Thesis Websites for Visually Impaired Users

Stefan Woods
2de Licentie Toegepaste Informatica

Academiejaar 2006-2007

Samenvatting

In onze moderne maatschappij kan het belang van Websites niet overschat worden. Zij zijn de digitale vertegenwoordigers van bedrijven en zijn 24h/24h actief, 7d/7d. Zij zijn ook het elektronische gezicht van de overheid waardoor ze gebruikt worden. Bovendien is ook de academische wereld een dankbare gebruiker voor zijn onderzoek, publications en informatieverspreiding voor studenten of het algemene publiek. Ontelbare andere organisaties, kleine ondernemingen en miljoenen individuen maken op de een of andere manier gebruik van het Internet en meer specifiek van Websites.
Veel Websites zijn geëvolueerd naar echte Web Toepassingen hetgeen veel meer voorstelt dan een samenraapsel van enkele webpagina's.

Tegelijkertijd heeft het Internet zich ontpopt tot een even belangrijk medium als radio en T.V. en overschaduwt het deze soms[1]. Maar in de loop van zijn snelle ontwikkeling zijn de behoeften van bepaalde gebruikersgroepen vergeten. Eén zo een groep zijn de blinden en slechtzienden. Deze mensen surfen op het internet op een andere manier dan mensen met een normaal zicht en worden daardoor ook geconfronteerd met andere, specifieke problemen. Recentelijk is er meer aandacht voor hun behoeften en worden er inspanningen gedaan om hun het leven als Internet surfer gemakkelijker te maken (b.v. het Web Accessibility Initiative (WAI) or W3C). Deze thesis is ook zo een inspanning.

Wij bekijken de ontwikkelingskant van het probleem, liever dan een hulpmiddel te ontwikkelen om blinden en slechtzienden te helpen bij hun Website navigatie. Een goede plaats om hier mee te beginnen is de Web Site Development Methodology ontwikkeld aan de Vrije Universiteit Brussel, beter bekend als WSDM. Deze methodologie bestaat uit 5 stappen en laat de designer toe om een Website vanaf nul op te bouwen tot het punt van implementatie. Een andere aanpak, Dante genaamd, neemt een bestaande Website en hervormt deze -met behulp van de designer- op een zodanige manier dat hij beter geschikt wordt om te navigeren voor blinden en slechtzienden. Eerder onderzoek [1] combineerde reeds deze 2 aanpakken met het ultieme doel een Website te bouwen die beter geschikt is voor blinden en slechtzienden.

Deze thesis focust op de link tussen the Dante en WSDM aanpak, meer bepaald op de overeenkomsten tussen hun respectievelijke ontologieën. WSDM gebruikt een specifieke ontologie als een soort container of databank om elementen in op te slaan gedurende het ontwikkelingsproces. Dante gebruikt een andere ontologie (de WAfA ontologie) om bestaande Webpagina's te annoteren om op die manier de pagina's te transformeren. WSDM kan met Dante gecombineerd worden door een WSDM tegenhanger te vinden van de concepten die we terugvinden in de WAfA ontologie. Dit doen we door mapping regels op te zetten en de WSDM methodologie uit te breiden waar nodig. Sommige van deze regels bestonden al (opgezet in eerder onderzoek [1]) maar zij omvatten niet alle WafA ontologie Authoring concepten. Wij zullen deze regels herbekijken en nieuwe opzetten waardoor we ook het WSDM proces zullen uitbreiden met de bedoeling zoveel mogelijk WAfA ontologie concepten te "dekken".

Eens deze regels opgezet, introduceren we enkele voorbeelden samen met een tool om de besproken regels uit te voeren en op die manier de validiteit van ons werk aan te tonen.

[1] Voor de eerste keer in de geschiedenis van de V.S. gingen presidentiële kandidaten een debat aan waar hen vragen werden gesteld in de vorm van video's gepost op de welbekende Website YouTube.com. Bij het ter perse gaan van deze thesis was de ronde voor de Democratische kandidaten voorbij (23 juli) terwijl het Republikeinse debat gepland is voor 17 september.

Abstract

In our modern society, the importance of Websites cannot be overstressed. They are the digital representatives of companies and are active 24 hours a day, 7 days a week. They are the electronic face of the government that uses them to inform the public. The academic world is also a grateful user for its research, publications and to inform students and the general public. Countless other organisations, small businesses and millions of individuals make use of the Internet in general and Websites in particular. Many Websites have evolved into Web Applications, being a lot more than a motley collection of Webpages.
At the same time, the Internet is rapidly becoming a medium of the same importance of radio and television, and regularly overshadowing the other two[2]. But in the course of it's rapid development, the requirements of certain user groups were ignored or forgotten. One specific user group whose needs were overlooked is the Visually Impaired. Blind people or people with bad eyesight surf the Internet in a different way than normally sighted users would and are therefore presented with a whole different set of problems. In recent years their specific requirements were acknowledged and efforts made to make their surfing life easier (e.g., the Web Accessibility Initiative (WAI) or W3C). This thesis is also such an effort.

Instead of developing tools to help visually impaired users navigate Websites, we target the design-side of the problem. The place to start is the Web Site Development Methodology developed at the Vrije Universiteit Brussel, a.k.a. WSDM. This methodology consists of 5 steps and allows designing a Website from scratch up to the point of implementation. Another approach called Dante takes an existing Website and –with the help of the designer- redesigns it in such a way that it becomes easier to travel for visually impaired users. Earlier research [1] already combined these two approaches with the ultimate goal to produce a Website that is better suited to the visually impaired user's needs.

This thesis focuses on the link between the Dante and WSDM approach, and more in particular on the correspondence between their respective ontologies. WSDM uses a specialised ontology as a sort of container to store elements during the design process. Dante uses another ontology (the WAfA ontology) to annotate existing Web pages to allow the pages to be "reshaped". The WSDM approach can be combined with the Dante approach by establishing a WSDM counterpart for the concepts found in the WAfA ontology. It is done by means of mapping rules and extending the WSDM ontology where necessary. Some of these mapping rules were already set up through earlier research [1] but they do not cover all the WAfA ontology Authoring concepts. We will review the existing rules, and create new ones extending the WSDM methodology in the process in an effort to cover as many of the WAfA ontology Authoring concepts as is possible.
Once these rules are in place we can introduce some examples along with a tool to execute the rules to show the validity of our work.

[2] For the first time in U.S. history presidential candidates engaged in a debate where questions were posed in the form of videos uploaded via the well-known Website YouTube.com. At the time of writing the Democratic candidates already completed their round (on july 23rd) with the Republican candidates debate scheduled to take place September 17th.

Acknowledgements

When I started my studies at the Vrije Universiteit Brussel as an evening student in Computer Science in 1999 I had no idea of the journey that lay ahead. The first academic year was in a certain sense a check – a test to see if university level studies could be combined at all with a full time job, a wife, sports and a social life. This proved feasible – but only just. The birth of my son in 2000 and my daughter in 2003 made the going even tougher. The duty of husband, father and full time consultant with a few international job assignments ate away the time I could devote to my study, lengthening my academic “career” as a result.
The road was long. It is appropriate at this point to extend my gratitude to my wife for putting up with an absent and preoccupied husband for all those years and to my parents for helping out where they could to facilitate my study.

Special thanks should also go to Dr. Sven Casteleyn and Prof. Dr. Olga De Troyer for answering my many questions, correcting my errors and generally guiding me through my research.

Table of Content

List of Figures

1 Preface

1.1 Motivation

At present, the Web is becoming the first and foremost source of information worldwide. It is consultable by anyone who can connect to the Internet and continues to expand. In recent years, it's value as an application platform is exploited, thereby expanding its functionality.

With the Web becoming a (large) part of every day life, it is becoming increasingly important for people with disabilities to be able to access the information or functionality as a normal user would. This is especially very important for visually impaired users; a blind person or someone with limited vision cannot use a computer (and therefore neither a Web browser) in the same way normally sighted users do. Moreover, in recent years the possibilities for visually impaired users with respect to computer usage in general and Web use in particular, have in fact diminished [3]. This is due to the fact that currently almost every interface is graphical, thereby limiting the usefulness of tools like e.g. screen readers or Braille output.

To assist visually impaired users, the industry developed several assistive tools like e.g., screen readers, magnifiers, Braille printers, etc… These tools provide visually impaired users with assistive technology for computer usage in general and Web browsing in particular.
Unfortunately, such assistive technologies have their limitations. A Web page with a flashy graphical intro cannot be "seen" by a visually impaired user. A solution to this problem would be to adapt the design of the Website and drop the flashy graphical stuff altogether or even better, provide an alternative. So, next to the assistive technology available to a disabled user, the design of a Web application is at least equally important.
Thus, Web applications need to be built with accessibility (for the visually impaired) in mind.

Another problem area is navigation: It is, for example, very difficult for a screen reader to recognize a collection of links as a menu because most of the time, these links are only *visually* grouped together and thereby form a menu. Things get even more complicated through the usage of top (main) menus, side menus and bottom menus or footers.
A solution would be to make a browser actually recognize navigational items on a Website.

1.2 Goal of the thesis

The purpose of this thesis is to embed VIA (*Visually Impaired Awareness*) in the Web design process. We do this by adapting an existing design methodology so that, when applied correctly, the navigational VIA is automatically build-in and one should not be worried about accessibility after the Website is implemented.

Previous research [1] focused on the so-called Dante approach, which sports semantic annotation of Web pages. This extra semantic knowledge can facilitate e.g. screen readers in their task of audio representation of a web page.
Dante analyses Web pages to identify objects that support navigation. These objects are then annotated with terms from an ontology, the so-called Web Authoring for Accessibility (or short: WAfA) ontology. The resulting markup is used to transcode pages into a form that is easy to travel (and therefore better suited to visually impaired users).

To eliminate its biggest drawback i.e. the manual extraction and annotation of objects, this process has been automated to a certain extent by combining the Dante approach with the WSDM methodology [1]. This was realized by creating a mapping between both the WSDM and WAfA ontologies, with the WSDM ontology forming an integral part of the WSDM design process. In this way WAfA annotations are generated "on the fly" by simply following the WSDM design methodology (WAfA annotations are generated from the design specifications collected during the design phase).

This integration allows using roughly 70% of the concepts defined in the WAfA ontology. The goal of the thesis is to raise this percentage. Therefore, it is necessary to adapt and expand the WSDM process.

A case study will be used to illustrate the main issues and the results obtained in this thesis. For this, the Website of the research group WISE will be used. First, the WISE Website is evaluated for its accessibility for visually impaired users by using evaluation tools, which check a Website against existing accessibility guidelines. Next, this Website will be used to illustrate and validate the WSDM extensions introduced.

1.3 Structure of this document

The structure of this thesis is as follows: Chapter 1 introduced the problem of accessibility of Websites for visually impaired users. In Chapter 2, an overview of tools and aids available to visually impaired users is presented, as well as related work. In particular, the current state of research in the field of Web design with an emphasis on visual-impaired awareness is discussed. In Chapter 3, we discuss the results of the accessibility evaluation of the WISE Website. Chapter 4 discusses the WSDM extension needed to support all WAfA concepts. In Chapter 5 the WISE Website is reconsidered in the context of the extension discussed in Chapter 4. Finally, Chapter 6 presents conclusions.

2 Accessibility for Visually Impaired Users – State of the Art

Accessibility can be a problem in Web design but one that needs to be addressed. For example, the U.S. government's Section 508 of the Rehabilitation Act requires "Federal electronic and information technology to be accessible to people with disabilities, including employees and members of the public" [8].

Most modern governments state that no citizens may be deprived of the access to information presented on governmental Websites. But this is exactly the case if such a Website has some serious accessibility issues and thus the government can be held accountable for discrimination.

Another but less noble drive to address accessibility issues on (private-funded) Websites is the economical potential of users with accessibility issues[3].

As a consequence of this new awareness, several components supporting accessibility are available to users as well as to Web developers. These components are meant to work together to make the Web more accessible to the (visually) impaired user. They can be divided into **Web browsers/assistive technology for users** on one hand, and **evaluation tools/authoring tools for Web developers** on the other. In this chapter we provide an overview of these components. We will also review related work in the context of the research on Web design and accessibility for visually impaired users.

2.1 Technical aids/assistive technology

The Web is geared towards people with normal vision4. This makes it harder for visually impaired people to browse Websites and find the information they are looking for. Technical aids or assistive technology are tools developed to help visually impaired people working with a computer and browsing the Web. An overview of available tools follows.

2.1.1 Braille Bars & Braille Keyboards

The Braille Bar (also called Braille Terminal or Braille Display) is the "computer screen for the blind". Applied to any computer, it enables the content of the screen to be "translated" in a Braille text. This can then be "read" by the user.

Usually a Braille Bar is combined with a Braille Keyboard. Braille Keyboards are available in different shapes and sizes (many resemble regular keyboards).

Few blind people read Braille however [3], making screen readers (see below) the preferred output device for the visually impaired.

2.1.2 Screen readers

Screen readers are software programs that present graphics and text as speech. A screen reader is used to verbalize, or "speak," everything on the screen including names and descriptions of control buttons, menus, text and punctuation.

The general problem with this type of "browsers" is that they do a poor job in conveying the logical structure and semantics of content in Web documents, nor do they provide users with easy ways to select which parts of a document to listen to. As a consequence, users with a visual disability waste a considerable amount of time and attention listening to irrelevant information [6].

Recently however, there is a tendency to make screen readers more "intelligent" – i.e. assist them in several ways so they can better convey the content or even structure of the visited Web page. The proposed annotation process Dante, mentioned earlier in this thesis is such an example.

2.1.3 Braille Printers

Braille printers (also called Braille Embossers) enable printing onto paper of any text from a word processor in the raised characters of Braille format.

[3] To name but one striking example : http://soundsdirty.com/ accessed 2007 – the fact that even a type of industry not usually renown for it's high standard or unquestionable morale makes efforts towards the visually impaired proves our case.

[4] Some information in this section is taken from the Microsoft Website (http://www.microsoft.com/enable/guides/vision.aspx , accessed 2006)

Various models differentiate by printing times and (the more sophisticated ones) by the ability to print on both sides of the paper. Even though functioning of the printer is not dissimilar from a regular printer, difficulties arise by the necessity to transcribe text in correct 6-point Braille format. It requires that upper case letters or numbers are preceded by a special symbol. Braille printing is also managed by software able to adapt the text to the needs of different Braille writings.

2.1.4 Scanners and OCR systems

A scanner is a tool that captures a graphic image and transforms it into digital information. OCR programs (Optical Character Recognition) recognize characters on paper and transform the image in a word processor document that can be saved onto disk, printed or read in Braille or by the screen reader. Scanners and OCR's are products for general use; however there are some OCR programs specifically made for blind people: they can decode text even if it's not correctly positioned on the scanner, they can recognize the page structure even if divided in columns, titles and paragraphs, as well as eliminate drawings, photographs and diagrams. There are also OCR-connected scanners that immediately read out the scanned text.

2.1.5 Speech Recognition Systems

Speech recognition systems, also called voice recognition programs, allow people to give commands and enter data using their voices rather than a mouse or keyboard. Though this is a general technology, it's use for the visually impaired and people having trouble working with normal input devices such as keyboard or mouse, is more than obvious.
Unfortunately speech as an interface still is a technology in its infancy, despite lots of research and other efforts in this area.

2.1.6 Video-magnifiers

Video Magnifiers are tools that film the image of a text and magnify it before projecting it onto a screen. With an electronic zoom it is possible to make enlargements. This can reduce the visual field; therefore the user must move the text around in order to read it under the view-finder. Video-magnifiers are essentially used for reading paper-printed text. It is a technology used by people who have bad eyesight but who are not blind.

2.1.7 Computer or Screen Magnifiers

Computer or screen magnifiers are programs that interface with the computer's graphical output, enlarging the screen content. Apart from enlarging screen content, these programs also offer other functionality like e.g., contrasting colors, smoothing (of enlarged text), offering different magnification modes etc…
Magnifiers in general are assistive technology for visually impaired people with some degree of functional vision. Users with no functional vision at all mostly use screen readers or other assistive technology.

2.1.8 Voice Browsers

Voice Browsers are essentially Web Browsers that are speech-driven. They allow users to access the Web using speech synthesis, pre-recorded audio and speech recognition. In this they are in a way the opposite of screen readers who use speech to convey Web content (Though voice browsers exist that also "speak" Web page content like screen readers making the distinction between them a grey, shady area). Voice Browsers are not developed specifically for visually impaired users.
At the time of writing voice browser technology is still in its infancy but is developing rapidly [6]. Associated with it is a special markup language called VoiceXML[5] that is designed for creating audio dialogs. VoiceXML has acquired industrial backing and is consequently becoming the industry standard in it's field[6].

[5] See http://www.w3.org/Voice and http://www.hitmill.com/internet/browsers.html (accessed 2006)
[6] See http://www.voicexml.org (accessed 2006)

2.2 Guidelines

The guidelines as referred to in this thesis are a set of rules or directives that aim at making Web content accessible to people with disabilities. Whereas assistive technology is intended for users, guidelines are intended for Web content developers, developers of authoring tools, browsers or media players [5] and therefore highlight the development side of the problem. These guidelines are general: they are not solely intended for the visually impaired but are intended to make Websites more accessible for people with disabilities in general.
The de facto standard in this field are the Web Access Initiative (WAI)[7] guidelines as published online by the World Wide Web Consortium (W3C) [7]. While these are general guidelines we will focus on their visually impaired aspect.

W3C proposes three different sorts of guidelines [5]:

1. **WCAG (Web Content Accessibility Guidelines)** describe how to make accessible Web content and Websites. Examples of requirements in WCAG include providing equivalent alternatives to auditory and visual content, providing clear and consistent navigation mechanisms, usage of features that enable activation of page elements via a variety of input devices.
2. **ATAG (Authoring Tool Accessibility Guidelines)** describe how to make Web authoring software that produces accessible content. Examples of requirements in UAAG is that access to content needs to be provided through a variety of navigation mechanisms
3. **UAAG (User Agent Accessibility Guidelines)** describe how to make browsers and media players accessible. Examples of requirements in ATAG include that Web authoring tools need to generate valid markup & that they can be configured to prompt for accessibility content such as e.g. alternative text for images, captions for audio, descriptions for video, etc …

Of the three above-mentioned guidelines, WCAG are the most widely used. At the time of writing up to version 2.0 they impact the HTML-code of a Website and advise a Web developer in some cases what HTML-code to write. An example is the use of an image on a Website. Whatever the image represents, it is meaningless to a blind user. The guidelines tell a developer to always add an HTML alt-tag (alt=”description”) to an image, i.e. the alt-tag should provide a textual description of the image. In this way, when a blind user navigates over the image, his screen reader can read out the supplied alt-text to him e.g., “picture of a coastal landscape”, in stead of just informing him there is an image present.
The more interesting part about WCAG is that some of its checks can be automated [5]. This in turn lead to the development of tools like e.g. Watchfire’s Bobby[8] or ATRC’s Web Accessibility Checker[9]. These tools are explained in more detail in chapter 3 Accessibility Evaluation of the WISE Website.

The guidelines as proposed by W3C/WCAG can be arranged in three levels of “severity” called priority 1, 2 and 3. We will explain them briefly:

- **Priority 1**: if the guidelines of this level aren’t followed, some user groups may find it *impossible* to access certain information on the site.
- **Priority 2**: if the guidelines of this level aren’t followed, some user groups may find it *(very) difficult* to access certain information on the site.
- **Priority 3**: if the guidelines of this level aren’t followed, some user groups may find it *somewhat difficult* to access certain information on the site.

In a perfect world, or rather on a perfect Website, all three priorities levels should be dealt with. The reality is that in most Websites, problems with priority level 2 and 3 occur and even problems of

[7] See http://www.w3.org/WAI/ (accessed 2006)
[8] See http://www.watchfire.com/products/desktop/accessibilitytesting/default.aspx (accessed 2005)
[9] See http://checker.atrc.utoronto.ca/servlet/Submit (accessed 2007)

priority 1 are commonplace, leaving much to be desired for the visually impaired surfer. So there is definitely room for improvement here.

2.3 Accessibility Tools

Accessibility tools are software programs or Websites that offer a service aimed at testing or enhancing/improving the accessibility of Websites. They analyze the degree of accessibility of a Website or of individual Web pages or assist in adapting a Web page in accordance with the accessibility guidelines mentioned earlier. Some tools even offer a different view of an existing Website better suited to the particular (visual) impairment of the user.

Accessibility Tools can be classified in 3 different categories:

- **Evaluation Tools** perform an analysis of pages or sites with regard to their accessibility. The result of this analysis is usually a report or a rating.
- **Repair tools** assist an author in making a page more accessible once the accessibility shortcomings are identified (through the use of an evaluation tool).
- **Filters and transformation tools** modify a page and are aimed at the user instead of the developer/designer. They are exlained in more detail in section 2.3.1.

There are many accessibility tools. Here are some examples:

Watchfire's Bobby[8] is a desktop evaluation tool aimed at Web masters and Web developers. It tests a Website page by page using the W3C's WAI guidelines and delivers a report as a result. Web masters or designers can then use this report to improve the accessibility of their Website. Bobby is a general tool, i.e. it focuses on all aspects of accessibility, not only those relevant to the visually impaired.
aDesigner[10] is a disability simulator that Web designers can use to test the accessibility of their Website for the visually impaired. aDesigner is a focused tool i.e. it focuses on the visually impaired and not on other disabilities.
A-Prompt[11] is a repair tool developed at the University of Toronto. It repairs Web pages automatically or with the assistance of the developer. Recently the creators of A-Prompt introduced an online accessibility checker[12] as a replacement of A-Prompt. Being Web-based it is more an evaluation tool than a repair tool.

2.3.1 Transcoders

On-line transcoders such as LIFT[13] transform Websites to text-only, eliminating the navigation problem. Others like e.g., {textualise;}[14] provide a transformation proxy service that adjusts the content of a Website before conveying it to the user's Web agent thereby providing a version more suited to the visually impaired. Users can even "tune" their view on certain Websites, enlarging fonts, using contrasting colors, etc...
Though undoubtedly very useful, also transcoders have their limitations, as it is very hard to recover the implicit semantics conveyed in Web pages by means of the visualization.

2.4 Related Work

As a result of the general ineffectiveness of existing screen readers for Web browsing tasks, several specialized Web audio browsers have been developed. The JAWS[15] system and IBM's Home Page Reader[16] e.g. permit hyperlink-based navigation.

[10] See http://www.alphaworks.ibm.com/tech/adesigner (accessed 2006)
[11] See http://aprompt.snow.utoronto.ca/ (accessed 2006)
[12] See http://checker.atrc.utoronto.ca/index.html (accessed 2006)
[13] See http://www.usablenet.com/products_services/text_transcoder/text_transcoder.html (accessed 2005)
[14] See http://aquinas.venus.co.uk/solutions/products/textualise/what.html (accessed 2006)
[15] See http://www.freedomscientific.com/fs_products/JAWS_HQ.asp (accessed 2007)
[16] See http://www-03.ibm.com/able/guidelines/web/webhprtest.html (accessed 2007)

Other systems like HearSay perform structural and semantic analysis on the HTML-documents. The resulting partition trees from this analysis are used to create VoiceXML dialogs which in turn facilitate audio browsing without information overload [6].

Other technologies like Aural Style Sheets[17] should help in writing content for speech-enabled browsers. Aural style sheets resemble regular style sheets with the difference that their markup/code is vocal-oriented. The added advantage here being the separation of the presentation (CSS properties) from the content.

Separating form from content is a school of thought that has proved its worth in general but also in terms of accessibility [7]. Apart from the obvious advantages it offers it also has the potential to at least ease problems of accessibility. When content is clearly distinct from format, it can be presented in numerous ways, including the ones needed or used by the visually impaired. It is, after all, the *medium* that is visual, not the information itself.

Apart from the assistive technology, guidelines and tools discussed above and in previous sections, research is done to tackle the stated VIA-problem[18] in a more fundamental way. Despite the available technologies and tools, the mobility, or ease of travel, of visually impaired Web users is reduced since Web pages are designed primarily for visual interaction and part of the information is only transferred in a visual way. Therefore, in a visually impaired person's environment objects that support travel are missing or inaccessible altogether. Screen readers, unlike sighted users, cannot see the implicit structural and mobility knowledge encoded within the visual presentation of Web pages.
Semantic Web technologies on the other hand allow making the implicit structural and mobility knowledge of a Web page explicit and accessible to screen readers. In this context, Semantic Web technology is not used to convey the semantics of the content, but to convey the structural and mobility properties of Web pages. E.g. Dante[19] is a tool that follows this approach.

Research was done to use WSDM in the context of accessibility for visually impaired users. To achieve this, WSDM was combined with the Dante-approach. Dante identifies objects that support navigation in Web pages. These objects are annotated with terms from an ontology and the resulting mark-up is used to transcode Web pages into a form that is easier to travel for a visually impaired user. The ontology used in this process is the WAfA-ontology.

[17] See http://www.w3.org/TR/REC-CSS2/aural.html (accessed 2007)
[18] VIA: Visually Impaired Awareness – see section 1.2
[19] See http://dante.man.ac.uk/ (accessed 2006)

2.5 Underlying concepts, technologies and approaches

Dante and WSDM, the two prominent approaches in this thesis, both rely on the ontology concept. Let us take a look at this in more detail.

An *Ontology* is a specification of a conceptualization of a knowledge domain; it is a controlled vocabulary that describes objects and the relations between them in a formal way. It has a grammar for using the vocabulary terms to express something meaningful within a specified domain of interest. Less formally defined an ontology is a special kind of language where concepts are defined with respects to specific user groups or knowledge domains. It can be thought of as a structured list of concepts.

Consider e.g. an ontology for describing all sorts of cars. The objects in this Car Ontology could be Steeringwheel, Seat, Gearbox, Engine, etc… Relations between these objects could be “Car has Gearbox” and furthermore Gearbox can have a Type property containing the values “Manual” or “Automatic”. This simplistic example allows describing if a car has an automatic or a manual gearbox.

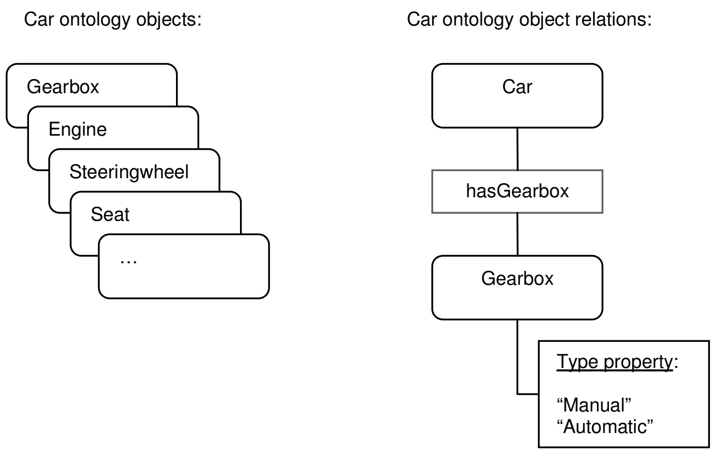


Figure 1 – the Car ontology

An automated system deployed in the automotive context could of course benefit from the usage of this ontology. But it gets really interesting when two systems (even different ones) use the same ontology because then their communication is facilitated by the very ontology usage – it’s so much easier to understand each other if we’re talking about the *same thing using the same language*. The same is true for computers.

A *web ontology language (like OWL[20])* allows defining an ontology. OWL is a language designed for processing information on the Web. It was build to be interpreted by computers, not to be read by people[21].
More formally we could say it is a language and framework for representing ontological knowledge and information about the way that “a world” (in this case the Web) is structured and fits together. In this case the ontology or domain of interest we are reasoning about is in fact the World Wide Web. As such the ontology will contain objects like Websites, pages, links, banners, etc…

[20] See http://www.w3.org/2004/OWL/ (accessed 2007)
[21] See http://www.w3schools.com/rdf/rdf_owl.asp (accessed 2007)

The *Semantic Web* is an enhancement/evolution of the current World Wide Web where information is annotated by means of metadata, making it understandable for machines. This enables e.g., automatic relations between and in documents.
RDF (Resource Description Framework)[22], developed by the W3C is a set of specifications that provide a lightweight ontology system for supporting exchange of knowledge on the Web – in other words RDF is a general-purpose language for representing information on and about the Web. This is one concrete example where the ontology concept is put to use.
Putting the definitions of OWL and RDF next to each other, we can see that the OWL language is actually an extension of RDF.

2.5.1 Dante

Dante is a semi-automated tool that encodes techniques for Web travel support. It focuses on navigation or mobility for visually impaired people, i.e. how they access Web pages and navigate through them.
The Dante-approach takes a Web page, annotates it using the WAfA ontology and then transforms that Web page based on the annotations to enhance the provided mobility support. This process is supported through four steps illustrated in Figure 2.

