element must have a *tr* parent element. These are the cells of our table. There can be any combination with such elements.

4.7 The Importance of Crosslets

In 4.4.2 we discussed the importance of changing the logical structure of a document and we concluded that we have a variety of reasons to prefer to do it during the adaptation procedure. However that is only one step in the adaptation procedure. We have noticed that despite the fact that by changing the structure of a document we gain flexibility in the adaptation process, this is not enough by itself to adapt content to different contexts. We need more powerful concepts to represent each resource in different ways for different contexts. In this section we discuss dynamic crosslet approach that is implemented by [François, 2014]. In Chapter 5 we discuss how we implemented our static crosslets. We divide crosslets functionality into three categories: *interactive*, *dynamic* and *static*. Since we do not have yet a concrete implementation in terms of interaction we do not discuss the interactive crosslets.

4.7.1 Dynamic Crosslets

In his master thesis [François, 2014] implemented crosslets which define extra functionalities to specific resources. The crosslets in his approach are responsible to make the resources more dynamic. In his implementation crosslets of a resource may change dynamically. Thus, a resource may support an action in a specific scenario and another kind of action in a different scenario. He takes into consideration the user preferences to assign an action to a resource and the environment that the user is surrounded by. In general his goal was to take static content and transform it into dynamic content. A definition of his vision of dynamic content is as follows:

Content that belongs to a document and has the capacity to change both itself and the structure of the document as a result of external factors that are linked to the content such as: a contextual element, an external resource, etc.

In his work each resource can be managed by several crosslets, meaning that crosslets add functionalities to his components in order to achieve a fully dynamic content. Furthermore he has introduced the concept of *Tubes* in order to specify how his components work together. His *Components* are interlinked with each other via his *Tubes*.

4.7.2 Functionality

A crosslet is a subtype of the resource concept in the RSL model. As a consequence it may support any kind of different digital media types as much as physical resources. The main reason that we use a crosslet is to add some functionality to our resources, since our resources may support different interactions or may have different representations in a variety of scenarios and we do not know prior how for instance an image will be presented or which kind of functionality it will have to support. Instead of treating all these scenarios for each of our resources separately we may create crosslets which offer these extra functionalities if needed to our resources.

4.8 Conclusion

In this chapter we have discussed the RSL metamodel. We have shown most of the concepts of the RSL metamodel. More specifically we illustrated how entities are treated in RSL through the granularity effect and how we can build general models for complex elements such as tables. We gave also the general approach of the modelling of an HTML table with RSL. In addition we discussed previous approaches from other students about the *document-component-link* model and the dynamic crosslets concept. In the next chapter we present our model in order to build structure and how we can add functionality to our resources.

Chapter 5

Conceptual Model

In this chapter we introduce our model. We further explain the different components of our conceptual model. Furthermore we discuss how our model can be extended for domain specific uses. Finally we conclude our chapter with the analysis of the model.

5.1 Elements in our Conceptual Model

We have already discussed some limitations of previous approaches in Chapter 3. We noticed that there is a lack of conceptual models in terms of granularity. Thus, the lack of a general model of treating elements individually encourages different interpretations which may be confusing. We present our conceptual modelling by describing the different elements and the use of them.

5.2 Components

We propose a component-based approach in order to get different representations of resources. In fact our components inherit all the functionalities of a resource from the RSL metamodel without any modification. We believe that *Components* are best seen as specialisation of resources in our document structure. Note that the implementation of a *Component* entity is there for clarity issues in our model.

It is the *Crosslet* approach that reveals our contribution in our model. Then, we implemented several components which follow the HTML element concepts. Each *Component* can be: *Article*, *Section*, *Subsection*, *Paragraph*, *Image*, *Table*, *Block Quote*, *Bulleted List*, *List Element*, *Audio* or *Div* component. An explanation of how we modelled each of our components follows in the next sections.

5.2.1 HTML Table Component

Modelling an HTML table in RSL way is challenging considering the possibility for adaptation. The complexity of a table raises many issues that we have to take into consideration. As already explained in section 4.6 a specific ordering of the different elements must be followed. Different combinations can be applied via nested elements. We may use also different elements to replace functionality of other elements, for instance we may use a *td* element with bold font at the top of our table to represent the headers of a table instead of a *th* elements. This flexibility that a table offers us sometimes becomes a weakness in terms of modelling. However we propose a solution that covers the majority of HTML tables in a simple way with our RSL solution. In our implementation we have a use case where we parse an HTML Wikipedia page. The reason that we chose such a page is that it offers us a variety of elements such as tables, images or audio files. During the parsing procedure we try to analyse all the elements of the page and store them in the iServer in an RSL way, meaning that each element that we analyse is stored as an individual entity in the RSL model. For the parsing procedure we have implemented different type of resources (e.g. paragraphs, images) by extending the general “resource” concept of RSL. One of the most challenging things during the implementation was to model HTML tables. In this section we present a general model that can be used for describing complex HTML table. There is also an illustrated example of our approach.

Table Model in RSL

Tables in RSL are treated like any other resource, but in a more sophisticated way. Our approach relies on the fact that we can analyse each element from higher to lower granularity levels and then since we already know how the different elements can be ordered in a table (discussed in section 4.6), we can easily put the *document-component-link* to construct our table.

We consider our root element to be the *table* element as defined by the general

model (section 4.6). In our implementation we do not consider head and foot sections since in our web page we have tables consisting only of body elements. However this is something that can easily change since the structure of header, footer and body section is identical. Moreover we consider that a table row is followed either by a *td* or *th* elements. In addition we analyse if a table data element consists of other elements like bulleted lists, paragraphs and images.

An Example of an HTML Table in RSL

We present an illustrated example in order to clarify our approach. Figure 5.1 shows one of the tables that we have parsed during our implementation. Unfortunately after our work is done the wikipedia page changed drastically so currently this table is represented in the page in a simpler way. In our HTML corresponding code we kept only the first three rows of the table. The following identical rows are omitted for clarity issues.

Year	Rank (Change)
2008	214 (▲ 15)
2009	227 (▼ 13)
2010	238 (▼ 11)
2011	204 (▲ 34)
2012	189 (▲ 15)
2013	172 (▲ 17)

(a) VUB rankings

```

<table class="wikitable">
  <tr>
    <th>Year </th>
    <th>Rank (Change)</th>
  </tr>
  <tr>
    <td>2008
    </td>
    <td>214 (
      <span title="Increase">
        <a class="image" href="File:Increase2.svg" >
          
        </a>
      </span> 15)
    </td>
  </tr>
  <tr>
    <td>2009
    </td>
    <td>227 (
      <span title="Decrease">
        <a class="image" href="File:Decrease2.svg" >
          
        </a>
      </span> 13)
    </td>
  </tr>
</table>

```

(b) HTML corresponding code

Figure 5.1: The HTML table from wikipedia page in 5.1a and the corresponding HTML code in 5.1b

First we identify the root element in our table which is the *table* element. Then we create an instance of a table object and store it in our model. Since we know that there will be other lower level elements inside our table we create an instance of a *document-component-link* named *tableComponentLink* and we add our *table*

⁰http://en.wikipedia.org/wiki/Vrije_Universiteit_Brussel

element as source of it. So now that we have our *tableComponentLink* we can analyse lower resources in our hierarchy and add more targets to that link. Since the general model of an HTML table defines that after a *table* element follows a *tbody* element we analyse this as well. Note that we do not store in our model the *tbody* element since this is somehow redundant information. We only need to analyse the contents of the body elements in order to avoid a lot of redundant objects in our model. As defined from the table model the *tbody* element contains one and possibly more *tr* elements. Rows in our model is the next level of granularity. Thus, again we create instances for every row resource and we instantiate *rowComponentLinks*. Once again we add the corresponding rows as source in our links.

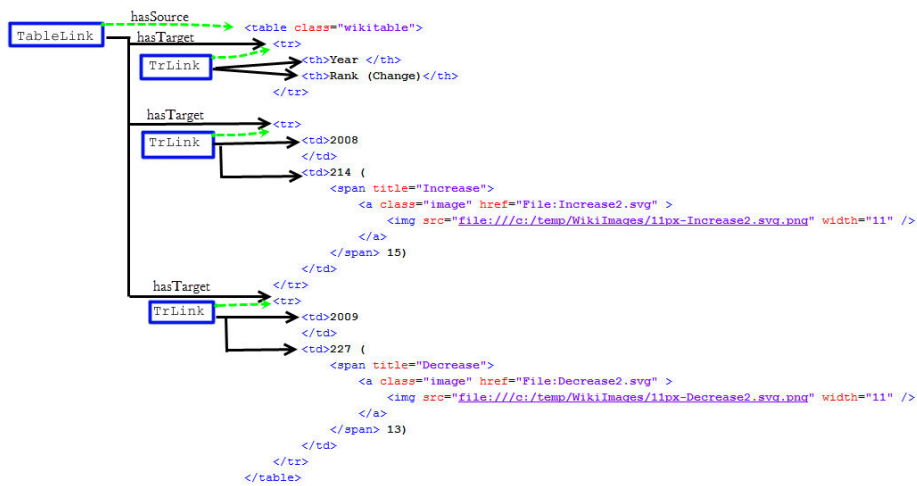


Figure 5.2: The resulting table of the example

For every row there are three possibilities for our next low level resources. One is to have as our next resource one or more *th* elements. Second to have one or more *td* elements. Third to have a combination of the two resources. In our example 5.1 we have to analyse only the first two cases. After the analysis of these elements we create instances of table data or table header resources. In this particular example we do not consider lower level elements of granularity because we do not create additional links for table data and table header. However in our use case he have a table that contains a table data element with a *div* element and a *bulleted list* element. Since *div* elements contain complex resources like paragraphs, images, audio files etc. It is easy to understand that our model covers most cases of an HTML table model. The *th* and *td* elements may contain any kind of information depending on the application that are used. In our case these elements contain bulleted lists, paragraphs, images and URL links.

