

The copyright of this thesis vests in the author. No quotation from it or information derived from it is to be published without full acknowledgement of the source. The thesis is to be used for private study or non-commercial research purposes only.

Published by the University of Cape Town (UCT) in terms of the non-exclusive license granted to UCT by the author.

Transcoding multilingual and non-standard web content to VoiceXML

Mduduzi E. Nxumalo

Speech Technology and Research Group
Department of Electrical Engineering
University of Cape Town
South Africa
February 2010



This dissertation is submitted to the University of Cape Town in fulfillment of the academic requirements for the Degree of Master of Science in Engineering.

Declaration

The work in this thesis is based on research carried out in the Speech Technology And Research (STAR) Group, in the Department of Electrical Engineering at the University of Cape Town, in South Africa. To the author's knowledge, there is no part of this thesis that has been submitted elsewhere for any other degree or qualification and it is all my own work unless referenced to the contrary in the text.

Full name of Author: Mduduzi Elias Nxumalo

Signature of Author:

Signed by candidate

Date: February 2010

Acknowledgements

I would like to thank the following people and organizations. Firstly, Dr Mashao, Prof Dlodlo, the members of the Speech Technology and Research Group and all people who sacrificed their time to participate in the experiments, without them I would not have been able to finish this work. Secondly, the Department of Science and Technology, the National Research Fund and the University of Cape Town's Center of Excellence for financial sponsorship. Thirdly, all the researchers and all people who develop open source software, for their contribution in making it possible for other people to make further contributions. Fourthly, Mr Andrew Robinson, Lehlohonolo Mohasi and Kevin Sikweza for proof reading and making comments on this thesis. Fifthly, Kevin, Earnest Zitha and Malibongwe Dywibiba for support while I was doing experiments. Sixly, my family, Babalwa Dlova, the Robinson family that I stay with and all my friends and all people for unconditionally loving and supporting me. Lastly and more importantly, God for giving me the opportunity to be alive and to do this work.

Dedicated to

My grandmother: Mrs Christina B. Nxumalo and my mother: Rachel Nxumalo who are both blind, my brothers: Mkhululi and Nkosi, my late sister: Fikile Mhlungu and all her kids, all the Nxumalo family, all my friends and all fellow South Africans.

Transcoding multilingual and non-standard web content to VoiceXML

Mduduzi Nxumalo

Abstract

Transcoding systems redesign and reformat already existing web interfaces into other formats so that they can be available to other audiences. For example, change it into audio, sign language or other medium. The benefit of such systems is less work on meeting the needs of different audiences. This thesis describes the design and the implementation details of a transcoding system called Dinaco. Dinaco is targeted at converting HTML web pages which are created using Extensible Markup Language (XML) technologies to speech interfaces. The differentiating feature of Dinaco is that it uses separated annotations during its transcoding process, while previous transcoding systems use HTML dependent annotations. These separated annotations enable Dinaco to pre-normalize non-standard words and to generate VoiceXML interfaces which have semantics of content. The semantics help Text-to-Speech (TTS) tools to read multilingual text and to do text normalization. The results from experiments indicate that pre-normalizing non-standard words and appending semantics enable Dinaco to generate VoiceXML interfaces which are more usable than those which are generated by transcoding systems which use HTML dependent annotations. The thesis uses the design of Dinaco to demonstrate how separating annotations makes it possible to write descriptions of content which cannot be written using external HTML dependent annotations and how separating annotations makes it easy to write, maintain, re-use and share annotations.

Contents

Declaration	ii
Acknowledgements	iii
Declaration	iii
Abstract	iv
1 Introduction	1
1.1 The motivation for speech enabling the web	2
1.1.1 Advantages of speech enabling the web	2
1.1.1.1 The web becomes accessible in different modalities	2
1.1.1.2 The web becomes ubiquitous	3
1.1.2 The growth of speech technology	3
1.2 Challenges of speech enabling the web	4
1.2.1 Communication devices	5
1.2.2 Interaction modes	5
1.2.3 Diversity of web users	6
1.3 Introduction to transcoding and research objectives	7
1.3.1 Creating multi-channel interface	7
1.3.2 Research objectives	7
1.3.3 Research scope and relationship to previous work	8
1.4 Contribution of this research	9
1.4.1 Transcoding web pages which feature multilingual text	9
1.4.2 Transcoding web pages which feature non-standard words	10

1.4.3	Summary of the solution	11
1.5	Summary of the remaining chapters	12
2	Background and literature review	13
2.1	Introduction to technologies	14
2.1.1	XML technologies	14
2.1.1.1	An introduction to XML	14
2.1.1.2	Linking XML resources	16
2.1.1.3	XML-based markup languages: XHTML	17
2.1.1.4	XML-based markup languages: VoiceXML	17
2.1.1.5	XML Schema language	23
2.1.1.6	Transforming XML documents	23
2.1.2	XML-Based User Interface Development	26
2.1.2.1	Design XML documents	27
2.1.2.2	Create XML documents	27
2.1.2.3	Define interfaces	27
2.1.2.4	Create user interfaces	28
2.1.2.5	Advantages of XinterfaceDevMeth	28
2.1.3	Semantic Web	30
2.1.3.1	Introduction to semantic web	30
2.1.3.2	Web Ontology Language	31
2.2	Transcoding	33
2.2.1	Making the web to be accessible through speech	33
2.2.1.1	Annotations-based transcoding	34
2.2.1.2	Navigation and usability	35
2.2.1.3	Markup languages	37
2.2.1.4	Substituting resources	38
2.2.1.5	Dynamic content and transactions	38
2.2.1.6	Simplifying the task of writing annotations	38
2.3	Related work	39
2.3.1	HTML dependent annotations	40
2.3.1.1	Costly maintenance and limited reuse	41

2.3.1.2	Limited knowledge	42
2.3.1.3	Limited separation of tasks	43
2.4	Summary of chapter 2	44
3	The design of the transcoding system	45
3.1	Separation of annotations	45
3.1.1	Introduction to separated annotations	46
3.1.1.1	Background on separated annotations	46
3.1.1.2	The first example of separated annotations	46
3.1.2	Annotations about content	48
3.1.2.1	Introduction to annotations about content	49
3.1.2.2	How to write annotations about content	50
3.1.2.3	An example of annotations about content	51
3.1.3	Annotations about the interface	51
3.1.3.1	Introduction to annotations about the interface	51
3.1.3.2	How to write AAI	52
3.2	The architecture	52
3.2.1	Inputs to Dinaco	53
3.2.1.1	Interface Repository	54
3.2.1.2	Content Repository	55
3.2.2	Pre-normalization of non-standard words	55
3.2.2.1	References to Bible scriptures	56
3.2.2.2	Dates	56
3.2.2.3	Currency	57
3.2.3	Outputs from Dinaco	57
3.2.3.1	Markup language	57
3.2.3.2	Limitations	58
3.2.3.3	Example	58
3.2.3.4	Navigation menu	65
3.2.3.5	Appending semantics	65
3.2.4	Deployment	66
3.3	The impact of separating annotations in the XinterfaceDevMeth	67

3.3.1	Separation of annotations and separation of concerns	67
3.3.2	It becomes easier to separate tasks	69
3.3.3	Managing changes is easier	71
3.3.4	Writing annotations without being limited by the HTML markup	71
3.3.5	Annotations become reusable and sharable	72
3.4	Summary of chapter 3	73
4	Implementation of the transcoding system	74
4.1	The Transcoding process	74
4.1.1	Load inputs	75
4.1.2	Create the intermediate HTML	75
4.1.3	Create the intermediate XSLT	78
4.1.3.1	Creating the intermediate xsl:stylesheet and xsl:output 78	
4.1.3.2	Creating the intermediate xsl:template	79
4.1.3.3	Creating the intermediate xsl:value-of and the inter- mediate xsl:apply-templates	80
4.1.4	Create the VoiceXML interface	82
4.2	Summary of chapter 4	83
5	Experimentation	84
5.1	The objective and the design of the experiments	84
5.1.1	The objective of the experiments	84
5.1.2	The hypothesis and the null hypothesis	85
5.1.2.1	Hypothesis	85
5.1.2.2	Null hypothesis	85
5.1.3	Design of the experiments	85
5.1.3.1	The sample web page	86
5.1.3.2	The control interface	89
5.1.3.3	The experimental interface	90
5.1.3.4	Evaluation of the control and the experimental in- terfaces	93

Contents	ix
5.1.4 Setting up the environment	94
5.1.4.1 The voice browser	95
5.1.4.2 The TTS tool	95
5.2 Summary of chapter 5	95
6 Analysis and interpretation of the results	97
6.1 Analysis and interpretation	97
6.1.1 Multilingual text	98
6.1.1.1 The results	98
6.1.1.2 Analysis and interpretation for multilingual text	99
6.1.1.3 Concluding remarks and limitations	100
6.1.2 The date value and the currency value	100
6.1.2.1 Recalling the currency and the date values	101
6.1.2.2 Overall assessment for non-standard words	102
6.1.2.3 Concluding remarks for non-standard words	104
6.1.3 Overall assessment of the interfaces	104
6.1.3.1 The results	105
6.1.3.2 Analysis and interpretation	106
6.2 Concluding remarks	106
6.3 Summary of chapter 6	107
7 Conclusions, future work and recommendations	108
7.1 Characteristics of HTML interfaces	108
7.2 Summary of the solution	109
7.3 Conclusions	109
7.4 Future work and recommendations	110
7.4.1 Future work	110
7.4.2 Recommendations	111
Bibliography	112
Appendix	120

A Evaluation sheet for system A

120

University of Cape Town

List of Figures

2.1	An example of the structure of XML document	15
2.2	VoiceXML architecture	18
2.3	VoiceXML example	20
2.4	The flow of the dialog between the user and the system	21
2.5	An example of the XSLT markup	24
2.6	The HTML document obtained by using a XSLT stylesheet to transform an XML document	24
2.7	The process of using the XinterfaceDevMeth to develop interfaces.	26
2.8	Using XSLT to generate multi-platform interfaces for the same web content in different languages	29
2.9	An example which demonstrates how to use the OWL syntax	32
2.10	The HTML Document to be annotated	40
2.11	Markup Dependent annotations limit the amount of annotations which can be written about text.	42
2.12	The interdependence of tasks when writing annotations in the XinterfaceDevMeth.	44
3.1	Using XSLT transformation to produce the HTML document	47
3.2	A transcoder which uses separated annotations	48
3.3	A transcoder which uses mixed annotations	48
3.4	Content Annotation Language	51
3.5	Overview of the inputs to Dinaco	53
3.6	The profiles HTML document.	60
3.7	Annotations about the profiles HTML interface	60