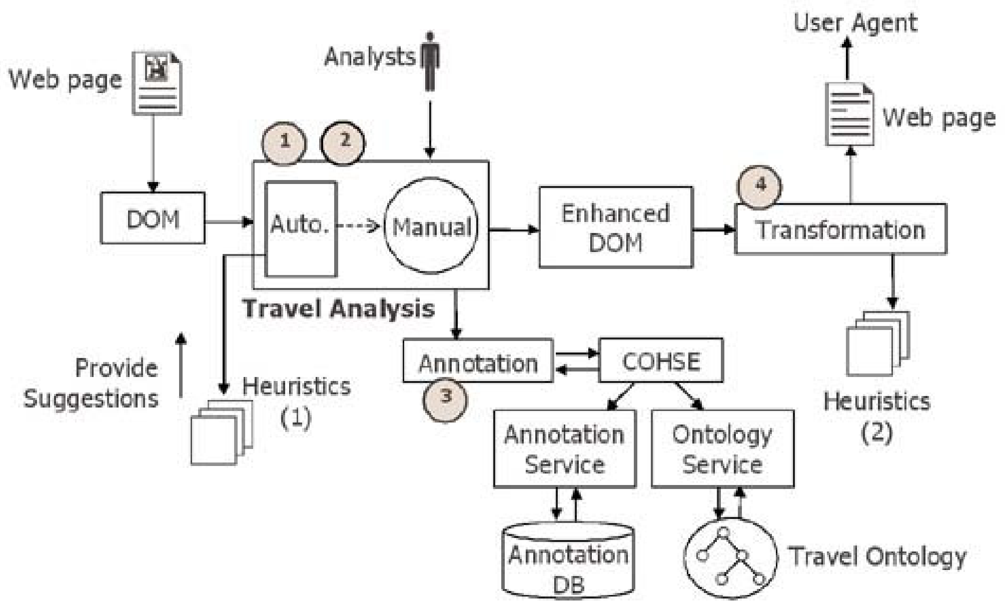


Figure 2 – the Dante approach[23]

The four Dante-steps explained:

1. Analyze Web pages to identify objects that support mobility and travel.
   Examples of such objects include links or menus …
2. Discover their roles.
   E.g. when a link is identified, what is its role? It can be an advertisement in which case it will point to the advertiser's homepage or it can be a breadcrumb, a favorite, part of a menu, etc…

[22] See http://www.w3.org/RDF/ (accessed 2006)
[23] the basic architecture of Dante as found on http://dante.man.ac.uk/index.htm

3. Annotate them with concepts from the WAfA ontology in order to make their roles explicit.

   Once the role of an object is known, i.e. we know to which *class* of objects it belongs, it can be annotated with a term from the WAfA ontology.

4. Transform pages based on the annotations to enhance the provided mobility support.
   This transformation takes as input the annotations of step 3 and the Webpage itself (for the sake of example, suppose in HTML-form). The output is the same Webpage but with different HTML better suited to navigation for visual impaired users.

In the case of the WAfA ontology, the domain of interest is the modeling of structural and navigational organization of Web pages.

Basically Dante makes a Website easier to navigate for blind people or people with reduced vision. Although the approach is promising, the drawback is that the annotation process (steps 1 → 3) must be performed manually. This is very time-consuming and therefore limits its applicability in practice.

2.5.2 WSDM

WSDM is a Web application design methodology developed at the research group WISE of the Vrije Universiteit Brussel. Unlike other such methodologies, it is audience-driven, i.e. it takes as starting point the needs and requirements of the intended audience and creates a Web structure based on this. The result is a Website that is better tailored to the user's needs.
Figure 3 gives an overview of the WSDM design process.

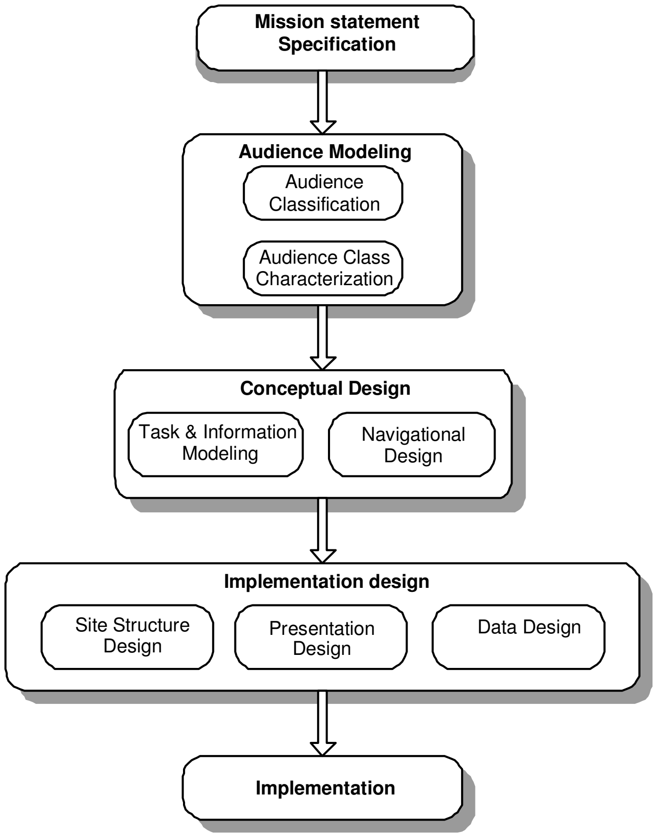


Figure 3 – the WSDM design process

The design process involves 5 steps:

1. **Mission statement Specification.**

   Here we define the purpose of the site and the site's main subject. Also the question what the target audiences are is answered here.
   Consider the following mission statement for a University Example: "*Provide general information about the available programs to attract more students and enhance the internal communication between students and lecturers by providing detailed information about programs and courses*".

2. **Audience Modeling.**

   As a first step, an Audience Class Hierarchy is built based on the target audiences identified in the mission statement. From each audience class, a set of Information requirements, Functional requirements and Usability requirements is derived. In the University Example consider the following Audience Class Hierarchy:

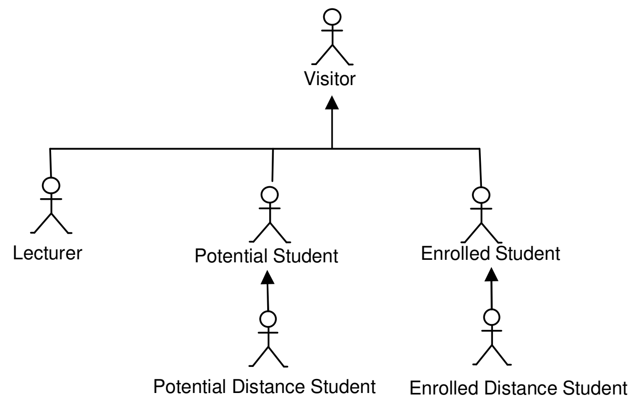


Figure 4 – university audience class hierarchy

Yielding as audience classes *Potential Students*, *Enrolled Students* and *Lecturers*; all subclasses of the class *Visitor.*

3. **Conceptual Design.**

The conceptual design step consists of 2 substeps:

i. Task & Information modeling
ii. Navigational design:

Task modeling & Information modeling *(or the conceptual "what")*

The information requirements of different audience classes are translated into so-called object chunks through tasks. First, for every information or functional requirement (resulting from the Audience Modeling step) a task must be defined. Each task needs to be broken down into elementary tasks, and for every elementary task a corresponding object chunk is created.
So Object chunks model information and/or functionality required by tasks. All object chunks combined together form the business object model (example Figure 5).

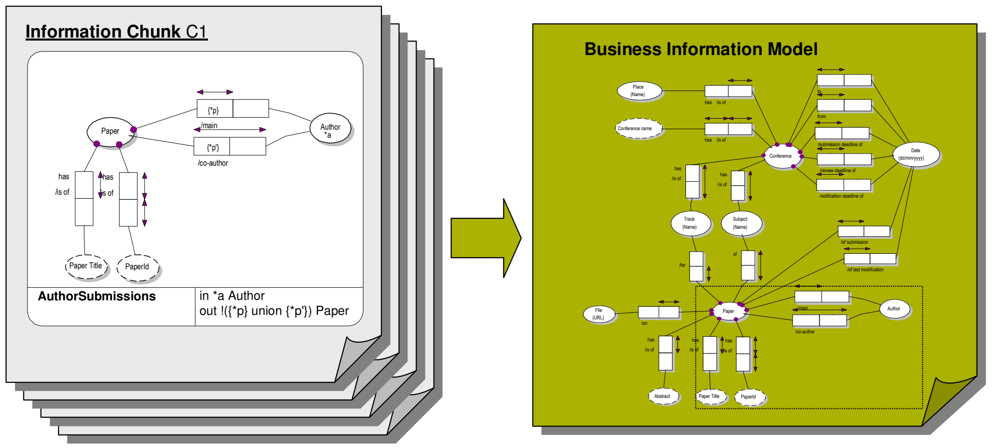


Figure 5 – Object Chunks and the Business Information Model

Navigational design *(or the conceptual "how")*

The navigational design models the conceptual structure of the Web site resulting in the so called Conceptual Structural Model. This model is a collection of navigation tracks where each navigation track consists of components and links. Going back to the same university example the main Web site structure looks as follows (Figure 6):

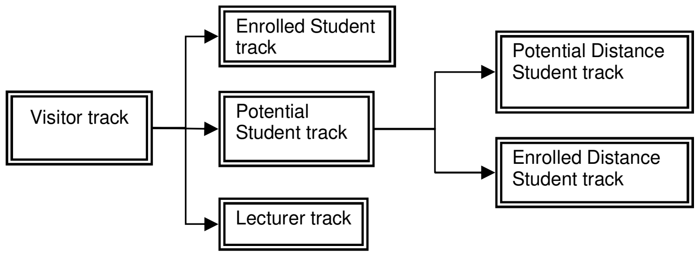


Figure 6 – Navigational Design

For every task (resulting from the Task Modeling & Information Modeling step) corresponding to an audience class, a task navigation model should be created (Figure 7).

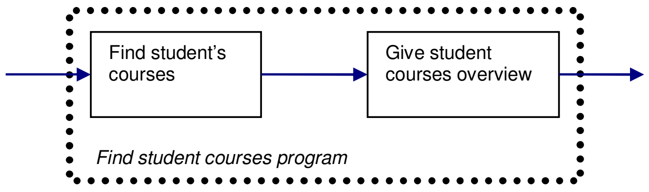


Figure 7 – Task Navigation Model

Each task navigation model consists of components and links.
Each elementary task corresponds with a component. But the *Task & Information Modeling step* also connected tasks to their corresponding object chunks. This allows us to link object chunks with their components.

4. **Implementation Design.**

The Implementation design consists of three different substeps:

i. Site structure design
ii. Presentation design
iii. Data design

The ***site structure design*** groups information into pages: Starting from the navigation model, the conceptual structure is translated into pages with the default setup being one component plus several links on one page.

The ***presentation design*** specifies the look and feel of the Website. This can be done e.g. through usage of templates (to base page layout on) in combination with CSS.

The ***data design*** comprises the definition of the data source structure – it is this step that defines the actual datastore (which does not have to be an actual database like e.g. Oracle or MySQL – it can also be achieved applying technologies like XML DTD, RDF definitions, etc…) with the business object model acting as the conceptual schema.

5. **Implementation.**

   At his point the actual (HTML) pages can be built. This can be performed manually or through some form of automated process.

Note that WSDM makes a distinction between the conceptual design and the design of the actual presentation. This effectively decouples the design and the technical implementation details/limitations and makes WSDM suitable for all types of Web applications.

3 Accessibility Evaluation of the WISE Website

To obtain a better insight in the problems related with the design of accessible Websites, we decided to first exam a Website that was developed in a systematic way (i.e. using a Web design method) but not developed with accessibility in mind. We wanted to detect what kind of accessibility problems such a Website would have. The WISE Website[24] is used for this purpose.

Evaluating a Website's accessibility is harder than it may seem. It is more than simply taking a look at the site and concluding it is ok or not. Nor is it sufficient to walk through the HTML-code to identify possible accessibility issues. Evaluating a Website and, more to the point, the WISE Website means checking if all accessibility guidelines are followed.
Evaluating the accessibility of a Website requires a systematic approach. One such approach, the *preliminary review*, is proposed by the W3C[25]. We explain the approach in the next sub section.

3.1 Preliminary Review

A preliminary review is a "quick way" to identify some accessibility problems on a Website. It combines manual checking of representative pages, along with the use of several semi-automatic accessibility evaluation tools. Reviewers do not need to know Web mark-up languages (hence, no extensive HTML-knowledge is required), but should be able to download software and familiarize themselves with some evaluation tools, and change certain settings on their browser. To conduct a preliminary review, the following tasks need to be completed:

1. Select representative page samples.
2. Examine pages using a graphical browser.
3. Examine pages using a specialized browser.
4. Use automated Web evaluation tools.
5. Summarize the results.

3.1.1 Selecting representative page samples

From the Website to be reviewed, in our case the WISE Website, a number of representative sampling of pages should be selected. The sampling pages should match the following criteria:

- Include all pages on which people are more likely to enter the site ("index.html", "default.html", etc.).
- Include a variety of pages with different layouts and functionality, for example:
    - Web pages with tables, forms, or dynamically generated results;
    - Web pages with informative images such as diagrams or graphs;
    - Web pages with scripts or applications that perform functionality.

3.1.2 Examining pages using a graphical browser

The pages selected must be examined using a graphical browser (such as Mozilla Firefox, Internet Explorer, Netscape Navigator/ Mozilla, Opera, Safari, or others). While examining the selection of pages some settings in browser or operating system should be adjusted as follows (some of these manual checks may require additional software):

- Turn off images, and check whether appropriate alternative text for the images is available.
- Turn off sound, and check whether audio content is still available through text equivalents. The WISE Website has no sound. Therefore this will not be included in our test.
- Use browser controls to vary font-size: verify that the font size changes on the screen accordingly; and that the page is still usable at larger font sizes.
- Test with different screen resolution, and/or by resizing the application window to less than maximum, to verify that horizontal scrolling is not required (caution: test with different

[24] See http://wise.vub.ac.be (accessed 2006)
[25] See http://www.w3.org/WAI/eval/preliminary.html (accessed 2006)

browsers, or examine code for absolute sizing, to ensure that it is a content problem and not a browser problem).

- Change the display color to gray scale (or print out page in gray scale or black and white) and observe whether the color contrast is adequate.
- Without using the mouse, use the keyboard to navigate through the links and form controls on a page (for example, using the "*Tab*" key), making sure that you can access all links and form controls, and that the links clearly indicate what they lead to.

3.1.3 Examining pages using specialized browsers

Next a voice browser (such as Home Page Reader[26]) or a text browser (such as Lynx[27]) should be used and the selection of pages should be examined while finding the answer to the following questions:

- Is equivalent information available through the voice or text browser as is available through the GUI browser?
- Is the information presented in a meaningful order if read serially?

3.1.4 Using automated Web evaluation tools

At least two automated Web accessibility evaluation tools should be considered to analyze the selection of pages. Note that these tools will only check the accessibility aspects that can be tested automatically, the results from these tools should not be used to determine a conformance level without further manual testing.

3.1.5 Summarize the results

Using the results obtained from the previous four tasks, the types of problems encountered, as well as positive aspects that should be continued or expanded on the site should be summarized. Indicate the method by which problems were identified. Follow-up steps, including full conformance evaluation which includes validation of markup and other tests, should be recommended as well as ways to address any problem identified.

[26] See http://www.soundlinks.com/hprgen.htm
[27] See http://lynx.browser.org/

3.2 Evaluation Results

The preliminary review method has been applied to the WISE Website.

3.2.1 Selecting representative page samples

From the WISE Website, the following pages were chosen as representative sample. Each page is followed by a brief explanation why it is chosen.

http://wise.vub.ac.be/default.htm: default page, the page most users will enter the site through;
http://wise.vub.ac.be/members/ : this page contains some pictures;
http://wise.vub.ac.be/researchers/researchtopics.html: of technical interest, so probably lots of hits;
http://wise.vub.ac.be/researchers/publications.php; php script, which means it is a dynamically generated page;
http://wise.vub.ac.be/students/proposals.html: popular page because of its content;
http://wise.vub.ac.be/students/courses.htm: popular page because of its content;
http://wise.vub.ac.be/metovr/default.htm: the default startpage for the MeTo-VR subsite;
http://wise.vub.ac.be/webmaster.html: several fields to fill out – different structure than the previous pages.

3.2.2 Examine pages using a graphical browser

Three different browsers were chosen: two based on an MS Windows platform and one based on a Linux platform. The combination of MS Internet Explorer and MS Windows was chosen because it will obviously be the most widely used combination. Versions are 6.0 for IE and XP for Windows. Through lack of time it was simply not feasible to test other version combinations like e.g. IE 5.0/Windows 2000 as this would lead us too far off track.
An alternative for MS Internet Explorer was needed on the MS Windows platform for people who cannot or will not use this combination. This alternative was Opera (v8.51); Opera is free at the time of printing.
Lastly an alternative for the MS Windows operating system was needed, so on the basis of popularity and cost Linux was an obvious choice. It is the most popular *free* alternative for Windows (for personal use). For this platform, Mozilla 1.1 was chosen as browser on a Mandrake Linux distribution.

The results of this test can be found in Appendices

Formatted: Bullets and Numbering

Appendix A: Browser Tests.

We can summarize the results as follows. In general, the site didn't produce dramatically different results over the three tested browsers. The font resizing proved the most "troublesome" as some elements didn't resize consistently over all three browsers. Textual descriptions of images are in some cases lacking and when reducing the color set (b/w) it becomes hard to distinguish the clickable hyperlinks because of the subtle grayscale difference.

3.2.3 Examining pages using specialized browsers

As specialized browser lynx was used on a Linux platform. The URL of the default page was entered at startup but all other pages were accessed via the Lynx browser.

On the http://wise.vub.ac.be/members/ page only Prof. Dr. Olga De Troyer and the former members at the bottom of the page had a text indicating the file name of a picture that was supposed to show. This was not the case for the others, so this is an inconsistency.
On the http://wise.vub.ac.be/webmaster.html page all fields are accessible.

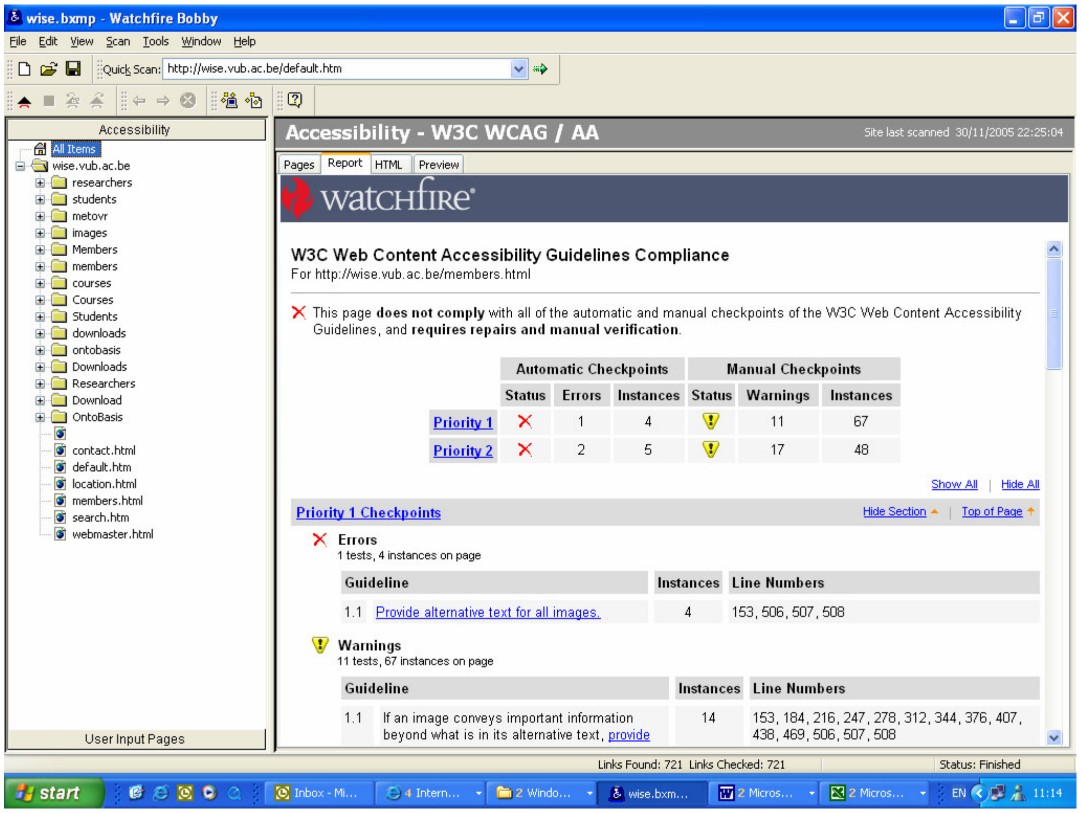
In summary it can be concluded that all features[28] are accessible, the links are ok and the order in which they are traversed is logical.

[28] Though in fairness it must be said that the content of the WISE-site is mainly text-based and thus better suited for this kind of “browsing”.

3.2.4 Use of automated Web evaluation tools

Our first tool of choice for automatic Web evaluation is the desktop version of Watchfire's Bobby (v5.30.4.16). It was chosen because of its popularity and the availability of a free (albeit time-limited) trial version that can be installed locally.
Bobby analyses individual Web pages starting from a designated starting page and following (and analyzing) all the links it encounters. It can be tuned to limit the amount of links to be followed/checked. In all, Bobby checked 721 pages but in this thesis we shall limit our findings to the 8 sample pages.
Each page is checked for Priority 1 and Priority 2 issues. Bobby reports errors or warnings. An error is something that can automatically be checked and that needs to be corrected. A warning is something that needs to be checked manually and may need correction (but that is not necessarily the case). Priority 3 is not tested in this free trial version.

A typical Bobby Report is given below.



Accessibility - W3C WCAG / AA

Site last scanned 30/11/2005 22:25:04

W3C Web Content Accessibility Guidelines Compliance
For http://wise.vub.ac.be/members.html

✗ This page **does not comply** with all of the automatic and manual checkpoints of the W3C Web Content Accessibility Guidelines, and **requires repairs and manual verification**.

	Automatic Checkpoints			Manual Checkpoints		
	Status	Errors	Instances	Status	Warnings	Instances
Priority 1	✗	1	4	!	11	67
Priority 2	✗	2	5	!	17	48

Priority 1 Checkpoints

✗ Errors
1 tests, 4 instances on page

Guideline		Instances	Line Numbers
1.1	Provide alternative text for all images.	4	153, 506, 507, 508

! Warnings
11 tests, 67 instances on page

Guideline		Instances	Line Numbers
1.1	If an image conveys important information beyond what is in its alternative text, provide	14	153, 184, 216, 247, 278, 312, 344, 376, 407, 438, 469, 506, 507, 508

Links Found: 721 Links Checked: 721 Status: Finished

Figure 8 - a typical Bobby report

In all, out of our 8 sample pages, two times a Priority 1 error was encountered. Each time the error was the same:

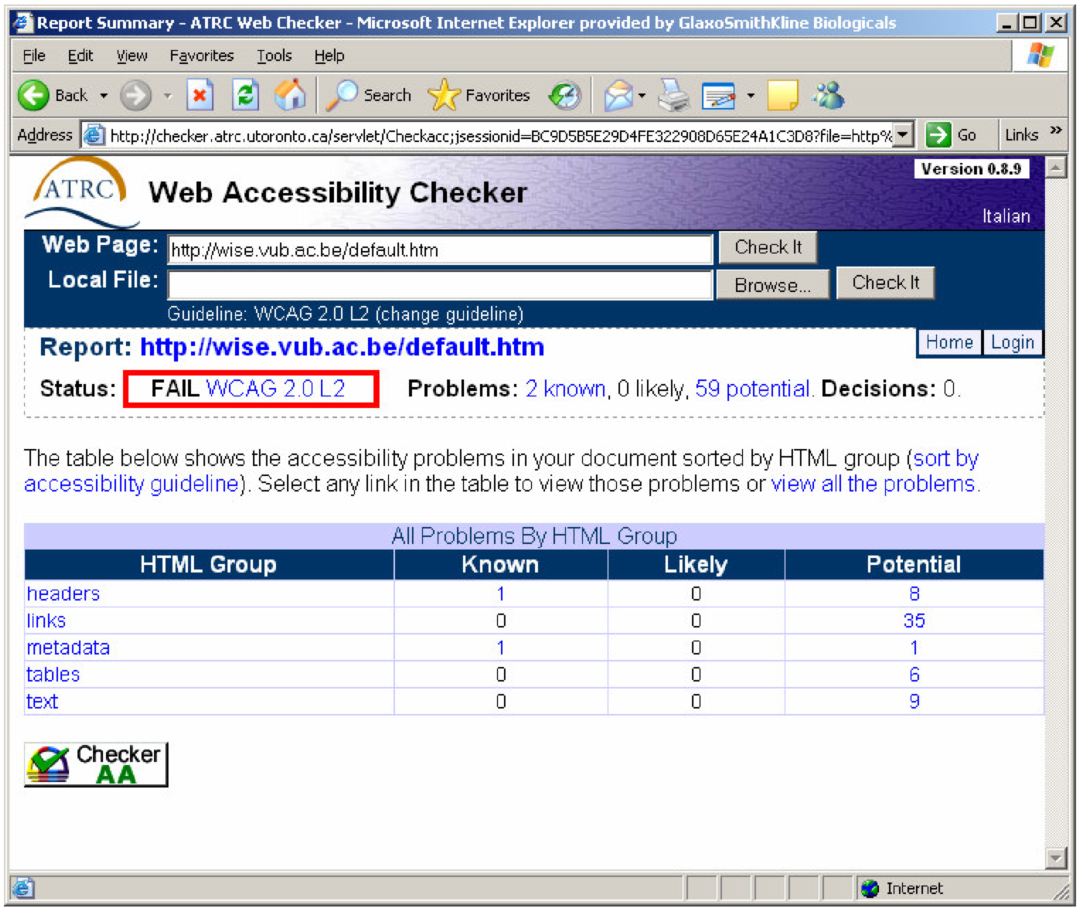
- No alternative text for an image. All images should contain a short alternative text description that represents the function of the graphic excepting e.g. bullets, spacers (where this would be distracting or unnecessary). Computers cannot interpret images and present them in a meaningful alternative format. Alternative text gives the computer something to present to the user.

Regarding Priority 2, eighteen errors were encountered. The most common error types were:

- Use relative size & positioning rather than absolute. Absolute sizes are units like pixels, points, etc… - they are relative to an absolute measurement and cannot be scaled. Relative units like % or ems are automatically scaled when the base unit is scaled, allowing text to change size or page layout to flow without running off the edge of the screen.
- Nest headings properly. E.g. H1 followed by H2 instead of H1 followed by H3. Some access aids extract the headings to create an outline of the page. Incorrect nesting will result in an incorrect outline structure which may disorient users.
- Same link phrase used more than once (on same page and points to different URL). If more than one link on a page shares the same link text, all those links should point to the same resource. This aids Web design as well as accessibility.
- Associate FORM controls & labels explicitly with the LABEL-element. For each FORM-control, place its label in a LABEL-element so the label's text gets associated explicitly with the form control. This allows a browser to tell the user definitively which label applies to the given control. E.g. clicking on the label positions the cursor in the form field or toggles the value of radio buttons or check boxes. Besides being intuitive it also provides a bigger target for the mouse thus aiding the visually impaired user.

Our second tool of choice for automated Web evaluation is the online version of ATRC's Web Accessibility Checker[29]. This is a free service provided by the Adaptive Technology Resource Center (ATRC) at the University Of Toronto[30].
Usage of the Web Accessibility Checker is simple: enter the URL of the Web page that needs checking, and click the "Check It"-button. Unlike Bobby, checks are performed on a page-by-page basis, and each page's URL needs to be entered manually which could be a problem for large Websites.
ATRC's Web Accessibility Checker features the choise of different guidelines. All checks in this paper were performed first with the WCAG 1.0 (Level AA) and second with the WCAG 2.0 L2 guidelines.
The check produces a list of *known*, *likely* and *potential* problems. *Known* problems necessitate a Web page change to solve, *likely* problems will probably (but not necessarily) result in Web page change and *potential* problems might be solved without having to adapt the Web page. We will focus mostly on the *known* problems.

A typical ATRC Web Accessibility Checker Report is given below.


Report Summary - ATRC Web Checker - Microsoft Internet Explorer provided by GlaxoSmithKline Biologicals
File Edit View Favorites Tools Help
Back Search Favorites
Address http://checker.atrc.utoronto.ca/servlet/Checkacc;jsessionid=BC9D5B5E29D4FE322908D65E24A1C3D8?file=http%
Go Links
Internet

ATRC Web Accessibility Checker Version 0.8.9 Italian

Web Page: http://wise.vub.ac.be/default.htm Check It

Local File: Browse... Check It

Guideline: WCAG 2.0 L2 (change guideline)

Home Login

Report: http://wise.vub.ac.be/default.htm

Status: FAIL WCAG 2.0 L2 **Problems:** 2 known, 0 likely, 59 potential. **Decisions:** 0.