5.2.2 Paragraph Component

The *Paragraph* component that we implemented is a component that contains fields such as *content* and *encoding*. The *content* of a paragraph is a string which contains all the text of the paragraph. The *encoding* field is also a string which has the value of “UTF-8”. Note that we store in the model the *Paragraph* component as an atomic entity. We could further analyse the paragraph in different levels of granularity. For example we could store the words that compose a paragraph, but that was not necessary for the purposes of this project.

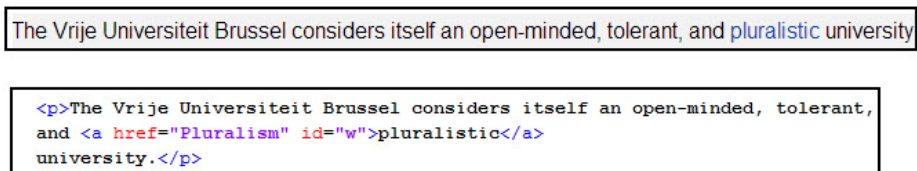


Figure 5.3: A paragraph example from Wikipedia VUB website with its corresponding HTML description

A paragraph may contain URL links inside its content. In Figure 5.3 we can see a paragraph element from a Wikipedia web page. As mentioned above the *Paragraph*'s content is a string value which contains only the text of the paragraph. If we store in the server the *href* anchor tag as part of the string, we lose the semantics of the paragraph. For instance if we want to describe a paragraph in a PDF file we do not want it to appear with the *href* anchor tag.

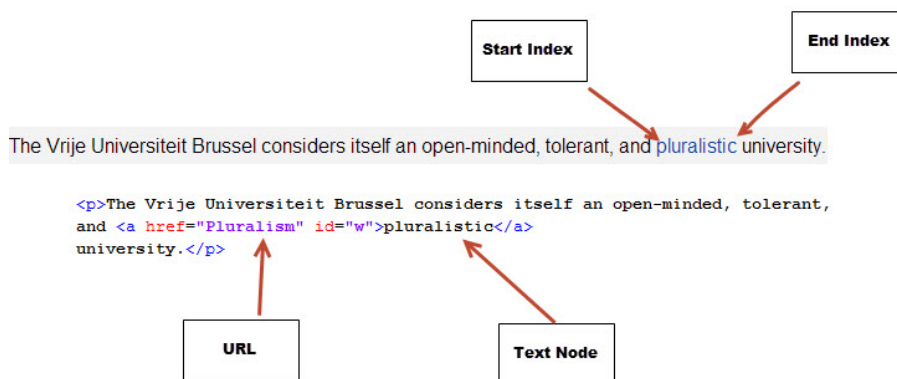


Figure 5.4: A paragraph example from Wikipedia VUB website with a URL link

In the RSL metamodel we can address specific parts of resources with selectors as discussed in 4.2.3. In order to keep the semantics of the paragraph, whenever we find an *href* anchor element inside a paragraph, we just append the text node of the *href* anchor element to the rest of its content and then we store its URL. Then we associate this URL with the part of the paragraph that is referenced to. For that purpose we have implemented a selector named *UrlSelector*. Our *UrlSelector* has fields such as *startIndex* (which is an integer), *endIndex* (which is an integer) and *url* (which is a URL address). The start index points to the position of the first character of the word that we want to address inside the paragraph and the end index the last character position of the word.

Figure 5.4 shows how we analyse a paragraph which contains a URL link inside. As it can be seen from the Figure 5.4 the *href* attribute has a value of a simple word and not a valid HTTP address. This is the case for the Wikipedia URL's. In order to overcome this issue and to store the actual URL in our server we do some processing on the *href* attributes. Each *href* anchor tag in Wikipedia that contains an *id* attribute equal to "w" means that this URL is a link in another Wikipedia page. The corresponding Wikipedia link of the example after the processing has the value of "http://en.wikipedia.org/wiki/Pluralism". Note that this is a Wikipedia link to the English version of the Wikipedia. We process cases for English, French and Dutch Wikipedia links. For instance in our implementation it may be the case that an *href* attribute has the value of "nl:link", the "nl:" prefix means that this is a Wikipedia link of the Dutch version of Wikipedia web page and the corresponding URL would be "http://nl.wikipedia.org/wiki/link".

When we finish with the processing of the *UrlSelector* we associate this selector to our *Paragraph* component. As defined from the RSL metamodel (see 4.2.3) each selector refers to only one resource while the resource can have zero or more referencing selectors. The RSL metamodel offers functionality to add as many selectors as we want in our resources. In Figure 5.5 we can see our *Paragraph* component and our *UrlSelector* along with their contents. We can also see the association between the paragraph and the URL selector.

Furthermore a paragraph element can also contain *i*, *span*, *strong* and *b* element tags. We do not store these tags in the server and we just append their text nodes to the rest of the paragraph.

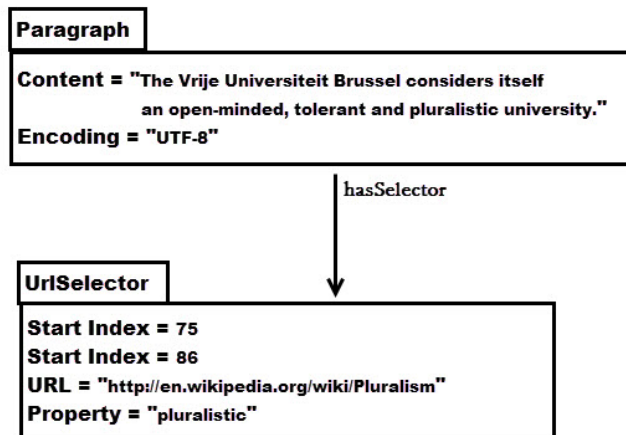


Figure 5.5: The association between a paragraph component and a URL selector

Elements that Inherit Paragraph Component

Our paragraph approach can cover a variety of different element HTML tags in our implementation. There are elements in our HTML page which follow exactly the same principles as our paragraph does, meaning that we can describe them with a *content* field (which is a string) and an *encoding* field.

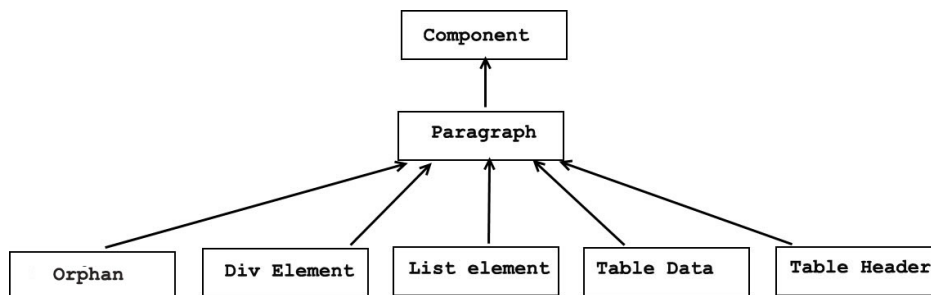


Figure 5.6: Elements that inherit from *Paragraph* component without any modification

As a consequence we created different specialisations of our *Paragraph* component. In Figure 5.6 we can see the specialised components that we implemented. Please note that they inherit all the functionalities of our *Paragraph* component without any modification. These elements are:

1. Orphan Paragraph

2. **Div Element**
3. **List Element** which is used in bulleted list.
4. **Table Data Element** which is used for tables
5. **Table Header Element** which is used for tables

5.2.3 Image Component

We have implemented an *Image* component in order to analyse our HTML images. Each *Image* object contains fields such as a *remoteImageLocation* (which is a URL address), a *remoteThumbnailLocation* (which is a URL address), a *localImageLocation* (which is the actual file stored in the server), a *localThumbnailLocation* (which is the actual file stored in the server), an *imageName* and a boolean value to check if the image has a thumbnail or not. These fields are enough to describe our images and retrieve all the information when needed. In our use case each image is contained in other elements such as div element, table data element, paragraph element and list element. Furthermore all of our images elements in the HTML page are inside *href* anchor tags. As a consequence we analyse them in our *UrlSelector* implementation. As discussed previously from a *UrlSelector* we keep only the text node of it and for example we append it in a paragraph content. In the case of a *UrlSelector* which contains an image element we append an empty string.

5.2.4 Other Components

Another component that we implemented is the *Bulleted List* component. This component contains an arraylist of *List Element* components. We can easily add, delete and retrieve *List Element* components from the arraylist. Since *List Element* components are already discussed it is not necessary to provide further details for this element.

Other components that we implemented are *Article*, *Section*, *Subsection* and *Table*. All these components are just specialisation of our *Component* approach. As a consequence are inherit all the functionality of the *Resource* type of the RSL model without any modification. In Figure 5.7 we can see some of the specialised components that we implemented. For clarity reasons we skip some of the components in our figure.

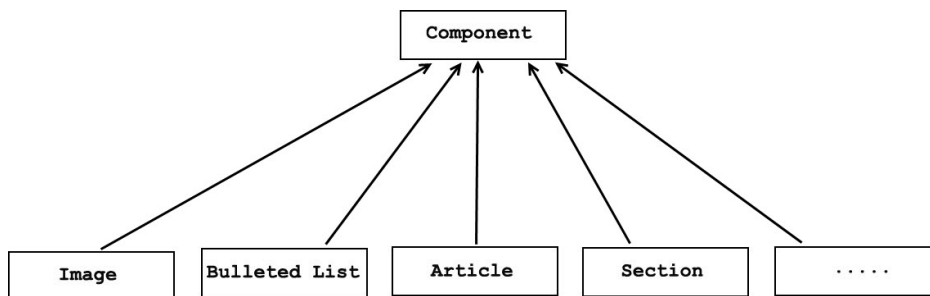


Figure 5.7: Elements that inherit from *Component* without any modification

Figure 5.8 shows our conceptual model of the *Component* that inherits the resource of our RSL metamodel.