3.8	The source XML document	61
3.9	Annotations about content for the profiles XML document	62
3.10	The XSLT document which generates the profiles HTML document .	63
3.11	The profiles VoiceXML document which Dinaco outputs	64
3.12	Mixed annotations overlap with the Interface Component and with the Content Component.	68
3.13	Separation of software components if annotations are separated. . . .	69
3.14	The process flow of the XinterfaceDevMeth if annotations are separated.	70
4.1	The Activities in Dinaco's transcoding process	75
4.2	The profiles source XML document	76
4.3	Annotations about content for the profiles XML document	77
4.4	The original HTML document	77
4.5	An example of the HTML document which is produced by the inter- mediate XSLT stylesheet	77
4.6	An example of the original XSLT stylesheet	78
4.7	The <code>xsl:stylesheet</code> element in the intermediate XSLT	79
4.8	The <code>xsl:template</code> element in the intermediate XSLT stylesheet	80
4.9	An example which shows how to replace the <code>xsl:value-of</code> select element	81
4.10	An example which shows how to add the <code>xsl:apply-templates</code> element into the intermediate XSLT	82
5.1	The sample HTML web page which was transcoded during experi- mentation	87
5.2	The XML document which was transformed by XSLT to produce the sample page	87
5.3	The XSLT stylesheet which was used to create the sample web page .	88
5.4	The VoiceXML interface for the control experiment	90
5.5	Annotations about content which were used by Dinaco while creating the experimental interface	92
5.6	The VoiceXML interface for the experiment	93
6.1	Comparing scores for the ability to read Afrikaans words	99

-
- 6.2 Comparing scores for the overall assessment of non-standard words . 102
- 6.3 The results of the overall assessment of the interfaces by subjects . . 105

University of Cape Town

List of Tables

2.1	A high level description of VoiceXML tags	22
2.2	A high level description of XSLT tags	25
2.3	A high level description of OWL tags	32
2.4	Example of markup dependent annotations.	40
3.1	Reading of the Bible verse by different speech synthesis tools	56
3.2	Reading of the dates by different speech synthesis tools	57
3.3	Reading of the South African currency	57
6.1	Approximate percentages of subjects per average score for Afrikaans words	98
6.2	Approximate percentages of subjects per average score for non-standard words	103
6.3	Approximate percentages of subjects per score for the overall assessment of the interfaces	105

Chapter 1

Introduction

The web has become popular as the virtual place where different stakeholders access and/or publish various kinds of electronic information and self-service systems. These stakeholders are from different parts of the world and play different roles in their communities. Amongst these stakeholders are different businesses, organizations, government agencies, academic institutions and even individuals. Consequently, the web has become important such that most people need it in order to do their daily business. People visit the web to access information which they use for different purposes, like socializing, living healthy lives, studying and exploring job opportunities. People also visit the web to gain access to online self-service systems which allow them to easily and efficiently do banking, trade, shop etc.

There are, however, still hindrances which make it difficult for some members of the community to easily access the web. One of these hindrances is that web pages are designed to be accessible visually. This makes websites to be accessible to sighted people who have access to devices like desktop computers and laptops. There are usually no interfaces which are designed specifically to make websites to be accessible through speech. The consequences of this are as follows: Firstly, it limits the ability of visually impaired people to also access the web, as sighted people do. Secondly, it limits the accessibility of the web to people who have access to the Internet and to computers.

This chapter and the subsequent chapters use the phrase “speech-enable” as a shorter way of saying “make accessible through speech”. The rest of this chapter is

organized as follows: Section 1.1 motivates why the web should be speech-enabled. Section 1.2 discusses the challenges that developers are still likely to face when creating speech-enabled websites. Section 2.2 gives an introduction to transcoding and gives more details about the objectives and the scope of this research. Section 1.4 discusses the contribution of this research. Section 1.5 gives an overview of the remaining chapters.

1.1 The motivation for speech enabling the web

Speech technology can be informally defined as the technology which gives machines/computers the ability to speak and/or to listen like human beings do. Currently, most developers create HTML [1] interfaces which are designed to be accessed by users of graphical web browsers, but they do not create interfaces which are specifically designed to be accessed by users of speech technologies. Section 1.1.1 motivates why it is important to create interfaces which make websites to be accessible through speech. Section 1.1.2 motivates why we are in a better position today to create speech-enabled websites.

1.1.1 Advantages of speech enabling the web

Speech enabling the web makes it possible for more people to be able to access the web. Speech enabling the web makes the web to be accessible in different modalities. It also makes the web to become ubiquitous.

1.1.1.1 The web becomes accessible in different modalities

Speech enabling the web makes it possible for people to interact with web applications without reading the screen of the devices which they use to connect to the web. This offers the following two benefits. Firstly, it makes it possible for people who live with disabilities to have means of accessing the web. For example, speech technology is used by the visually impaired as the means of accessing the web and using computers. This allows people who live with such disabilities to become active members in their communities, to be independent, to compete for employment

opportunities etc.

Secondly, speech enabling the web does not only benefit people who live with visual disabilities, it benefits most people. It gives people a convenient way of communicating with web-based applications. For example, it allows people to access the web even if they are far from their devices or their hands are busy on something else.

1.1.1.2 The web becomes ubiquitous

Speech enabling the web increases the number of people who can access information anywhere and anytime. It opens the door for the creation of websites which are not only accessed by people who have access to computers and to the Internet. It makes it possible to create websites which are accessible telephonically, as it is explained in section 2.1.1.4. Making the web to be accessible telephonically makes it possible for people to access information even if they do not have access to the Internet at the time of their need of information. For example, it makes it possible for people to access the web even if they live in areas where there is no Internet connection.

1.1.2 The growth of speech technology

There has been growth in the number of tools and standards which are created to support the development and deployment of speech-enabled applications, compared to the last two decades. Consequently, it is now much easier for developers to create speech-enabled websites. In addition, many web users have access to speech-enabled technologies. This can be deduced from the following three factors:

Firstly, most modern devices have capabilities to install and run speech hardware and software. For example, desktop computers, laptops, cellular phones and even cars and elevators have speech capabilities. This makes it possible for users of speech technology to access speech-enabled websites without being limited by the communication device that they are using. It also makes technology users to be familiar with speech-enabled applications.

Secondly, the number of open source and private Text-to-Speech (TTS) tools has grown in the last decade. Some of the currently available open source TTS tools

are Festival [2] and espeak [3]. These TTS tools have different voices which give them the capability to read content in different spoken languages. As a result, it is possible for developers to create speech-enabled websites which are accessible in different spoken languages.

Thirdly, the World Wide Web Consortium (W3C) has defined a set of markup languages to be used when developing speech-enabled applications. The Speech Synthesis Markup Language [4] (SSML) and the Voice Extensible Markup Language (VoiceXML) [5] are examples of these markup languages.

SSML is designed to give authors of synthetic speech-enabled applications a way of controlling aspects of speech output such as pronunciation, volume, pitch, rate etc. across different speech synthesis-enabled platforms. VoiceXML is a markup language which gives developers the capability to develop Interactive Voice Response (IVR) systems in a similar way to developing web based applications. Chapter 2 gives a more detailed introduction to VoiceXML and also gives an example which demonstrates how to create VoiceXML-based IVRs.

As these examples show that speech technology is growing, so it is important to take initiatives which will encourage the development of speech-enabled websites.

1.2 Challenges of speech enabling the web

This section discusses factors which are the likely causes of the slow down in the development of speech-enabled websites. Speech-enabling the website is a task which competes with many other tasks in web development. Speech enabling the website adds to the cost and the amount of time required to design and develop web interfaces. It also adds to the cost of maintaining the website.

The cost of creating the website which is meant to be accessed by a wide and diverse audience anywhere and anytime is high. This is because of the number of issues which need to be taken into consideration and be catered for when creating a website such that it becomes accessible in this way [6] [7] [8] [9] [10]. The first issue is the diversity of the communication devices which are used by users when they access the web. The second issue is the diversity of the interaction modes which can

be used during the interaction between the user and a web application. The third issue is the diversity of web users themselves.

1.2.1 Communication devices

There is a wide range of communication devices that people can choose to use when they access the web. They can choose from desktop computers, personal digital assistants, cellular phones, telephones etc. As a result, it is desired that websites be designed to be accessible to users of all these different devices. The advantage of this is that it increases the number of people who are able to access information and e-commerce services, and it makes the web to be accessible anywhere and anytime. The disadvantage is that these communication devices have different characteristics. Their display screens are of different sizes. They support different ways of receiving user inputs. They are suitable to be used with different networks.

The impact of the differences between communication devices is that web developers have to create multiple web interfaces in order to support connections from multiple client devices. The interfaces for different client devices can differ in the following ways:

- The amount of input that the user can give to the system.
- The amount and type of content suitable for rendering in the client device.
- The suitable layout and font sizes for displaying content.

1.2.2 Interaction modes

Modern communication devices have the capability to run advanced hardware and software systems. This gives them the ability to support different modes of interaction, for example, graphical, speech only and multi-modal interactions. This capability allows users to choose their preferred mode of interaction when browsing the web. The user's choice can depend on many things. It can depend on the body parts that they want to involve while browsing the web. This helps to improve the accessibility of the web in the following ways. First it extends the accessibility of

the web to people who live with disabilities. Second it makes users to choose the mode of interaction depending on their context. For example, a user in a noisy environment can prefer to use a graphical interaction, while a user whose hands are busy on something can prefer to use speech.

Interaction modes have different characteristics. These characteristics are different in the way that web developers have to create a different interface for each mode. For example, a graphical interaction is different from a speech-only interaction. It is different in terms of how the user accesses the output from the system, how the user accesses the navigation menu, the types of supported media and software tools used during the interaction. The differences are such that different markup languages are used to develop interfaces for different interaction modes. For example, graphical web interfaces are written in XHTML, multi-modal web interfaces are written in XHTML plus Voice [11] (X+V) and voice dialogs are written in VoiceXML.

1.2.3 Diversity of web users

Most commercial and some non-commercial websites are created primarily to be visited by people who connect from anywhere in the world. There are many benefits of creating such websites. Acquiring customers from all over the world is often one of them. Examples of such websites are Multinational Online Stores [12] [13] and websites owned by Multinational Companies [14] .

It is however, a challenge to make these websites to present each user with content which he/she is able to understand. The cause of this challenge is the differences between users who connect from different countries. Some of these differences are their spoken languages, cultures and data formats and units of measurement used in their countries.

The amount of time and cost of creating websites such that they become accessible to different people is high. Different versions of the same content and of web interfaces are often required. This requires the website to be carefully designed to allow automatic conversion of data values [15]. Web content needs to be written by people who have different language skills.

The research results published by Internet World Stats [16] showed that by June

2008, English was spoken by most Internet users, 29.4% of them. These results also showed that the number of Internet users who speak languages like Chinese, Spanish and French grew more than those who speak English, between 2000 and 2008. These results indicate a growing need to publish web content in many different languages, not in English alone. See [13, 14, 17] for information on the importance of taking language and culture into consideration when designing websites.

1.3 Introduction to transcoding and research objectives

1.3.1 Creating multi-channel interface

Researchers have proposed different approaches which can help to simplify the task of creating multi-channel web interfaces. There are techniques which help developers who design and develop multi-channel interfaces themselves. For example, there is on-going research on how to use modal based interface design techniques [7] [8] and how to use one markup language to write interfaces for multi-web platforms e.g. the User Interface Markup Language [18] [19].