The table below shows the accessibility problems in your document sorted by HTML group (sort by accessibility guideline). Select any link in the table to view those problems or view all the problems.

All Problems By HTML Group

HTML Group	Known	Likely	Potential
headers	1	0	8
links	0	0	35
metadata	1	0	1
tables	0	0	6
text	0	0	9

Checker AA

Figure 9 - a typical ATRC report

To summarize, he following errors were encountered on the 8 sample pages.

[29] See http://checker.atrc.utoronto.ca/servlet/Submit (accessed 2007)
[30] See http://www.utoronto.ca/ (accessed 2007)

- No alternative text for an image. Computers cannot interpret images and present them in a meaningful alternative format. Alternative text gives the computer something to present to the user.
- Make sure text content is readable and understandable: the document language is not identified. The lang attribute serves to specify the base language of an element's attribute values and text content[31].
- Make sure the information & structure can be separated from presentation. Header nesting was found to be inconsistent (e.g. H1 element not followed by an H2 element, etc. …).
- Form controls & labels are not associated explicitly. E.g. text boxes, radio buttons etc. need labels explicitly to them. A browser can thus tell the user definitively which label applies to the given control.
- Provide mechanisms to help users find content & navigate through it – anchor element <a> was found that contains no text. Each source anchor must contain text.
- Possible misuse of <p>-element: if all the text in the paragraph is marked with a presentational element, text might be better marked as header.

In conclusion, both tools although different in usage yield very much the same results. Given the fact that these results only relate to the 8 tested sample pages, it is the author's opinion that part or all of the WISE Website pages would benefit from a Priority 1 and 2 overhaul in terms of accessibility. The need to follow the new university Website layout-style does present limitations however so not all issues that arose may be addressed.
Color contrast is inadequate with regards to hyperlinks: the university's house Style does not underline hyperlinks unless the mouse is positioned directly over them. Their color is a light green/dark yellow which makes it difficult to distinguish the clickable hyperlinks in a b/w environment. We recommend to change the color of the hyperlinks to a more contrasting tint of green/yellow.

[31] See http://www.w3.org/TR/html4/struct/dirlang.html (accessed 2007)

4 Extending WSDM

This chapter explains how the WSDM methodology is extended in combination with the Dante approach and what this extension means for the combined WSDM/Dante process as a whole.
Some basic proposition logic understanding is assumed.

4.1 WSDM & Dante combined

WSDM is a Website design methodology with an alternative approach (see earlier) whilst Dante focuses on the accessibility of web pages for Visual impaired users. Combining both approaches could result in a rather unique Web site design methodology taking the user and his requirements as starting point for the design and ending with a Web site that is better suited for visually impaired web travelers. Combining two approaches that at first glance have very few things in common is no easy feat.
To accomplish this, we look at both approaches from an ontology point of view. In the course of the WSDM design process a lot of ontology concepts, items and objects result from the different design steps. Dante starts with manually identifying certain objects – in concreto this means one or more persons have to review an existing Web site, taking note of any objects that are of interest to Dante and entering them in some form into the WAfA ontology. These WAfA annotations are then used as input for the transformation of the web pages as the final step in the Dante approach.
It was noted that a lot of the objects entered in the WAfA ontology resemble those existing in the WSDM ontology which indicated some form of similarity.

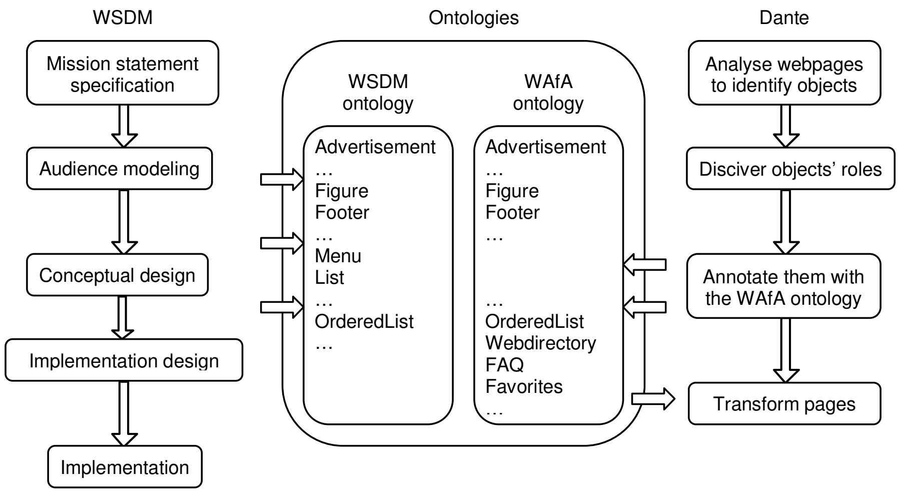

WSDM
Ontologies
Dante
Mission statement specification
Audience modeling
Conceptual design
Implementation design
Implementation
WSDM ontology
WAfA ontology
Advertisement … Figure Footer … Menu List … OrderedList …
Advertisement … Figure Footer … … OrderedList Webdirectory FAQ Favorites …
Analyse webpages to identify objects
Disciver objects' roles
Annotate them with the WAfA ontology
Transform pages

Figure 10

After taking a closer look at these similarities, a mapping was conceived starting from the WSDM ontology objects and projecting them onto the WAfA ontology objects.
So combining the WSDM and Dante approach starts primarily with the definition of a mapping between their respective ontologies. In this way objects from the WSDM ontology can be translated into objects in the WAfA ontology. The WAfA ontology still functions as an input for the Dante transformation step but is now populated by the mapping of objects from the WSDM ontology. This means the first 3 steps of the Dante approach can be discarded as shown in Figure 11.

The bottom line is the ability to automatically generate WAfA annotations whilst applying the WSDM methodology.

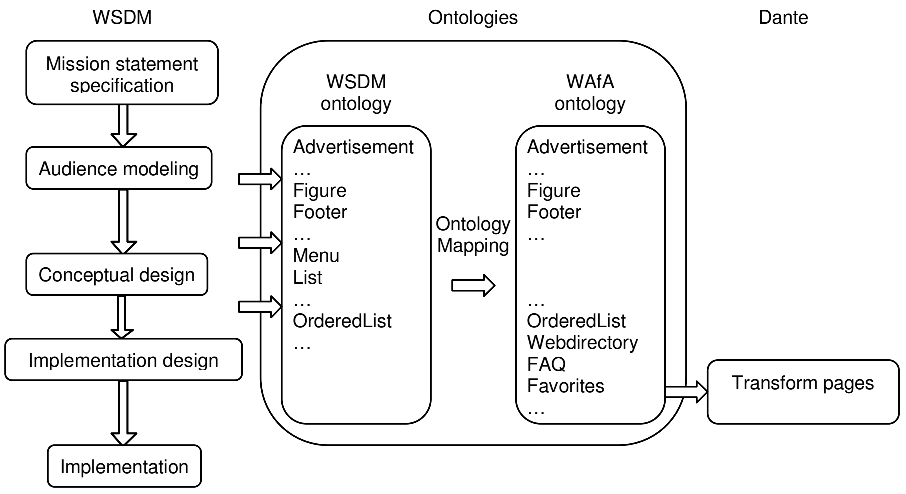

WSDM
Ontologies
Dante
Mission statement specification
Audience modeling
Conceptual design
Implementation design
Implementation
WSDM ontology
Advertisement … Figure Footer … Menu List … OrderedList …
Ontology Mapping
WAfA ontology
Advertisement … Figure Footer … … OrderedList Webdirectory FAQ Favorites …
Transform pages

Figure 11

As shown in the resulting WSDM + Dante methodology (see Figure 12) the resulting combined approach consists of 6 consecutive steps. The advantage here is the automation of the Dante annotation process and achieving this almost without additional effort from the designer (as a side-effect so to speak).

However, this combination is not yet 100% waterproof. As of yet, the existing mappings from WSDM to WAfA [1] cover about 69% of WAfA concepts, so additional work is needed to raise this percentage and attain a higher degree of “coverage” and thus automation.

How this is achieved is explained in the following sections.

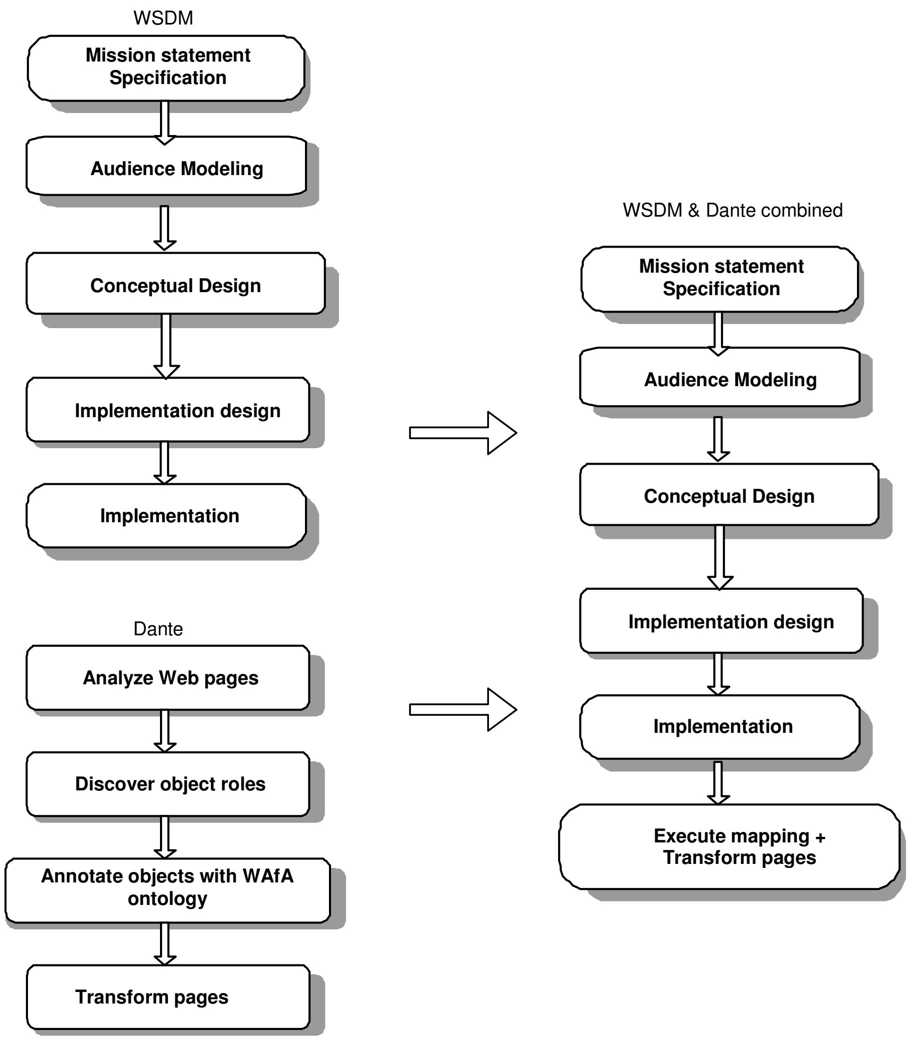

WSDM
Mission statement Specification
Audience Modeling
Conceptual Design
Implementation design
Implementation
Dante
Analyze Web pages
Discover object roles
Annotate objects with WAfA ontology
Transform pages
WSDM & Dante combined
Mission statement Specification
Audience Modeling
Conceptual Design
Implementation design
Implementation
Execute mapping + Transform pages

Figure 12

4.2 The combined WSDM-Dante methodology - Mapping rules explained

Section 2.5.2 explains what WSDM stands for and how it works. To summarize briefly we can state it is a Web Site design methodology developed at the Vrije Universiteit Brussel, consisting of 5 distinct phases:

1. Mission statement specification
2. Audience modeling
3. Conceptual design
4. Implementation design
5. Implementation.

WSDM also uses the ontology concept. The WSDM ontology was especially created to store the information collected during the design process.

Section 2.5.1 explains what Dante stands for and how it works. Dante is an approach consisting of four steps:

1. Analyze Web pages to identify objects that support mobility and travel.
2. Discover their roles.
3. Annotate them with the WAfA ontology in order to make their roles explicit.
4. Transform pages based on the annotations to enhance the provided mobility support.

Dante also relies heavily on the ontology concept; the WAfA ontology (formerly known as the Travel ontology) is used as an integral part of the annotation (step 3) process.

A further step (Section 4.1) combined these two methods by means of their ontologies: A mapping was conceived to map WSDM ontology objects onto WAfA ontology objects.

Now, let us take a closer look at how this concept works:

One-to-one relation/mapping

Some objects exist in the WAfA ontology and in the WSDM ontology under the same name and meaning the same thing. This means a simple one-to-one relation is possible between the WSDM ontology object and the WAfA ontology object.
Take for example the Advertisement object. This exists in both WAfA and WSDM ontologies under exactly the same name. In a formal definition using first-order predicate logic, this one-to-one relation is translated into a mapping rule as follows:

```
∀ i ∈ I: wsdm:Advertisement(i) → wafa:Advertisement(i)
```

Where *I* stands for the set of all instances of the collection of all WSDM modeling concepts for a Web site [1].
This rule means the following: if we have an Advertisement object (or instance would be more correct here) in the WSDM ontology, the rule maps it to its corresponding WAfA ontology object (again, instance applies better than object).
In practice, as a result of the WSDM design process, we would have a collection of objects (or instances) in the WSDM ontology. Then, after building the Web site, in stead of performing the first 3

Dante steps manually (Analyze Web pages, Discover object roles & Annotate objects with WAfA ontology), we see that this information is already present, not in the Dante-usable form (meaning not as instances of WAfA-objects) but as instances of WSDM ontology objects. So in order to get the information in a Dante-usable form, the WAfA ontology instances are created starting from the WSDM ontology instances through the execution of the before-mentioned mapping rules.
Coming back to our example, this means the WAfA ontology Advertisement instance (`wafa:Advertisement(i)`) is created starting from the WSDM ontology Advertisement instance (`wsdm:Advertisement(i)`). This WAfA instance is now readily available for the fourth and final step of the Dante process (Transform pages), completing the combined methodology.

More complex relation/mapping.

Not all objects in the WSDM ontology have a WAfA counterpart. Nevertheless a mapping rule can still be defined provided we get a bit more "creative". Consider the following example: in the WAfA ontology several symbol separators are defined:

- CommaSeparator
- DashSeparator
- TriangleLeftSeparator
- TriangleRightSeparator
- VerticalBarSeparator

They have no immediate WSDM ontology counterpart. However, the WSDM ontology does have the CustomSeparator as an object. If this CustomSeparator were to be represented by a string having the same value as the value the corresponding WAfA ontology object represents, they would both mean the same thing and these "conditions" could be combined into a mapping rule.
Taking the TriangleLeftSeparator as an example from the WAfA ontology, we would need a WSDM ontology CustomSeparator object represented by a "<" to have a correspondence. This is exactly what the following mapping rule represents:

```
∀ i ∈ I, ∃ x ∈ I: wsdm:CustomSeparator(i) ^ wsdm:representedBy(i, x) ^
  wsdm:String(x) ^ wsdm:hasValue(x, "<") → wafa:TriangleLeftSeparator(i)
```

Although a bit more complex than the one-to-one mapping no "special magic" is needed. In fact, with these two kinds of mapping we were able to generate about 70% of the relevant WAfA ontology objects starting from our existing WSDM ontology.

The complete list of mappings can be found in Appendix B: Mapping Rules between WSDM and WAfA.

4.3 The combined WSDM-Dante methodology – Extending WSDM

At the time of writing there are 84 WAfA ontology concepts that are subject of this investigation. 3 WAfA concepts are abstract classes however (*Object, AuthoringConcept, Atom*), meaning they do not need to be mapped. This leaves us with 81 concepts to take into account. The mapping rules explained before account for 56 of those yielding coverage of 69%.

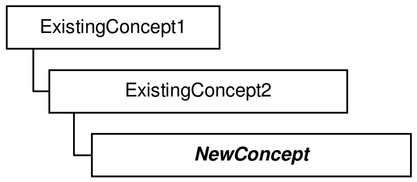
This leaves us the following 25 WAfA ontology Concepts that do not have a WSDM ontology counterpart/mapping:

1. OrderedList
2. UnorderedList
3. AttributeBreadcrumb
4. LocationBreadCrumb
5. PathBreadCrumb
6. SpecialGraphic
7. Skiplink
8. Bookmark
9. Favorites
10. ShoppingCart
11. Index
12. SiteIndex
13. FAQ
14. Note
15. Citation
16. NB
17. PS
18. Abstract
19. PageSummary
20. SiteSummary
21. SearchEngine
22. HistoryList
23. DataTable
24. LayoutTable
25. Headline

“Extending WSDM” to also support these 25 remaining concepts means adding classes to the WSDM ontology, creating new mapping rules for these newly-added classes and specifying what part of the WSDM design process is impacted and how. We shall do this for the WAfA ontology objects specified above.

Typographical conventions:

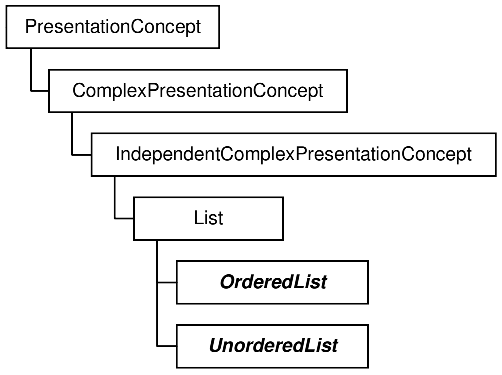
When adding concepts to the WSDM ontology, existing concepts will also be shown to clearly demonstrate where in the WSDM ontology concept hierarchy the new concepts are added. These new concepts will have a typeface of *italics* and **bold**. This is illustrated by the following figure:



4.3.1 OrderedList & UnorderedList

Ontology extension:

We add the two objects (classes is a more appropriate term here and we will use it henceforth) OrderedList & UnorderedList as a subtype of the existing List object (again, class is a better term here). Since every list is either ordered or unordered, we could make the list an abstract class, forcing the use of either OrderedList or UnorderedList instead. This however would impact the existing mapping rules. Furthermore there are cases where nothing about the order in the list is known. It is exactly for these cases that the List is retained as a concrete class.


PresentationConcept
ComplexPresentationConcept
IndependentComplexPresentationConcept
List
OrderedList
UnorderedList

New mapping rules:

```
∀ i ∈ I: wsdm:OrderedList(i) → wafa:OrderedList(i)
∀ i ∈ I: wsdm:UnorderedList(i) → wafa:UnorderedList(i)
```

Methodology impact:

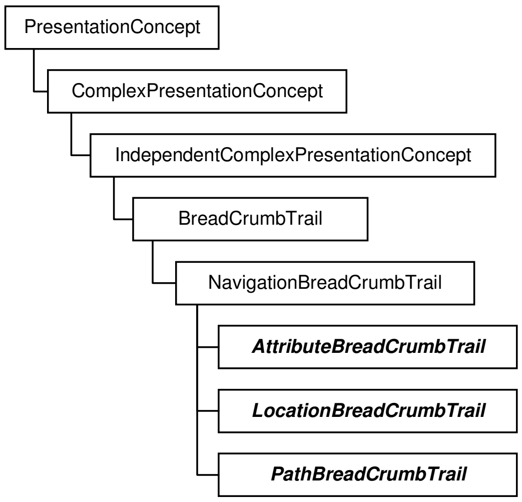
In the Implementation Design, the Presentation Design is impacted. The List class is a non-basic presentation concept [10]. Now however, when adding lists to the design, the designer has the added option of adding ordered or unordered lists. If nothing is known about the list ordering, the existing list concept can still be used.

4.3.2 AttributeBreadcrumb, LocationBreadCrumb & PathBreadCrumb

Ontology extension:

In the WAfA ontology, these three concepts are subtypes of the NavigationalBreadCrumbTrail concept, a concept that also exists in the WSDM ontology. As such it is a bit strange that the suffix "Trail" is missing in all three concept names since their class inheritance and their respective (WAfA ontology) definitions clearly state they are in fact breadcrumb trails. To illustrate this, take e.g. the WAfA ontology definition of the LocationBreadCrumb concept: "Conveys the position of the current page within the site hierarchy. A page has the same breadcrumb trail, no matter how users get there." – this is clearly a LocationBreadCrumb*Trail*.
So our first recommendation, surprisingly, would be to alter the WAfA ontology to add the suffix "Trail" to all three concept names. To avoid confusion, we will refer to the WAfA ontology concepts AttributeBreadCrumb, LocationBreadCrumb and PathBreadCrumb as AttributeBreadCrumb[Trail], LocationBreadCrumb[Trail] and PathBreadCrumb[Trail].

Coming back to WSDM, we add the three classes AttributeBreadCrumbTrail, LocationBreadCrumbTrail & PathBreadCrumbTrail as a subclass of the existing NavigationBreadCrumbTrail class thereby replicating the 4 concepts' WAfA ontology hierarchy in the WSDM ontology.
The PresentationConcept is chosen over the NavigationConcept because in most cases, breadcrumbs are used for reasons of clarity or explanatory nature in stead of pure navigation. Breadcrumbs don't even *have* to be clickable hyperlinks. But even when they are, they seldom form an integral part of a site's navigation design.



New mapping rules:

```
∀ i ∈ I: wsdm:AttributeBreadcrumbTrail(i) → wafa:AttributeBreadcrumb[Trail](i)
∀ i ∈ I: wsdm:LocationBreadcrumbTrail(i) → wafa:LocationBreadcrumb[Trail](i)
∀ i ∈ I: wsdm:PathBreadcrumbTrail(i) → wafa:PathBreadcrumb[Trail](i)
```

Methodology impact:

In the Implementation Design, the Presentation Design is impacted. These three new types of non-basic presentation concepts [10] offer new BreadCrumbTrail options to the designer to choose from, while retaining the existing NavigationBreadCrumbTrail.

4.3.3 SpecialGraphic

Ontology extension:

This is an abstract class in WAfA meaning it is not instantiated. Therefore it does not need a WSDM ontology counterpart and hence no mapping.

New mapping rules:

N/A.

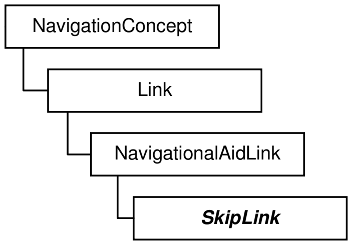
Methodology impact:

N/A.

4.3.4 SkipLink

<u>Ontology extension:</u>

In the WSDM ontology, we add the class SkipLink as a subtype of the existing NavigationalAidLink class.



<u>New mapping rules:</u>

```
    ∀ i ∊ I: wsdm:SkipLink(i) → wafa:SkipLink(i)
```

<u>Methodology impact:</u>

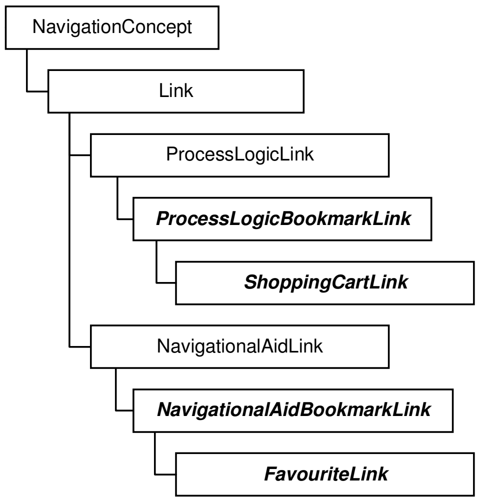
In the WAfA ontology, a SkipLink is defined as follows: "A link that enables users, in particular visually impaired users, to avoid certain areas that are considered as obstacles or not of interest. For example, a skip link is usually provided to skip a header or sidebar. Since it provides movement within the page, it is also a child of the Intra concept."
In the Conceptual Design, the Navigational Design is impacted [10]. This new type of link is something that is especially geared towards *aiding* the *navigation* of visually impaired users. Therefore it is considered a NavigationalAid link and should be used in this context (e.g. for skipping recurring headers as stated in the WAfA ontology definition).
As far as WSDM is concerned, it might be considered a drawback to introduce a concept that has little use outside the visual impaired "sphere of interest". The designer should be (made) aware of the fact that this link needs to be introduced manually and thus does not get created automatically.
A possible solution might be to introduce SkipLinks as a standard feature in e.g. headers or rather the WSDM ontology Header concept since SkipLinks will be very often used in this way.

4.3.5 Bookmark, Favourites, ShoppingCart

<u>Ontology extension:</u>

When looking at the WAfA ontology definition, Favourites and ShoppingCart are two "special cases" of a Bookmark collection of links. In [10] the example of the Shopping Cart is taken as a suspend-resume link which in turn is a special case of a process logic link. E.g. a user might *suspend* his buying activity temporarily to check out his shopping cart or basket, to *resume* his shopping activities afterwards. Hence the type suspend-resume link. It is obvious the ShoppingCart should be created as a ProcessLogicLink subtype.
Favourites however is something different, this is more often used as navigational aid, thereby making it a candidate for a NavigationalAidLink subtype. The problem here is we lose the WAfA ontology grouping of the 3 concepts Bookmark, Favourites and ShoppingCart.

This is solved by creating two types of bookmark: one used for process logic purposes with ShoppingCart as a subtype or subclass and one used for navigational aid purposes with the Favourites as subclass.
One last issue remains: note that the WAfA concept speaks of Favourites as a collection or list of links and not of a single link. This means in the WSDM ontology a list of FavouriteLinks constitutes a WAfA ontology Favourites concept. The same applies to Bookmark/BookmarkLink and ShoppingCart/ShoppingCartLink. So all 4 WAfA concepts are created in the WSDM ontology as individual links and it is the collection of them bundled in a WSDM ontology list concept that maps onto the corresponding WAfA concept. Though we now have 4 new WSDM ontology concepts as opposed to 3 existing WAfA concepts this is solved by mapping both WSDM ontology bookmark concepts to the existing WAfA ontology bookmark concept.



New mapping rules:

In the WSDM ontology, a list consists of a collection of elements/items which in turn point to a link. This link can be a.o. a ProcessLogicBookmarkLink, NavigationalAidBookmarkLink, FavouriteLink or ShoppingCartLink. Note that these rules resemble the NavigationalList rule, where NavigationalList is a WAfA ontology concept and it's WSDM ontology counterpart is a list consisting of a collection of (regular) links. This reasoning yields the following 4 rules:

```
∀ a ∈ I, ∃ b, c, d, e ∈ I: wsdm:List(a) ^ wsdm:hasChild(a, b) ^ wsdm:ListElement(b) ^
wsdm:hasChild(b, c) ^ wsdm:ListItem(c) ^ wsdm:hasNavigationReference(c, d) ^
wsdm:NavigationReference(d) ^ wsdm:pointingToLink(d, e) ^
wsdm:NavigationalAidBookmarkLink(e) → wafa:Bookmark(a)

∀ a ∈ I, ∃ b, c, d, e ∈ I: wsdm:List(a) ^ wsdm:hasChild(a, b) ^ wsdm:ListElement(b) ^
wsdm:hasChild(b, c) ^ wsdm:ListItem(c) ^ wsdm:hasNavigationReference(c, d) ^
wsdm:NavigationReference(d) ^ wsdm:pointingToLink(d, e) ^
wsdm:ProcessLogicBookmarkLink(e) → wafa:Bookmark(a)

∀ a ∈ I, ∃ b, c, d, e ∈ I: wsdm:List(a) ^ wsdm:hasChild(a, b) ^ wsdm:ListElement(b) ^
wsdm:hasChild(b, c) ^ wsdm:ListItem(c) ^ wsdm:hasNavigationReference(c, d) ^
wsdm:NavigationReference(d) ^ wsdm:pointingToLink(d, e) ^ wsdm:FavouriteLink(e) →
wafa:Favourites(a)

∀ a ∈ I, ∃ b, c, d, e ∈ I: wsdm:List(a) ^ wsdm:hasChild(a, b) ^ wsdm:ListElement(b) ^
wsdm:hasChild(b, c) ^ wsdm:ListItem(c) ^ wsdm:hasNavigationReference(c, d) ^
wsdm:NavigationReference(d) ^ wsdm:pointingToLink(d, e) ^ wsdm:ShoppingCartLink(e) →
wafa:ShoppingCart(a)
```

Methodology impact:

In the Conceptual Design, the Navigational Design is impacted. During construction of the navigation track of an audience class, a task navigation model is build for each task in this audience class. Task navigation models are basically defined in terms of components and navigational aid links or process logic links [10]. It is precisely here that the designer has extra choices with the addition of the four proposed WSDM ontology concepts.