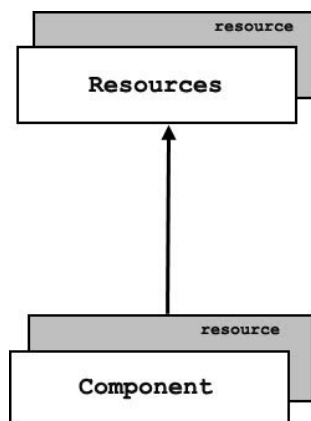


Figure 5.8: *Component* in our model inherits from the RSL *Resources* without any modification

5.3 Crosslets

5.3.1 Static Crosslets

In section 4.7 we discussed dynamic crosslets that are implemented by another student. In this section we present our approach for static crosslets. We focus on the static part of the content. In fact our crosslets will add some functionality to our resources (e.g. *getPresentation* of a paragraph). Someone could argue that this functionality makes our content fully dynamic. However, in our implementation

each resource can accept only one crosslet each time. For instance a paragraph crosslet is associated with only one paragraph resource. We will consider our approach static for that reason and from the fact that crosslets cannot be assigned after the initial adaptation. This is the main difference between our work and Jochen’s [François, 2014] work.

The advantages of our approach relies on the fact that each resource is uniquely managed by a crosslet. Note that even same types of resources can be treated differently by the crosslet. An example will be that two paragraphs may have different fonts depending on their content (e.g number of words) or even different coloured text. If we consider that paragraphs are exactly the same resource in terms of type we can conclude that this is a flexible way to treat our resources. Note that since we have our crosslet which encapsulates a paragraph resource we can call its method to give us back a specific representation for a specific device.

Since we have already observed the need of crosslets, we divided our component concept into two parts. The actual content of a resource is the *Component*, the functionality that we add to the each resource in order to get different representations of a resource is its *Crosslet*. Each instance of a crosslet defines a representation of a component. This cardinality constraint is expressed in Figure 5.9 in the association *managedByCrosslet*. Note that the same instance of *Crosslet* cannot be used for different components, meaning that each *Crosslet* can manage only one instance of *Component*.

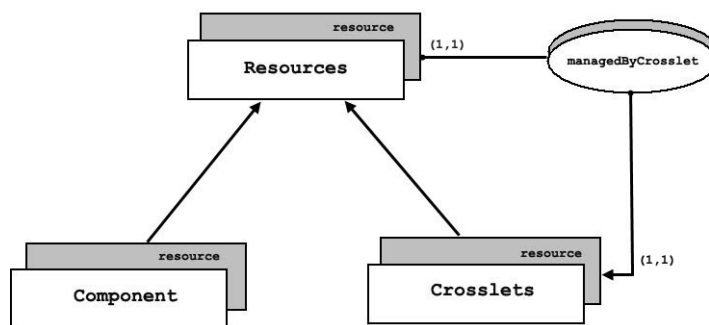


Figure 5.9: Conceptual model: Component and Crosslets

In Figure 5.10 we illustrate a paragraph component with its crosslet. From the paragraph crosslet we can get different representations of our resource component. Device specific capabilities are calculated inside the crosslet and then the crosslet comes up with a concrete representation of the paragraph.

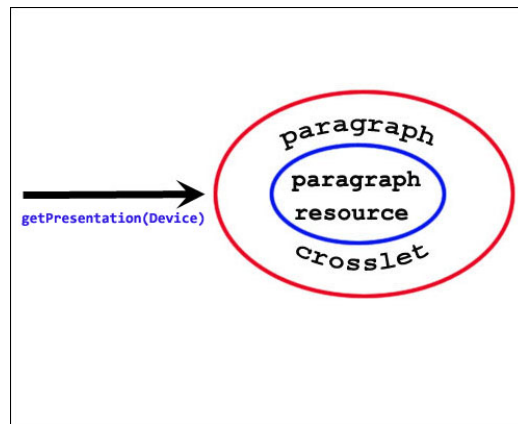


Figure 5.10: An example of a paragraph crosslet

Furthermore we have implemented a variety of crosslets to manage our components. Figure 5.11 shows some of the component resources and their corresponding crosslets. Note that we did not implement crosslets for *Article*, *Div* and *Block-Quote* elements because there was not necessary for these elements to get different representations.

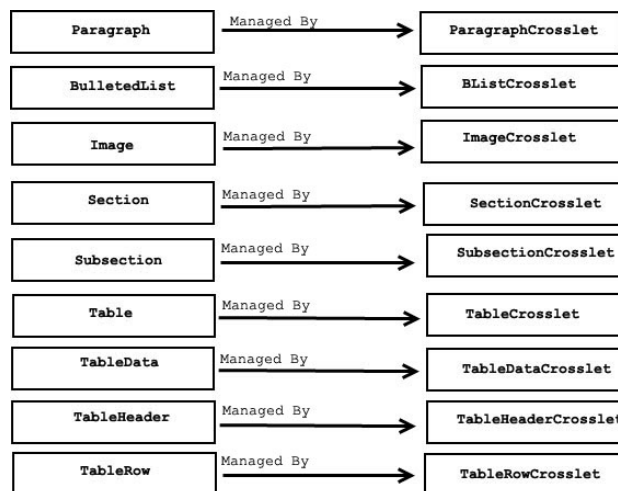


Figure 5.11: Our component resources along with their corresponding crosslets

Our approach of crosslets follows the Decorator Design Pattern [Freeman et al., 2004]. We implemented a variation of the Decorator Pattern. The Decorator Pattern is used in order to add some functionality in objects. Each object can have zero or many decorators. In our implementation our *Components* can have only one decorator. This is the main different between the actual implementation of the

Decorator Pattern and our implementation of *Crosslets*. In Figure 5.12 we can see that the constructor of a paragraph crosslet can accept an instance of a paragraph component. Then inside the crosslet there is a method *getRepresentation(Device d)* which returns the paragraph instance with a specific representation according to the device capabilities.

```
public class WikiParagraphCrosslet extends Crosslet{  
    private Paragraph para;  
    public WikiParagraphCrosslet(Paragraph p){  
        this.para = p;  
    }  
}
```

Figure 5.12: The constructor of a paragraph crosslet accepts an instance of paragraph component

An Example of Crosslet

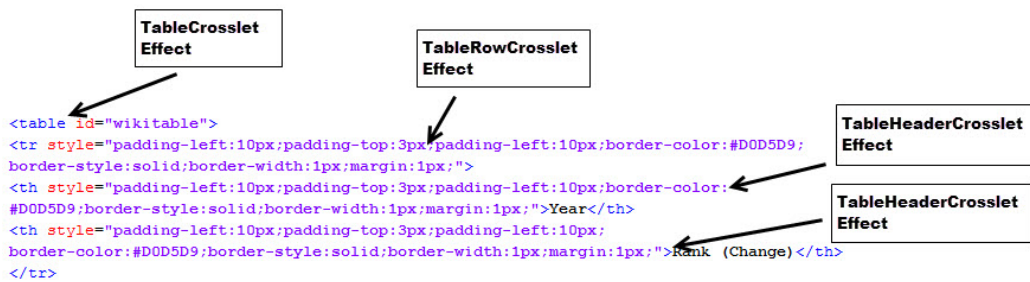
At this point it is worth to explain our powerful concept of crosslets with an illustrative example. We discuss our table implementation since tables are the most complicated components in our approach. Figure 5.13 shows the corresponding XML elements that we get when we apply our crosslets given that our device is the Nexus 7 Tablet. As it can be seen crosslets add to our elements different styles in order to achieve an nice presentation in the Android device. Note that our table crosslet does not add any style to the root table element. In our XML description we show only the effect for the first row of our HTML table for clarity reasons.

5.4 Structures

As already explained in 4.4.2 there are many reasons to escape from the logical structure of a document. Our intention is not to escape from the logical structure concept but rather to extend it. In this section we explain how to create different structures of a document.

Year	Rank (Change)
2008	214 (▲ 15)
2009	227 (▼ 13)
2010	238 (▼ 11)
2011	204 (▲ 34)
2012	189 (▲ 15)
2013	172 (▲ 17)

(a) An HTML table of our use case



(b) The corresponding XML description after the crosslet effect in each element

Figure 5.13: An example an HTML table (a) after we apply crosslets in our elements (b)

5.4.1 How to Create Structures

We propose a solution such that different structures can coexist in a more general structure that has the logical structure as core. The reason that we choose for structures to coexist is that we cannot directly define unique structures of a document. For instance we may need to change our chapter structure of a document and take chapters in reverse order but at the same time to use a structure of transclusion in that document because our screen size is small and we need to take parts of content from different sources. Furthermore for the same example we may need an additional structure which says for instance that we do not need also the links that are included in our document since this time our device is an e-book ¹ reader.

If we try to define all these different structures uniquely we will end up with an explosion of different structures and probably we will not cover most of the cases. Of course if we use a domain specific approach in our model we may define a structure which can manage a specific case of structure. However our intention is to cover as many cases as possible for a diversity of devices in a simple and manageable way.