There is also on-going research on how to use transcoding systems to convert/adapt already existing web interfaces from one form to the other. Transcoding can be used to make web interfaces which were designed to be accessible in one platform to become accessible in another platform. Transcoding has been used to adapt web interfaces such that they become accessible in multiple client devices, mostly in mobile devices [20–25]. Transcoding is a cheaper way of creating interfaces which make websites to be accessible in multiple platforms.

1.3.2 Research objectives

This research investigates how to design a transcoding system which converts HyperText Markup Language (HTML) interfaces which are created using Extensible Markup Language (XML) technologies into speech interfaces. The objective of doing this investigation is to design a system which makes it easier and cheaper for

developers who create interfaces with an aim of making websites to be accessible visually, to also create interfaces which make the same website to be accessible through speech.

Achieving the above-mentioned objective is intended at reducing the overall cost and the amount of time required to create and maintain websites which are created to be accessible from different platforms. The research intends to help developers to create websites which are accessible to a wider audience of web users. Websites will have interfaces which make web content to be accessible both visually and non-visually. People who access the website from graphical platforms will access HTML interfaces. People who access the same website from speech platforms will access speech interfaces which results from doing transcoding.

1.3.3 Research scope and relationship to previous work

The primary purpose of transcoding systems which were reviewed during this research is to convert an already existing HTML interface into a speech interface, for example, [20, 26–32] . These systems do transcoding without taking into consideration the processes and the technologies which were used to create the HTML. Doing transcoding this way, makes it possible to transcode any HTML document and transcoding can be done both in the client side and in the server side. However, it also has disadvantages.

The annotations (semantics/descriptions/guidelines) which help these transcoding systems to produce usable speech interfaces are dependent on the HTML markup. This dependency on the HTML markup makes it difficult and expensive to write, maintain and reuse annotations. It also limits the descriptions of web content to those which can be written using the HTML elements of the transcoded document.

This research investigates how to design and implement a transcoding system which transcodes HTML interfaces which are created by using the Extensible Stylesheet Language Translation [33] (XSLT) to transform XML documents to HTML. The investigation is targeted at XSLT version 1 because the investigation was started before version 2 became a W3C recommendation. Chapter 2 discusses more details about the technologies which are used and the process which is fol-

lowed when creating interfaces which the investigation is targeted at. The aim of targeting web pages which are created using specific technologies is to investigate the problem on a small scale: how to efficiently write and manage annotations and how transcoding systems can effectively use these annotations.

1.4 Contribution of this research

The aim of doing transcoding (in the scope of this research) is to create usable interfaces which make websites to be accessible to users of speech technologies. This thesis describes how to design and implement a transcoding system such that the following two objectives which are listed below are met.

1. The transcoding system is able to transcode HTML interfaces which feature multilingual text and non-standard words (NSWs) into usable speech interfaces. See sections 1.4.1 and 1.4.2 for more details on multilingualism and on reading NSWs. TTS tools play an important role in making it possible for people to access the web through speech. They are used to read text content to users [34]. Usable speech interfaces in the context of the stated objective are speech interfaces which are created such that TTS tools become able to read multilingual text and NSWs for improved user understanding.
2. The transcoding system is designed such that it uses annotations which are easy to create, maintain, reuse and share.

1.4.1 Transcoding web pages which feature multilingual text

Speech interfaces which result from doing transcoding are targeted to be used by users who use speech-based user agents. The problem is that it is not easy for these agents to identify the spoken language which they have to use when reading text web content, during the interaction between the user and the speech interface. Current transcoding techniques have not come with efficient solutions on how to generate speech interfaces which have semantic descriptions of content which can be used by

TTS tools while reading web content. The following are the factors which contribute in making it difficult to solve this problem:

- HTML interfaces are usually created with an aim of making it easy for human beings to read web content on their own. The consequences of this is that these interfaces serve their purpose even if they do not have information on which language should be used when reading web content. The user uses his/her own judgement to identify the language used when reading. This makes it difficult for transcoding systems to adapt these interfaces into speech interfaces which have information on which language should be used by voice browsers when reading text web content.
- There are web pages which have multilingual text. Even web pages which have text which is predominantly from one spoken language can have proper nouns, like the names of people and of places from other spoken languages. Web pages for news websites are an example of pages of this nature. These pages have stories about people and names of places from all over the world. Transcoding systems which transcode these web pages should create speech interfaces which have information which help TTS tools to switch between different spoken languages while reading text web content.
- There is on-going research in using language identification techniques to create multilingual TTS tools. See [35] for an example. However, there are too many spoken languages in the world. More than six thousand of the spoken languages are listed in the Ethnologue [36]. This makes it difficult to create intelligent software systems which can automatically identify the language of each piece of text in any given web page.

1.4.2 Transcoding web pages which feature non-standard words

Web content consists of standard words (SWs) and non-standard words (NSWs). Standard words have a specific meaning that can be described phonetically. The

examples of SWs are ordinary words and proper nouns. Unlike SWs, NSWs consist of numerical patterns and alphabetical strings which usually do not have a consistent pronunciation that can be phonetically described. The examples of NSWs are numbers, currency amounts, dates, acronyms and abbreviations.

TTS engines are more likely to be able to convert SWs from their written form into their correct spoken form, compared to NSWs. This is because the correct interpretation and pronunciation of NSWs can depend on the context in which they are used. For example, there is more than one way of reading IV and 1/2. As explained by [37], IV can be read as “fourth”, like in “Henry the IV” or as “the fourth”, like in “Henry IV” or as “four”, like in “section IV”. 1/2 can be read as “half” if it is interpreted as a fraction; or as “the first of February” or “the second of January” if it is interpreted as the date. More background information about text normalization can be found in [4] [37] [38] [39].

Some countries use different numeric formats of date values, for example, dd/m-m/ccyy is used in the United Kingdom and mm/dd/ccyy is used in the United States. These differences also make it difficult to convert dates from their numerical form into their spoken form. An example made by [15] shows these differences with 10/01/2006, which is interpreted as “October 1, 2006” in the United States, and as “January 10, 2006” in the United Kingdom.

HTML interfaces do not have adequate descriptions of content which can enable transcoding systems to deal with the problem of text normalization, for example, by creating speech interfaces which have descriptions of content which can help TTS tools to deal with the problem of text normalization.

1.4.3 Summary of the solution

This research contributes by discussing the design of Dinaco, which is a transcoding system which converts HTML interfaces to VoiceXML. Dinaco uses annotations which guide it on how to produce VoiceXML interfaces. Dinaco uses annotations to produce VoiceXML interfaces such that they have semantics which help TTS engines to read multilingual text and to do text normalization. Dinaco also has the ability to use annotations to pre-normalize text such that it becomes readable by

TTS tools.

Dinaco separates annotations about content from annotations about the interface. The aim of separating annotations this way is to make it easy to write, maintain, reuse and share these annotations.

1.5 Summary of the remaining chapters

The rest of this thesis is organized as follows. Chapter 2 firstly introduces the reader to various XML technologies which are used when developing web interfaces which are transcoded by the discussed transcoding system. Secondly, it introduces the reader to the markup language which is a W3C's recommendation for creating web ontologies. This markup language has been used to write annotations which are used by previous transcoding systems and it is also used to write annotations which are used by the transcoding system which is discussed in this thesis. Thirdly, chapter 2 reviews and critiques previous research on transcoding systems which convert HTML interfaces into speech interfaces. Fourthly, it discusses more details about the research problem that this report endeavours to solve.

Chapter 3 discusses the design of the transcoding system. Chapter 4 discusses the implementation details of the transcoding system. Chapter 5 discusses the experiment which was conducted to measure the ability of the designed transcoding system to produce speech interfaces which are usable in speech-based platforms. Chapter 6 discusses the results from experimentation. Chapter 7 concludes this thesis.

Chapter 2

Background and literature review

This chapter is organized as follows:

- Section 2.1 gives the reader a high level introduction to technologies which are used when developing web user interfaces using XML-based technologies and when doing transcoding. This introduction is intended to help the reader to be able to understand the research problem which is described later in this chapter and the solutions to the problem which are discussed in chapter 3.
- Section 2.2 reviews research done on transcoding HTML interfaces into speech user interfaces, with an aim of making them usable in speech-based platforms.
- Section 2.3 discusses some of the problems with the manner in which transcoding systems which were reviewed during this research are designed.
- Section 2.4 summarizes this chapter.

Web pages often feature content from more than one language. This led to a need to make some examples in this chapter and in subsequent chapters to have phrases or proper nouns which do not come from the English language. This is done to illustrate the real life scenarios. However, the reader is not expected to understand or to be able to pronounce proper nouns and phrases which are not from the English language in order to understand the examples.

This chapter and subsequent chapters feature figures which show examples of documents which are written in a particular markup language. Some of these figures

have line numbers on the left. These line numbers do not form part of the markup of the document in question. They are there to make it easier to refer to different parts of the figure.

2.1 Introduction to technologies

This section is organized as follows:

- Section 2.1.1 introduces the reader to the technologies which are used when creating raw XML documents, linking XML resources: XLink and XPath, creating user interfaces: XHTML and VoiceXML, creating the XML schema, and transforming XML documents from one form to the other: XSLT. Harold and Means [40] give a more detailed introduction on XML-based technologies.
- Section 2.1.2 defines the process and a combination of technologies used when developing web interfaces using the methodology which this thesis calls XinterfaceDevMeth. The objective of this is to investigate how to transcode HTML interfaces which are created following the XinterfaceDevMeth into usable speech interfaces.
- Section 2.1.3 introduces the reader to semantic web technologies which are used when doing annotations-based transcoding.

2.1.1 XML technologies

2.1.1.1 An introduction to XML

This section describes the Extensible Markup Language (XML). The section mentions the benefits of using XML and uses an example to describe the XML syntax and terminology.

XML is a data format used to encode structured electronic documents. XML is a meta-language whose syntax rules are a subset of the Standard Generalized Markup Language (SGML) [41] syntax rules. The XML syntax rules are defined by the W3C's XML specification [42]. One of the main design goals of XML is to make XML documents to be usable over the Internet.

There are many benefits of using XML. This paragraph mentions four benefits. Firstly, the author of an XML document has an unlimited vocabulary that he/she can use to describe the structure of textual content. This is made possible because the XML specification does not predefine the tags that can be used to describe the structure of the XML document. The author of the XML document invents his/her own tags. Secondly, XML content can be stored as a plain text document which can be easily transported from one system to the other. Thirdly, XML documents are structured in a way which makes it easy for human beings to understand and for software systems to process. Fourthly, XML is a popular open standard which has already received support from major players in the Information Technology sector, for example, governments and various major software companies like International Business Machines (IBM) Corporation, Sun Micro Systems and Microsoft. As a result, it has become easier to obtain software tools which can be used to view and process XML documents.