4.3.6 Index & SiteIndex

Ontology extension:

Consider the example of reading a book. A table of content is a standard feature of a book (in the beginning), but so is an index (at the end). An index is usually an alphabetical list of references. A site index is also an alphabetical list of references, all pointing to pages within that site.
It is thanks to the addition of the OrderedList concept that we can map to both WAfA concepts through usage of mapping rules only. The WSDM ontology does not need to be extended.

New mapping rules:

```
∀ i ∈ I, ∃ x, y ∈ I: wsdm:OrderedList(i) ^ wsdm:hasChild(i, x) ^ wsdm:ListItem(x) ^
wsdm:hasNavigationReference(x, y) ^ wsdm:NavigationReference(y) ^
(wsdm:pointingToNode(y, ) ∨ wsdm:pointingToLink(y, )) → wafa:Index(i)

∀ i ∈ I, ∃ x, y, z ∈ I: wsdm:OrderedList(i) ^ wsdm:hasChild(i, x) ^ wsdm:ListItem(x)
^ wsdm:hasNavigationReference(x, y) ^ wsdm:NavigationReference(y) ^
wsdm:pointingToNode(y, z) ^ wsdm:InternalNode(z) → wafa:SiteIndex(i)
```

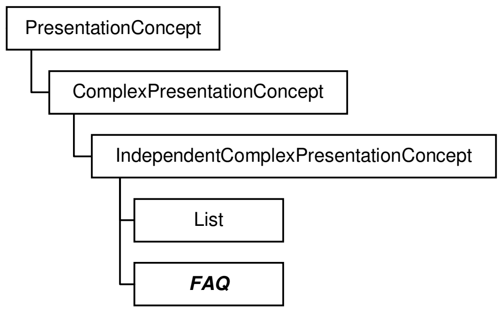
Methodology impact:

As explained above, the WSDM ontology is not extended, hence the methodology is not impacted.

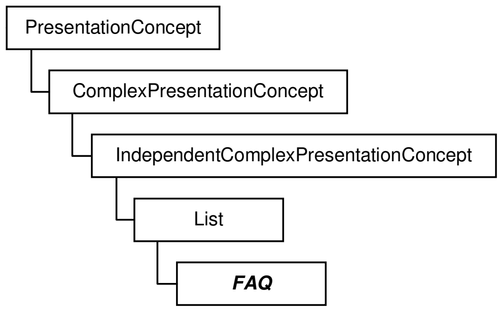
4.3.7 FAQ

Ontology extension:

A frequently asked questions-list as such is an unknown concept in the WSDM ontology. It is impossible to create a single mapping rule to cover this WAfA concept. Therefore we add it to the WSDM ontology as a separate class. The FAQ shall be added as another non-basic presentation concept and is thus a subclass of the existing IndependentComplexPresentationConcept.
Whether it should be added as an IndependentComplexPresentationConcept or as a List subclass has no impact on the new mapping rule (it is always a one-to-one relation) nor on the methodology, since in both cases the same phase of the methodology is impacted.


PresentationConcept
ComplexPresentationConcept
IndependentComplexPresentationConcept
List
FAQ

-or-


PresentationConcept
ComplexPresentationConcept
IndependentComplexPresentationConcept
List
FAQ

<u>New mapping rules:</u>

$\forall$ `i` $\in$ `I: wsdm:FAQ(i)` $\rightarrow$ `wafa:FAQ(i)`

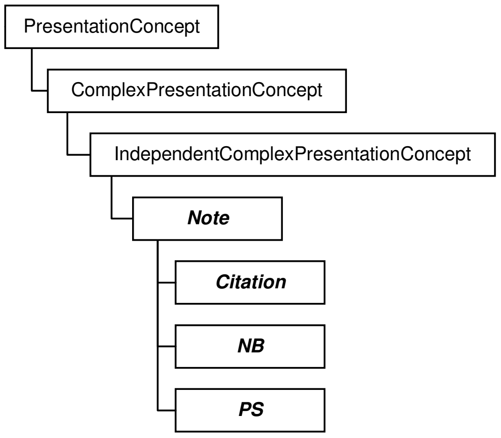
<u>Methodology impact:</u>

In the Implementation Design, the Presentation Design is impacted. As stated above, the FAQ is a non-basic presentation concept offering an alternative to the existing List concept when it comes to adding a frequently asked questions part to the Web site design.

4.3.8 Note, Citation, NB, PS

<u>Ontology extension:</u>

The WAfA ontology describes the Note concept as follows: "A comment that is usually added as supporting information to the main content of the document- additional information".
There are no concepts in the WSDM ontology that have any resemblance to this one. The WSDM Summary comes closest but even that differs too much to create the Note as a subclass of the Summary class. Therefore the Note class is created as a separate IndependantComplexPresentationConcept. In the WAfA ontology, Citation, NB and PS are subclasses of the Note class. This hierarchy is retained, so in the WSDM ontology they are also created as subclasses. We add the Note class to the WSDM ontology with Citation, NB and PS as subclasses.


PresentationConcept
ComplexPresentationConcept
IndependentComplexPresentationConcept
Note
Citation
NB
PS

New mapping rules:

```
∀ i ∊ I: wsdm:Note(i) → wafa:Note(i)
∀ i ∊ I: wsdm:Citation(i) → wafa:Citation(i)
∀ i ∊ I: wsdm:NB(i) → wafa:NB(i)
∀ i ∊ I: wsdm:PS(i) → wafa:PS(i)
```

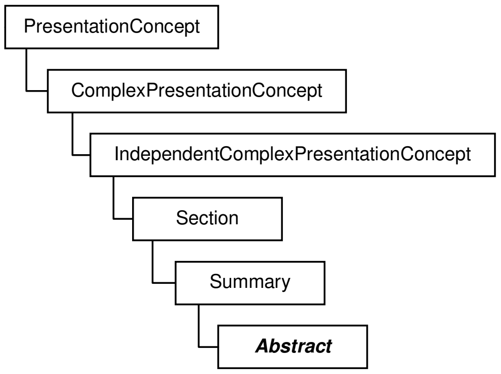
Methodology impact:

Adding these classes impacts the Presentation Design of the Implementation Design. Notes, Citations, NB and PS are all new concepts offered to the designer to use at his or her own discretion.

4.3.9 Abstract

Ontology extension:

In the terms of WAfA, the Abstract concept is defined as follows: “a sketchy summary of the main points of an argument or theory”. This means an abstract is *a sort of summary*. As a consequence, in the WSDM ontology, the Abstract class is created as a subclass of the existing Summary class.


PresentationConcept
ComplexPresentationConcept
IndependentComplexPresentationConcept
Section
Summary
Abstract

New mapping rules:

```
∀ i ∈ I: wsdm:Abstract(i) → wafa:Abstract(i)
```

Methodology impact:

In the Implementation Design, the Presentation Design is impacted. Abstract as a new WSDM ontology concept is another non-basic presentation concept as a "special summary" which in turn is a special case of the section concept. Note that the Abstract concept is not tied to a page or a site like the PageSummary or SiteSummary concepts (see below).

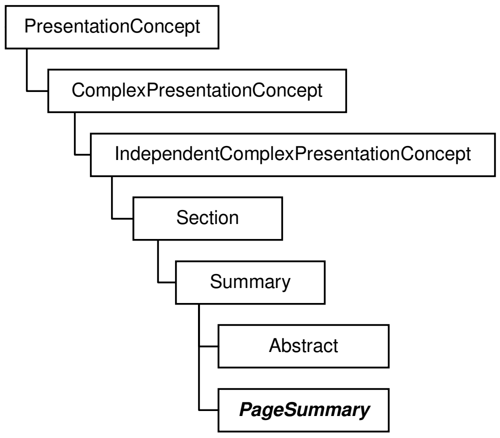
4.3.10 PageSummary

Ontology extension:

The WAfA ontology defines the PageSummary concept as follows: "a chunk that provides the summary of the page".
In WSDM, a WAfA chunk is represented by a grid that represents an object chunk as is illustrated by the following rule:

```
∀ i ∈ I, ∃ x ∈ I: wsdm:Grid(i) ^ wsdm:representsChunk(i, x) ^ wsdm:ObjectChunk(x) →
wafa:Chunk(i)
```

So we can think of a PageSummary as a part of that grid containing the summary. In the WSDM ontology, the general Summary concept already exists. It is not hard to see the PageSummary as a special case meaning it should be added as a subclass of the Summary class. Like a grid represents an ObjectChunk it would be nice to be able to tie the PageSummary to it's corresponding page.
This means that apart from the addition of the PageSummary concept, a new relation needs to be created: summarizesPage(x, y) which should be read in the following way: "a PageSummary x summarizes the Page y".


PresentationConcept
ComplexPresentationConcept
IndependentComplexPresentationConcept
Section
Summary
Abstract
PageSummary

New mapping rules:

```
∀ i ∈ I: wsdm:PageSummary(i) → wafa:PageSummary(i)
```

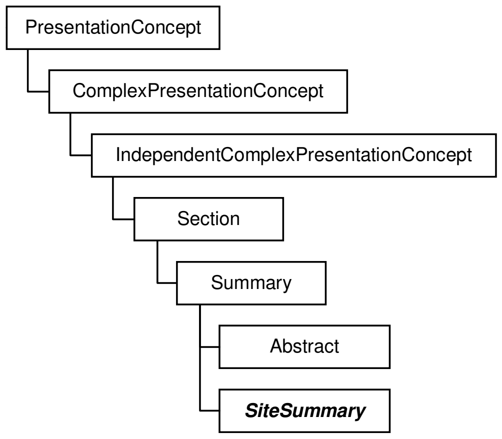
Methodology impact:

Again, the Presentation Design of the Implementation Design phase is impacted since another non-basic presentation concept is added [10]. Note that Page and ObjectChunk should not be confused: in WSDM-terms a page can consist of one or more ObjectChunks so it would be incorrect to tie the PageSummary to the WSDM ontology ObjectChunk concept.

4.3.11 SiteSummary

Ontology extension:

The WAfA ontology defines the SiteSummary concept as follows: “a chunk that provides a summary of the site”.
In the WSDM ontology, a site is represented by the SiteStructureConcept WebSite. Much like with the PageSummary, this new concept can be added as a subclass of the existing Summary class but with some kind of “link” to a Web site. This link is formed by the relation summarizesWebSite. As with the PageSummary addition, a new relation needs to be created: summarizesWebSite(x, y) which should be read in the following way: “a SiteSummary x summarizes the site WebSite y”.


PresentationConcept
ComplexPresentationConcept
IndependentComplexPresentationConcept
Section
Summary
Abstract
SiteSummary

New mapping rules:

```
∀ i ∈ I: wsdm:SiteSummary(i) → wafa:SiteSummary(i)
```

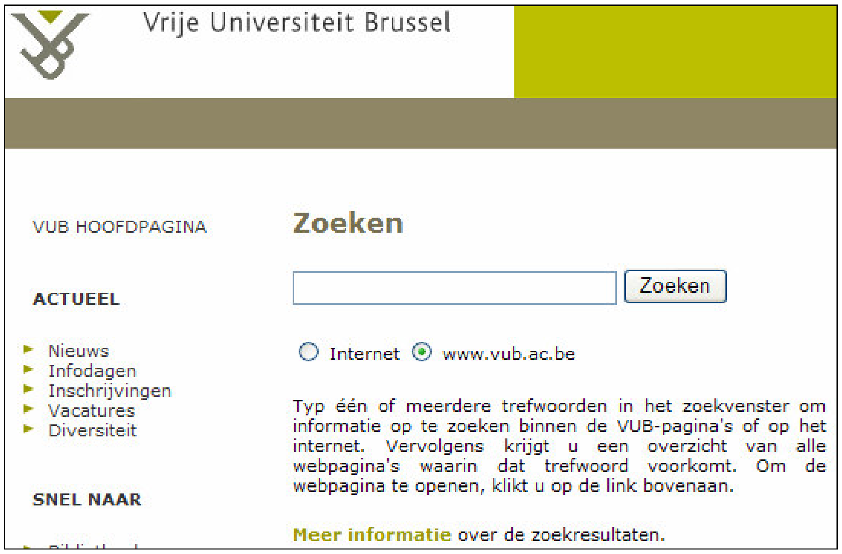
Methodology impact:

Again, the presentation design of the implementation design phase is impacted since another non-basic presentation concept is added [10].
Apart from this, it is not hard to see that a summary about the entire Website would resemble the Mission Statement (first phase of the WSDM methodology). The Mission Statement is not modelled in the WSDM ontology though so this phase is not impacted. For the designer however it should be noted that the “inspiration” for the content of the Site Summary can be found there.

4.3.12 SearchEngine

Ontology extension:

The WAfA definition of a Search Engine is “Consists of an edit box, a button and is usually identified by a label. It could be used to search the site or the Web”. Plotted against this definition, consider the following examples:



Typ één of meerdere trefwoorden in het zoekvenster om informatie op te zoeken binnen de VUB-pagina's of op het internet. Vervolgens krijgt u een overzicht van alle webpagina's waarin dat trefwoord voorkomt. Om de webpagina te openen, klikt u op de link bovenaan.

Meer informatie over de zoekresultaten.

Figure 13 - VUB Website Search

Figure 13 shows the standard VUB Website search. Notice however that apart from the standard textbox/button combination there are also 2 radio buttons offering a choice between searching the Internet or searching the VUB Website.

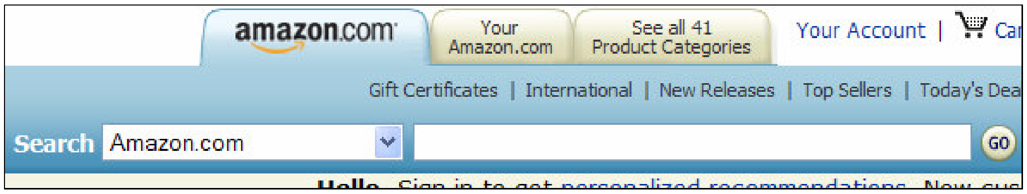

amazon.com
Your Amazon.com
See all 41 Product Categories
Your Account
Gift Certificates | International | New Releases | Top Sellers | Today's Dea
Search
Amazon.com
GO

Figure 14 - Search on Amazon.com

-and-

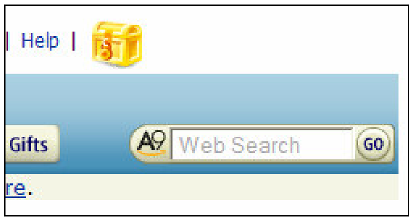
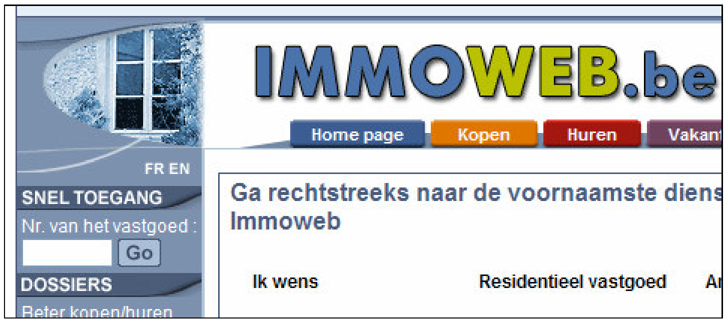


Figure 15 - another Search on Amazon.com

Figure 14 and Figure 15 both are from the Amazon.com Web site and both reside in the same header but one can clearly see the difference in build between the two.



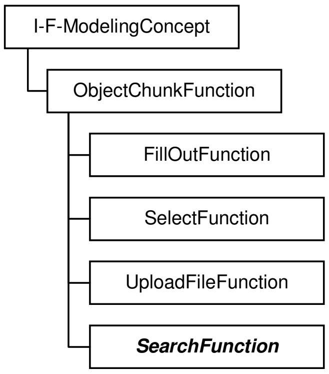
Ga rechtstreeks naar de voornaamste diens
Immoweb

Figure 16 - a special search on www.immoweb.be

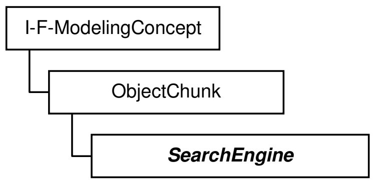
Figure 16 on the left hand side allows users to input a number related to a property featured on the Website www.immoweb.be (Belgian real estate site). Upon clicking the Go button the site opens the relevant property page which is faster than having to browse through all the properties in your area of choise. So this too, has something of a search functionality, rapidly finding the desired property.

The above examples demonstrate the SearchEngine concept definition needs to be extended. The button can be a link, represented by some text (e.g. Go!, Get It!, Search, Find, etc…) or a graphic. The label itself can also be a graphic. In lots of cases there is some text present indicating the presence of a search possibility (e.g. “enter your search criteria here”, etc…).
Next consider the placement – the accompanying text or even the button/link are not always placed in the same area.

The need to be able to search for something, i.e. to have a search capability or search engine is a functional requirement. When taking a look at the I-F-ModelingConcept part of the WSDM ontology, notice the ObjectChunkFunction concept. This concept has several subconcepts or subclasses like e.g. FillOutFunction or UploadFileFunction. The uploading of a file resembles a search somewhat: in most cases we’ll find a label or some accompanying text, a textbox and an upload button or link. Based on these observations, the class SearchFunction is added as a subclass of the ObjectChunkFunction superclass.



The SearchEngine concept represents a very specific kind of functionality or in other words *functional requirement*. Since in WSDM-terms an ObjectChunk is a model of information and/or functional requirements, we can think of a search engine as a special kind of ObjectChunk. This means in the WSDM ontology the SearchEngine is created as a subclass of the existing ObjectChunk concept.



New mapping rules:

```
∀ i ∈ I: wsdm:SearchEngine(i) → wafa:SearchEngine(i)
```

Methodology impact:

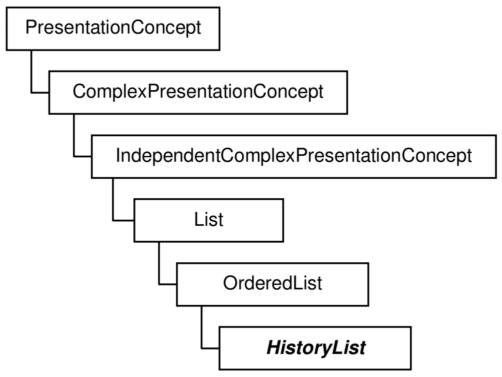
The addition of the SearchFunction and SearchEngine concepts impact the Conceptual Design, Task & Information Modeling (also functional modelling). This is the conceptual "what" of the design.

As an afterthought, the SearchEngine concept could be split up in an InternalSearchEngine and ExternalSearchEngine – depending if the engine searches the site or the Internet. This was not implemented however as the SearchEngine concept suffices for our current research.

4.3.13 HistoryList

Ontology extension:

The WAfA ontology definition[32] : "Provides links to the pages that the user has visited before. It usually presents the links in a particular time order therefore it is also a child of the OrderedList". The good news is the OrderedList was added as a concept to the WSDM ontology. The bad news is that there is no distinction between the types of ordering.
This poses a problem because in order to make a distinction between e.g., a HistoryList concept and an Index concept the only thing separating them is the type of ordering: they are both ordered lists of links. The links can point both internally as well as externally. The only difference is an Index is sorted alphabetically (or alfanumerically) while a HistoryList is sorted according to timestamp of link-access. This leaves us no alternative but to create the concept as a subtype of the existing (albeit recently created) OrderedList WSDM ontology concept.


PresentationConcept
ComplexPresentationConcept
IndependentComplexPresentationConcept
List
OrderedList
HistoryList

[32] See http://www.schemaweb.info/schema/SchemaInfo.aspx?id=275 (accessed 2007)

New mapping rules:

```
∀ i ∈ I: wsdm:HistoryList(i) → wafa:HistoryList(i)
```

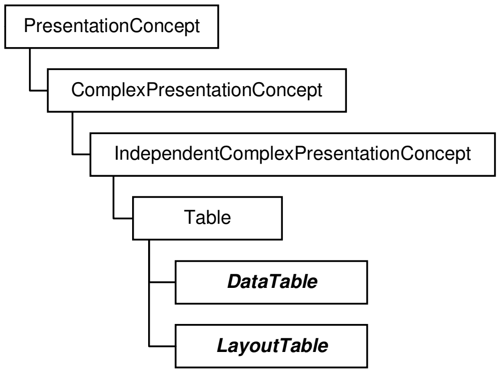
Methodology impact:

Since the HistoryList is a subtype of the List class which is a non-basic presentation concept [10], in the Implementation Design, the Presentation Design is impacted. Apart from the added OrderedList and UnorderedList concepts, the designer now has an additional or more specific OrderedList class available.

4.3.14 DataTable & LayoutTable

Ontology extension:

The Table concept already exists in the WSDM ontology. It is not hard to see that both WAfA concepts are specialised cases of the table concept so they are added in the WSDM ontology as subtypes of the Table concept.


PresentationConcept
ComplexPresentationConcept
IndependentComplexPresentationConcept
Table
DataTable
LayoutTable

New mapping rules:

```
∀ i ∈ I: wsdm:DataTable(i) → wafa:DataTable(i)
∀ i ∈ I: wsdm:LayoutTable(i) → wafa:LayoutTable(i)
```

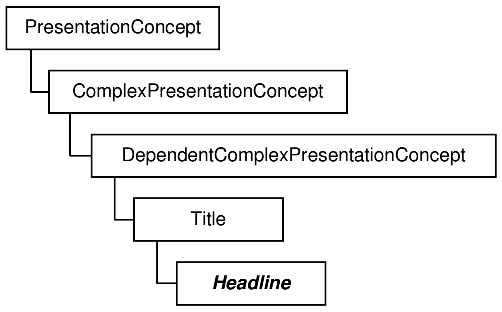
Methodology impact:

Since a Table is a presentation concept, the presentation design is impacted. It is left to the discretion of the designer to choose the purpose of the table. The existing Table concept is retained as a class in case it is unclear which type of table to choose from.

4.3.15 Headline

Ontology extension:

The WAfA ontology defines the headline concept as follows: “Is a widely known concept in newspapers; it is usually set in large type to indicate an important or sensational piece of news. In the context of Web pages, it is the highlighted heading which can be the latest updated or the most important heading in the page.”
In the WAfA ontology, Headline is a subtype of the Heading concept, as is SectionHeading. In WSDM ontology terms, the WAfA Heading and SectionHeading concepts are mapped to the Title concept. The difference between the WAfA concepts Heading and SectionHeading in the WSDM ontology mapping is the object the Title is linked to. Simply put, in WSDM ontology terms a (WAfA)Heading is a (WSDM)Title on a (WSDM)Page and a (WAfA)SectionHeading is a (WSDM)Title on a (WSDM)Section.
This however leaves us no room for mapping a WAfA ontology Headline concept. We therefore create this concept in the WSDM ontology as a subtype of the existing Title class.


PresentationConcept
ComplexPresentationConcept
DependentComplexPresentationConcept
Title
Headline

New mapping rules:

Unlike Heading and SectionHeading, no mapping rule deduction is needed. There exists a one-to-one mapping.

$\forall$ i $\in$ I: wsdm:Headline(i) $\rightarrow$ wafa:Headline(i)

Note that we do not require the Headline to be tied to a Page or Section object. It can occur on either one, hence the simple one-to-one mapping rule.

Methodology impact:

Since the new WSDM ontology Headline class is another subtype of the PresentationConcept class, the presentation design is impacted. The newly added concept offers the designer a choice in titles: regular title or headline. There is no need for another SectionHeading subtype since a WAfA ontology SectionHeading concept can be mapped to from existing WSDM ontology elements.

4.4 Conclusion

In this chapter, we presented an extension of the WSDM methodology. This was achieved by adding classes to the ontology and creating new mapping rules with the purpose of attaining a higher coverage of WAfA ontology elements.
Of the 81 WAfA concepts, a mapping and/or similar WSDM ontology concept was presented for 80 of them giving a coverage of almost 99%. To say the least this is a marked improvement over the initial 69% we set out to increase.

5 Test Case: WISE Website

The previous chapter explained how mapping rules and some extensions to WSDM (the method and the ontology) allows to combine both the WSDM and the Dante approach. This, in turn, would lead to the development of Web sites that are much better suited for the visual impaired Web user.
But does the mapping rules concept actually work? And how is this done in practice, how is this concept implemented? Chapter 5 will attempt to answer these questions. Firstly a simple example testing the mapping rule implementation is given, secondly more complex mapping rule examples and implementation problems are discussed, and thirdly a larger example (or rather test case) is set up and discussed.

5.1 The combined WSDM-Dante methodology - Mapping rules put into practice

5.1.1 The search for automation.

In the previous section, it was explained how mapping rules were set up to have a formalized way of tying WSDM ontology concepts to the WAfA ontology counterpart, meaning that starting from WSDM ontology individuals we would be able to generate WAfA ontology individuals.
Even nicer would be to find some automatic way to do this. For example a tool that understands both the WSDM and WAfA ontologies and would also understand and be able to execute the mapping rules as defined before.
In this search for automation, SWRL presented itself as a useful means to accomplish this.

SWRL[33] or the Semantic Web Rule Language is in a fact a combination of two things: the Web ontology Language OWL and the Rule Language RuleML[4].
OWL is intended for use by applications that need to process the content of information in stead of presenting this information to humans. OWL consists of 3 sublanguages[34] OWL Lite, OWL DL and OWL Full.
RuleML is a markup language for publishing & sharing rule bases on the World Wide Web.
Molded together as they are in SWRL, they offer the advantage of combining Horn-like rules (as provided by RuleML) with the OWL knowledge base (as provided by OWL).

We were able to map WSDM concepts onto their WAfA counterparts using SWRL. Moreover, and of much more practical use, SWRL allowed us to create WAfA ontology individuals automatically, taking as input both WSDM and WAfA ontologies and the WSDM ontology individuals.
Since the WAfA and WSDM ontologies are a given, and WSDM ontology individuals are generated as part of the WSDM methodology, this only leaves the effort to enter the rules in the SWRL system. This however, is only a one-time effort. The result is the quasi automatic generation of WAfA individuals requiring no additional actions from the designer/user.

To illustrate this an implementation of SWRL was needed. Protégé-OWL[35] is a program that meets this need. This Java-based tool supports a.o. the usage (edit, create, etc…) of ontologies (see Figure 17) and the editing and execution of rule sets much to our convenience exactly like the sort of rule sets we created with the WSDM/WAfA mapping rules.

5.1.2 Rule execution – a simple example

[33] See http://www.daml.org/rules/proposal/rules-all.html (accessed 2006/2007)
[34] See http://www.w3.org/TR/owl-features/ (accessed 2007)
[35] See http://protege.stanford.edu/overview/protege-owl.html (accessed 2006/2007)

We will first use a small example to illustrate how a mapping rule can be implemented with Protégé-OWL. Suppose that, after the design phase, we have an instance (in protégé terminology also called an individual) of the WSDM ontology Advertisement class. Since there exists a one-to-one mapping, we expect to see, after execution of the mapping rules, an instance of the WAfA ontology Advertisement class.
Our example consists of 4 steps:

1. Start a new Protégé project and import both WAfA and WSDM ontologies.
2. Add the WSDM Advertisement individual (an instantiation of the WSDM Advertisement class)
3. Enter the necessary mapping rules (using the protégé build-in SWRL-tab)
4. To execute the rules: activate the reasoner (called Jess)

Let us take a look at these steps in detail.

Step 1: starting from a new project, import both ontologies (WAfA & WSDM).

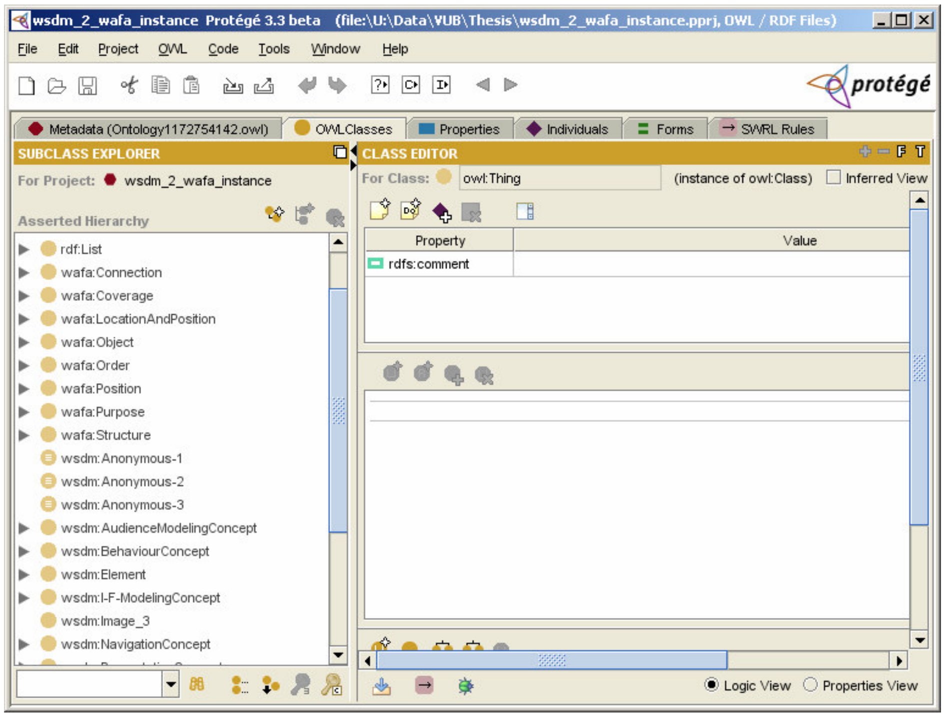


Figure 17 - WSDM & WAfA Ontologies

Step 2: create the WSDM Advertisement individual. Note that at this point there are no WAfA individuals.