¹http://en.wikipedia.org/wiki/E-book_reader

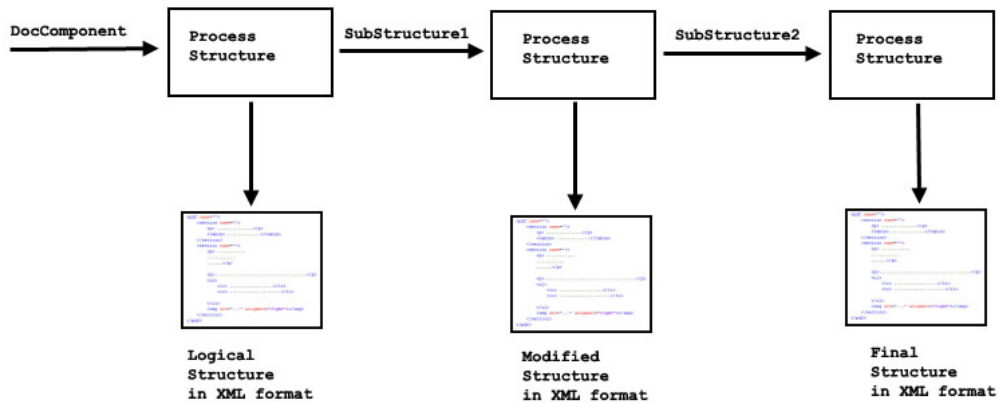


Figure 5.14: Different steps to modify the logical structure of a document

Thus, in the example of the e-book reader we may apply a structure of reversing the chapters and a structure which will exclude the existing links of our document. These two different structures will be handled by a more general structure which will process the structures. First the general structure will be taken from the logical structure of the document. Then we apply the reverse structure to the logical structure and we take the document chapters in reverse order. Afterwards, we apply the next substructure which will exclude all the links of our document. Having this kind of structure then, it is easier to further adapt each resource to specific device capabilities (e.g. screen size).

The order of the structures that will be added to our general structure plays an important role. Everytime that we applied structures we applied in the previously modified structure, meaning that the first structure is always the logical structure of the document. Then the second structure is applied to the logical structure and we get a new structure, then the third structure is applied to the second structure and we get a new structure and so on. An illustration of such an approach can be seen in Figure 5.14. For instance if you apply on PDF document a reverse structure of our chapters and then again a reverse structure of the chapters we will end up with a document that has the chapters in the same order as it initially had. However applying many substructures to our general structure helps us to solve a variety of complex cases.

5.4.2 Elements in Structure

We have already discussed the importance of having different structures in our document rather than having only the logical structure of it (section 4.4.2). In this section we present a model that can define different structures in a document and modify the logical structure of it. We define *Substructure* as an entity that we can build different structures. Figure 5.15 shows the *Substructure* entity along with some of the structures that we have implemented.

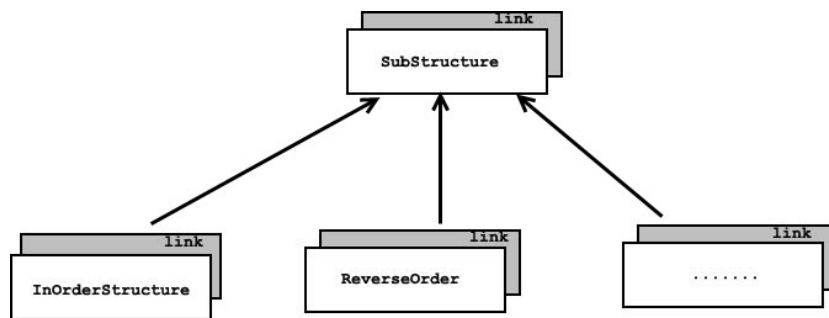


Figure 5.15: Some of the *Substructure* specialisations

In our implementation we have defined structures as follows:

- **InOrderStructure** which returns specific resources in document order. This link has almost the same functionality as the *document-component-link*. However the main difference is that we can choose only specific resources that we can get in document order. For instance we can define to take all the images in document order.
- **NoSelectors** to exclude the existing selectors (links) in our document (if any).
- **ReverseOrder** to return specific resources in reverse order. In this link we can define specific resources that we can take in reverse order.
- **WithSelectors** which includes selectors (links) in our document (if any).
- **ReferenceResource** which references resources in other documents. With this link we can get resources that are referenced by other documents. Note that we get back selector objects and not the actual resources, meaning that if we specify to get for instance paragraph resources, we get back selectors which are references to the paragraphs.

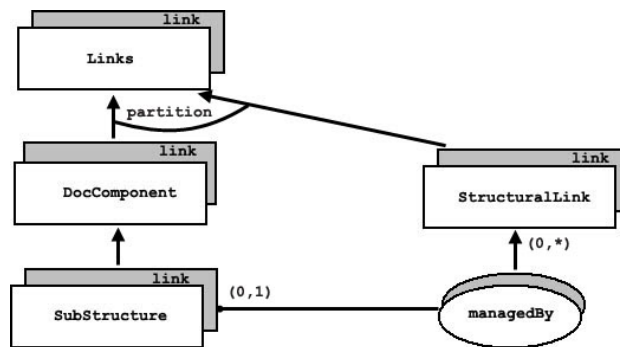


Figure 5.16: Conceptual model: Structure

Our conceptual model is illustrated in Figure 5.16. As it can be seen, we have put a partition constraint to distinguish instances of *document-component-link* and *Structural-link*. A *StructuralLink* can manage zero or many substructures, while a *Substructure* may be managed by zero or one *StructuralLink*.

5.4.3 An Example of Our Approach

Let us explain our approach with an illustrative example. Figure 5.17 shows the corresponding XML logical structure from an HTML document which contains three paragraphs. Note that the paragraph structure contains only the text of the paragraph and not the *href* anchor tags. This is the logical structure that we modify when we apply structures. Our contribution reveals how we modify this logical structure.

```

The university's name is sometimes abbreviated by "VUB" or translated to "Free University of Brussels".
The university is organised into 8 faculties that accomplish the three central missions of the university: education, research, and service to the community.
It is also a strongly research-oriented institute, which has led to its top-189th position among universities worldwide.

<p>The university's name
  is sometimes abbreviated by "VUB" or translated to "Free University of Brussels".</p>
<p>The university is organised into 8 faculties that accomplish the three central missions of the university:
  education, research, and service to the community.</p>
<p> It is also a strongly research-oriented institute,
  which has led to its top-189th position among universities worldwide.</p>
  
```

Figure 5.17: The XML logical structure of an HTML document

We change structure in our paragraphs by adding their *href* anchor tags. We do that by applying a *WithSelectors* structure as Figure 5.18 shows. The structure of our paragraphs has changed, and at this point we get our paragraph elements modified from the initial logical structure. The *WithSelectors* structure will process all

the *UrlSelector* objects that we have implemented. If a paragraph object has no selectors then the paragraph remains unmodified.

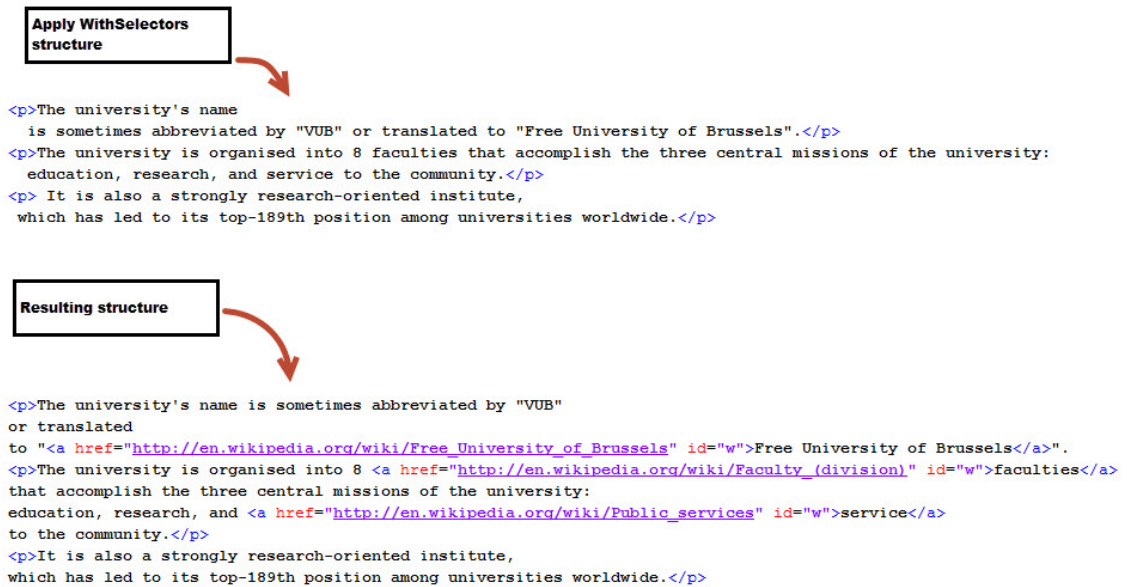


Figure 5.18: The resulting structure after apply a *WithSelectors* structure to the logical structure

Then in a second step we can apply another structure to our modified XML document. Let us assume that our paragraphs contain a lot of text and the device has very small size. In this case we want only one paragraph at a time to appear on the mobile device. In order to achieve that we can apply a *ReferenceResource* structure that we implemented which modifies the structure of specific resources. These resources are modified in such a way that we get back selectors instead of the previously defined resources. The paragraphs of our example become selectors and they point to the server's REST interface that we implemented. The REST interface can generate then our paragraph object when requested and in the mobile device only one paragraph at a time is displayed. Figure 5.19 shows the resulting structure after we apply our *ReferenceResource* structure. Note that in the example with the paragraphs we may not need to reference the paragraphs from different sources but someone can imagine that chapter or section resources are best candidates for this structure since these resources are normally very long to fit all of them in a small screen.