Figure 2.1 shows an example of the XML document called people.xml. The remainder of this section uses the contents of and the line numbers in people.xml to introduce the reader to the basic XML terminology and syntax rules.

```
1 <?xml version="1.0" ?>
2 <people>
3   <person id="jsmith">
4     <firstName>John</firstName>
5     <lastName>Smith</lastName>
6   </person>
7 </people>
```

Figure 2.1: An example of the structure of XML document

The XML specification recommends that the XML document should begin with the XML declaration. This declaration gives some information about the document itself e.g. the version number of the XML specification that the document conforms to. People.xml has an XML declaration in line 1.

An XML element is a logical component of an XML document which has structured data about a particular object. A non-empty element can contain one or more other elements. For example, the “people” element which starts in line 1 contains

the “person” element. A non-empty element can also contain textual content. For example, the “firstName” element which starts in line 4 contains the “John” textual content. A non-empty element can also contain a combination of both the textual content and other element(s). An element is called a child element of an element which contains it, for example, the “person” element is a child element of the “people” element.

A non-empty element starts with a start-tag and ends with an end-tag. For example, the start-tag of the “people” element is <people> (line 2) and its end tag is </people> (line 7).

An element can have one or more attributes associated with it. The attributes are written in the start-tag or in an empty-tag of the element. Each attribute has a name and a value associated with it. For example, in line 3, the start-tag of the “person” element contains an attribute which has “id” as its name and “jsmith” as its value.

2.1.1.2 Linking XML resources

There is often a need to be able to create an address which points to a specific fragment within an XML resource. W3C has created specifications, like the XML Pointing Language (XPointer) [43] and the XML Path Language (XPath) [44] which define how to write these addresses.

XPath is used to write an address to a specific fragment of the XML document, for example, a single node or a group of nodes or an attribute of a particular node. The understanding of XPath which is required for the purposes of this thesis is that the location path of a node can be constructed using its name, the name of its parent node and the names of its ancestor nodes separated by slashes. The names of its ancestor nodes and the name of its parent node appear in the left in relation to their location in the tree of nodes of the XML document. For example, the XPath address: `people/person/lastName` can be used to identify elements of `people.xml` which have last names of people.

For the purposes of this thesis, the reader is required to understand that XPointer can be used to identify a node using the value of its ID attribute. For example, the

“person” element of `people.xml` (figure 2.1) which has information about John can be identified using the `”people.xml#jsmith”` XPointer expression.

2.1.1.3 XML-based markup languages: XHTML

The W3C has defined different XML compliant markup languages which can be used to create different kinds of interfaces. This section discusses the Extensible Hypertext Markup Language (XHTML) [45].

The Hypertext Markup Language (HTML) is the markup language used to write web pages. The author of the web page uses HTML to specify the content of the web page and to describe how this content should be displayed by user agents (web browsers). HTML is also used to define hyper linking from the web page to other Internet resources like remote web pages and different types of media. XHTML version 1.0 is a group of markup languages which are a reformulation of HTML version 4.01. The major difference between HTML and XHTML is that the meta-language of HTML is SGML, while the meta-language of XHTML is XML. This means that all markup languages which form XHTML are XML compliant, while HTML is not. More details about the differences between XHTML and HTML can be found in [46].

The markup languages which resulted from XHTML include XHTML Plus Voice (X+V) [11] and the XHTML Mobile Profile (XHTML-MP) [47]. XHTML-MP was created by the Open Mobile Alliance to replace the Wireless Markup Language (WML), as the standard for writing web pages which are intended for mobile devices and other resource constrained devices. X+V is a markup language for writing multi-modal web interfaces which feature both visual and speech based-interactions. X+V had not achieved the W3C recommendation status at the time of writing this thesis.

2.1.1.4 XML-based markup languages: VoiceXML

This section discusses another XML-based user interface markup language: the Voice Extensible Markup Language [5](VoiceXML).

VoiceXML is a W3C standard used to create voice dialogs between a human and a computer. The main difference between VoiceXML applications and XHTML

applications is that XHTML applications are designed to be viewed in graphical web browsers; while VoiceXML applications are designed to be accessed from voice browsers in a non-visual manner. VoiceXML applications can play synthesized and/or pre-recorded speech to users. VoiceXML applications can receive user input by doing automatic recognition of speech user inputs or by interpreting the user's Dual Tone Multi Frequency (DTMF) key inputs. The main goal of VoiceXML is to make the development of Interactive Voice Response (IVR) applications to be similar to web development.

VoiceXML applications can be deployed in the server side infrastructure such that they become accessible by users of different client devices. This is depicted in figure 2.2. The client devices can be different in terms of their physical characteristics and in terms of the networks that they are connected to. The client devices can be desk computers and laptops which are connected to the Internet. Client devices can also be telecommunications devices like telephones and cellular phones.

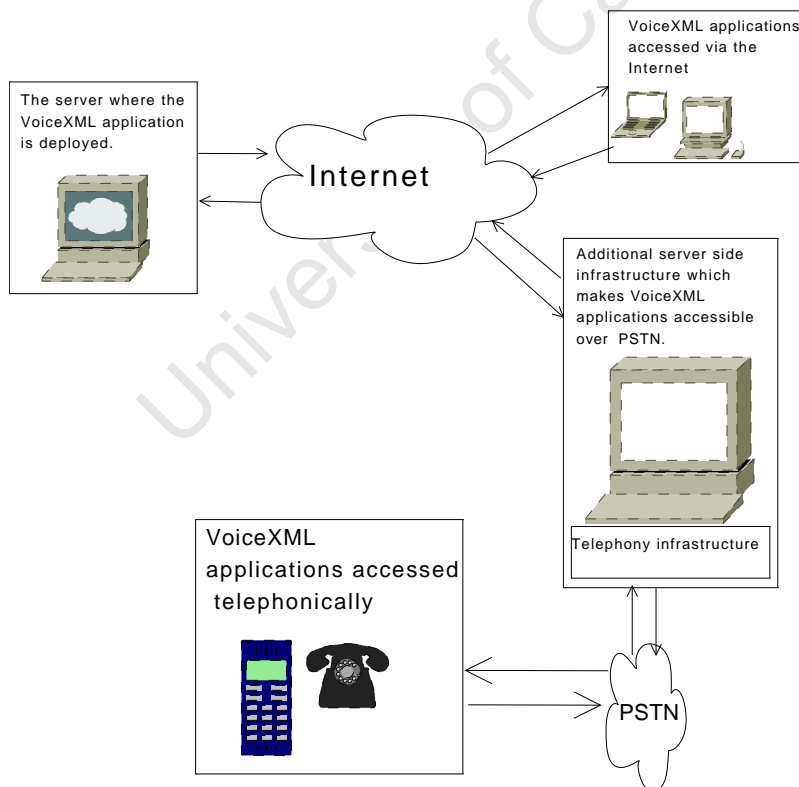


Figure 2.2: VoiceXML architecture

The client devices which are connected to the Internet can access VoiceXML applications by connecting straight to the web server where the VoiceXML applications are deployed. An additional server side infrastructure is required in order to make VoiceXML applications to be accessible from devices which are connected to the Public Switched Telephone Network (PSTN). This server side infrastructure can be composed of a computer with an installed Private Branch Exchange (PBX) and a telephone line.

The remainder of this section uses an example to give the reader a high level understanding of the VoiceXML markup. Figure 2.3 shows an example of the VoiceXML markup which can be used to model a dialog between the user and the system. This dialog is depicted in figure 2.4. The main aim of the system is to play a greeting message, either in English or in isiZulu. The dialog is started by the system. The system prompts the user to choose the language in which he/she wants to be greeted. The user gives inputs to the system through DTMF key inputs. The system replays the main menu every time it has successfully played the greeting message to the user. The system plays an error message and reprompts the user for input if an invalid input was encountered, or if the user does not give input until the waiting time expires.

```
1 <?xml version="1.0" encoding="UTF-8"?>
2 <vxml version="2.0" xmlns="http://www.w3.org/2001/vxml"
   xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
   xsi:schemaLocation="http://www.w3.org/2001/vxml http://
   www.w3.org/TR/voicexml20/vxml.xsd">
3 <menu dtmf="true" id="mainMenu">
4   <prompt>
5     <enumerate>
6       For <value expr="_prompt"/> greeting press <value expr="
       _dtmf"/>.
7     </enumerate>
8   </prompt>
9   <choice next="#engForm">English</choice>
10  <choice next="#zulForm">isiZulu</choice>
11  <noinput>Please listen to the menu and make your choice.
12    <reprompt/>
13  </noinput>
14  <nomatch>
15    Invalid input, please listen to the menu and re-enter your
16      choice.
17    <reprompt/>
18  </nomatch>
19  </menu>
20  <form id="engForm">
21    <block>
22      Hi there.<goto next="#mainMenu"/>
23    </block>
24  </form>
25  <form id="zulForm">
26    <block>Sawubona.<goto next="#mainMenu"/></block>
27  </form>
28 </vxml>
```

Figure 2.3: VoiceXML example

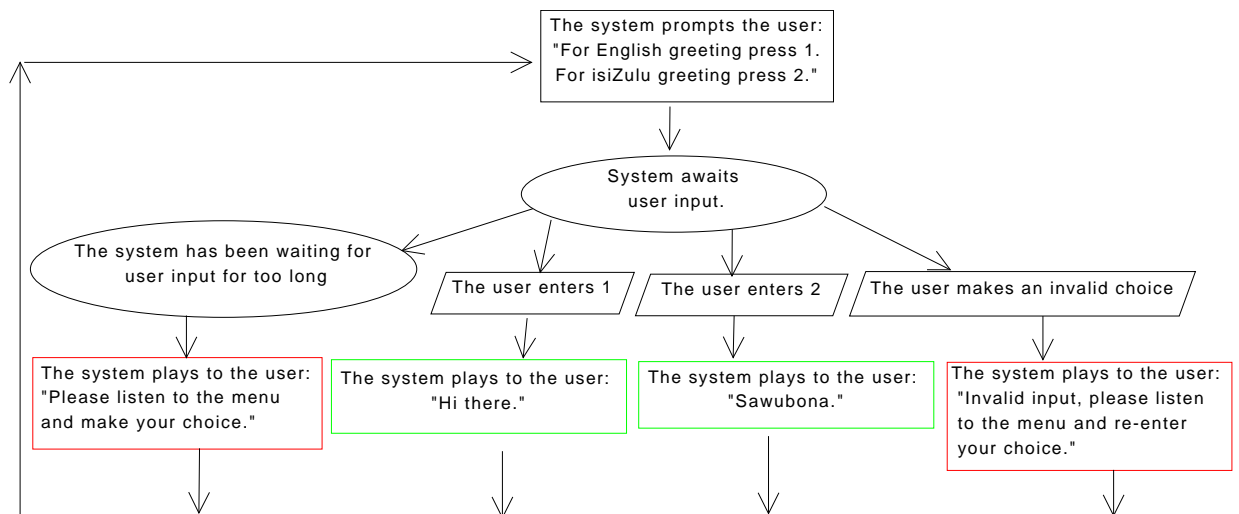


Figure 2.4: The flow of the dialog between the user and the system

Table 2.1 gives a high level description of VoiceXML elements which appear in figure 2.3. The descriptions of the elements given in table 2.1 are not general, they are based on how elements are used in the context of the example. Formal descriptions of the VoiceXML elements can be found in the VoiceXML specification [5].