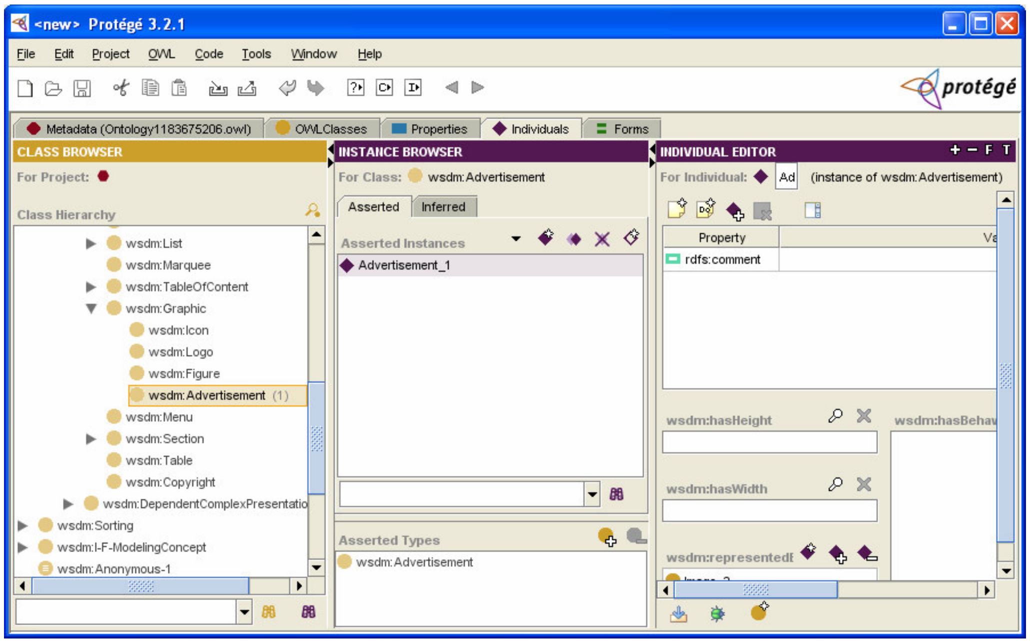
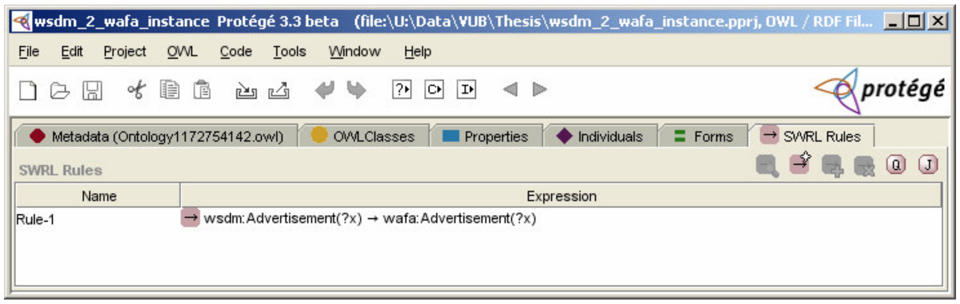


Figure 18 - WSDM Ontology Advertisement Instance

Step 3: enter the mapping rules We will at this point only enter the rule that is relevant to this example.



wsdm_2_wafa_instance Protégé 3.3 beta (file:\U:\Data\VUB\Thesis\wsdm_2_wafa_instance.pprj, OWL / RDF Fil...
File Edit Project OWL Code Tools Window Help
Metadata (Ontology1172754142.owl)
OWLClasses
Properties
Individuals
Forms
SWRL Rules
SWRL Rules

Name	Expression
Rule-1	wsdm:Advertisement(?x) → wafa:Advertisement(?x)

Figure 19 - a simple mapping rule in Protégé using Jess/SWRL

Step 4: activate the reasoner (called Jess, it is activated by clicking on the “J”-button on the right of Figure 19) which is the actual execution of the mappig rule(s). As a result of our example, a new WAfA ontology Advertisement individual is automatically created.

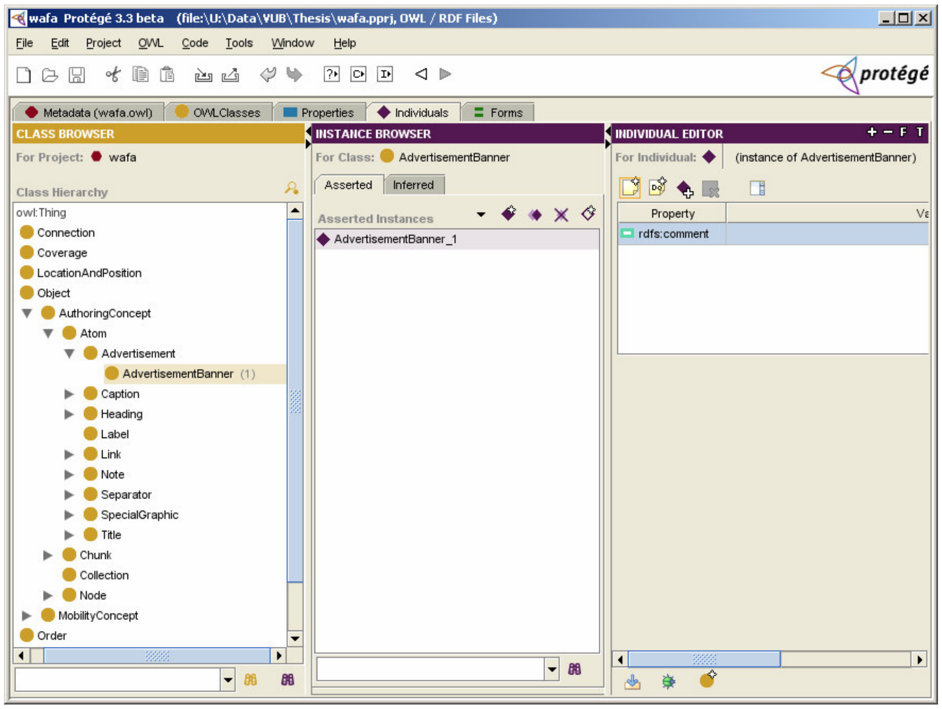


Figure 20 - WAfA Ontology Advertisement Instance

This simple example proves the applicability of the automated rules concept in the form of the Protégé/SWRL tool. It is not hard to imagine the gains to be made after the complete design with dozens or hundreds of WSDM individuals is subjected to this automatic mapping rule execution.

However, one obvious drawback is that if one or both ontologies change, then also the mapping rules will need to be re-examined and -if necessary- changed. But this is standard maintenance inherent to any IT-system and should not be considered as a drawback specifically related to our mapping approach. The advantages of the mapping approach still far outweigh the drawbacks.

5.2 Rule execution – more complex examples/rules

As explained above, SWRL extends OWL-axioms to include Horn-like rules. Such a rule axiom consists of a so-called antecedent and a consequent written in the following form: *antecedent* → *consequent*. Which is read as "if *antecedent* is true, then *consequent* must also be true". Both antecedent and consequent are conjunctions of atoms, so can be written in the form `a1 ^ a2 ^ a3` … `^ an` where `a1` to `an` represents the atoms and the ^ symbol represents the conjunction. This, in turn should be read as "`if a1 and a2 and` … `and an`". This is well suited for our intended usage. In all, of the 81 WAfA concepts we were able to map 69% of them through usage of rules. Most of those rules are written as the so called "conjunctions of atoms" explained above and can therefore directly be applied through the SWRL/Protégé combination explained in 5.1.2. This was not the case for *all* rules however.

The current SWRLTab/Jess implementation only supports conjunction(AND), no disjunction(OR) nor negation(NOT).

Observe the following rule:

```
∀ i ∈ I, ∃ x ∈ I: (wsdm:String(i) ∨ wsdm:Image(i)) ^ (wsdm:Textbox(x) ∨
wsdm:Checkbox(x) ∨ wsdm:Radiobutton(x) ∨ wsdm:Pushbutton(x)) ^
wsdm:hasLabel(x, i) → wafa:Label(i)
```

This rule means that, starting from the WSDM ontology, a string or image that functions as a label for a textbox, a checkbox, a radio button or a push button maps to a WAfA ontology Label concept. The problem here is the disjunction (symbol ∨). We cannot write one SWRL rule that corresponds to the above-mentioned mapping rule, since SWRL only recognizes the concept of conjunction. This can be solved however in the following way:

In Propositional Logic, 2 formulae are *Logically Equivalent* if they both yield the same result with the same value of input parameters [9].
This means that

```
p ^ (q ∨ r)
```

Is logically equivalent to

```
(p ^ q) ∨ (p ^ r)
```

Which, coming back to our problematic rule, means that

```
(wsdm:String(i) ∨ wsdm:Image(i)) ^ (wsdm:Textbox(x) ∨ wsdm:Checkbox(x) ∨
 wsdm:Radiobutton(x) ∨ wsdm:Pushbutton(x)) ^ wsdm:hasLabel(x, i)
```

can be rewritten as

```
(wsdm:String(i) ^ wsdm:Textbox(x) ^ wsdm:hasLabel(x, i))   ∨
(wsdm:String(i) ^ wsdm:Checkbox(x) ^ wsdm:hasLabel(x, i))  ∨
(wsdm:String(i) ^ wsdm:Radiobutton(x) ^ wsdm:hasLabel(x, i)) ∨
(wsdm:String(i) ^ wsdm:Pushbutton(x) ^ wsdm:hasLabel(x, i)) ∨
(wsdm:Image(i) ^ wsdm:Textbox(x) ^ wsdm:hasLabel(x, i))    ∨
(wsdm:Image(i) ^ wsdm:Checkbox(x) ^ wsdm:hasLabel(x, i))   ∨
(wsdm:Image(i) ^ wsdm:Radiobutton(x) ^ wsdm:hasLabel(x, i)) ∨
(wsdm:Image(i) ^ wsdm:Pushbutton(x) ^ wsdm:hasLabel(x, i))
```

which gives us one complex rule consisting of 8 disjunctions. However, if the following holds

```
A ∨ B → C
```

then the following also holds if both rules are executed independent from each other at the same time:

```
A → C
B → C
```

So the solution for us lies in simply creating extra rules for disjunctions. Coming back to our WSDM/WAfA ontology example this leads to the creation of 8 simple, separate rules that SWRL understands:

```
∀ i ∈ I, ∃ x ∈ I: wsdm:String(i) ^ wsdm:Textbox(x) ^ wsdm:hasLabel(x, i) →
wafa:Label(i)
∀ i ∈ I, ∃ x ∈ I: wsdm:String(i) ^ wsdm:Checkbox(x) ^ wsdm:hasLabel(x, i) →
wafa:Label(i)
∀ i ∈ I, ∃ x ∈ I: wsdm:String(i) ^ wsdm:Radiobutton(x) ^ wsdm:hasLabel(x, i) →
wafa:Label(i)
∀ i ∈ I, ∃ x ∈ I: wsdm:String(i) ^ wsdm:Pushbutton(x) ^ wsdm:hasLabel(x, i) →
wafa:Label(i)
```

```
∀ i ∈ I, ∃ x ∈ I: wsdm:Image(i) ^ wsdm:Textbox(x) ^ wsdm:hasLabel(x, i) →
wafa:Label(i)
∀ i ∈ I, ∃ x ∈ I: wsdm:Image(i) ^ wsdm:Checkbox(x) ^ wsdm:hasLabel(x, i) →
wafa:Label(i)
∀ i ∈ I, ∃ x ∈ I: wsdm:Image(i) ^ wsdm:Radiobutton(x) ^ wsdm:hasLabel(x, i) →
wafa:Label(i)
∀ i ∈ I, ∃ x ∈ I: wsdm:Image(i) ^ wsdm:Pushbutton(x) ^ wsdm:hasLabel(x, i) →
wafa:Label(i)
```

Executing these rules would have the same outcome as executing the original rule. The small theory explained above states that in general all disjunction problems can be solved in the described way, which solves our “disjunction-in-rules” problem.

This leaves us with the negation.

As explained before, the SWRLTab/Jess implementation does not support the negation (NOT). However, we do have some rules (4 of them) that have a negation element as one of their components.
A possible solution is to create specific *fact-denouncing* WSDM ontology classes or relations that the designer must instantiate explicitly. This would then just be another class or relation with it’s corresponding individual. Consider e.g. the following rule:

```
∀ a ∈ I, ∃ b, c, d, e ∈ I: wsdm:List(a) ^ wsdm:hasChild(a, b) ^ wsdm:ListElement(b) ^
wsdm:hasChild(b, c) ^ wsdm:ListItem(c) ^ wsdm:hasChild(b, d) ^ wsdm:ListItem(d) ^
¬wsdm:hasNavigationReference(c, ) ^ ¬wsdm:hasNavigationReference(d, ) →
wafa:DefinitionList(a)
```

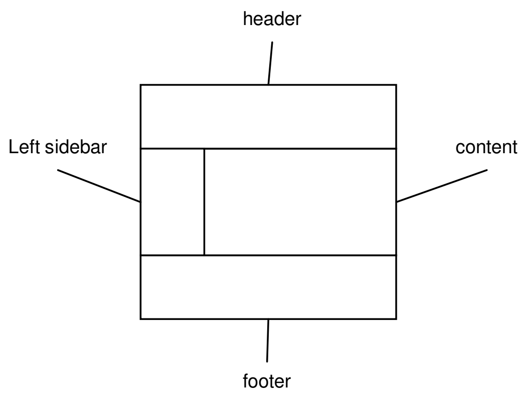
When a new relation `hasNotNavigationReference` is created which e.g. can only contain one string value (“`true`”, “`noref`”, etc…). It is however up to the designer’s discretion to see to it that an element does not have both relations `hasNavigationReference` and `hasNotNavigationReference` activated simultaneously.
To say the least this is not an elegant solution.

5.3 *The WISE Website example*

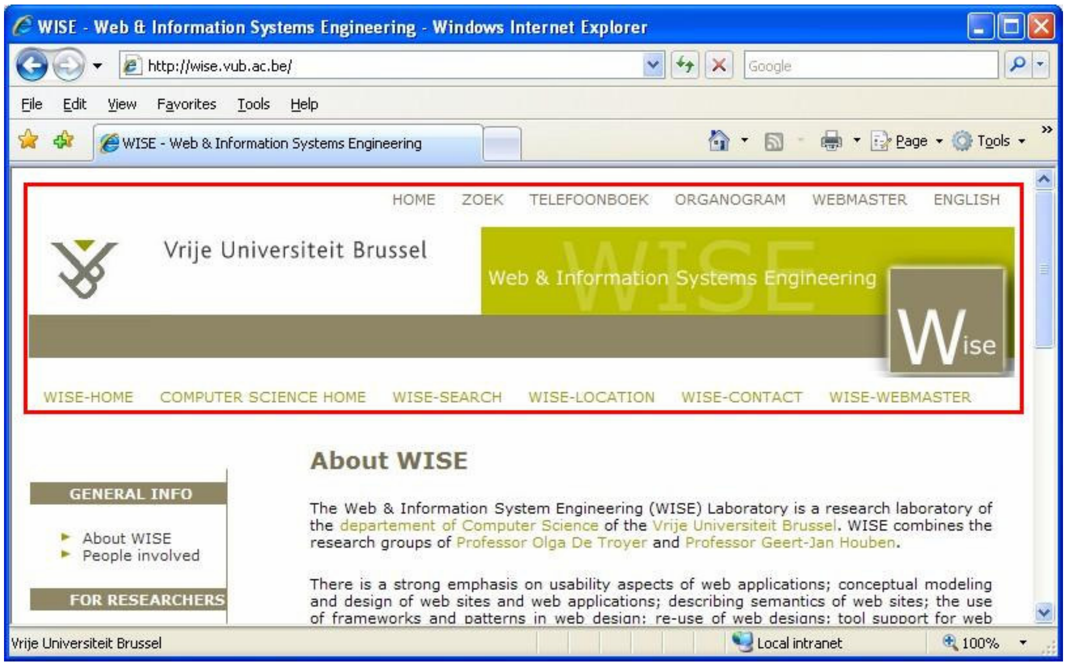
As a final example, a few pages from the WISE Website are modeled in the WSDM ontology. Then the mapping rules seen earlier are executed on these pages and the results will be discussed.

5.3.1 Modeling the pages

The pages included in this example are: the WISE Homepage, the WISE members page and finally Sven Casteleyn's Homepage.
It can be clearly seen that all three pages use the same basic header-sidebar-footer layout.



All pages use the same header and footer (Figure 21 and Figure 22). Sven's homepage uses a different sidebar (menu) than the other two WISE pages (Figure 23 and Figure 24).

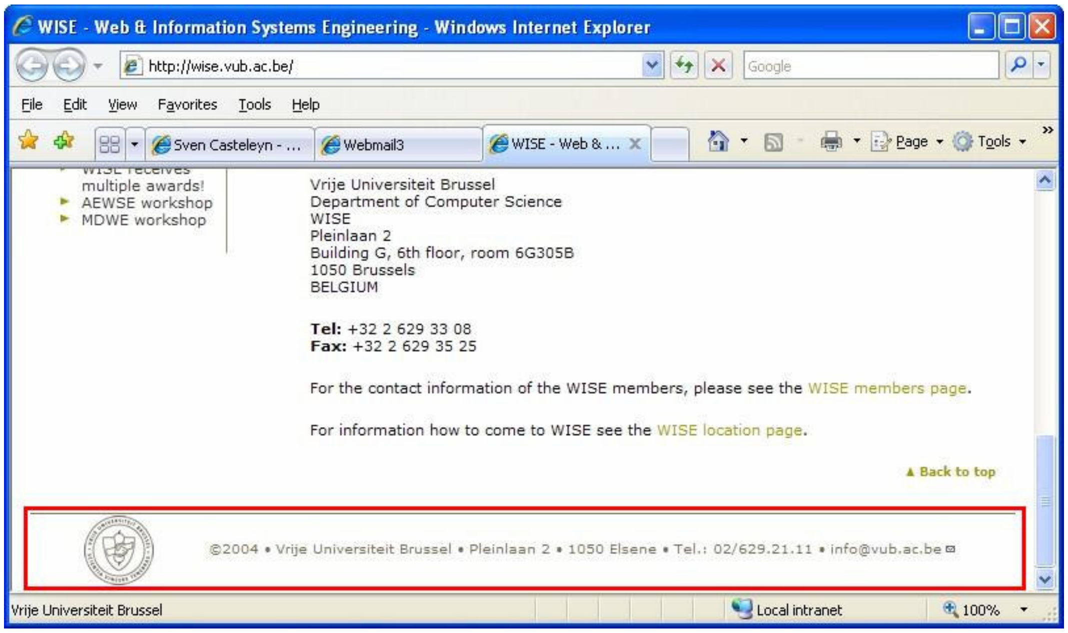

WISE - Web & Information Systems Engineering - Windows Internet Explorer
http://wise.vub.ac.be/
File Edit View Favorites Tools Help
HOME ZOEK TELEFOONBOEK ORGANOGRAM WEBMASTER ENGLISH
Vrije Universiteit Brussel
Web & Information Systems Engineering
Wise
WISE-HOME COMPUTER SCIENCE HOME WISE-SEARCH WISE-LOCATION WISE-CONTACT WISE-WEBMASTER
GENERAL INFO
About WISE
People involved
FOR RESEARCHERS
Local intranet

About WISE

The Web & Information System Engineering (WISE) Laboratory is a research laboratory of the departement of Computer Science of the Vrije Universiteit Brussel. WISE combines the research groups of Professor Olga De Troyer and Professor Geert-Jan Houben.

There is a strong emphasis on usability aspects of web applications; conceptual modeling and design of web sites and web applications; describing semantics of web sites; the use of frameworks and patterns in web design; re-use of web designs; tool support for web

Figure 21 - WISE Homepage and Header



Vrije Universiteit Brussel
Department of Computer Science
WISE
Pleinlaan 2
Building G, 6th floor, room 6G305B
1050 Brussels
BELGIUM

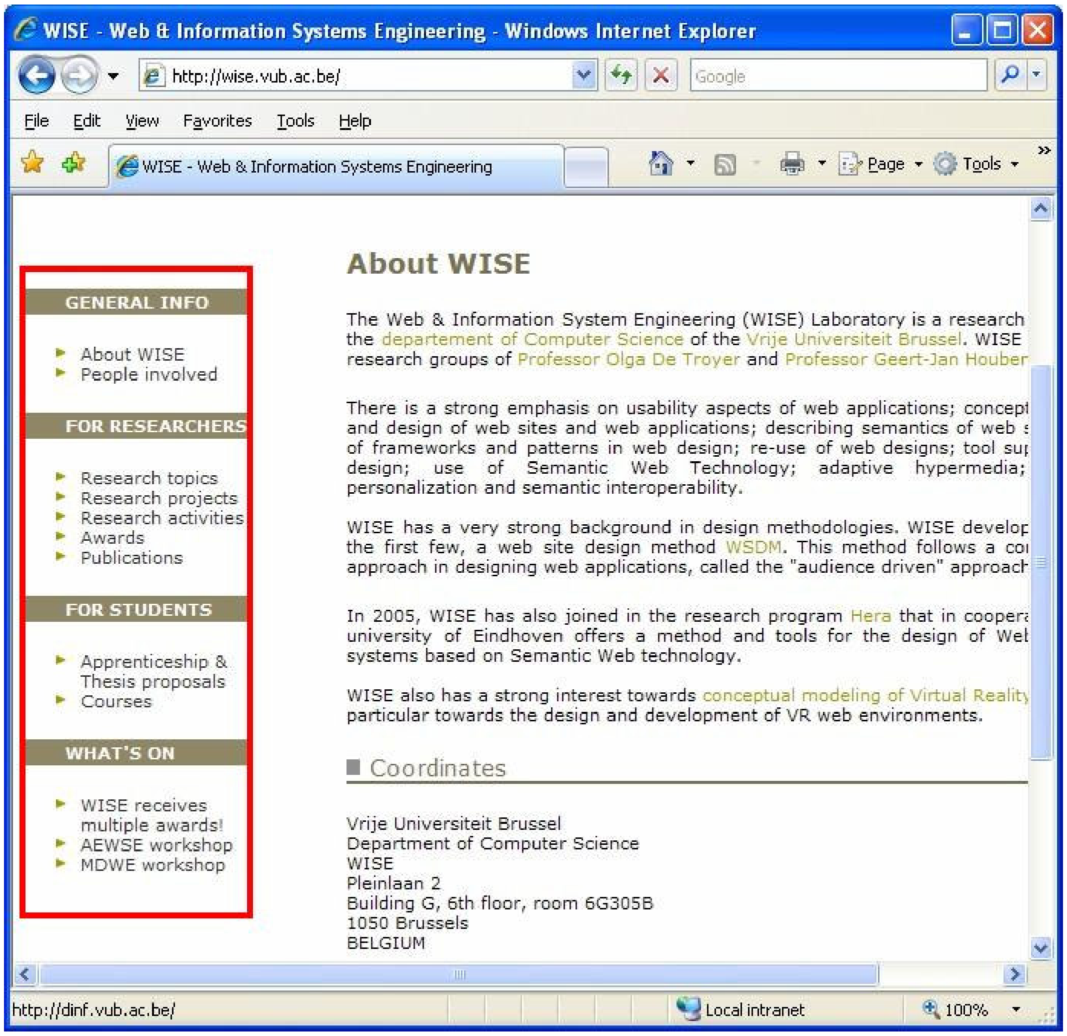
Tel: +32 2 629 33 08
Fax: +32 2 629 35 25

For the contact information of the WISE members, please see the WISE members page.

For information how to come to WISE see the WISE location page.

©2004 • Vrije Universiteit Brussel • Pleinlaan 2 • 1050 Elsene • Tel.: 02/629.21.11 • info@vub.ac.be

Figure 22 - WISE Homepage and Footer


WISE - Web & Information Systems Engineering - Windows Internet Explorer
http://wise.vub.ac.be/
Google
File Edit View Favorites Tools Help
WISE - Web & Information Systems Engineering
Page Tools
http://dinf.vub.ac.be/
Local intranet
100%

GENERAL INFO

- About WISE
- People involved

FOR RESEARCHERS

- Research topics
- Research projects
- Research activities
- Awards
- Publications

FOR STUDENTS

- Apprenticeship & Thesis proposals
- Courses

WHAT'S ON

- WISE receives multiple awards!
- AEWSE workshop
- MDWE workshop

About WISE

The Web & Information System Engineering (WISE) Laboratory is a research
the departement of Computer Science of the Vrije Universiteit Brussel. WISE
research groups of Professor Olga De Troyer and Professor Geert-Jan Houber

There is a strong emphasis on usability aspects of web applications; concept
and design of web sites and web applications; describing semantics of web s
of frameworks and patterns in web design; re-use of web designs; tool sup
design; use of Semantic Web Technology; adaptive hypermedia;
personalization and semantic interoperability.

WISE has a very strong background in design methodologies. WISE develop
the first few, a web site design method WSDM. This method follows a co
approach in designing web applications, called the "audience driven" approach

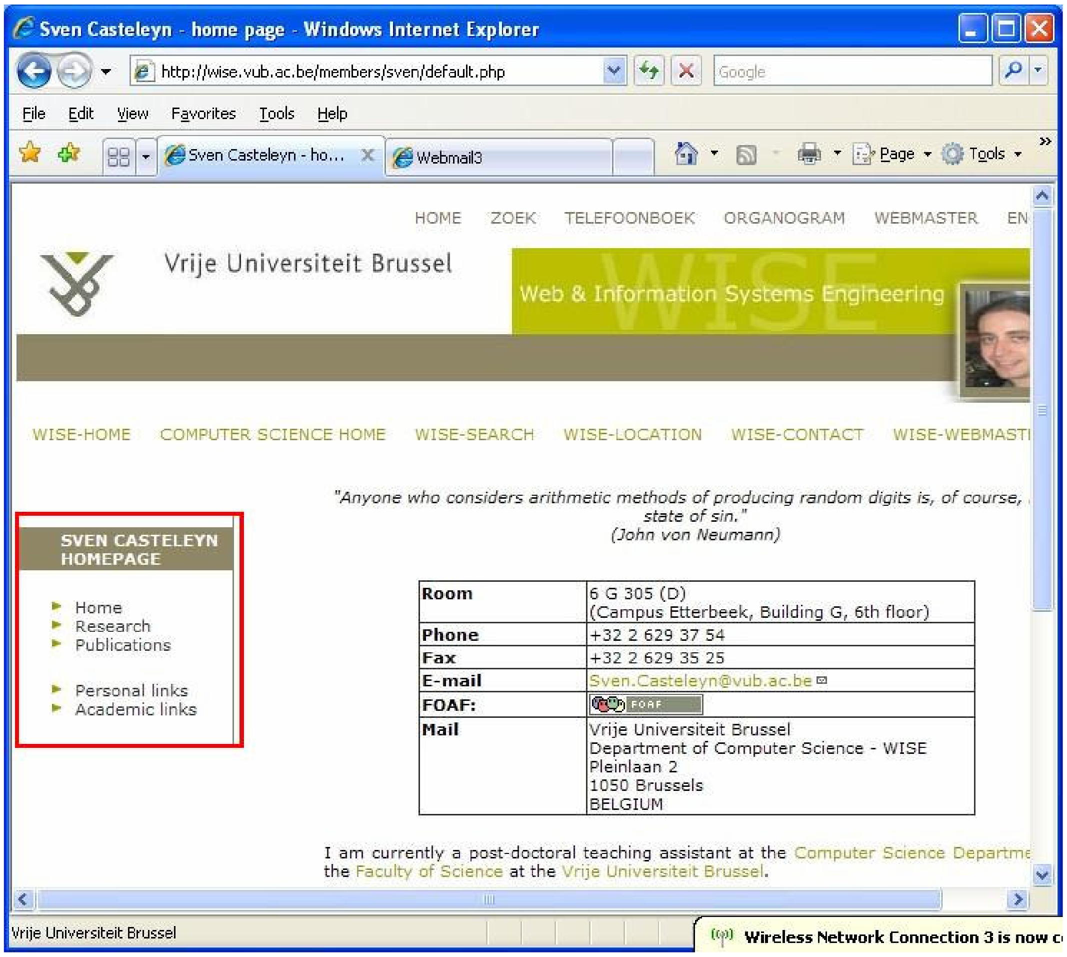
In 2005, WISE has also joined in the research program Hera that in coopera
university of Eindhoven offers a method and tools for the design of Wel
systems based on Semantic Web technology.