Furthermore we can apply any number of structures in order to modify our resulting structure. For instance we can also apply a *ReverseOrder* structure for our



Figure 5.19: The resulting structure after apply a *ReferenceResource* structure to the previously modified structure

paragraphs in order to get our paragraphs in reverse order. Figure 5.20 shows the resulting document structure after we apply the *ReverseOrder* structure.

5.5 Conclusion

In this chapter we have proposed our conceptual model. After analysing previous research topics we came up with the conclusion that there is a lack in terms of modelling of how we can create and modify content. Furthermore we believe that structure plays a crucial role in adaptation process.

Our contribution is based on two discrete factors. One of them is how to represent different information. We achieve that when we tie our components with crosslets that can affect the representation of them. Secondly, we believe that structure should be on the document level.

In order to prove the efficiency of such a model, we further discuss a specific use case that we implemented and we explain the implications that our approach has in the next chapter.

Apply ReverseOrder(Paragraph)



```
<a href="http://192.168.0.101:9998/iserwer/wiki/VUB/ParagraphID">Paragraph_1 Description</a>  
<a href="http://192.168.0.101:9998/iserwer/wiki/VUB/ParagraphID">Paragraph_2 Description</a>  
<a href="http://192.168.0.101:9998/iserwer/wiki/VUB/ParagraphID">Paragraph_3 Description</a>
```

Resulting structure



```
<a href="http://192.168.0.101:9998/iserwer/wiki/VUB/ParagraphID">Paragraph_3 Description</a>  
<a href="http://192.168.0.101:9998/iserwer/wiki/VUB/ParagraphID">Paragraph_2 Description</a>  
<a href="http://192.168.0.101:9998/iserwer/wiki/VUB/ParagraphID">Paragraph_1 Description</a>
```

Figure 5.20: The resulting structure after apply a *ReverseOrder* structure to the previously modified structure

Chapter 6

Implementation

In this chapter we discuss a use case that we implemented to clarify our contribution. Our main goal is to show the efficiency of our approach and how someone can easily extend our approach in other domains. We first begin by explaining our use case and our motivation in detail. Then, technologies that are used for this implementation are presented. Afterwards, we illustrate our approach through examples on different devices. Furthermore, we give motivation for possible extensions via some extra substructures that we implemented. Finally we give our conclusions.

6.1 Use Case

6.1.1 Motivation

Web applications are widely used not only by desktop users but also by mobile users. In our use case we chose to adapt a web page in Android mobile devices. Android devices are chosen because of their well-known programming language (Java) and their wide availability. Since our metamodel supports java applications it was easy to come up with such an idea. Furthermore we tried to find rich web pages in terms of content and complexity and we decided that the VUB (Vrije Universiteit Brussel) in Wikipedia is a good candidate. Wikipedia pages offer a variety of resources such as images, sections, audio, tables and most important many hyperlinks.

One of the limitations in Wikipedia pages is that they cannot be parsed by standard DOM parsers. The reason is that these pages are written in a special *WikiText*¹ language which is a markup language for wikipedia². As a consequence we needed to use a special wiki page parser to get the contents of the page.

6.1.2 Infrastructure

In Figure 6.1 we present our infrastructure. We use two Android devices (one mobile phone and a tablet) with different screen sizes and capabilities to motivate our work.

Integrated Development Environment	Android Developer Tools V22.2.1
Programming Language	Java
OS Developing Machine	Windows 8
Testing Devices	Nexus 7, Motorola G
Nexus 7 Resolution	1920 * 1200 pixels
Nexus 7 Physical Screen Size	7 inches
Motorola G Resolution	720 * 1280 pixels
Motorola G Physical Screen Size	4.5 inches

Figure 6.1: Tools and Platform

6.1.3 Scenario

Our scenario follows a client-server approach. After parsing the web page the server stores all its contents in our model. We have implemented REST services to distribute our adapted web page across different clients. We demonstrate the steps of the adaptation process in Figure 6.2.

At first, the client sends an XML file to our server with the device capabilities (1). This XML file is stored in the server (2). The XML file that we store for the device profile is shown in 6.3.

We also store a unique id for the client. This request is a *post* request from the client side to send a message to the server and says “who” is the client that asks for the web page. Then, through an HTTP request the client asks for the adapted

¹http://en.wikipedia.org/wiki/Wiki_markup

²Wikipedia page that we used in our use case is now implemented in standard html markup language

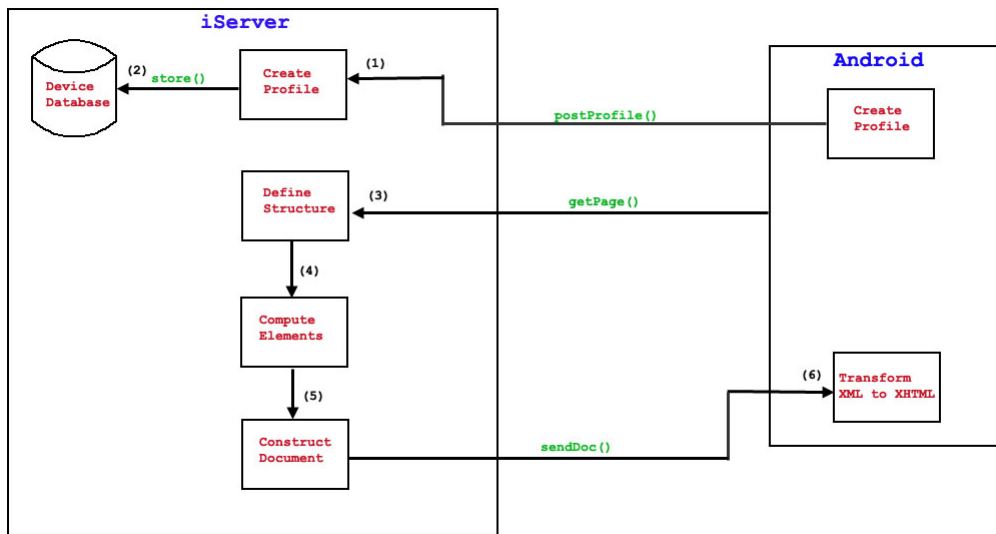


Figure 6.2: Steps in adaptation process

VUB web page (3). This request includes also the unique id that is previously stored into the server. Since the server has this unique id, it analyses the capabilities of the device and decides which structure could best match the device. Then, for each element that is added to the structure of our document calculations are made through our crosslet concept in order to get the representation of our elements (4). Since the server knows both the structure that will be followed and the presentation of the elements it begins to construct our final document (5). The server sends the corresponding XML file to the mobile device (6). Then, through some XSLT transformations the web page is displayed to the client.

At this point it is worth mentioning once again the power of our crosslet concept. In our related work (Chapter 3) we mentioned that XSLT is not a good candidate for adaptation. Our XSLT does nothing more than translating the XML elements into HTML tags since the structure of our elements have already been defined. Furthermore, we do not make extended use of CSS files, we just give style to some general elements such as *body* and *header*, our elements already contain information about their representation through our crosslet concept (font size, position, etc.).

```

<?xml version="1.0" encoding="UTF-8"?>
<device brand="google" id="JSS15Q" model="Nexus 7" name="asus">
  <cpu>
    <mhz>1512.0</mhz>
    <nrcores>4</nrcores>
  </cpu>
  <memory>1822</memory>
  <screen>
    <type>large</type>
    <width>1200</width>
    <height>1824</height>
    <screendpi>xhdpi</screendpi>
    <screenltr>true</screenltr>
    <touchscreen>true</touchscreen>
    <collaborativescreen>true</collaborativescreen>
  </screen>
  <camera>true</camera>
  <bluetooth>true</bluetooth>
  <gps>true</gps>
  <microphone>true</microphone>
  <nfc>true</nfc>
  <wifi>true</wifi>
  <audio>true</audio>
  <video>true</video>
  <keyboard>false</keyboard>
  <navpad>nonav</navpad>
</device>

```

Figure 6.3: Device profile

6.1.4 Tablet Device

We have used a Nexus 7³ tablet device in order to illustrate the outcome of our adapted web page. On the tablet device our algorithm decides that there is no need to change the structure of the document. Thus, an *InOrder* structure that we implemented follows exactly the same principles as our *document-component-link*, meaning that our target entities are retrieved in the same order that we inserted them during the parsing procedure. In addition a *WithSelectors* structure is defined to take our structure with the external links of wikipedia.

In terms of presentation though, we notice some modifications. Since sections are too long and contain a lot of content (paragraphs, images or tables), each section crosslet decides to give to each section a list presentation instead of displaying all the contents at once. Furthermore paragraphs contain a font size that is relative to the width of the screen, images scaling depends on the screen resolution, bulleted list elements are having circle elements in front of them. In general all our elements follow different representations for different devices. In Figure 6.4 we illustrate the outcome of such adaptation on our Nexus 7 device.