Tag name	Line number	High level description of how VoiceXML tags are used for in the VoiceXML example of figure 2.3 .
vxml	2	The vxml element is the root element of the VoiceXML document.
menu	3	The menu element is used to play the choices available to the user, prompt the user for a choice and to choose the next form to be processed, based on the user's choice.
prompt	4	The prompt element is used to make the system to play synthesized speech to the user.
enumerate	5	The enumerate element is used to construct a template for playing the available choices to the user. The template has changing prompting text and numbers to be pressed by the user when choosing a particular choice. These are constructed as follows: The <value expr="_prompt"/> element gets the prompting text for the current choice. The prompting text for the first choice is "English". The <value expr="_dtmf"/> element gets the number which the user has to press in order to choose the current choice. This number is 1 for the first choice.
choice	9	Each choice element is used to specify an available choice to the user. The choice specifies information about the prompting text, the value which the user has to press when choosing it and the next form to be executed if it is chosen by the user.
noinput	11	The noinput element gets executed when the system has waited for the user's input until the acceptable waiting time has expired. In this case, it makes the system to play the error message and to replay the main menu to the user.
reprompt	12	The reprompt element is used to make the system to replay the menu to the user.
nomatch	14	The nomatch element gets executed when the user enters an invalid input. In this case, it makes the system to play the error message and to replay the main menu to the user.
form	19	The form element contains a set of instructions which can be executed together as a unit.
block	20	The block element contains text to be synthesized into speech and played to the user. It also contains an instruction on what the system must do after playing text to the user .
goto	25	The goto element is used to go to a specified element in the VoiceXML document.

Table 2.1: A high level description of VoiceXML tags

2.1.1.5 XML Schema language

This section gives the reader an understanding of what one can do with the XML schema, without covering details on the syntax.

The XML schema is the technology used to set constraints on the contents of the XML document. The XML document which does not violate the constraints set by the given XML schema is said to conform to that XML schema. The XML Schema Language is called the XML Schema Definition (XSD). The XSD is used to define constraints on the elements and the attributes which a conforming XML document is allowed to have. XSD has language constructs to be used to define constraints on the data type of content, the attributes and the child elements that each element of the conforming XML document is allowed to have. The constraints on attributes of the element can be on the attributes that it is allowed to have and the values of those attributes. The constraints on child elements can be on the child elements which an element is allowed to have, their quantity and their order of appearance. The second version of the W3C Schema Language is available in two parts: part 1 in [48] and part 2 in [49].

2.1.1.6 Transforming XML documents

The Extensible StyleSheet Language Transformations [33] (XSLT) is a language recommended by W3C for transforming XML documents from one form to the other. This section uses an example to help the reader to understand how to use XSLT to transform XML documents. Section 2.1.2 discusses the advantages of using XSLT together with other XML-based technologies when developing web interfaces.

Figure 2.5 shows an XSLT document which was used to transform `people.xml` (figure 2.1) into an HTML document shown in figure 2.6. Table 2.2 gives a high level description of XSLT tags which feature in figure 2.5. The line numbers in this table, are the line numbers where the tag first appears in figure 2.5.

```

1 <?xml version="1.0" encoding="UTF-8"?>
2 <xsl:stylesheet version="1.0" xmlns:xsl="http://www.w3.org/1999/
   XSL/Transform">
3   <xsl:template match="/">
4     <xsl:apply-templates />
5   </xsl:template>
6   <xsl:template match="people">
7     <html>
8       <body>
9         <h1>Profiles</h1>
10        <xsl:apply-templates select="person" />
11      </body>
12    </html>
13  </xsl:template>
14  <xsl:template match="person">
15    <h2>Personal details</h2>
16    Full name: <xsl:value-of select="firstName" /><xsl:text> </
       xsl:text><xsl:value-of select="lastName" />
17  </xsl:template>
18 </xsl:stylesheet>

```

Figure 2.5: An example of the XSLT markup

```

<html>
<body>
  <h1>Profiles</h1>
  <h2>Personal details</h2>
  Full name: John Smith
</body>
</html>

```

Figure 2.6: The HTML document obtained by using a XSLT stylesheet to transform an XML document

Tag name	Line number	Description
xsl:stylesheet	2	The root element of the XSLT document.
xsl:template	3	<p>It defines how to transform elements of the source XML document whose paths match the value of the match attribute. The value of the match attribute is evaluated in relation to the element that is currently being transformed. In the context of people.xml, it is used as follows:</p> <ul style="list-style-type: none"> • In lines 3 to 5: it defines how to start transforming people.xml. • In lines 6 to 13: it defines how to transform elements whose XPath matches people.xml/people. • In lines 14 to 17: it defines how to transform elements whose XPath matches people.xml/person.
xsl:apply-templates	4	<p>It is an instruction which is used to apply templates for elements. It is used as follows in the context of people.xml.</p> <ul style="list-style-type: none"> • To apply templates for all child elements of the element which is currently being transformed. For example, in line 4, it is called to apply templates for people.xml/*. • To apply templates for selected child elements, for example in line 10, it is used to apply templates for people.xml/person.
xsl:value-of	16	It is used to include in the result tree the value which is generated by evaluating its select attribute. In line 16, it is used to select the values of people.xml/person/firstName and people.xml/person/lastName.
xsl:text	16	It is used to write text as it is in the result tree. In line 16, it is used to create an empty space.

Table 2.2: A high level description of XSLT tags

2.1.2 XML-Based User Interface Development

This section discusses the XML-Based User Interface Development Methodology (XinterfaceDevMeth). XinterfaceDevMeth is defined for the purposes of this research as the process to be followed and the combination of XML technologies used when developing web interfaces. The XinterfaceDevMeth is targeted at developers who work as teams to develop multi-channel web interfaces which are designed to be accessed by diverse web users. The technologies which are used in XinterfaceDevMeth are already introduced in sections 2.1.1.1 to 2.1.1.6. Chapter 3 discusses the design of a transcoding system which converts HTML interfaces which are created using the XinterfaceDevMeth.

The XinterfaceDevMeth process of developing web interfaces is divided into four activities: design XML documents, create XML documents, define interfaces and create interfaces. This process is depicted in figure 2.7. The details about each activity are discussed in sections 2.1.2.1 to 2.1.2.4. The advantages of following this process are discussed in section 2.1.2.5.

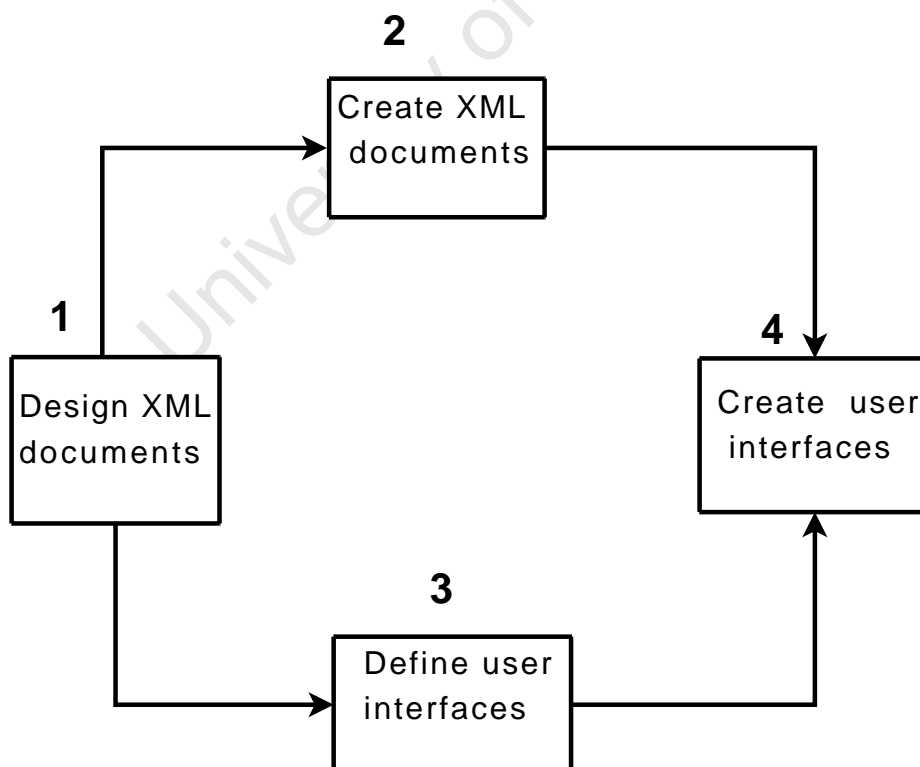


Figure 2.7: The process of using the XinterfaceDevMeth to develop interfaces.

2.1.2.1 Design XML documents

The first activity is to design the data that is to be created and published for users to access. The end product of this activity is the XML schema which describes the details of the data that will be published. The XML schema is required when creating the XML documents and when defining the interfaces. As a result, this activity should be done before the second and the third activities.

2.1.2.2 Create XML documents

The second task is to create the actual content. The content is created as raw XML documents, which do not have the presentation or the formatting markup. The XML documents are created following the syntax rules and the vocabulary which was defined in the XML schema. That is, the XML documents are created such that they conform to the XML schema from the first activity. This activity depends on the first activity but not on other activities. The team members who create these XML documents are called Content Creators (CCs).

2.1.2.3 Define interfaces

The third activity is to define the template rules for creating interfaces which are suitable to be accessed in different user contexts. The end product of this activity is the XSLT stylesheets which define how to transform XML documents into interfaces. There can be as many stylesheets as possible, depending on the interfaces that are intended to be created. There can be stylesheets which define how to transform XML documents into interfaces for speech based platforms, for graphical access in mobile devices, graphical access in desktops and laptops and more. The stylesheets are created from knowing the XML schema from the first activity, but without knowing the actual XML documents from the second activity. The team members who create the stylesheets are called Interface Creators (ICs).

2.1.2.4 Create user interfaces

The actual interfaces are created automatically by the XSLT processor. An example of the open source XSLT processor which is available in SourceForge [50] is SAXON. The XSLT processor generates interfaces by applying the transformation rules from the XSLT stylesheet(s) into the XML document(s). It can be done immediately after both the XML documents and the stylesheets have been created or at run time when a particular interface is required.

2.1.2.5 Advantages of XinterfaceDevMeth

This section discusses the advantages of using the XinterfaceDevMeth. The main advantage of XinterfaceDevMeth is that it makes it possible to divide the labour intensive task of creating multilingual content and interfaces for multiple web platforms into multiple software components. These software components are written using W3C standards, are reusable and can be easily created by different teams of developers. The results are that it is possible to speed things up and to reduce the cost when developing interfaces. The rest of this section uses an example to illustrate the advantages of using the XinterfaceDevMeth.