WISE also has a strong interest towards conceptual modeling of Virtual Reality
particular towards the design and development of VR web environments.

Coordinates

Vrije Universiteit Brussel
Department of Computer Science
WISE
Pleinlaan 2
Building G, 6th floor, room 6G305B
1050 Brussels
BELGIUM

Figure 23 - WISE Homepage and Sidebar


Sven Casteleyn - home page - Windows Internet Explorer
http://wise.vub.ac.be/members/sven/default.php
Google
File Edit View Favorites Tools Help
Sven Casteleyn - ho...
Webmail3
Page Tools
HOME ZOEK TELEFOONBOEK ORGANOGRAM WEBMASTER EN
Vrije Universiteit Brussel
Web & Information Systems Engineering
WISE-HOME COMPUTER SCIENCE HOME WISE-SEARCH WISE-LOCATION WISE-CONTACT WISE-WEBMASTI
SVEN CASTELEYN HOMEPAGE
Home
Research
Publications
Personal links
Academic links
Vrije Universiteit Brussel
Wireless Network Connection 3 is now c

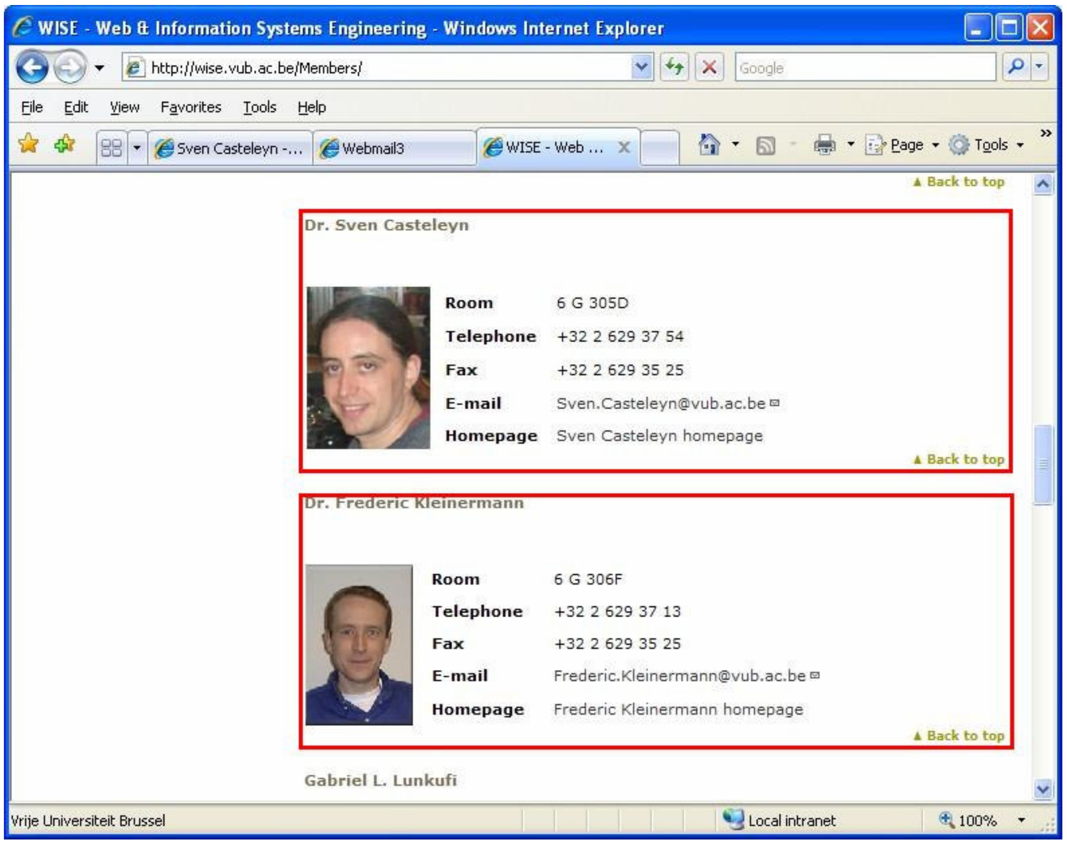
"Anyone who considers arithmetic methods of producing random digits is, of course, state of sin."
(John von Neumann)

Room	6 G 305 (D) (Campus Etterbeek, Building G, 6th floor)
Phone	+32 2 629 37 54
Fax	+32 2 629 35 25
E-mail	Sven.Casteleyn@vub.ac.be
FOAF:	FOAF
Mail	Vrije Universiteit Brussel Department of Computer Science - WISE Pleinlaan 2 1050 Brussels BELGIUM

I am currently a post-doctoral teaching assistant at the Computer Science Departme
the Faculty of Science at the Vrije Universiteit Brussel.

Figure 24 - Sven Casteleyn's Homepage

The main difference obviously lies in the main content of the pages. Whereas the WISE homepage's main content is basically a big chunk of text, the members' main content is somewhat more complex, consisting of repeating blocks of information for each member (Figure 25).


WISE - Web & Information Systems Engineering - Windows Internet Explorer
http://wise.vub.ac.be/Members/
Google
File Edit View Favorites Tools Help
Sven Casteleyn -...
Webmail3
WISE - Web ...
Page
Tools
Vrije Universiteit Brussel
Local intranet
100%

▲ Back to top

Dr. Sven Casteleyn

Room	6 G 305D
Telephone	+32 2 629 37 54
Fax	+32 2 629 35 25
E-mail	Sven.Casteleyn@vub.ac.be
Homepage	Sven Casteleyn homepage

▲ Back to top

Dr. Frederic Kleinermann

Room	6 G 306F
Telephone	+32 2 629 37 13
Fax	+32 2 629 35 25
E-mail	Frederic.Kleinermann@vub.ac.be
Homepage	Frederic Kleinermann homepage

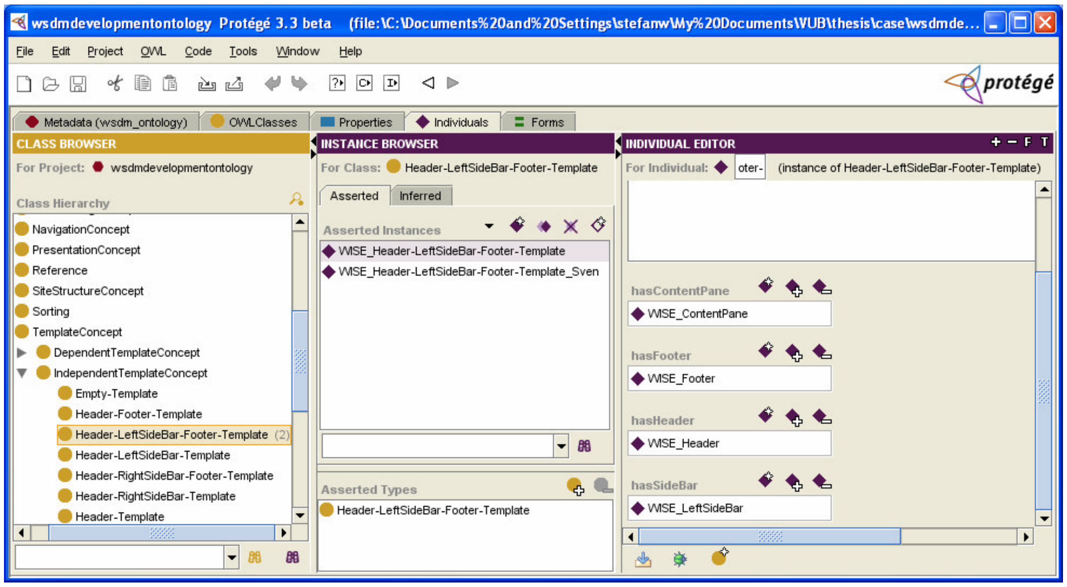
▲ Back to top

Gabriel L. Lunkufi

Figure 25 - Members page main content

The next step is a bit of reverse engineering – in stead of going through the complete WSDM design process for the entire WISE Website, we describe a few existing pages using the WSDM ontology, which has exactly the same result as doing it the other way round. The point is to have a few real-life examples modeled into the WSDM ontology to test our rules on.

Looking at the page structure from a WSDM ontology perspective, the header-footer-sidebar combination can be modeled as such (Figure 26).



For Project: wsdmdevelopmentontology

Class Hierarchy:
- NavigationConcept
- PresentationConcept
- Reference
- SiteStructureConcept
- Sorting
- TemplateConcept
  - DependentTemplateConcept
  - IndependentTemplateConcept
    - Empty-Template
    - Header-Footer-Template
    - Header-LeftSideBar-Footer-Template (2)
    - Header-LeftSideBar-Template
    - Header-RightSideBar-Footer-Template
    - Header-RightSideBar-Template
    - Header-Template

For Class: Header-LeftSideBar-Footer-Template

Asserted Instances:
- WISE_Header-LeftSideBar-Footer-Template
- WISE_Header-LeftSideBar-Footer-Template_Sven

Asserted Types: Header-LeftSideBar-Footer-Template

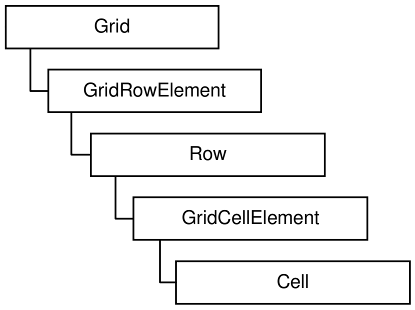
For Individual: oter- (instance of Header-LeftSideBar-Footer-Template)

hasContentPane: WISE_ContentPane
hasFooter: WISE_Footer
hasHeader: WISE_Header
hasSideBar: WISE_LeftSideBar

Figure 26 - Header-Sidebar-Footer model

To model this template, we create instances (or individuals as they are called in our tool of choice) of the appropriate WSDM ontology concepts.
The WSDM ontology Header-LeftSideBar-Footer-Template concept has a ContentPane, a Footer, a Header and a LeftSideBar, for all of which individuals are created. This modeling combination can be used for the WISE pages as well as Sven's Homepage. In fact, the Header and Footer can be used on both templates.

For the main content we use a Grid. The grid is a basic building block in the WSDM ontology's presentation design. In fact, every webpage can be considered to consist mainly of a grid, the grid dividing the page in areas which in turn can contain grids etc… This is reflected in the WSDM ontology Grid concept shown by the following hierarchy:



A WSDM ontology Grid concept can have one or more GridRowElements as a child concept. A GridRowElement can have one or more Rows as a child. The same goes for the GridCellElement and finally the individual Cell as building block in the WSDM ontology Grid concept hierarchy. A Cell, in turn, can contain several different IndependentComplexPresentationConcepts (e.g., a List, Table, Banner, etc…), a ConceptReference, a MultimediaConcept (String, Email, Image, etc…) or another Grid.
This is shown in Figure 27.

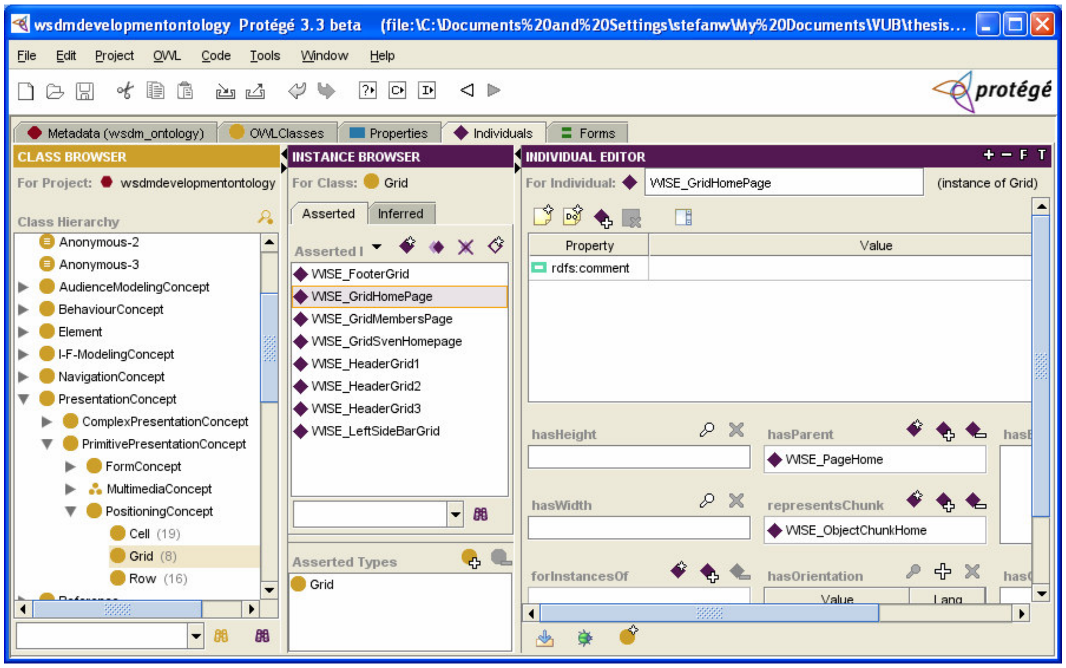


Figure 27- Grid individuals

Coming back to our 3 pages, the WISE Homepage Grid content is basically a big block of text. We can think of the main content part as a grid containing one row with one cell that contains a big lump of text.

The members page is somewhat more complex. Simplified for the sake of not getting lost in too many details it can be thought of as a Grid, consisting of several rows per GridRowElement. Basically each member's data represents a GridRowElement with its accompanying GridCellElement and Cell concepts (Figure 28 and Figure 29).

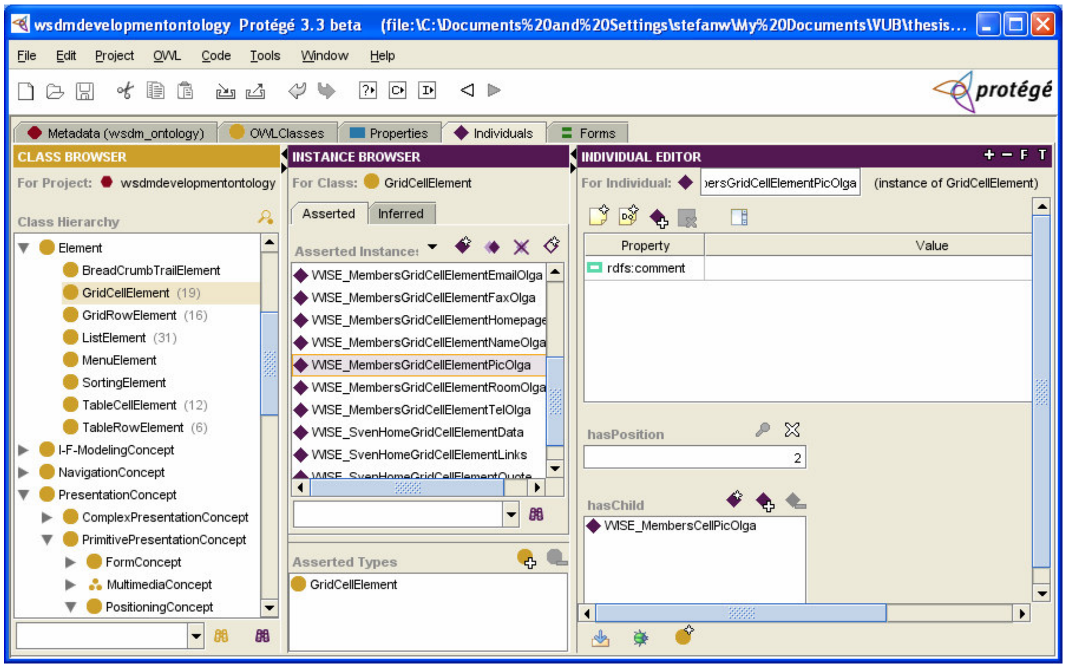

wsdmdevelopmentontology Protégé 3.3 beta (file:\C:\Documents%20and%20Settings\stefanw\My%20Documents\VUB\thesis...
File Edit Project OWL Code Tools Window Help
protégé
Metadata (wsdm_ontology) OWLClasses Properties Individuals Forms
CLASS BROWSER
For Project: wsdmdevelopmentontology
Class Hierarchy
Element
BreadCrumbTrailElement
GridCellElement (19)
GridRowElement (16)
ListElement (31)
MenuElement
SortingElement
TableCellElement (12)
TableRowElement (6)
I-F-ModelingConcept
NavigationConcept
PresentationConcept
ComplexPresentationConcept
PrimitivePresentationConcept
FormConcept
MultimediaConcept
PositioningConcept
INSTANCE BROWSER
For Class: GridCellElement
Asserted Inferred
Asserted Instances
WISE_MembersGridCellElementEmailOlga
WISE_MembersGridCellElementFaxOlga
WISE_MembersGridCellElementHomepage
WISE_MembersGridCellElementNameOlga
WISE_MembersGridCellElementPicOlga
WISE_MembersGridCellElementRoomOlga
WISE_MembersGridCellElementTelOlga
WISE_SvenHomeGridCellElementData
WISE_SvenHomeGridCellElementLinks
Asserted Types
GridCellElement
INDIVIDUAL EDITOR
For Individual: ersGridCellElementPicOlga (instance of GridCellElement)
Property Value
rdfs:comment
hasPosition
2
hasChild
WISE_MembersCellPicOlga

Figure 28 - The Members page Grid structure

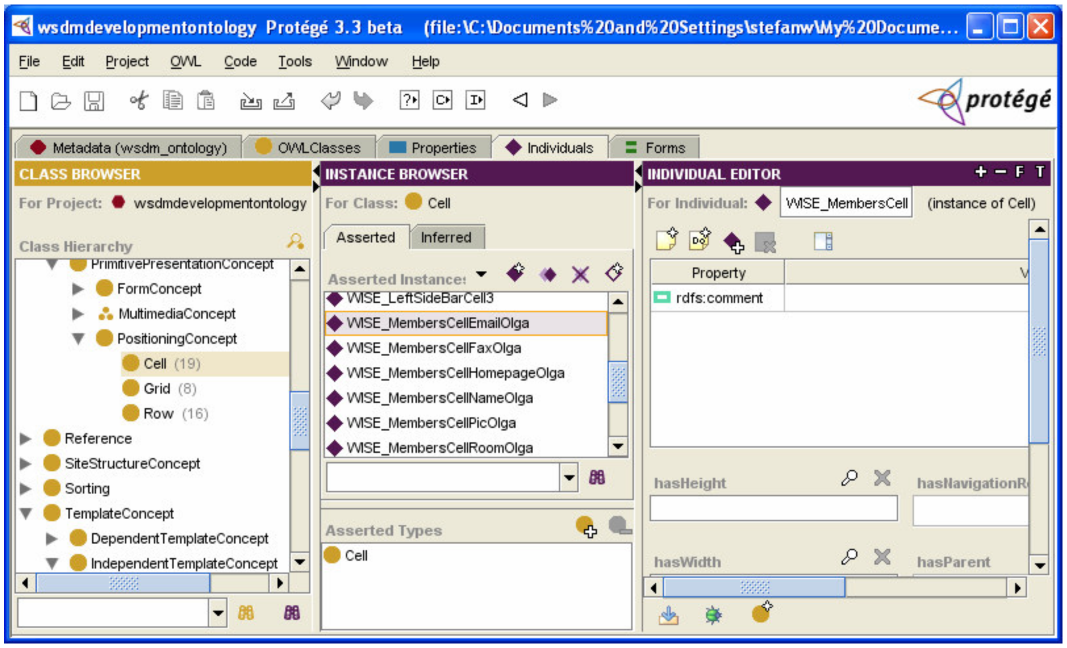
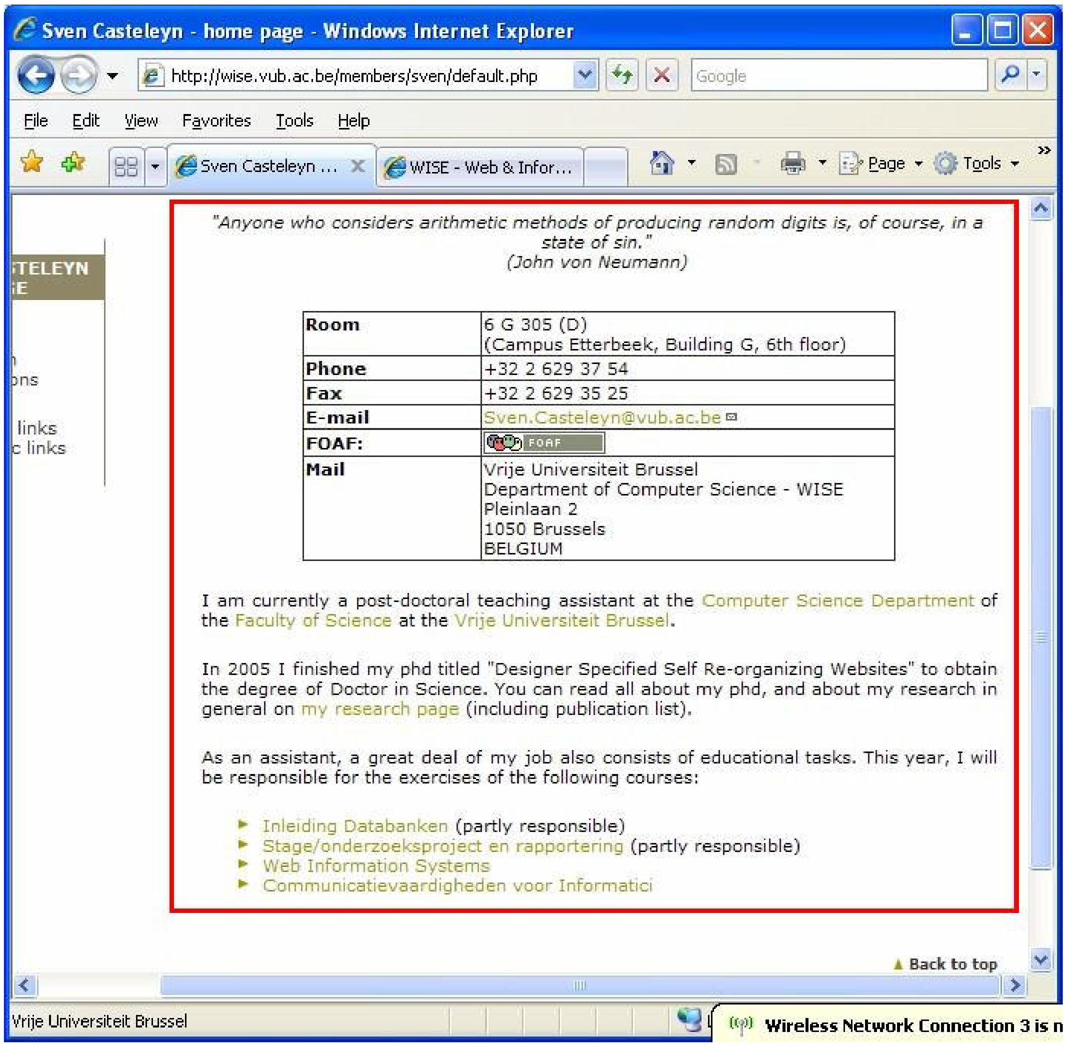


Figure 29 - The Members page Grid structure (cont'd)

On Sven Casteleyn's Homepage we can clearly distinct a citation, a table (used for layout purposes only), some text and a list of links as being part of the main Grid/Layout.


Sven Casteleyn - home page - Windows Internet Explorer
http://wise.vub.ac.be/members/sven/default.php
File Edit View Favorites Tools Help
Sven Casteleyn ...
WISE - Web & Infor...
Page
Tools
TELEYN
E
ons
links
c links
Back to top
Vrije Universiteit Brussel
Wireless Network Connection 3 is n

"Anyone who considers arithmetic methods of producing random digits is, of course, in a state of sin."
(John von Neumann)

Room	6 G 305 (D) (Campus Etterbeek, Building G, 6th floor)
Phone	+32 2 629 37 54
Fax	+32 2 629 35 25
E-mail	Sven.Casteleyn@vub.ac.be
FOAF:	FOAF
Mail	Vrije Universiteit Brussel Department of Computer Science - WISE Pleinlaan 2 1050 Brussels BELGIUM

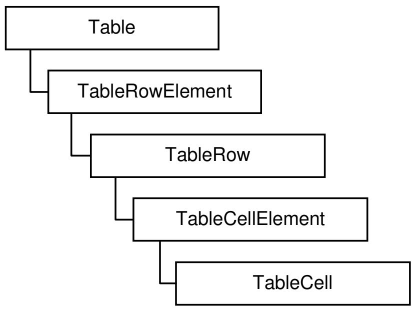
I am currently a post-doctoral teaching assistant at the Computer Science Department of the Faculty of Science at the Vrije Universiteit Brussel.

In 2005 I finished my phd titled "Designer Specified Self Re-organizing Websites" to obtain the degree of Doctor in Science. You can read all about my phd, and about my research in general on my research page (including publication list).

As an assistant, a great deal of my job also consists of educational tasks. This year, I will be responsible for the exercises of the following courses:

- Inleiding Databanken (partly responsible)
- Stage/onderzoeksproject en rapportering (partly responsible)
- Web Information Systems
- Communicatievaardigheden voor Informatici

All these elements are modeled in the WSDM ontology, the Citation and the LayoutTable actually being new concepts that were introduced into WSDM in the course of this research.
The Table concept (of which DataTable and LayoutTable are children concepts thereby inheriting the same properties/structure) actually shows a lot of similarities in construction with the Grid element:



The table containing information like room, phone, fax, email, etc… is modeled as a LayoutTable: the Table element is primarily used for visual purposes.

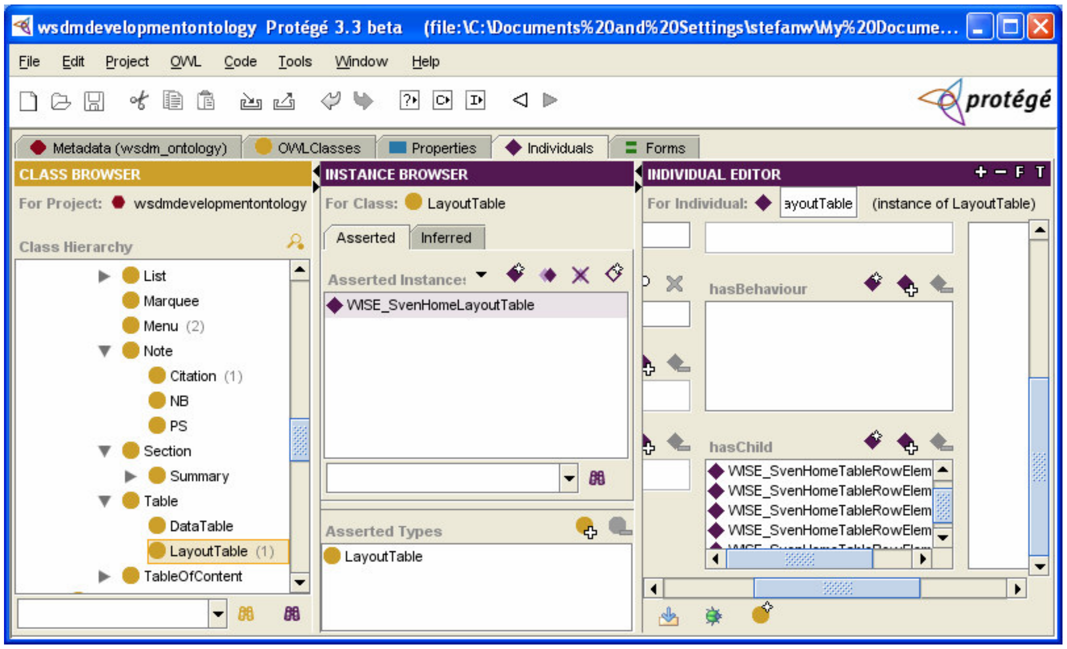
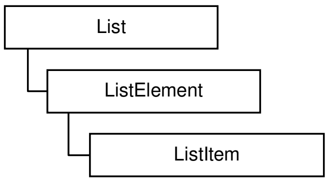

wsdmdevelopmentontology Protégé 3.3 beta (file:\C:\Documents%20and%20Settings\stefanw\My%20Docume…
File Edit Project OWL Code Tools Window Help
protégé
Metadata (wsdm_ontology) OWLClasses Properties Individuals Forms
CLASS BROWSER
For Project: wsdmdevelopmentontology
Class Hierarchy
List
Marquee
Menu (2)
Note
Citation (1)
NB
PS
Section
Summary
Table
DataTable
LayoutTable (1)
TableOfContent
INSTANCE BROWSER
For Class: LayoutTable
Asserted Inferred
Asserted Instances
WISE_SvenHomeLayoutTable
Asserted Types
LayoutTable
INDIVIDUAL EDITOR
For Individual: ayoutTable (instance of LayoutTable)
hasBehaviour
hasChild
WISE_SvenHomeTableRowElem
WISE_SvenHomeTableRowElem
WISE_SvenHomeTableRowElem
WISE_SvenHomeTableRowElem

Figure 30 - Sven's Homepage LayoutTable

Another type of element we encounter frequently on the WISE pages (header/footer/sidebar) and Sven Casteleyn's Homepage is the List of links. The hierarchical structure of a WSDM ontology List concept looks like this:



The List concept has several subtypes, some of which are new elements introduced in the course of this research. Of those, we used the UnorderedBullettedList and the UnorderedList concept the most. The basic list element, the ListItem, can refer to a NavigationReference concept which in turn can point to a Link concept that can have an InternalNode or an ExternalNode as its target (Figure 31).