³<http://www.google.com/nexus/7/>

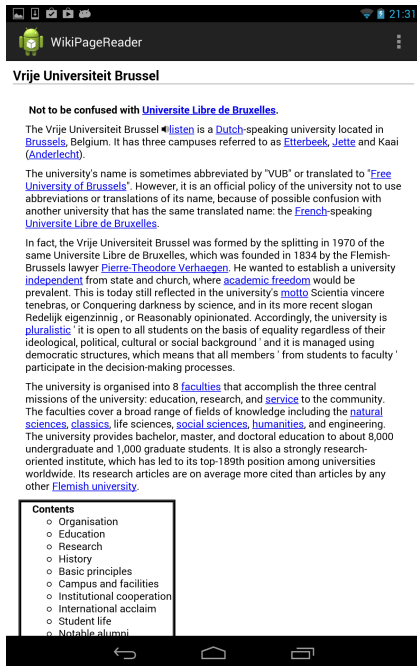


Figure 6.4: Main page until *table of contents* element

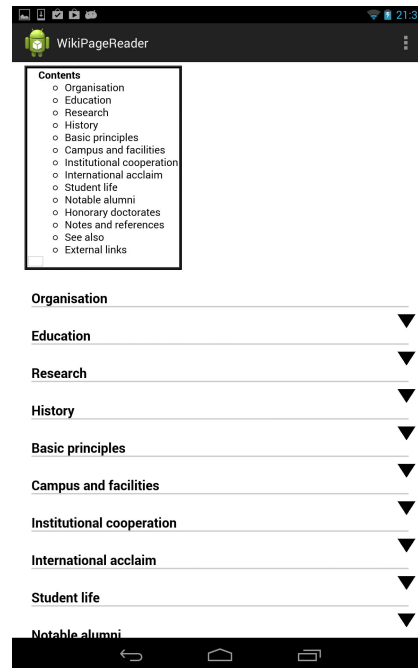


Figure 6.5: The remaining VUB page

As can be seen from Figure 6.5 the user can see the contents of each section by pressing the down arrow in the drop-down list element and then if he wants to switch it back he can press the up arrow of the element. An illustration of the contents of the section “*Organisation*” is shown in Figure 6.6. Our structure approach combined with crosslets nicely represents our document based on the device capabilities.

6.1.5 Smartphone

Another device that we used in order to illustrate the different presentations of our web page is a Motorola G mobile device. The main reason that we used such a device is the screen size of the mobile device which is almost two times smaller than a tablet device. Here our adaptation follows a very different approach both in terms of structure and element presentation. In Figures 6.7 and 6.8 as it can be seen the contents of the web page are too long to be displayed in the screen. Even if we assume that a list manner approach for each section would be followed, our entire page will be too long to fit the screen size, thus the user would extensively scroll down to find a section.

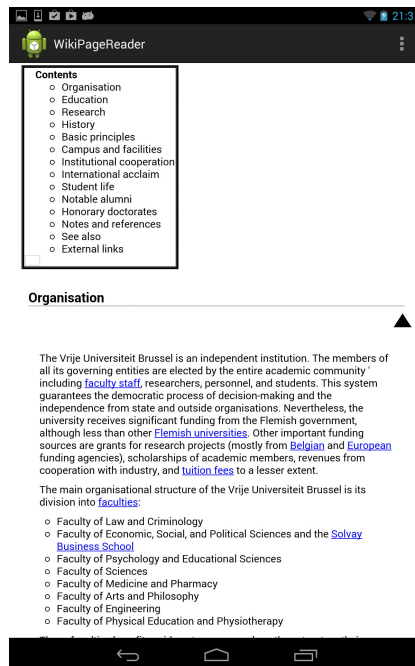


Figure 6.6: Section “Organisation” of the VUB web page

Our algorithm decides that full sections of the web page should not be presented on our document, but should be presented in separate documents (*Reference-Source* structure). Thus, whenever we have sections, a selector should be presented. This selector contains a link to our server. When the user clicks such a selector, an HTTP request to our server is made. Our server contains a REST interface implementation for sections. Thus, the XML file that is sent to the client is only a file that contains the content of a section. The user can switch back to the main page and select another section to read. Note that we chose a section entity to be displayed in separate pages, but we can do that implementation for any kind of resources (images, paragraphs, tables e.t.c.) or entities as long as we implement the corresponding REST interface. An example would be if we want our tables in separate pages, the only thing that we need to do is to create the corresponding URL selector every time that we have for instance a long table.

Figures 6.9 and 6.10 show two different sections of our document. Indeed retrieving each section in separate pages is a good candidate for such a device since the contents of a section is too long to be displayed on the device screen. As it can be seen also our crosslet approach nicely scales the images in a manner that can fit on the screen size. Furthermore all the Wikipedia links are presented in the document. The user can click on any of them and navigate to the corresponding

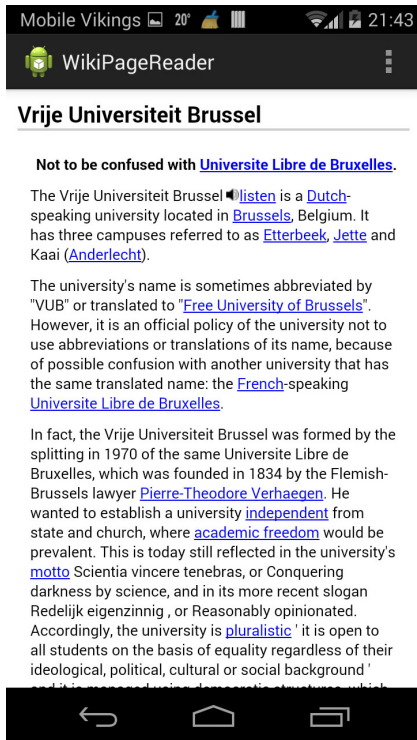


Figure 6.7: Main VUB page until *table of contents* element

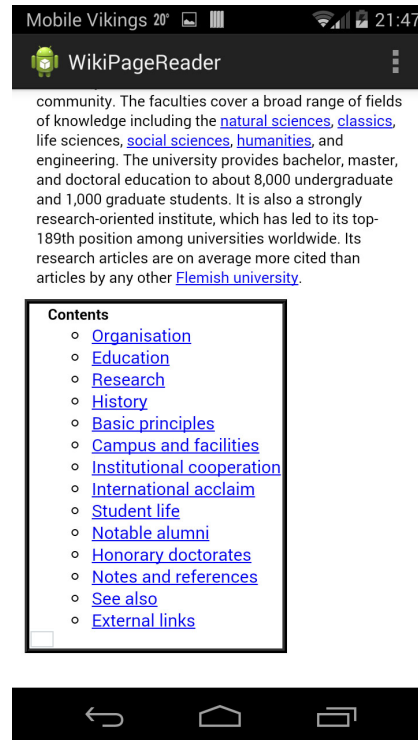


Figure 6.8: The remaining VUB page with URLs

Wikipedia page. Thus, a *WithSelectors* structure is also added to our substructures list.

6.1.6 Possible Structures

In our use case our document follows first an *InOrder* structure along with a *WithSelectors* structure on the tablet device and the same structure plus our *Reference* structure to create separate documents for different sections on the mobile device. However, in order to illustrate more structures that can be applied on our document we implemented some additional structures to show the effectiveness of our approach.

We implemented two extra substructures named *Reverse* and *NoSelectors*. The first structure can be used to take some specific entities (in this case we chose sections) in a reverse order manner and the latter to exclude selectors of the external links that we have in our document. Figure 6.11 shows our adapted web

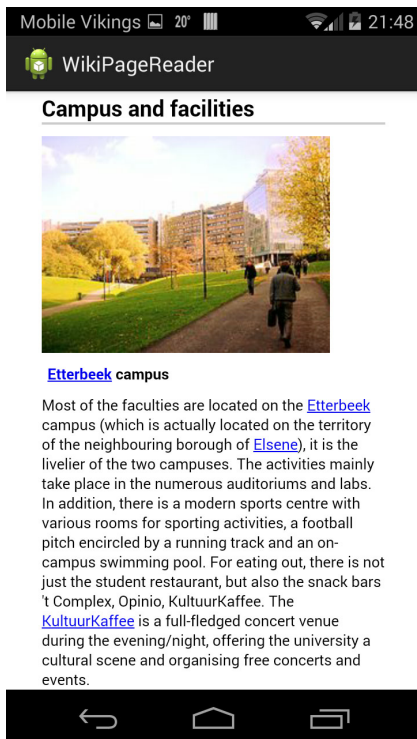


Figure 6.9: Campus and Facilities section

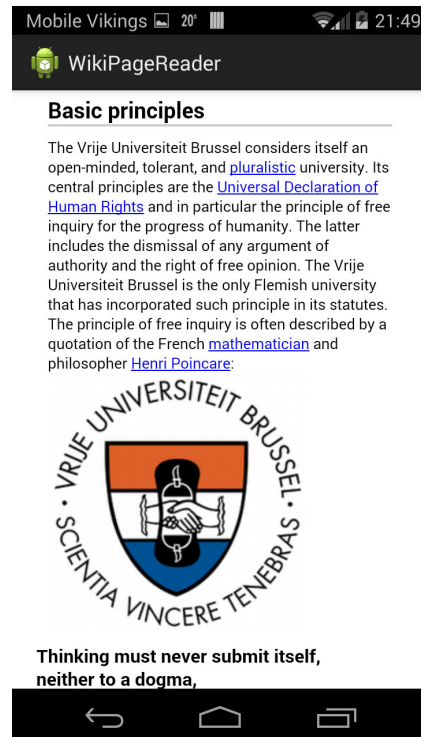


Figure 6.10: Basic principles section

page with the sections taken in reverse order. This would make sense maybe in an application that uses an alphabetical ordering of chapters or sections. However, we illustrate this example only to show how easy it is to apply some extra substructure in our document structure.

Another example would be to have a e-book reader as a device. E-book readers are widely used for reading digital books. In that case our document may have an extra substructure of *NoSelectors* which excludes our selectors of our document, meaning that all the URLs are not presented in the document. Our crosslet approach will focus then only with the presentation of the different elements. In Figure 6.12 it can be seen that the web page is nicely displayed with only the text of our elements excluding all the external links. This way of reading an article is more comfortable for the user without having many blue texts that may cause distraction. Note that this kind of presentation we could achieve by changing the presentation of the elements. However we believe that “hacking” *href* elements to get a nice presentation is not a good idea. Our proposal is to change the document on the structural level rather than on the presentation level.