Figure 2.8 shows how to reuse XML artifacts when creating content in different spoken languages and interfaces for multiple web platforms. In this figure, the aim is to publish content in two different official South African languages: English and isiZulu. The content is first written in two different XML documents: eng.xml and zulu.xml. Both these XML documents conform to the same schema and have content in English and in isiZulu, respectively. There are three XSLT stylesheets. These stylesheets are web.xslt, wap.xslt and voice.xslt and are used to transform the XML documents into HTML interfaces for bigger devices, HTML interfaces for smaller devices and VoiceXML interfaces for speech based platforms, respectively.

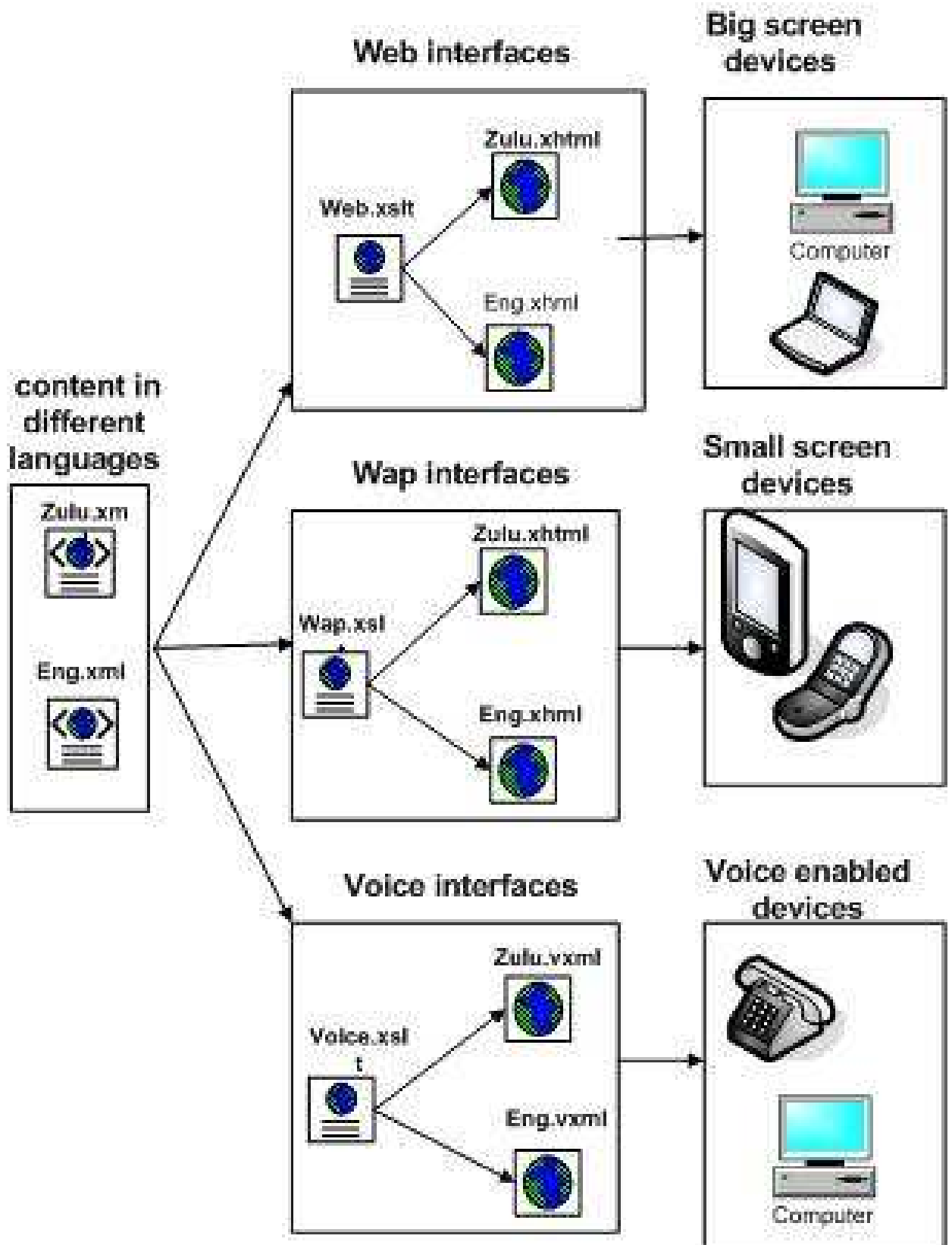


Figure 2.8: Using XSLT to generate multi-platform interfaces for the same web content in different languages

The next three points use the above mentioned and illustrated scenario to specify the advantages of using XinterfaceDevMeth.

1. XinterfaceDevMeth makes it easier to assign tasks to different people who work independent of each other, and to base the assignments on their skills and expertise. In the illustrated scenario, it is easy to assign members of the development teams based on their human language skills and their interface development skills and expertise. The examples are as follows: Team members of the Content Creators who know isiZulu can be assigned to write zulu.xml and those who know English to write eng.xml. Team members of the Interface Creators who know VoiceXML and have expertise in creating speech-based interfaces can be assigned the task of creating voice.xlst.
2. The same XSLT stylesheet can be used to transform multiple XML documents which conform to the same schema. In the illustrated scenario, all three XSLT stylesheets are used to transform both the zulu.xml and the eng.xml documents to different interfaces.
3. XML documents which are created during Activity 2 do not have presentation or formatting markup. This makes these documents to be created such that the same XML document can be transformed to different interfaces by different XSLT stylesheets. In the illustrated scenario, both the zulu.xml and the eng.xml documents were transformed by three XSLT stylesheets.

2.1.3 Semantic Web

2.1.3.1 Introduction to semantic web

One of the limitations of the HTML markup is that it is used to describe how to display content in the web browser without giving the semantics (the meaning) of content. For example, HTML can be used to describe how and where to display the today's date on the web page, without giving any information which will help software tools to know that it is the date. Semantic Web technologies are seen as

being amongst the technologies which form part of Web 3.0 [51], which is the next generation of web applications.

The term Semantic Web refers to the web in which web applications are created together with machine understandable meanings (semantics) of their contents. The Semantic Web promises to make it possible to develop systems which can do smart processing of the web data, for example, searching. The Semantic Web originates from Sir Tim Berners-Lee's vision of the Web. Berners-Lee is a director in W3C.

Semantic Web technologies have also been used to help transcoding systems to convert HTML interfaces into usable speech interfaces. These technologies have been used to write annotations which help transcoding systems to understand the contents of web pages. Annotations help transcoding systems to create usable speech interfaces.

2.1.3.2 Web Ontology Language

The Web Ontology Language (OWL) is one of the markup languages which was created by the W3C to support the development and deployment of semantically rich web applications. The OWL language reference document [52] describes OWL as “a semantic markup language for publishing and sharing ontologies on the World Wide Web. OWL is developed as a vocabulary extension of RDF (the Resource Description Framework) and is derived from the DAML+OIL Web Ontology Language” [52]. Chapter 3 discusses the design of a transcoding system which uses semantic descriptions of web content which are written in OWL version 1 [53].

This section uses an example shown in figure 2.9 to introduce the reader to the OWL syntax. The example shows the short version of the ontology which was used during this research to describe the HTML elements. The example demonstrates how to declare a class called Element and how to define a property of this class. It also shows how to create an instance of the Element class. Table 2.3 gives a high level description of the OWL elements which are used in the example.

```

1 <?xml version="1.0"?>
2 <!DOCTYPE rdf:RDF [
3   <!ENTITY rdf "http://www.w3.org/1999/02/22-rdf-syntax-ns#">
4   <!ENTITY rdfs "http://www.w3.org/2000/01/rdf-schema#">
5   <!ENTITY owl "http://www.w3.org/2002/07/owl#"> ]>
6 <rdf:rdf xmlns:rdf="&rdf;" xmlns:rdfs="&rdfs;" xmlns:owl="&
   owl;">
7   <owl:Class rdf:ID="Element"/>
8   <owl:DatatypeProperty rdf:ID="hasTagName">
9     <rdfs:domain rdf:resource="#Element"/>
10    <rdfs:range rdf:datatype="http://www.w3.org/2001/XMLSchema
      #string"/>
11  </owl:DatatypeProperty>
12  <Element rdf:ID="myElement">
13    <hasTagName rdf:datatype="http://www.w3.org/2001/XMLSchema
      #string">
14      body
15    </hasTagName>
16  </Element>
17 </rdf>

```

Figure 2.9: An example which demonstrates how to use the OWL syntax

Tag name	Line number	Description
rdf:rdf	6	The root element of the OWL document. This element makes the OWL elements to be sub-elements of RDF.
owl:Class	7	It declares an OWL class called Element.
owl:DatatypeProperty	8	It declares a datatype property called hasTagName.
rdfs:domain	9	It specifies that the hasTagName property is applicable to instances of the Element class.
rdfs:range	10	It specifies that the data type of the hasTagName property is a String.
Element	12	It declares an instance of the Element OWL class. The URI reference of this instance is #myElement.
hasTagName	13	It specifies the value of the hasTagName property of the myElement instance. This value is "body".

Table 2.3: A high level description of OWL tags

2.2 Transcoding

This section reviews and critiques previous research on transcoding. The focus of the review and critique is on transcoding systems which use annotations to convert HTML interfaces into speech interface.

2.2.1 Making the web to be accessible through speech

The private sector and different academic institutions have been doing research on using transcoding as one of the ways of making the web to be accessible through speech. Transcoding has been used to adapt web pages which were designed to be accessed visually such that they become accessible non-visually through speech-based technologies. This section reviews previous research on transcoding of this nature. The findings from the literature review are summarized below:

- It is difficult to design transcoding systems which can convert any interface which was designed to be accessed visually into a usable speech interface. The recommended solution to this problem is to design transcoding systems such that they are given hints (semantic annotations) which guide them during the transcoding process. More details are covered in section 2.2.1.1.
- There are transcoding systems which retain the original markup of the transcoded document and there are also those which do not. More details are covered in section 2.2.1.3.
- It is important for transcoding systems to have a mechanism of creating a speech-friendly navigation menu which can help users to easily find information when they browse the web. More details are covered in section 2.2.1.2.
- Transcoding systems may have to do transcoding such that the resulting interface does not have exactly the same content as the original interface in order to achieve better usability in speech-based platforms. The details are covered in sections and 2.2.1.4.

- Special techniques are required when transcoding web pages which generate dynamic content and which have transactions. More details are covered in section 2.2.1.5.
- Techniques and tools which can help to simplify the task of writing annotations are required in order to make it easy for people to use annotations-based transcoding systems. More details are covered in section 2.2.1.6.

2.2.1.1 Annotations-based transcoding

Transcoding systems adapt already existing interfaces from one form to the other. One of the main components of a transcoder is its transcoding policy. The transcoding policy is the set of rules which determine how the transcoder produces a new interface from the transcoded interface. For example, the transcoding policy can determine how to organize, prioritize and order contents in the interface which results from transcoding of the original interface. Annotations are the semantic descriptions of content which are taken by the transcoder as one of its inputs. Annotations help the transcoder with information about the contents of the transcoded interface, which it will not be easy for it to infer on its own. The examples of how transcoders have been using annotations are mentioned in sections 2.2.1.2 and 2.2.1.4.