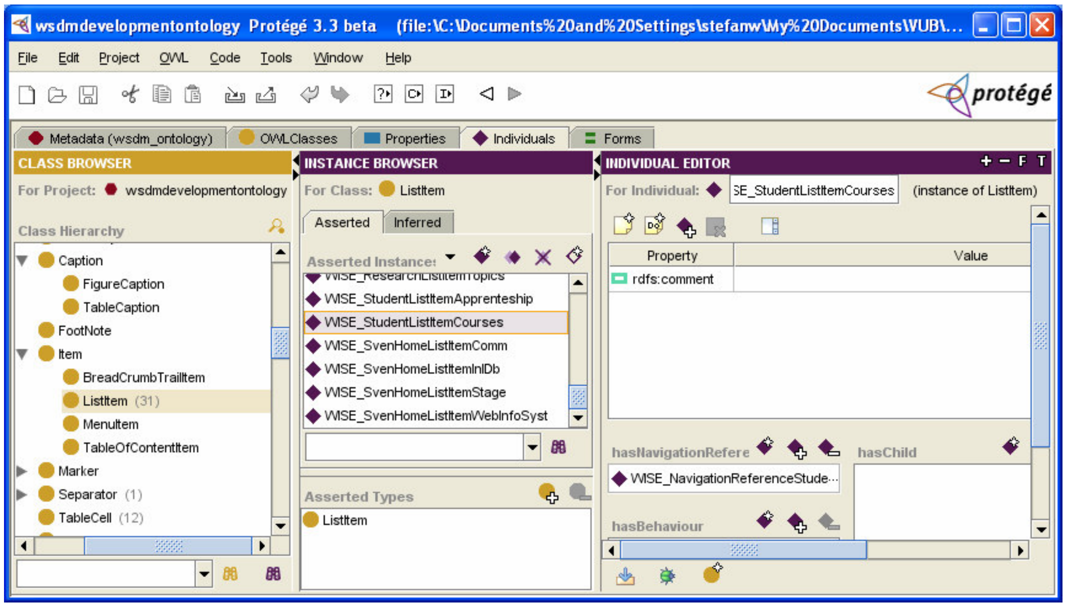

wsdmdevelopmentontology Protégé 3.3 beta (file:\C:\Documents%20and%20Settings\stefanw\My%20Documents\VUB\...
File Edit Project OWL Code Tools Window Help
Metadata (wsdm_ontology) OWLClasses Properties Individuals Forms
CLASS BROWSER
For Project: wsdmdevelopmentontology
Class Hierarchy
Caption
FigureCaption
TableCaption
FootNote
Item
BreadCrumbTrailItem
ListItem (31)
MenuItem
TableOfContentItem
Marker
Separator (1)
TableCell (12)
INSTANCE BROWSER
For Class: ListItem
Asserted Inferred
Asserted Instances
WISE_ResearchListItemTopics
WISE_StudentListItemApprenteship
WISE_StudentListItemCourses
WISE_SvenHomeListItemComm
WISE_SvenHomeListItemInIDb
WISE_SvenHomeListItemStage
WISE_SvenHomeListItemWebInfoSyst
Asserted Types
ListItem
INDIVIDUAL EDITOR
For Individual: SE_StudentListItemCourses (instance of ListItem)
Property Value
rdfs:comment
hasNavigationRefere
WISE_NavigationReferenceStude...
hasChild
hasBehaviour

Figure 31 - ListItem/NavigationReference combination

A WSDM Link concept has several subtypes, some of which were introduced during the course of this research. The one we use here is the NavigationalAidLink (Figure 32).

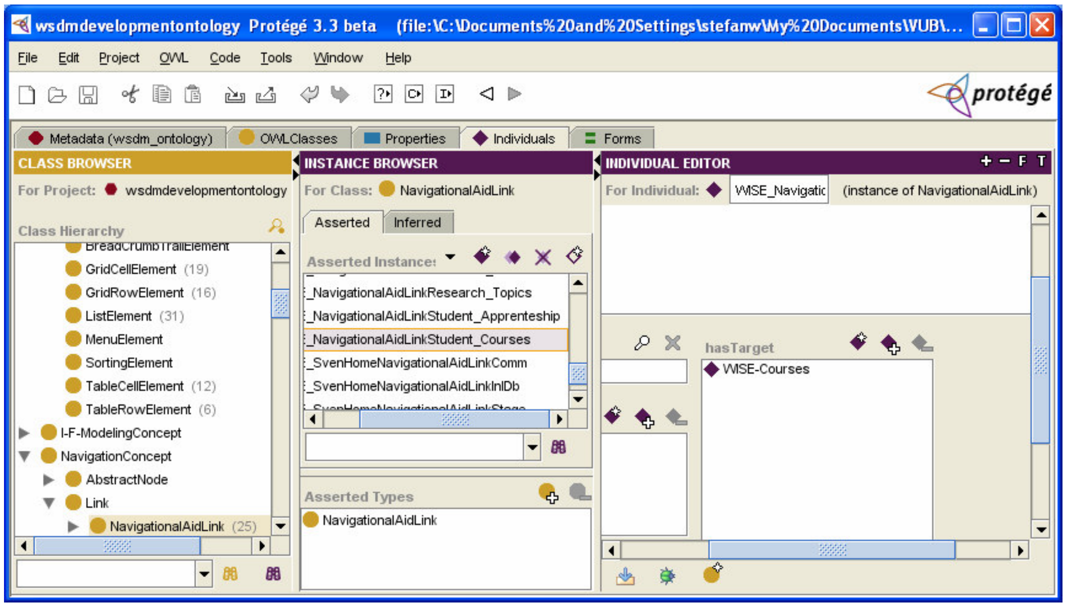

wsdmdevelopmentontology Protégé 3.3 beta (file:\C:\Documents%20and%20Settings\stefanw\My%20Documents\VUB\...
File Edit Project OWL Code Tools Window Help
Metadata (wsdm_ontology) OWLClasses Properties Individuals Forms
CLASS BROWSER
For Project: wsdmdevelopmentontology
Class Hierarchy
BreadCrumbTrailElement
GridCellElement (19)
GridRowElement (16)
ListElement (31)
MenuElement
SortingElement
TableCellElement (12)
TableRowElement (6)
I-F-ModelingConcept
NavigationConcept
AbstractNode
Link
NavigationalAidLink (25)
INSTANCE BROWSER
For Class: NavigationalAidLink
Asserted Inferred
Asserted Instances
_NavigationalAidLinkResearch_Topics
_NavigationalAidLinkStudent_Apprenteship
_NavigationalAidLinkStudent_Courses
_SvenHomeNavigationalAidLinkComm
_SvenHomeNavigationalAidLinkInIDb
Asserted Types
NavigationalAidLink
INDIVIDUAL EDITOR
For Individual: WISE_Navigatic (instance of NavigationalAidLink)
hasTarget
WISE-Courses

Figure 32 - Link example

Although not going into the smallest detail of every page of the WISE Website, this section's purpose is to give the reader an insight into how a Webpage can be described using the WSDM ontology. The end result is a collection of WSDM ontology individuals. This is also exactly what would happen if a new Website is developed using the WSDM design methodology: the end result in the WSDM ontology would be a collection of individuals.

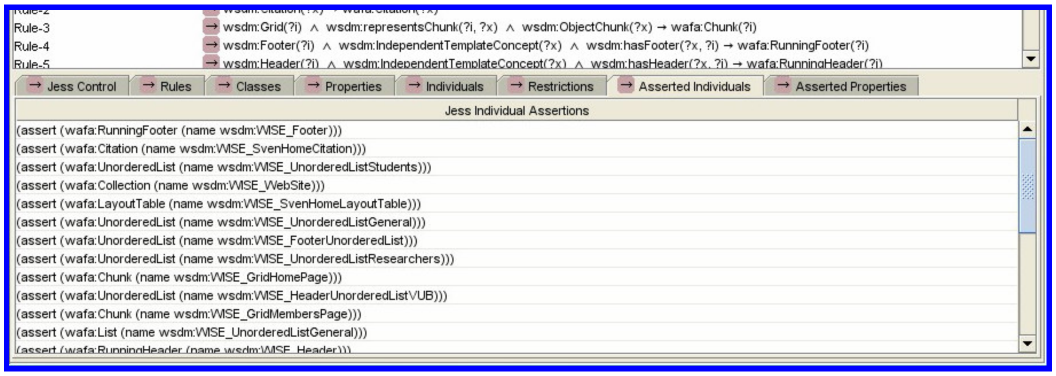
In total, 237 WSDM ontology individuals were created. Broken down in separate classes/concepts this gives us the following list:

WSDM Class/Concept	Individuals
Cell	8
Citation	1
ContentPane	3
CustomSeparator	1
Email	1
ExternalNode	7
Footer	1
Grid	8
GridCellElement	19
GridRowElement	16
Header	1
Header-LeftSideBar-Footer-Template	2
Image	7
InternalNode	12
LayoutTable	1
LeftSideBar	2
ListItem	31
Menu	2
NavigationalAidLink	25
NavigationReference	26
ObjectChunk	3
Page	3
PageRegion	1
RootNode	2
Row	16
Separator	2
String	6
TableCell	12
TableRow	6
Title	2
UnorderedBulletedList	3
UnorderedList	6
Website	1
Total	**237**

5.3.2 Running the rule set

The next step is the introduction and execution of the mapping rules that map WSDM ontology concepts or combinations of WSDM ontology concepts to their WAfA ontology counterparts. How a mapping rule is introduced is already explained in the beginning of this chapter. Important to note is that although more mapping rules were introduced, it is not expected that all of them would actually fire because in many cases the WSDM ontology individual necessary to trigger the rule is not present. For example the pages modeled here do not contain anything in the form of BreadCrumbTrails so we do not expect any of those rules to fire (and consequently we do not expect any of those WAfA individuals to be created as a result).

The result after the execution of our rule set is a collection of new WAfA ontology individuals (which can be clearly identified as “Asserted Individuals” in our tool of choice – see Figure 33, Figure 34 and Figure 35 - New WAfA individuals). These WAfA ontology individuals have no other origin than the execution of the rules in combination with the introduced WSDM ontology individuals.

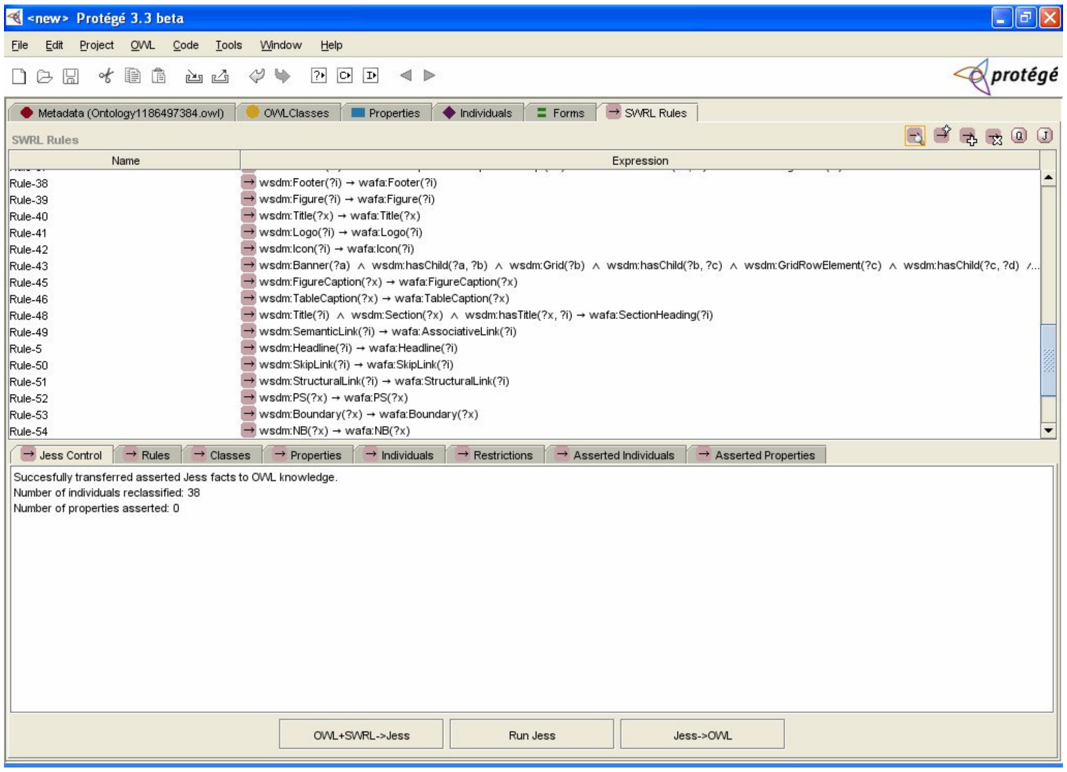


Rule-3	wsdm:Grid(?i) ∧ wsdm:representsChunk(?i, ?x) ∧ wsdm:ObjectChunk(?x) → wafa:Chunk(?i)
Rule-4	wsdm:Footer(?i) ∧ wsdm:IndependentTemplateConcept(?x) ∧ wsdm:hasFooter(?x, ?i) → wafa:RunningFooter(?i)
Rule-5	wsdm:Header(?i) ∧ wsdm:IndependentTemplateConcept(?x) ∧ wsdm:hasHeader(?x, ?i) → wafa:RunningHeader(?i)

Jess Individual Assertions

```
(assert (wafa:RunningFooter (name wsdm:WISE_Footer)))
(assert (wafa:Citation (name wsdm:WISE_SvenHomeCitation)))
(assert (wafa:UnorderedList (name wsdm:WISE_UnorderedListStudents)))
(assert (wafa:Collection (name wsdm:WISE_WebSite)))
(assert (wafa:LayoutTable (name wsdm:WISE_SvenHomeLayoutTable)))
(assert (wafa:UnorderedList (name wsdm:WISE_UnorderedListGeneral)))
(assert (wafa:UnorderedList (name wsdm:WISE_FooterUnorderedList)))
(assert (wafa:UnorderedList (name wsdm:WISE_UnorderedListResearchers)))
(assert (wafa:Chunk (name wsdm:WISE_GridHomePage)))
(assert (wafa:UnorderedList (name wsdm:WISE_HeaderUnorderedListVUB)))
(assert (wafa:Chunk (name wsdm:WISE_GridMembersPage)))
(assert (wafa:List (name wsdm:WISE_UnorderedListGeneral)))
(assert (wafa:RunningHeader (name wsdm:WISE_Header)))
```

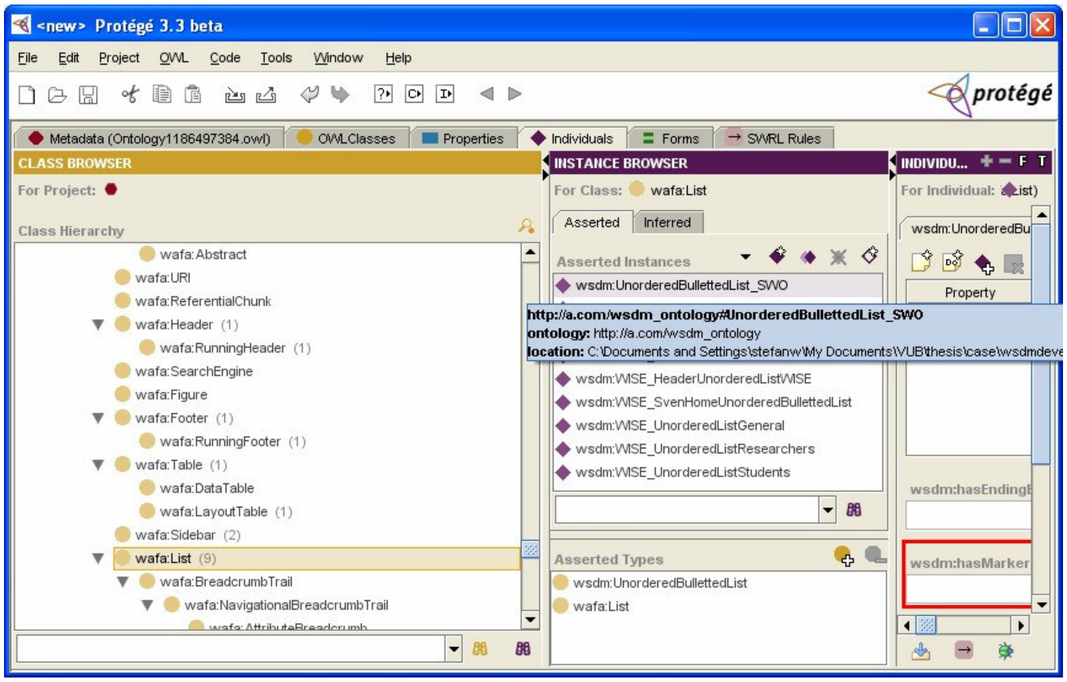
Figure 33 - Rule Execution Results



Name	Expression
Rule-38	wsdm:Footer(?i) → wafa:Footer(?i)
Rule-39	wsdm:Figure(?i) → wafa:Figure(?i)
Rule-40	wsdm:Title(?x) → wafa:Title(?x)
Rule-41	wsdm:Logo(?i) → wafa:Logo(?i)
Rule-42	wsdm:Icon(?i) → wafa:Icon(?i)
Rule-43	wsdm:Banner(?a) ∧ wsdm:hasChild(?a, ?b) ∧ wsdm:Grid(?b) ∧ wsdm:hasChild(?b, ?c) ∧ wsdm:GridRowElement(?c) ∧ wsdm:hasChild(?c, ?d) ∧...
Rule-45	wsdm:FigureCaption(?x) → wafa:FigureCaption(?x)
Rule-46	wsdm:TableCaption(?x) → wafa:TableCaption(?x)
Rule-48	wsdm:Title(?i) ∧ wsdm:Section(?x) ∧ wsdm:hasTitle(?x, ?i) → wafa:SectionHeading(?i)
Rule-49	wsdm:SemanticLink(?i) → wafa:AssociativeLink(?i)
Rule-5	wsdm:Headline(?i) → wafa:Headline(?i)
Rule-50	wsdm:SkipLink(?i) → wafa:SkipLink(?i)
Rule-51	wsdm:StructuralLink(?i) → wafa:StructuralLink(?i)
Rule-52	wsdm:PS(?x) → wafa:PS(?x)
Rule-53	wsdm:Boundary(?x) → wafa:Boundary(?x)
Rule-54	wsdm:NB(?x) → wafa:NB(?x)

```
Succesfully transferred asserted Jess facts to OWL knowledge.
Number of individuals reclassified: 38
Number of properties asserted: 0
```

Figure 34- Rule Execution Results (cont'd)



wafa:Abstract
wafa:URI
wafa:ReferentialChunk
wafa:Header (1)
wafa:RunningHeader (1)
wafa:SearchEngine
wafa:Figure
wafa:Footer (1)
wafa:RunningFooter (1)
wafa:Table (1)
wafa:DataTable
wafa:LayoutTable (1)
wafa:Sidebar (2)
wafa:List (9)
wafa:BreadcrumbTrail
wafa:NavigationalBreadcrumbTrail
wafa:AttributeBreadcrumb

wsdm:UnorderedBullettedList_SWO

http://a.com/wsdm_ontology#UnorderedBullettedList_SWO
ontology: http://a.com/wsdm_ontology
location: C:\Documents and Settings\stefanw\My Documents\VUB\thesis\case\wsdmdeve

wsdm:WISE_HeaderUnorderedListWISE
wsdm:WISE_SvenHomeUnorderedBullettedList
wsdm:WISE_UnorderedListGeneral
wsdm:WISE_UnorderedListResearchers
wsdm:WISE_UnorderedListStudents

wsdm:UnorderedBu

wsdm:UnorderedBullettedList
wafa:List

Figure 35 - New WAfA individuals

After running the rules and transferring that knowledge back to the ontologies (using SWRLTab/Jess) we observe 38 newly created WAfA ontology individuals.

WAfA Class/Concept	Individuals
Chunk	3
Citation	1
Collection	1
Footer	1
Header	1
Heading	2
LayoutTable	1
List	9
Node	3
Note	1
RunningFooter	1
RunningHeader	1
Separator	2
SideBar	2
Table	1
Title	2
UnorderedList	6
Total	38

The newly-created WAfA ontology individuals have the same name as the WSDM ontology individuals they originated from, so their conception can always be traced back to the WSDM ontology individual. For example, the WSDM ontology Page concept *WISE_PageSvenHome* maps to the WAfA ontology Node concept *WISE_PageSvenHome*. This leaves us some sort of structure so we are not left with a bag of WAfA ontology individuals all called *Generated_Individual_1*, *Generated_Individual_2*, etc…

SWRLTab/Jess does not explicitly say which rules were fired and which weren't. This can be deducted however by looking at the "Asserted Individuals" Tab. The fired rules are the following:

WSDM/WAfA fired rules
∀ i ∈ I: wsdm:Page(i) → wafa:Node(i)
∀ i ∈ I: wsdm:WebSite(i) → wafa:Collection(i)
∀ i ∈ I: wsdm:Header(i) → wafa:Header(i)
∀ i ∈ I, ∃ x ∈ I: wsdm:Header(i) ^ wsdm:IndependantTemplateConcept(x) ^ wsdm:hasHeader(x, i) → wafa:RunningHeader(i)
∀ i ∈ I: wsdm:Footer(i) → wafa:Footer(i)
∀ i ∈ I, ∃ x ∈ I: wsdm:Footer(i) ^ wsdm:IndependantTemplateConcept(x) ^ wsdm:hasFooter(x, i) → wafa:RunningFooter(i)
∀ i ∈ I: wsdm:Separator(i) → wafa:Separator(i)
∀ i ∈ I, ∃ x ∈ I: wsdm:Grid(i) ^ wsdm:representsChunk(I, x) ^ wsdm:ObjectChunk(x) → wafa:Chunk(i)
∀ i ∈ I, ∃ x ∈ I: wsdm:Title(i) ^ wsdm:Page(x) ^ wsdm:hasTitle(x, i) → wafa:Heading(i)
∀ i ∈ I: wsdm:Note(i) → wafa:Note(i)
∀ i ∈ I: wsdm:Citation(i) → wafa:Citation(i)
∀ i ∈ I: wsdm:List(i) → wafa:List(i)
∀ i ∈ I: wsdm:UnorderedList(i) → wafa:UnorderedList(i)
∀ i ∈ I: wsdm:Title(i) → wafa:Title(i)
∀ i ∈ I: wsdm:SideBar(i) → wafa:Sidebar(i)
∀ i ∈ I: wsdm:Table(i) → wafa:Table(i)
∀ i ∈ I: wsdm:LayoutTable(i) → wafa:LayoutTable(i)

So at first sight, the rules executed correctly. Not only the basic one-on-one rules but also the more complex ones (e.g. ∀ i ∈ I, ∃ x ∈ I: wsdm:Header(i) ^ wsdm:IndependantTemplateConcept(x) ^ wsdm:hasHeader(x, i) → wafa:RunningHeader(i)). Note however that the rules in bold are not supposed to fire because they have no corresponding WSDM ontology individual to prompt their execution. This problem is discussed in the next section. In all it can be concluded that too many rules executed given the existing WSDM ontology individuals (there were no List, Table or Note individuals created during the WISE pages modeling phase).

5.3.3 Problems with rule execution

For some concepts which are the subtype of a parent concept, we expect the subtype to be created in the WAfA ontology. However, in practice, both subtype and supertype (or parent type) are instantiated as WAfA ontology individuals. This was the case with the List/UnorderedList and Table/LayoutTable concepts: in the WSDM ontology, we started from an UnorderedList individual *WISE_UnorderedListGeneral*. After rule execution however, we noticed a WAfA ontology UnorderedList individual *WISE_UnorderedListGeneral* with the same name (which is correct) but also a List individual *WISE_UnorderedListGeneral* with the same name (which is not correct). The same occurred with the LayoutTable individuals (e.g. *WISE_SvenHomeLayoutTable*).

Of the 81 rules, 4 had negations in them. With the remaining 77 rules it was discovered that there was a problem running some of them. In the end the problem was broken down to the wsdm:hasNavigationReference relationship and wsdm:String class. Any rule containing one or both caused errors and stopped the complete rule set from working. After elimination there remained 54 rules that executed without problems. This set was run for our example case.

The cause of the errors with the above mentioned elements was unclear. The wsdm:hasNavigationReference relationship error is probably due to an incorrectly edited WSDM ontology file. With more research this could be solved. However the cause of the wsdm:String class error is unclear, also necessitating more research.

5.3.4 Conclusion

In this chapter we set up and explained a few examples including a test case to test the viability of the mapping rule concept. It has been shown that this concept works outside a purely theoretical environment but its implementation is not 100% trouble-free and thus more work/research in this field is needed. It is our belief the hasNavigationReference/String problem can be solved, but the negation and super/subtype double creation both present a more fundamental problem. The next chapter draws a general conclusion and explains what future research would be of interest.

6 Conclusions and further work

This chapter presents a general conclusion about this thesis and possible avenues of future research.

6.1 Conclusion

The ever-increasing importance of Internet and the use of Websites have also put more emphasis on the problems that visually impaired users encounter while travelling the Web. During the course of this research we gave an overview of the technology and tools employed to help Visually impaired users with their internet travels.
By way of test, the WISE Website was evaluated using a specialised method to see how it would score with regards to the visually impaired. As a result of this test some suggestions were made as to what could be improved.
After determining the nature of the problem and the need to do something about it, we looked more closely at the WSDM Methodology. The Website Development Methodology developed at the VUB is a good place to start if you want to create a Website that is suitable for the visually impaired to navigate. For this reason, earlier research [1] linked this methodology to the Dante approach. The Dante approach is another effort to make the surfing life for the visually impaired a bit easier. It takes existing Webpages that are annotated using the WAfA ontology to produce “transformed” Webpages better suitable to be navigated by the blind or people with bad eyesight. WSDM also uses its own ontology as a container of the results of the design process. With both ontologies showing some similarities, they formed the connecting point between the two approaches, the goal being for Dante to continue where WSDM stops. To achieve this the information contained in the WSDM ontology needs to be transferred to the WAfA ontology. This is done by mapping the objects in the WSDM ontology to the objects in the WAfA ontology through special mapping rules. Some of these mapping rules already existed [1] attaining a coverage of 69%. The goal of this thesis was to achieve as high a coverage as possible, extending the WSDM methodology in the process.
We achieved this goal and set up a proof of concept to test the applicability of these mapping rules in practice. This example case showed that although difficult to implement at this stage the mapping rule concept does work (The complete set of mapping rules can be found in Appendix B).
The advantages of this mapping rule approach are:

- The combined effort of WSDM and Dante combined through the mapping rule concept is less than the sum of both separate efforts: the WAfA annotations do not need to be inserted manually because they are generated by the mapping rules.
- The mapping rule concept is a flexible one: ontologies and individual mapping rules can be changed, allowing fine tuning and facilitating maintenance – a necessity inherent to any IT system.
- Setting up the mapping rules is a one-time effort. It does not need to be repeated with every run of the combined WSDM/Dante process.

6.2 Future work

The completion of our experimental research opens up the path for some interesting future work.

Given the problems we encountered in the mapping rule implementation through the Protégé SWRLTab/Jess combination, this implementation needs to be re-evaluated. The development of an entirely new tool is unnecessary, since Protégé is still very useful for editing ontologies (e.g. the WSDM and WAfA ontologies). However, a bug-free Protégé plug-in (much like SWRLTab/Jess) that understands conjunction, disjunction and negation would be a step forward.

This in turn, though beyond the scope of this thesis, might prompt the research and development of a general tool that connects two ontologies through a set of mapping rules. Where two systems using the same ontology speak the same language, two systems using a different ontology speak a different language. So a translator is necessary in the form of a tool executing a set of mapping rules.

The research in this thesis focussed on a mapping between the WSDM ontology concepts and the WAfA ontology *Authoring* concepts. Future work can examine the possibilities of mappings between the WSDM ontology concepts and the WAfA ontology *Mobility* concepts.

6.3 Final word

After presenting our experimental research içn the form of this thesis it is the author's hope and belief that another step is made towards the design of Websites that make the surfing life for the visually impaired easer.