Figure 6.11: Sections from the VUB web page in reverse manner

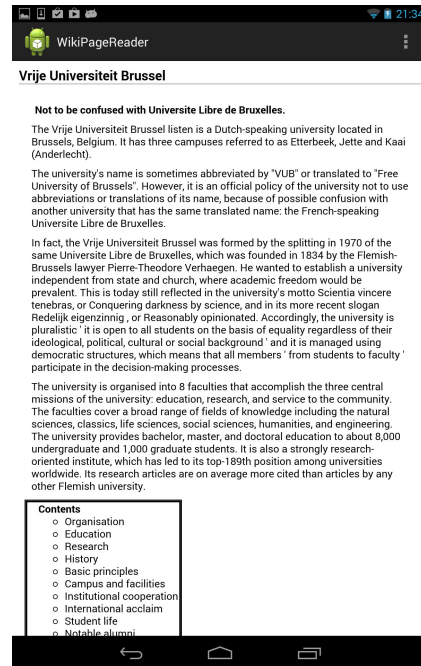


Figure 6.12: The VUB web page with no links

6.2 Conclusion

In this chapter we demonstrated our work with several use cases. Indeed changing the structure of the document can be beneficial in the adaptation procedure. Since the structure is not enough to the adaptation procedure we have shown how the same elements (sections, paragraphs, images, etc.) can be represented in different devices.

In addition we demonstrated the flexibility of our approach and we have shown how easily substructures can be built to serve different kind of devices (such as e-book readers). We believe that our model can be extended also to serve much more complicated structures for specific applications domains.

Conclusion and Future Work

In this chapter we conclude our work with a summary of our research questions and how we answer them. Furthermore we discuss limitations of our model that we faced during the implementation and we propose some possible improvements. Finally we give some proposals for future work which will make our model more powerful for further use.

7.1 Objectives of the Research

In this thesis we analysed in Chapter 3 different approaches of adaptation techniques. We examined how they consider adaptation in terms of structure of a document and how they adapt their elements in order to get different presentations for different mobile devices.

We have analysed a variety of different domains such as migratory interfaces, user/device centric approaches, multimedia adaptation etc. Most of the approaches that we analysed were implemented for web applications. All of the papers have at least one concrete implementation of a use case which illustrates their work. One of the first things that we noticed was that almost all of the approaches tackle well the mobile device capabilities which was our core concept in our research.

However, most of the approaches do not analyse in depth the structure of their documents. Thus they propose solutions such as to adapt the initial logical structure of a document. We argue that the initial document structure cannot be power-

ful enough for complex adaptations. By changing the document structure we gain flexibility and simplicity. Other approaches propose to use techniques such as extended XSLT transformations to get different structures. We have already mentioned during this thesis that extended use of XSLT leads in a very complex implementation via XSLT templates due to the different device capabilities considerations. Some of the approaches use interaction techniques where the user can select the elements that can be presented on a document. We believe that giving the user full control of the adaptation procedure should be clear enough both to the user and to the application, otherwise it may lead in an unexpected result.

The second step of our analysis focussed on the element granularity, meaning how the applications treat each element of a document. We have noticed that all of the reviewed research papers do not take into account the element granularity effect. Thus, they treat all the elements of the entire document as one unique unit. Thus, there is no clear separation of the elements and the adaptation procedure is hiding into the implementation. Since there is no clear conceptual model of how we can adapt elements of a document as unique units, this may lead in different interpretations of the approaches.

After our research we proposed a conceptual model to define structures of a document and to represent differently its elements. Our conceptual model has as a basis the Resource-Selector-Link (RSL) metamodel which has proven itself a good candidate with its hypermedia approach.

7.2 Contributions

Research question 1:

How can we define structure over a document?

Our proposal of defining structures is to use the power of the RSL metamodel and build a generic structural link-based modelling which serves as a core concept of our structure and then build several substructures which can be easily applied one by one to our document resulting in a document structure that is then easily handled by the application. The first substructure will be applied to the initial logical structure of our document. The rest of the substructures will be applied to the previous modified substructure. This conceptual model captures the semantics of a structure and serves as a guide for further use.

Research question 2:

How can we represent information in different contexts?

The representation of information is also another issue that was revealed in Chapter 3. We have realised that in previous approaches representations do not follow a clear conceptual model in terms of presentation of information. We state that our conceptual model of crosslets is a good candidate for representing information of any kind of resources. Crosslets have been very powerful since the individual elements can be represented in different ways. Furthermore programming individual elements at various level of granularity gives more control to the adaptation procedure.

7.3 Limitations

7.3.1 Order of the Structures

Since our substructures are analysed in order and one by one, it may be the case that changing the order of the substructures results in a unexpected structure. Thus, the developer should be aware of how many substructures applies to a document and in which order they have to be displayed to get the expecting result. One possible solution would be to define only one complex substructure if it is possible, however this complex substructure it is possible that it cannot be reused by other applications since probably contains application specific rules. With our conceptual model we tried to keep things as simple as possible. Nevertheless we believe that modelling complex structures is a challenging attempt that we tried to tackle and we think that we have achieved to solve it to some extent.

7.3.2 Future Work

In our conceptual model we do not introduce at all the user model which the RSL metamodel offers. Device capabilities are our core concept in our implementation. However it would be interesting if we could involve also the users in the adaptation procedure. One example would be to create a user interface and give the user control to change the structure of an application dynamically. Internally this would happen by changing links between different entities. We believe that

our conceptual model can already support this functionality. For example if the user might not need the external links to be presented in a document, he could “add” one *NoSelector* substructure to the existing structures and get the corresponding result. However for more complex structures more assumptions should be taken into consideration. It would be also interesting to test the user model on a collaborative screen when multiple interactions from the users are performed.

The introduction of a user model will also arise the need of a new conceptual model of our crosslet concept. For example to give the user the possibility to change representations among the entities of an application. Our crosslet concept then will also support dynamic modifications of different granularity levels which seems a powerful solution.

Last but not least, it would be interesting that our structures and crosslets would consider also the environment that a target device is surrounded by. It would be for example attractive to see how an audio crosslet would react in a noisy environment.

To conclude, all the assumptions that are proposed above require more abstract models that will cover a variety of cases both in terms of structure and by providing alternative presentations.

Bibliography

- Gregory D. Abowd, Anind K. Dey, Peter J. Brown, Nigel Davies, Mark Smith, and Pete Steggle. Towards a Better Understanding of Context and Context-Awareness. In Hans W. Gellersen, editor, *Handheld and Ubiquitous Computing*, volume 1707 of *Lecture Notes in Computer Science*, pages 304–307. Springer Berlin Heidelberg, 1999.
- Renata Bandelloni and Fabio Paternò. Migratory User Interfaces Able to Adapt to Various Interaction Platforms . *International Journal of Human-Computer Studies*, 60(56):621 – 639, 2004. HCI Issues in Mobile Computing.
- Tim Berners-Lee and Mark Fischetti. *Weaving the Web: The Original Design and Ultimate Destiny of the World Wide Web by Its Inventor*. Harper Information, 2000.
- Silvia Berti, Francesco Correani, Giulio Mori, Fabio Paternò, and Carmen Santoro. TERESA: A Transformation-based Environment for Designing and Developing Multi-Device Interfaces. In *CHI 2004 Extended Abstracts on Human Factors in Computing Systems*, pages 793–794, Vienna, Austria, April 2004. ACM.
- Peter Brusilovsky. Adaptive Hypermedia. *User Modeling and User-Adapted Interaction*, 11(1-2):87–110, 2001.
- D Bulterman and L Rutledge. *SMIL 2.0-Interactive Multimedia for Web and Mobile Devices*. Springer, 2004.
- Pablo Cesar, Ishan Vaishnavi, Ralf Kernchen, Stefan Meissner, Cristian Hesselman, Matthieu Boussard, Antonietta Spedalieri, Dick C.A. Bulterman, and Bo Gao. Multimedia Adaptation in Ubiquitous Environments: Benefits of

- Structured Multimedia Documents. In *Proceedings of the eighth ACM symposium on Document engineering*, DocEng 2008, pages 275–284, Sao Paulo, Brazil, 2008. ACM.
- Tarak Chaari, Frédérique Laforest, and Augusto Celentano. Adaptation in Context-Aware Pervasive Information Systems: The SECAS Project. *International Journal on Pervasive Computing and Communications (IJPCC)*, 3(4): 400–425, December 2007.
- James Clark. XSL Transformations (XSLT), Version 1.0. W3C Recommendation. *World Wide Web Consortium*, <http://w3c.org/TR/xslt>, 1999.
- Dov Dori, David Doermann, Christian Shin, Robert Haralick, Ihsin Phillips, Mitchell Buchman, and David Ross. The Representation of Document Structure: A Generic Object-process Analysis. Technical report, College Park, MD, USA, 1995.
- Jochen François. Dynamic Content in Fluid Cross-Media Documents. Master’s thesis, Vrije Universiteit Brussel, Brussels, Belgium, 2014.
- Elisabeth Freeman, Eric Freeman, Bert Bates, and Kathy Sierra. *Head First Design Patterns*. O’ Reilly & Associates, 2004.
- R. Furuta, V. Quint, and J. Andre. Interactively Editing Structured Documents. *Electron. Publ. Origin. Dissem. Des.*, 1(1):19–44, April 1989.
- Shudi Gao, C Michael Sperberg-McQueen, Henry S Thompson, Noah Mendelsohn, David Beech, and Murray Maloney. W3C XML Schema Definition Language (XSD). *W3C Candidate Recommendation*, 30, 2009.
- K. Geihs, P. Barone, F. Eliassen, J. Floch, R. Fricke, E. Gjørven, S. Hallsteinen, G. Horn, M. U. Khan, A. Mamelli, G. A. Papadopoulos, N. Paspallis, R. Reichle, and E. Stav. A Comprehensive Solution for Application-level Adaptation. *Software: Practice and Experience*, 39(4):385–422, March 2009.
- Giuseppe Ghiani, Fabio Paternò, and Carmen Santoro. On-demand Cross-device Interface Components Migration. In *Proceedings of the 12th International Conference on Human Computer Interaction with Mobile Devices and Services*, Mobile HCI 2010, pages 299–308, Lisbon, Portugal, September 2010. ACM.
- Frank Halasz and Mayer Schwartz. The Dexter Hypertext Reference Model. *Communications of the ACM*, 37(2):30–39, February 1994.