Annotations can be written inside or outside of the annotated document. Annotations which are written inside the annotated document are called internal annotations. Annotations which are written outside of the annotated document (in a separate annotations document) are called external annotations. There are pros and cons for both the internal and the external annotations. However, most research papers (for example, [20], [21], [54] and [55]) which were reviewed during this research are in favor of using external annotations because of the reasons listed below:

- External annotations bring about the separation of content from presentation.
- External annotations do not require changes to be done directly on the annotated document. This makes it possible to write these annotations without making changes on the already existing markup language for writing web sites:

HTML. As a result, the annotated HTML document can still be rendered correctly by web browsers without a need to ignore some of the elements or attributes.

- Authors of external annotations do not need to have permission to modify the annotated document [56].
- Internal annotations can increase the size of the annotated document, which will make it not suitable for rendering in small devices [55].

2.2.1.2 Navigation and usability

There are noticeable differences between browsing the web from graphical web browsers and browsing the web non-visually from speech-based platforms [55]. Thus, creating a speech-friendly navigation menu is one of the tasks of transcoding systems which adapt HTML interfaces into interfaces which are usable in speech-based platforms.

Web designers usually use visual artifacts (for example, colours and fonts) to divide a web page into visual fragments. These fragments can be easily spotted by sighted users who browse the web from graphical web browsers. For example, sighted users can easily see the menu, the main content and the advertisements on the web page. The problem is that these visual artifacts are not visible if the same web pages are accessed non-visually. This section refers to this problem as the visual grouping problem. Researchers have proposed transcoding techniques which can help to solve different aspects of the visual grouping problem.

IBM's patent [57] describes the annotations-based transcoding system which converts HTML interfaces into VoiceXML. This patent focuses mostly on how to derive the navigation menu for a VoiceXML interface from the HTML interface. The patent describes how to divide the HTML page into fragments which can be accessed individually from speech-based platforms. The patent also describes how to create headings for these fragments and how to convert the HTML markup into VoiceXML markup.

In [32], IBM researchers describe a transcoding system which tries to resolve the

visual grouping problem. This system divides the web page into fragments and also adds text which helps users of speech platforms to recognize these fragments.

In [26] and [55], researchers from the University of Manchester describe a transcoding system called SADie. SADie was designed to also resolve this problem for web pages which use XHTML and the Cascading Style Sheets (CSS) [58]. SADie uses the CSS to identify and to get the semantic meaning of the visual groupings. SADie can be accessed as a Firefox extension. The other research on web page fragmentation is in [20].

In [28] and [59], researchers from Virginia Technicon describe transcoding techniques which they used to transcode HTML interfaces into usable VoiceXML interfaces. Their transcoding techniques also deal with the problem of creating a navigation menu. They describe how to create a navigation menu which allows users to navigate across VoiceXML documents. Their VoiceXML navigation menu has a “back” choice. This “back” choice makes it possible for users to go back to the VoiceXML document that they were previously in. Their related work is in [30]. The other research on generating usable speech interfaces is described by Borodin [60]. Borodin describes the dialog generation and text summarization techniques which are used in the HearSay project.

Researchers have also been using annotations to create usable speech interfaces. Transcoders have been using annotations which give them hints on how to prioritize the contents of the HTML document which they transcode. These hints help them to generate speech interfaces which make it easier for users to find the most important content in the web page. The exact details on how the transcoder achieves this depends on its transcoding policy. The following two techniques have been used the most. The first technique is to organize the menu such that the user is likely to be prompted for the most important information first. The second technique has been to eliminate the less important content. Examples of transcoding systems which use annotations this way are in [20], [21], [32], [55] and [59].

2.2.1.3 Markup languages

Researchers in the field of transcoding have been making different decisions on the markup of the document to be transcoded and the markup of the document which results from doing transcoding. There are transcoding systems which make changes on the original document but retain the original markup language.

IBM is one of the companies which have also been doing research on transcoding. IBM has a software system called IBM WebSphere Transcoding Publisher [61] (WTP). WTP can be deployed in the proxy server. There are several transcoding software components which form part of WTP. These transcoding components can be used to convert web content from one format into different other formats. There is a HTML-to-VoiceXML transcoder which forms part of IBM's WTP. The guidelines on how to use this transcoder can be found in [27].

A masters thesis [62] from the University of Texas at Dallas describes a transcoding system which converts HTML to VoiceXML. This transcoding system is not annotations-based. The thesis addresses the processes required when converting HTML to VoiceXML, for example, parsing of the HTML document and matching of HTML tags to VoiceXML tags. Other transcoding systems which convert HTML to VoiceXML are discussed in [28], [31] and [59].

In [29], researchers from the Chung-Hsing University describe an OWL based transcoding system. The paper describes how to use multiple layers of OWL ontologies to design the transcoding system such that it has an extensible transcoding policy. The system is designed such that it is possible for developers to customize its transcoding policy and to use its components when creating transcoding systems for other markup languages. As an example, the paper describes the framework which was used to convert HTML forms to WML and to VoiceXML.

There are transcoding systems which do not change the original markup of the transcoded document, that is, the speech-friendly interface has the same markup language as the original document. SADIE and [54] are the examples of such transcoding systems.

2.2.1.4 Substituting resources

Annotations can help the transcoder to get information about existing alternative representations of the same resource [20] [21] [27]. The transcoder can use this information to choose the resource which best suits the characteristics of the new target platform. For example, a transcoder which generates interfaces for small screen devices can use annotations to choose smaller images instead of bigger images, which may have been used for bigger screen devices.

2.2.1.5 Dynamic content and transactions

In [31], researchers from the Stony Brooks University describe techniques which they used to help visually impaired users to find information from web pages. Their techniques are targeted at transcoding HTML web pages which generate web content dynamically. Their techniques also deal with the problem of transcoding websites which have transactions which span multiple web pages. They also give the details about the transcoding module which implements their techniques and converts the HTML markup to VoiceXML. This transcoding module is incorporated into the HearSay Voice Browser [60] [63].

2.2.1.6 Simplifying the task of writing annotations

The reasons why external annotations are preferred over internal annotations were mentioned in section 2.2.1.1. However, there are disadvantages of using external annotations. Firstly, external annotations add an extra document which is likely to be changed when the annotated document is changed. Secondly, it is not easy to write external annotations because of the need to construct addressing expressions (for example XPath and XPointer) which help to identify specific locations within the annotated document.

Most websites are designed primarily with an aim of making them to be usable and appealing to visually sighted users, who view them in devices with big screens like desktops and laptops. This is achieved by encoding web content in HTML, without writing annotations. In [26] and [55] it was pointed out that developers are

not keen to write annotations because they see writing annotations as an unnecessary additional task.

Researchers have proposed different solutions which can help to simplify the task of writing annotations. Researchers in [64] proposed a way of automatically annotating HTML documents. The paper states that the solution is targeted at HTML documents which are “machine generated and contain rich semantic data” . There are also annotation software tools which can be used to semi-automate the task of capturing annotations [20] [56].

The SADIE project [26] [55] proposed a different way of simplifying the task of writing annotations. SADIE was designed to use annotations which can be easily captured by developers who use XHTML and CSS. In SADIE, annotations are written using the ID and the class attributes which developers use to specify the styles of XHTML elements. This makes it possible to write annotations without knowing the XHTML markup and eliminates the need to write addressing expressions (for example XPath or XPointer) for each XHTML element. It also makes annotations to be reusable when the same CSS styles are used to style different XHTML elements and different web pages.

This research mentions and attempts to address other challenges which make it difficult to write external annotations used by transcoding systems to generate usable speech interfaces. See section 2.3.1 for more details.

2.3 Related work

Annotations-based transcoding systems have a potential to generate speech-friendly interfaces such that user agents are able to read multilingual text and to normalize text during the interaction between the user and the speech-friendly interface. The challenge is on overcoming the difficulties which are encountered when writing annotations for these transcoders. Section 2.3.1 argues that annotations are written in a manner which makes them to dependent on HTML and it also discusses some of the drawbacks of writing annotations this way.

2.3.1 HTML dependent annotations

Previous research has proposed that annotations should be written in an external RDF/OWL document. The annotations document is written such that it has statements which are used by the transcoder to understand how to convert the HTML markup from one form to the other. Each statement can be understood as giving two pieces of information. The first piece is a URL which identifies an individual element or a group of elements of the annotated document. The second piece is the actual information about the element or the group of elements identified by the URL. URLs are written as either the XPath or the XLink expressions. See [20] and [56] for examples.

The XLink expressions are constructed using the ID attributes of elements of the annotated document. The XPath expressions are constructed using tag names of HTML elements and positions which help to distinguish elements with the same tag name. Fig 2.10 shows doc1.html which is a short HTML document. Table 2.4 uses examples to show how to use XPath and XLink URLs to write annotations about doc1.html.

```
<body>
  <p id="paragraph1">
    The first line of the first paragraph
  </p>
  <p>
    Umugqa wokuqala kupharagurafu yesibili.
  </p>
</body>
```

Figure 2.10: The HTML Document to be annotated

Style	URL	Information
XLink	doc1.html#paragraph1	has English text
XPath	doc1.html#p[2]	has isiZulu text

Table 2.4: Example of markup dependent annotations.

The style of writing annotations described above requires knowledge about HTML

elements to write statements about text between HTML elements. This can be seen in Table 2.4, where the ID attribute of the first paragraph of doc1.html was required to construct the XLink URL (doc1.html#paragraph1) which is part of the statement about the contents of this paragraph. This section uses the phrase “MD annotations” to refer to these annotations. MD annotations can be expanded as Markup Dependent annotations. The dependency on knowledge about HTML elements to write annotations about content makes this style of writing annotations to have the following characteristics:

- Changes in the annotated documents result in changes in annotations. This can lead to costly maintenance and limited reuse of annotations. More details are covered in section 2.3.1.1.
- The amount of knowledge which can be written about content depends on the ability to identify it using XPath or XLink expression which are constructed using HTML elements. More details are covered in section 2.3.1.2.
- Annotations can only be written after the HTML interface to be annotated already exists. This can lead to limited separation of tasks. More details are covered in section 2.3.1.3.

2.3.1.1 Costly maintenance and limited reuse

MD annotations are not easy to maintain and to reuse. The details are described below.

Maintenance: in MD annotations, statements about constituents of HTML elements have XLink or XPath expressions of HTML elements as subjects. The problem with using XLink or XPath expressions this way is that they can change when some HTML elements of the annotated document change and lead to a need to change annotations. Changes can be required even if the contents of the elements have not been changed. Examples of changes on the HTML element which can lead into a need to change XPath or XLink expressions are when either changes on the tag name or the value of its ID attribute of one of the HTML elements are made. Changes in the tag name affects the XPath expression of the changed element, the

XPath expression of its child elements and of some of its siblings which have the same tag name as itself. Changes in the value of the ID attribute affect the XLink expressions of the changed element.

Reuse: statements about content in the MD annotations are written to specifically describe content when it is in the annotated document. For example, the same statements about the same content may need to be duplicated in different annotations documents if the same content appears in different HTML documents.