7 References

[1] Peter Plessers, Sven Casteleyn, Yeliz Yesilada, Olga De Troyer, Robert Stevens, Simon Harper and Carole Goble – Accessibility: A Web Engineering Approach. WWW 2005, May 10-14, 2005 Chiba -Japan. pages 357 - 361

[2] Sven Casteleyn, Peter Plessers and Olga De Troyer – Generating Semantic Annotations during the Web Design Process. ICWE'06 July 11-14, 2006 Palo Alto, California, USA.

[3] Human-Computer Interaction (second edition) – Alan Dix, Janet Finlay, Gregory Abowd and Russel Beale. ISBN 0-13-239864-8

[4] Christine Golbreich - Combining Rule and Ontology Reasoners for the Semantic Web. RuleML 2004. International workshop N°3, Hiroshima , Japan (08/11/2004)

[5] Judy Brewer – Web Accessibility Highlights and Trends. W4A at WWW2004, may 18th, 2004, New York, U.S.A.

[6] I.V. Ramakrishnan, Amanda Stent and Guizhen Yang – HearSay: Enabling Audio Browsing on Hypertext Content. WWW2004, may 17th – 22nd 2004, New York, U.S.A.

[7] Joseph Scheuhammer – Accessibility and Separating Form from Content. Adaptive Technology Resource Centre, july 6th, 2000.

[8] U.S. Department of Justice – A Guide to Disability Rights Laws, Rehabilitation Act, Section 508. http://www.usdoj.gov/crt/ada/cguide.htm#anchor65610 accessed 2007

[9] Logica voor Informatici (tweede editie) – J. F. A. K. Van Benthem, H. P. Van Ditmarsch, J. Ketting, W. P. M. Meyer-Viol. ISBN 90-7869-484-2

[10] Web Site Design Method WSDM – Prof. Dr. Olga De Troyer. Syllabus for the course Design Methods for Internet Based Information Systems – 2007.

[11] Use and Advantages of Ontology-based Web Design – unpublicised paper, see WISE

[12] The W3C Web Accessibility Initiative: http://www.w3.org/WAI/ (accessed 2006)

8 Appendices

8.1 Appendix A: Browser Tests

	MS Windows/IE	MS Windows/Opera	Linux/Mozilla
Normal	Everything looks ok though on http://wise.vub.ac.be/members/ some photos look distorted. Furthermore on the Web master and publications pages the menu fonts are different from the other pages.	Everything looks ok, though on http://wise.vub.ac.be/members/ some photos look distorted.	Everything looks ok, though on http://wise.vub.ac.be/members/ some photos look distorted.
Images off	http://wise.vub.ac.be/members.html no textual description for pictures; All other pages ok	http://wise.vub.ac.be/members.html some pictures have an “image” description, but most have no description at all.	http://wise.vub.ac.be/members/ no textual description replacing any of the images.
Varying font size	Internet Explorer offers limited font sizes: only 5 different settings ranging from smallest to largest. But even the “largest” setting comes nowhere near the 400%-setting of e.g. Mozilla thus limiting the use of the browser somewhat. Nevertheless, all pages were checked with the “largest” text size setting. On some pages certain portions of text did not resize (e.g. names on http://wise.vub.ac.be/members/ page). On the following pages nothing resizes at all: http://wise.vub.ac.be/researchers/publications.php, http://wise.vub.ac.be/metovr/default.htm, http://wise.vub.ac.be/webmaster.html On the proposals page http://wise.vub.ac.be/students/proposals.html titles do not resize either. The footer page does not resize on any page.	Opera offers a zoom function that enlarges everything on the page: not only the font size but images as well. A 300% as well as a 400% resize is possible. All pages check out ok with the radio buttons on the http://wise.vub.ac.be/webmaster.html page resizing as well. One minor drawback though is that the page as a whole resizes, resizing borders as well and making horizontal and vertical scrolling necessary.	Fonts at 400%: left column overflows in right column on most pages. Overall text resizes according to chosen specification. On http://wise.vub.ac.be/webmaster.html page, pushbuttons also resize, but not the radio buttons (their text resizes though).

Different screen resolution/resizing application	The browser was resized to a size which corresponds roughly to full screen mode in a 640*480 resolution screen setting[36]. A horizontal (as well as a vertical) scrollbar appeared on all pages. When resizing the browser, the content re-centers automatically as long as the site width is less than the browser screen width. When site width is larger than browser width, text is aligned left.	As with Internet Explorer, the browser was resized to a size which corresponds roughly to full screen mode in a 640*480 resolution screen setting. A horizontal (as well as a vertical) scrollbar appeared on all pages. When resizing the browser, the content re-centers automatically as long as the site width is less than the browser screen width. When site width is larger than browser width, as on IE, text is aligned left.	As with both Internet Explorer and Opera, the browser was resized to a size which corresponds roughly to full screen mode in a 640*480 resolution screen setting[37]. A horizontal (as well as a vertical) scrollbar appeared on all pages. When resizing the browser, the content re-centers automatically as long as the site width is less than the browser screen width. When site-width is larger than browser width, text is aligned left.
Change display color to gray scale (or print out page in gray scale or black and white	Due to the inability to change the color settings to grayscale on the test platform (Windows XP Laptop) all test pages were printed on a grayscale laser printer. Contrast is adequate (dark/black text on light/white background) but it is difficult to distinguish the clickable hyperlinks.	Same platform as Internet Explorer, so same problem: due to the inability to change the color settings to grayscale on the test platform (Windows XP Laptop) all test pages were printed on a grayscale laser printer in stead. Contrast is adequate (dark/black text on light/white background) but it is difficult to distinguish the clickable hyperlinks.	The printouts made from the Linux-running Mozilla were horrible: the fonts were much too small, the links of the left-hand side menu were barely visible and links in the main content weren't visible at all! To rule out printer problems a newer version was installed (now called Mozilla Firefox) on the Windows XP machine and again, printouts were made using the same laser printer as used with Internet Explorer and Opera. This produced better results; very much the same as with Internet Explorer but with much smaller fonts. The dark text on white background contrasts well but the color difference of hyperlinks is hardly noticeable.
No mouse	Mouseless navigation was found to be ok on all tested pages. The TAB order was logical (first top	The Opera browser can also be operated without a mouse but the	As with Internet Explorer, navigation without use of a mouse was found to

[36] Both Internet Explorer and Opera tests were performed on a Windows XP Laptop. The laptop did not allow the Windows resolution to be set to 640*480 so it was kept at 1024*768. In stead, the browser window was resized.

[37] The browser was resized to about 60% of the total screen area. Given the fact the linux platform ran in a 1024*768 pixels environment, this roughly corresponds to full screen mode in a 640*480 resolution screen setting.

	menu, then main menu on the left, then page content) and all links could be activated by pressing the ENTER-key.	key combinations differ somewhat from the other 2 tested browsers, requiring more time to get used to. As an alternative Opera does offer voice commands – it's a feature that can be installed and activated separately and allows the user to give voice commands to the browser[38] and/or have the browser read out selected sentences or words. Since things to be read out need to be selected with the mouse first however, the usability of this feature for the visually impaired is limited which makes it a doubtful alternative for a screen reader.	be ok on all tested pages. The TAB order was logical (first top menu, then main menu on the left, then page content) and all links could be activated by pressing the ENTER-key. Oddly, pushbuttons don't display the same visual effect when activated through the keyboard contrary to using a mouse. Given the context and the intended usage (visually impaired) this is not a problem.

Table 1: browser comparison

[38] See also the section on Voice Browsers 2.1.8

8.2 Appendix B: Mapping Rules between WSDM and WAfA

Nr	WAfA Ontology Concept	WSDM Ontology Concept	Mapping Rule
1	Object	N/A	N/A
2	AuthoringConcept	N/A	N/A
3	Atom	N/A	N/A
4	Advertisement	Advertisement	∀ i ∊ I: wsdm:Advertisement(i) → wafa: Advertisement(i)
5	AdvertisementBanner	Banner *	∀ a ∊ I, ∃ b, c, d, e, f, g ∊ I: wsdm:Banner(a) ^ wsdm:hasChild(a, b) ^ wsdm:Grid(b) ^ wsdm:hasChild(b, c) ^ wsdm:GridRow(c) ^ wsdm:hasChild(c, d) ^ wsdm:Row(d) ^ wsdm:hasChild(d, e) ^ wsdm:GridCellElement(e) ^ wsdm:hasChild(e, f) ^ wsdm:Cell(f) ^ hasChild(f, g) ^ wsdm:Advertisement(g) → wafa:AdvertisementBanner(a)
6	Caption	Caption	∀ i ∊ I: wsdm:Caption(i) → wafa:Caption(i)
7	FigureCaption	FigureCaption	∀ i ∊ I: wsdm:FigureCaption(i) → wafa:FigureCaption(i)
8	TableCaption	TableCaption	∀ i ∊ I: wsdm:TableCaption(i) → wafa:TableCaption(i)
9	Heading	Title *	∀ i ∊ I, ∃ x ∊ I: wsdm:Title(i) ^ wsdm:Page(x) ^ wsdm:hasTitle(x, i) → wafa:Heading(i)
10	Headline	Headline	∀ i ∊ I: wsdm:Headline(i) → wafa:Headline(i)
11	SectionHeading	String or Image *	∀ i ∊ I: wsdm:Title(i) ^ wsdm:Section(x) ^ wsdm:hasTitle(x, i) → wafa:SectionHeading(i)
12	Label	String or Image *	∀ i ∊ I, ∃ x ∊ I: (wsdm:String(i) ∨ wsdm:Image(i)) ^ (wsdm:Textbox(x) ∨ wsdm:Checkbox(i) ∨ wsdm:Radiobutton(x) ∨ wsdm:Pushbutton(x)) ^ wsdm:hasLabel(x, i) → wafa:Label(i)
13	Link	Link	∀ i ∊ I, ∃ x ∊ I: wsdm:hasNavigationReference(i, x) ^ wsdm:NavigationReference(x) → wafa:Link(i)
14	AssociativeLink	SemanticLink	∀ i ∊ I: wsdm:SemanticLink(i) → wafa:AssociativeLink(i)
15	ReferentialLink	Link *	∀ i ∊ I, ∃ x, y ∊ I: wsdm:Link(i) ^ wsdm:hasSource(i, x) ^ wsdm:hasTarget(i, y) ^ wsdm:Node(x) ^ wsdm:Node(y) ^ ∀ u, v ∊ I: wsdm:hasChunk(x, u) ^ wsdm:hasChunk(y, v) ^ (∀ a, b ∊ C: (wsdm:isComposedOf(u, a) → wsdm:isComposedOf(v, b)) ^ ∃ c ∊ C: wsdm:isComposedOf(v, c) ^ ¬wsdm:isComposedOf(u, c)) → wafa:ReferentialLink(i)
16	SkipLink	SkipLink	∀ i ∊ I: wsdm:SkipLink(i) → wafa:SkipLink(i)
17	StructuralLink	StructuralLink	∀ i ∊ I: wsdm:StructuralLink(i) → wafa:StructuralLink(i)

18	ToTextOnlyPage	Link *	∀ l, is, it, cs, ct, is, it, gs, gt, cls, clt, jt ∊ I, ∃ js ∊ I: wsdm:Link(l) ^ wsdm:hasSource(l, is) ^ wsdm:InternalNode(is) ^ wsdm:hasTarget(l, it) ^ wsdm:InternalNode(it) ^ wsdm:hasChunk(is, cs) ^ wsdm:ObjectChunk(cs) ^ wsdm:hasChunk(it, ct) ^ wsdm:ObjectChunk(ct) ^ wsdm:Grid(gs) ^ wsdm:representsChunk(gs, cs) ^ wsdm:Grid(gt) ^ wsdm:representsChunk(gt, ct) ^ wsdm:hasChild(gs, cls) ^ wsdm:Cell(cls) ^ wsdm:hasChild(gt, clt) ^ wsdm:Cell(clt) ^ wsdm:hasChild(cls, js) ^ (wsdm:MultiMediaConcept(js) ∨ wsdm:Graphic(js)) ^ wsdm:hasChild(clt, jt) ^ ¬(wsdm:Graphic(jt) ^ wsdm:Image(jt) ^ wsdm:Video(js)) à wafa:ToTextOnlyPage(l)
19	Note	Note	∀ i ∊ I: wsdm:Note(i) → wafa:Note(i)
20	Citation	Citation	∀ i ∊ I: wsdm:Citation(i) → wafa:Citation(i)
21	FootNote	Footnote	∀ i ∊ I: wsdm:Footnote(i) → wafa:Footnote(i)
22	Copyright	Copyright	∀ i ∊ I: wsdm:CopyRight(i) → wafa:CopyRight(i)
23	NB	NB	∀ i ∊ I: wsdm:NB(i) → wafa:NB(i)
24	PS	PS	∀ i ∊ I: wsdm:PS(i) → wafa:PS(i)
25	Separator	Separator	∀ i ∊ I: wsdm:Separator(i) → wafa:Separator(i)
26	Boundary	Boundary	∀ i ∊ I: wsdm:Boundary(i) → wafa:Boundary(i)
27	Banner	Banner	∀ i ∊ I: wsdm:Banner(i) → wafa:Banner(i)
	AdvertisementBanner		see 5
28	TitleBanner	Banner *	∀ i ∊ I, ∃ x, y, z ∊ I: wsdm:Banner(i) ^ wsdm:hasChild(i, x) ^ (wsdm:String(x) ∨ wsdm:Image(x)) ^ wsdm:Page(y) ^ wsdm:hasTitle(y, z) ^ wsdm:representedBy(z, x) → wafa:TitleBanner(i)
29	Space	CustomSeparator *	∀ i ∊ I, ∃ x ∊ I: wsdm:CustomSeparator(i) ^ wsdm:representedBy(i, x) ^ wsdm:String(x) ^ wsdm:hasValue(x, “ “)→ wafa:Space(i)
30	SymbolSeparator (used to be Symbol)	Separator *	∀ i ∊ I, ∃ x ∊ I: wsdm:Separator(i) ^ wsdm:representedBy(i, x) ^ wsdm:String(x) → wafa:Separator(i)
31	CommaSeparator	CustomSeparator *	∀ i ∊ I, ∃ x ∊ I: wsdm:CustomSeparator(i) ^ wsdm:representedBy(i, x) ^ wsdm:String(x) ^ wsdm:hasValue(x, “,”) → wafa:CommaSeparator(i)
32	DashSeparator	CustomSeparator *	∀ i ∊ I, ∃ x ∊ I: wsdm:CustomSeparator(i) ^ wsdm:representedBy(i, x) ^ wsdm:String(x) ^ wsdm:hasValue(x, “–”) → wafa:CommaSeparator(i)
33	TriangleLeftSeparator	CustomSeparator *	∀ i ∊ I, ∃ x ∊ I: wsdm:CustomSeparator(i) ^ wsdm:representedBy(i, x) ^ wsdm:String(x) ^ wsdm:hasValue(x, “<”) → wafa:CommaSeparator(i)
34	TriangleRightSeparator	CustomSeparator *	∀ i ∊ I, ∃ x ∊ I: wsdm:CustomSeparator(i) ^ wsdm:representedBy(i, x) ^ wsdm:String(x) ^ wsdm:hasValue(x, “>”) → wafa:CommaSeparator(i)
35	VerticalBarSeparator	CustomSeparator *	∀ i ∊ I, ∃ x ∊ I: wsdm:CustomSeparator(i) ^ wsdm:representedBy(i, x) ^ wsdm:String(x) ^ wsdm:hasValue(x, “|”) → wafa:CommaSeparator(i)
36	SpecialGraphic	N/A	N/A

	Banner		see 27
	AdvertisementBanner		see 5
	TitleBanner		see 28
37	Icon	Icon	∀ i ∈ I: wsdm:Icon(i) → wafa:Icon(i)
38	Logo	Logo	∀ i ∈ I: wsdm:Logo(i) → wafa:Logo(i)
39	Title	Title	∀ i ∈ I: wsdm:Title(i) → wafa:Title(i)
	TitleBanner		see 28
40	Chunk	Grid *	∀ i ∈ I, ∃ x ∈ I: wsdm:Grid(i) ^ wsdm:representsChunk(I, x) ^ wsdm:ObjectChunk(x) → wafa:Chunk(i)
41	Figure	Figure	∀ i ∈ I: wsdm:Figure(i) → wafa:Figure(i)
42	Footer	Footer	∀ i ∈ I: wsdm:Footer(i) → wafa:Footer(i)
43	RunningFooter	Footer *	∀ i ∈ I, ∃ x ∈ I: wsdm:Footer(i) ^ wsdm:IndependantTemplateConcept(x) ^ wsdm:hasFooter(x) → wafa:RunningFooter(i)
44	Header	Header	∀ i ∈ I: wsdm:Header(i) → wafa:Header(i)
45	RunningHeader	Header *	∀ i ∈ I, ∃ x ∈ I: wsdm:Header(i) ^ wsdm:IndependantTemplateConcept(x) ^ wsdm:hasHeader(x, i) → wafa:RunningHeader(i)
46	List	List	∀ i ∈ I: wsdm:List(i) → wafa:List(i)
47	BreadcrumbTrail	BreadcrumbTrail	∀ i ∈ I: wsdm:BreadCrumbTrail(i) → wafa: BreadCrumbTrail(i)
48	DefinitionList	List *	∀ a ∈ I, ∃ b, c, d, e ∈ I: wsdm:List(a) ^ wsdm:hasChild(a, b) ^ wsdm:ListElement(b) ^ wsdm:hasChild(b, c) ^ wsdm:ListItem(c) ^ wsdm:hasChild(b, d) ^ wsdm:ListItem(d) ^ ¬wsdm:hasNavigationReference(c,) ^ ¬wsdm:hasNavigationReference(d,) → wafa:DefinitionList(a)
49	NavigationalList	List *	∀ a ∈ I, ∃ b, c, d, e ∈ I: wsdm:List(a) ^ wsdm:hasChild(a, b) ^ wsdm:ListElement(b) ^ wsdm:hasChild(b, c) ^ wsdm:ListItem(c) ^ wsdm:hasNavigationReference(c, d) ^ wsdm:NavigationReference(d) ^ wsdm:pointingToLink(d, e) ^ wsdm:Link(e) → wafa:NavigationalList(a)
50	Bookmark	List *	∀ a ∈ I, ∃ b, c, d, e ∈ I: wsdm:List(a) ^ wsdm:hasChild(a, b) ^ wsdm:ListElement(b) ^ wsdm:hasChild(b, c) ^ wsdm:ListItem(c) ^ wsdm:hasNavigationReference(c, d) ^ wsdm:NavigationReference(d) ^ wsdm:pointingToLink(d, e) ^ wsdm:NavigationalAidBookmarkLink(e) → wafa:Bookmark(a)

			∀ a ∈ I, ∃ b, c, d, e ∈ I: wsdm:List(a) ^ wsdm:hasChild(a, b) ^ wsdm:ListElement(b) ^ wsdm:hasChild(b, c) ^ wsdm:ListItem(c) ^ wsdm:hasNavigationReference(c, d) ^ wsdm:NavigationReference(d) ^ wsdm:pointingToLink(d, e) ^ wsdm:ProcessLogicBookmarkLink(e) → wafa:Bookmark(a)
51	Favourites	List *	∀ a ∈ I, ∃ b, c, d, e ∈ I: wsdm:List(a) ^ wsdm:hasChild(a, b) ^ wsdm:ListElement(b) ^ wsdm:hasChild(b, c) ^ wsdm:ListItem(c) ^ wsdm:hasNavigationReference(c, d) ^ wsdm:NavigationReference(d) ^ wsdm:pointingToLink(d, e) ^ wsdm:FavouriteLink(e) → wafa:Favourites(a)
52	ShoppingCart	List *	∀ a ∈ I, ∃ b, c, d, e ∈ I: wsdm:List(a) ^ wsdm:hasChild(a, b) ^ wsdm:ListElement(b) ^ wsdm:hasChild(b, c) ^ wsdm:ListItem(c) ^ wsdm:hasNavigationReference(c, d) ^ wsdm:NavigationReference(d) ^ wsdm:pointingToLink(d, e) ^ wsdm:ShoppingCartLink(e) → wafa:ShoppingCart(a)
53	Directory	List *	∀ a ∈ I, ∃ b, c, d, e, f ∈ I: wsdm:List(a) ^ wsdm:hasChild(a, b) ^ wsdm:ListElement(b) ^ wsdm:hasChild(b, c) ^ wsdm:ListItem(c) ^ wsdm:hasNavigationReference(c, d) ^ wsdm:NavigationReference(d) ^ wsdm:pointingToLink(d, e) ^ wsdm:Link(e) ^ wsdm:hasTarget(e, f) ^ wsdm:AbstractNode(f) → wafa:Directory(a)
54	SiteMap	List *	∀ a ∈ I, ∃ b, c, d, e, f ∈ I: wsdm:List(a) ^ wsdm:hasChild(a, b) ^ wsdm:ListElement(b) ^ wsdm:hasChild(b, c) ^ wsdm:ListItem(c) ^ wsdm:hasNavigationReference(c, d) ^ wsdm:NavigationReference(d) ^ wsdm:pointingToLink(d, e) ^ wsdm:Link(e) ^ wsdm:hasTarget(e, f) ^ wsdm:InternalNode(f) → wafa:Directory(a)
55	WebDirectory	List *	∀ a ∈ I, ∃ b, c, d, e, f ∈ I: wsdm:List(a) ^ wsdm:hasChild(a, b) ^ wsdm:ListElement(b) ^ wsdm:hasChild(b, c) ^ wsdm:ListItem(c) ^ wsdm:hasNavigationReference(c, d) ^ wsdm:NavigationReference(d) ^ wsdm:pointingToLink(d, e) ^ wsdm:Link(e) ^ wsdm:hasTarget(e, f) ^ wsdm:ExternalNode(f) → wafa:Directory(a)
56	FAQ	FAQ	∀ i ∈ I: wsdm:FAQ(i) → wafa:FAQ(i)
57	HistoryList	HistoryList	∀ i ∈ I: wsdm:HistoryList(i) → wafa:HistoryList(i)
	PathBreadcrumb		see 65
58	Index	OrderedList	∀ i ∈ I, ∃ x, y ∈ I: wsdm:OrderedList(i) ^ wsdm:hasChild(i, x) ^ wsdm:ListItem(x) ^ wsdm:hasNavigationReference(x, y) ^ wsdm:NavigationReference(y) ^ (wsdm:pointingToNode(y,) ∨ wsdm:pointingToLink(y,)) → wafa:Index(i)

59	SiteIndex	OrderedList	∀ i ∊ I, ∃ x, y, z ∊ I: wsdm:OrderedList(i) ^ wsdm:hasChild(i, x) ^ wsdm:ListItem(x) ^ wsdm:hasNavigationReference(x, y) ^ wsdm:NavigationReference(y) ^ wsdm:pointingToNode(y, z) ^ wsdm:InternalNode(z) → wafa:SiteIndex(i)
60	LinkMenu	Menu *	∀ i, x ∊ I, ∃ y ∊ I: wsdm:Menu(i) ^ wsdm:representedBy(i, y) ^ wsdm:List(y) ^ ¬wsdm:hasBehavior(y, x) → wafa:LinkMenu(i)
61	DropDownLinkMenu	Menu *	∀ i ∊ I, ∃ x, y, z, u ∊ I: wsdm:Menu(i) ^ wsdm:representedBy(i, x) ^ wsdm:List(x) ^ wsdm:hasBehavior(x, y) ^ wsdm:Behavior(y) ^ wsdm:onEvent(y, z) ^ wsdm:Event(z) ^ wsdm:hasValue('onClick') ^ wsdm:doAction(y, u) ^ wsdm:Action(u) ^ wsdm:hasValue(u, 'dropDown') → wafa:LinkMenu(i)
62	NavigationalBreadcrumbTrail	NavigationalBreadcrumbTrail	∀ i ∊ I: wsdm:NavigationBreadCrumbTrail(i) → wafa:NavigationalBreadcrumbTrail(i)
63	AttributeBreadcrumb	AttributeBreadcrumbTrail	∀ ι ∊)ι(λιαρΤβμυρχδαερΒετυβιρττΑ:μδσω :Ι ◊)ι(βμυρχδαερΒετυβιρττΑ:αφαω
64	LocationBreadcrumb	LocationBreadcrumbTrail	∀ ι ∊)ι(λιαρΤβμυρχδαερΒνοιταχοΛ:μδσω :Ι ◊)ι(βμυρχδαερΒνοιταχοΛ:αφαω
65	PathBreadcrumb	PathBreadcrumbTrail	∀ ι ∊)ι(λιαρΤβμυρχδαερΒηταΠ:μδσω :Ι ◊)ι(βμυρχδαερΒηταΠ:αφαω
66	TableOfContent	NavigationTableOfContent	∀ i, ∊ I: wsdm:NavigationTableOfContent(i) → wafa:TableOfContent(i)
67	Toolbar	Menu *	∀ i ∊ I, ∃ x, y, a, b, c, d ∊ I: wsdm:Menu(i) ^ wsdm:representedBy(i, x) ^ wsdm:List(x) ^ wsdm:hasBehavior(x, y) ^ wsdm:Behavior(y) ^ wsdm:onEvent(y, a) ^ wsdm:Event(a) ^ hasValue(a, 'onClick') ^ wsdm:doAction(y, b) ^ wsdm:Action(b) ^ wsdm:hasChild(i, c) ^ wsdm:MenuItem(c) ^ wsdm:hasIcon(c, d) ^ wsdm:Icon(d) → wafa:Toolbar(i)
68	OrderedList	OrderedList	∀ i ∊ I: wsdm:OrderedList(i) → wafa:OrderedList(i)
	AttributeBreadcrumb		see 63
	Directory		see 53
	SiteMap		see 54
	WebDirectory		see 55
	HistoryList		see 57
	PathBreadcrumb		see 65
	Index		see 58
	SiteIndex		see 59
	LocationBreadcrumb		see 64
	TableOfContent		see 66
69	UnorderedList	UnordenedList	∀ i ∊ I: wsdm:UnorderedList(i) → wafa:UnorderedList(i)

70	ReferentialChunk	Grid represents ObjectChunk	∀ a ∈ I, ∃ b, c, d, e, f, g, h, I, j, k, l ∈ I: wsdm:Grid(a) ^ wsdm:representsObjectchunk(a, b) ^ wsdm:ObjectChunk(b) ^ wsdm:hasChild(a, c) ^ wsdm:GridRowElement(c) ^ wsdm:hasChild(c, d) ^ wsdm:Row(d) ^ wsdm:hasChild(d, e) ^ wsdm:GridCellElement(e) ^ wsdm:hasChild(e, f) ^ wsdm:Cell(f) ^ wsdm:hasChild(f, g) ^ wsdm:IndependentComplexPresentationConcept(g) ^ wsdm:hasChild(g, h) ^ wsdm:Section(h) ^ wsdm:hasChild(h, i) ^ wsdm:Summary(i) ^ wsdm:hasNavigationReference(f, j) ^ wsdm:NavigationReference(j) ^ wsdm:pointingToNode(j, k) ^ wsdm:AbstractNode(k) ^ wsdm:hasChild(k, l) ^ wsdm:InternalNode(l) → wafa:ReferentialChunk(a)
71	SearchEngine	SearchEngine	∀ i ∈ I: wsdm:SearchEngine(i) → wafa:SearchEngine(i)
72	Section	Section ??	∀ i ∈ I: wsdm:Section(i) → wafa:Section(i)
73	Abstract	Abstract	∀ i ∈ I: wsdm:Abstract(i) → wafa:Abstract(i)
74	Sidebar	Sidebar	∀ i ∈ I: wsdm:Sidebar(i) → wafa:Sidebar(i)
75	Summary	Summary	∀ i ∈ I: wsdm:Summary(i) → wafa:Summary(i)
	Abstract		see 73
76	PageSummary	PageSummary	∀ i ∈ I: wsdm:PageSummary(i) → wafa:PageSummary(i)
77	SiteSummary	SiteSummary	∀ i ∈ I: wsdm:SiteSummary(i) → wafa:SiteSummary(i)
78	Table	Table	∀ i ∈ I: wsdm:Table(i) → wafa:Table(i)
79	DataTable	DataTable	∀ i ∈ I: wsdm:DataTable(i) → wafa:DataTable(i)
80	LayoutTable	LayoutTable	∀ i ∈ I: wsdm:LayoutTable(i) → wafa:LayoutTable(i)
81	URI	ExternalNode	∀ i ∈ I: wsdm:ExternalNode(i) ^ wsdm:refersToURI(i, string) → wafa:URI(i)
82	Collection	Website	∀ i ∈ I: wsdm:Website(i) → wafa:Collection(i)
83	Node	Page	∀ i ∈ I: wsdm:Page(i) → wafa:Node(i)
84	HomePage	Page *	∀ i ∈ I, ∃ x ∈ I: wsdm:Page(i) ^ wsdm:hasNode(i, x) ^ wsdm:RootNode(x) → wafa:HomePage(i)