- Iyad Khaddam and Jean Vanderdonckt. Adapting UsiXML User Interfaces to Cultural Background. In *Proceedings of 1st International Workshop on User Interface eXtensible Markup Language UsiXML*, pages 163–170, Berlin, Germany, 2010.
- Tayeb Lemlouma and Nabil Layada. Universal profiling for content negotiation and adaptation in heterogeneous environments. In *W3C Workshop on Delivery Context, W3C/INRIA*, pages 4–5, Sophia-Antipolis, France, March 2002.
- Tayeb Lemlouma and Nabil Layaida. Content Adaptation and Generation Principles for Heterogeneous Clients. In *OPERA Project, INRIA Rhone Alpes, Position Paper for the W3C Workshop on Device Independent Authoring Techniques*, 2002.
- Tayeb Lemlouma and Nabil Layaida. Adapted Content Delivery for Different Contexts. In *Proceedings of the 2003 Symposium on Applications and the Internet, SAINT 2003*, pages 190–197, Washington, DC, USA, 2003. IEEE Computer Society.
- Quentin Limbourg, Jean Vanderdonckt, Benjamin Michotte, Laurent Bouillon, and Víctor López-Jaquero. USIXML: A Language Supporting Multi-path Development of User Interfaces. In *Engineering Human Computer Interaction and Interactive Systems*, volume 3425 of *Lecture Notes in Computer Science*, pages 200–220. Springer Berlin Heidelberg, 2005.
- Dave E. Millard, Luc Moreau, Hugh C. Davis, and Siegfried Reich. FOHM: A Fundamental Open Hypertext Model for Investigating Interoperability Between Hypertext Domains. In *Proceedings of the Eleventh ACM on Hypertext and Hypermedia*, pages 93–102, San Antonio, Texas, USA, 2000. ACM.
- T. H. Nelson. Complex Information Processing: A File Structure for the Complex, the Changing and the Indeterminate. In *Proceedings of the 1965 20th National Conference*, ACM 1965, pages 84–100, Cleveland, Ohio, USA, 1965. ACM.
- Ted Nelson. On the xanadu project. *BYTE Magazine*, 15(9):298–299, 1990.
- Theodor Holm Nelson. *Literary machines 93.1.: the report on, and of, project Xanadu concerning word processing, electronic publishing, hypertext, thinker-toys, tomorrow's intellectual revolution, and certain other topics including knowledge, education and freedom*. Mindful Press, 1992.
- Moira C. Norrie. An Extended Entity-Relationship Approach to Data Management in Object-Oriented Systems. In Ramez A. Elmasri, Vram Kouramajian, and Bernhard Thalheim, editors, *Entity-Relationship Approach ER 1993*,

- volume 823 of *Lecture Notes in Computer Science*, pages 390–401. Springer Berlin Heidelberg, 1994.
- Moira C. Norrie. General Framework for the Rapid Development of Interactive Paper Applications. In *Proceedings of CoPADD 2006, 1st International Workshop on Collaborating over Paper and Digital Documents*, volume 6, pages 9–12. Citeseer, 2006.
- Moira C. Norrie, Beat Signer, and Nadir Weibel. Print-n-link: Weaving the Paper Web. In *Proceedings of the 2006 ACM Symposium on Document Engineering, DocEng '06*, pages 34–43, Amsterdam, The Netherlands, 2006. ACM.
- Moira C. Norrie, Beat Signer, Michael Grossniklaus, Rudi Belotti, Corsin Decurtins, and Nadir Weibel. Context-aware Platform for Mobile Data Management. *Journal of Wireless Networks*, 13(6):855–870, December 2007.
- Peter J. Nürnberg and M. C. Schraefel. Relationships Among Structural Computing and Other Fields. *Journal of Network and Computer Applications*, 26(1): 11–26, January 2003.
- James M. Nyce and Paul Kahn, editors. *From Memex to Hypertext: Vannevar Bush and the Mind's Machine*. Academic Press Professional, Inc., San Diego, CA, USA, 1991.
- Fabio Paternò. *Model-Based Design and Evaluation of Interactive Applications*. Springer-Verlag, London, UK, 1st edition, 1999.
- Fabio Paternò and Giuseppe Zichittella. Desktop-to-Mobile Web Adaptation through Customizable Two-Dimensional Semantic Redesign. In Regina Bernhaupt, Peter Forbrig, Jan Gulliksen, and Marta Lrusdttir, editors, *Human-Centred Software Engineering*, volume 6409 of *Lecture Notes in Computer Science*, pages 79–94. Springer Berlin Heidelberg, 2010.
- Fabio Paternò, Carmen Santoro, and Lucio Davide Spano. MARIA: A Universal, Declarative, Multiple Abstraction-level Language for Service-oriented Applications in Ubiquitous Environments. *ACM Transactions on Computer-Human Interaction*, 16(4):19:1–19:30, November 2009.
- Fabio Paternò, Carmen Santoro, and Antonio Scordia. Ambient Intelligence for Supporting Task Continuity across Multiple Devices and Implementation Languages. *The Computer Journal*, 53(8):1210–1228, 2010.
- Barry Redmond and Vinny Cahill. Iguana/J: Towards a Dynamic and Efficient Reflective Architecture for Java. In *Proceedings of ECOOP 2000 Workshop on*

- Reflection and Metalevel Architectures*, Sophia Antipolis and Cannes, France, June 2000.
- Brian Keith Reid. *Scribe: A Document Specification Language and Its Compiler*. PhD thesis, Pittsburgh, PA, USA, 1981.
- Mazeiar Salehie and Ladan Tahvildari. Self-adaptive Software: Landscape and Research Challenges. *ACM Transactions on Autonomous and Adaptive Systems (TAAS)*, 4(2):14:1–14:42, May 2009.
- B. Schilit, N. Adams, and R. Want. Context-aware Computing Applications. In *Proceedings 1994 Workshop on Mobile Computing Systems and Applications*, pages 85–90, Santa Cruz, CA, USA, December 1994.
- Albrecht Schmidt, Michael Beigl, and H. W. Gellersen. There is More to Context than Location. *Computers & Graphics*, 23(6):893 – 901, November 1999.
- Ron Shacham, Henning Schulzrinne, Srisakul Thakolsri, and Wolfgang Kellerer. Ubiquitous Device Personalization and Use: The Next Generation of IP Multimedia Communications. *ACM Transactions on Multimedia Computing, Communications, and Applications*, 3(2), May 2007.
- Beat Signer. *Fundamental Concepts for Interactive Paper and Cross-Media Information Spaces*. PhD thesis, ETH Zürich, Switzerland, 2005.
- Beat Signer. What Is Wrong with Digital Documents? A Conceptual Model for Structural Cross-Media Content Composition and Reuse. In Jeffrey Parsons, Motoshi Saeki, Peretz Shoval, Carson Woo, and Yair Wand, editors, *Conceptual Modeling ER 2010*, volume 6412 of *Lecture Notes in Computer Science*, pages 391–404. Springer Berlin Heidelberg, 2010.
- Beat Signer and Moira C. Norrie. As We May Link: A General Metamodel for Hypermedia Systems. In Christine Parent, Klaus-Dieter Schewe, Veda C. Storey, and Bernhard Thalheim, editors, *Conceptual Modeling - ER 2007*, volume 4801 of *Lecture Notes in Computer Science*, pages 359–374. Springer Berlin Heidelberg, 2007.
- Beat Signer et al. PaperPoint: A Paper-based Presentation and Interactive Paper Prototyping Tool. In *Proceedings of the 1st International Conference on Tangible and Embedded Interaction*, TEI 2007, pages 57–64, Baton Rouge, Louisiana, 2007. ACM.
- Ahmed A. O. Tayeh. *A Metamodel and Prototype for Fluid Cross-Media Document Formats*. PhD thesis, Faculty of Science, Department of Computer Science. Vrije Universiteit Brussel, 2012.

- David Thevenin and Joëlle Coutaz. Plasticity of User Interfaces: Framework and Research Agenda. In *Proceedings of INTERACT 1999 - IFIP TC13 Seventh International Conference on Human-Computer Interaction*, pages 110–117, Edinburgh, Scotland, September 1999.
- Jean Vanderdonckt. A MDA-Compliant Environment for Developing User Interfaces of Information Systems. In Oscar Pastor and João Falcão e Cunha, editors, *Advanced Information Systems Engineering*, volume 3520 of *Lecture Notes in Computer Science*, pages 16–31. Springer Berlin Heidelberg, 2005.
- Nadir Weibel, Moira C. Norrie, and Beat Signer. A Model for Mapping Between Printed and Digital Document Instances. In *Proceedings of the 2007 ACM Symposium on Document Engineering, DocEng '07*, pages 19–28, Winnipeg, Manitoba, Canada, 2007. ACM.
- Dongsong Zhang. Web Content Adaptation for Mobile Handheld Devices. *Communications of the ACM*, 50(2):75–79, February 2007.