2.3.1.2 Limited knowledge

Fig 2.11 shows the HTML document (doc2) which has two paragraphs. Doc2 is written to illustrate that the amount of knowledge which can be written about text content found between HTML elements is limited. In MD annotations, statements about constituents of the HTML element refer to all the constituents of the element, for example, the statement {doc2#p1 hasLanguage English} means that the whole sentence of the first paragraph is in English. This is not true, because the only English text in the first paragraph is “was born in”. It is not clear how to write statements about individual words of the second paragraph, that is, “Nelson” is the English text, “Mandela” is the isiXhosa text and that “1994” is a year.

```
<body>
  <p id="p1">
    Gedleyihlekisa Zuma was born in Nkandla.
  </p>
  <p id="p2">
    Nelson Mandela: was a president in 1994.
  </p>
</body>
```

Figure 2.11: Markup Dependent annotations limit the amount of annotations which can be written about text.

2.3.1.3 Limited separation of tasks

MD annotations make no provision for writing annotations about the contents of the HTML independent of the markup. This makes it difficult to separate annotations which describe how to convert the HTML interface into a usable speech interface, from annotations which describe the text content which is found between HTML elements. This section illustrates how this limits the separation of tasks in the XinterfaceDevMeth.

The annotator who writes annotations about content requires knowledge about HTML elements which leads into the following problem.

1. **Interdependence between the task of writing annotations and the task of creating the interface.** In MD annotations, the annotations about content can only be written after the annotated document exists. This requires the Annotator who writes these annotations to wait until the stylesheet has been finished and used to transform the XML document into HTML. The interdependence of tasks of this nature, does not allow for two different entities (companies, for example) to work on different tasks of the same project without waiting for each other. For example it is not possible for one entity to do all tasks which are related to content, for example, creating XML content and writing annotations about it and the other entity to do all tasks which are related to the interface, for example, writing the XSLT stylesheet. Figure 2.12 depicts that the task of writing annotations about content of XML documents which get created in step 2 are written in step 5.

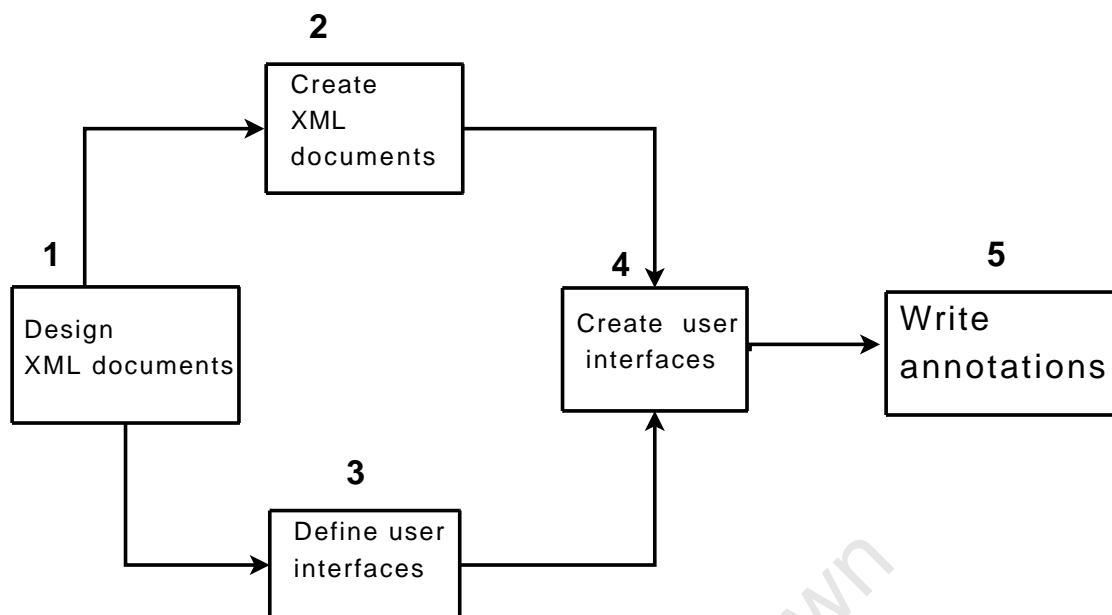


Figure 2.12: The interdependence of tasks when writing annotations in the XinterfaceDevMeth.

2.4 Summary of chapter 2

This chapter introduced the reader to XML and Semantic Web technologies in preparation of chapter 3 where the design of the transcoding system is discussed. The chapter also reviewed previous research on transcoding and discussed the drawbacks of using HTML dependent annotations. The chapter focused on transcoding systems which convert HTML interfaces into speech interfaces.

Chapter 3

The design of the transcoding system

This chapter describes the design of an annotations-based transcoding system which converts HTML interfaces to speech interfaces. This transcoding system is given a name “Dinaco” (**D**evice **I**ndependent **A**ccess to **C**ontent Transcoder). Dinaco is designed specifically to transcode HTML interfaces which are created using the XinterfaceDevMeth. The decision to limit the target HTML interfaces was to investigate, on a smaller scale, how to design an annotations-based transcoding system such that its annotations can be written and managed without experiencing the disadvantages of HTML dependent annotations. The disadvantages of HTML dependent annotations are mentioned in section 2.3.1.

Dinaco is an annotations-based transcoding system. That is, Dinaco uses annotations which guide it on how to transcode a given HTML interface. More details about these annotations are covered in section 3.1. Section 3.2 describes the artifacts which Dinaco takes as inputs and how it uses these artifacts while doing transcoding. This section also describes the speech interface that Dinaco produces as output. Section 3.3 concludes this chapter, by discussing the advantages of the annotations which are used by Dinaco.

3.1 Separation of annotations

Dinaco was designed such that it uses annotations which are separated into annotations about content and annotations about the interface. Section 3.1.1 gives an

introduction to these separated annotations. This section also uses an example to further explain how to separate annotations. Section 3.1.2 describes annotations about content in detail. Section 3.1.3 describes annotations about the interface in detail.

3.1.1 Introduction to separated annotations

3.1.1.1 Background on separated annotations

The question which led to the design which is proposed in this research is, how to take advantage of the fact that in XinterfaceDevMeth, the XSLT stylesheets and the XML documents are created and managed separately and independently of each other. This question led in a need to view the HTML interface that Dinaco has to transcode as having contents which come from two different sources. The first source is the XSLT stylesheet (assuming that only one stylesheet was used) which was used to transform XML documents to HTML. The second source is the XML document (assuming that only one XML document was used) which was transformed into HTML, by applying the XSLT stylesheet.

A decision was taken to design Dinaco such that it takes two groups of annotations. These groups of annotations can be created and managed separately and independently of each other. The first group of annotations is called the annotations about content (AAC). The distinguishing characteristic of the AAC is that they are written to describe the contents of the HTML documents which come from the XML document(s) which was/were transformed into HTML by applying the XSLT stylesheet. The second group is called the annotations about the interface (AAI). The distinguishing characteristic of the AAI is that they are written to describe the contents of the HTML document which come from the XSLT stylesheet. Section 3.1.1.2 uses an example to demonstrate how to separate annotations.

3.1.1.2 The first example of separated annotations

Figure 3.1 shows the source XML document, `source.xml`, the XSLT transformations and the HTML document, `results.html`. `Results.html`, is created by applying the

XSLT transformations on source.xml. These transformations take the phrase “Hello World” from the `<greeting id="enGreeting">` element of source.xml and makes it to become the contents of the `<p id="p1">` element of results.html. In this case, the content of results.html which came from source.xml is the “Hello World” phrase. The remainder of the contents of results.html came from the XSLT stylesheet.

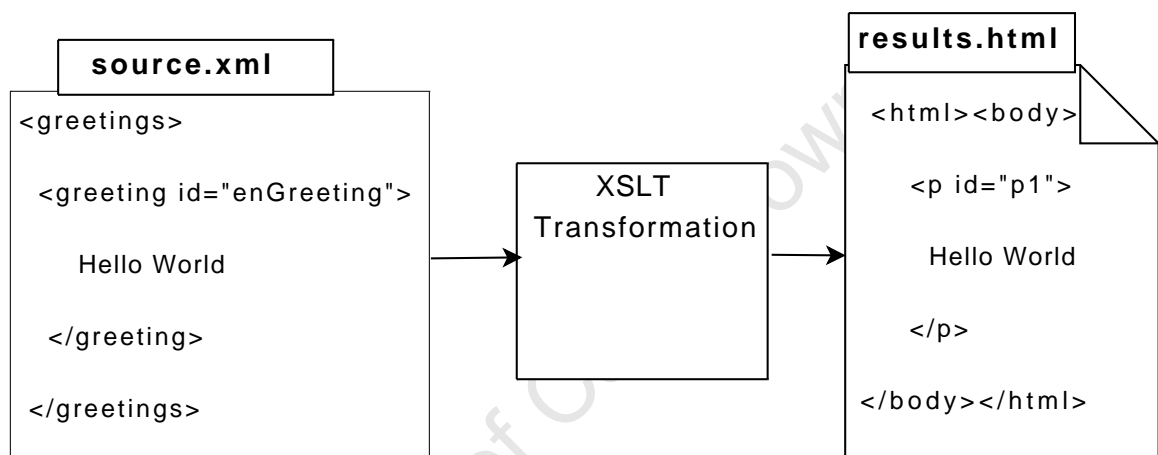


Figure 3.1: Using XSLT transformation to produce the HTML document

In the above example, the AAC would be written to describe the “Hello World” phrase. The AAI would be written to describe the p, body and html elements of results.html. The AAC are further described in section 3.1.2. The AAI are further described in section 3.1.3.

Based on the descriptions of the annotations that Dinaco takes as inputs, Dinaco can be seen as using separated annotations. Transcoders which take annotations that are not separated, can be seen as using mixed annotations. Figure 3.2 shows the annotations that a transcoder which uses separated annotations takes as input. Figure 3.3 shows the annotations that a transcoder which uses mixed annotations takes as input.

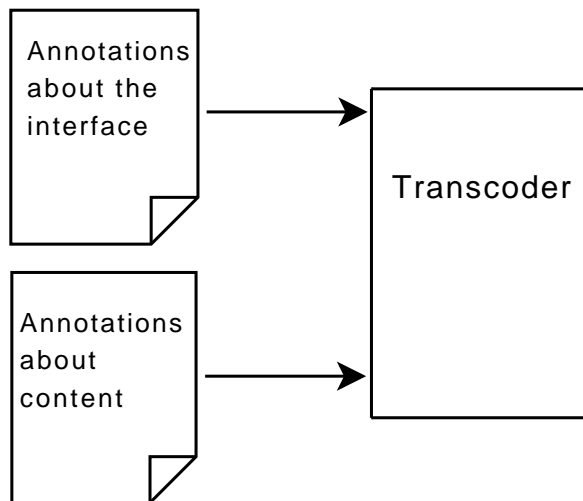


Figure 3.2: A transcoder which uses separated annotations

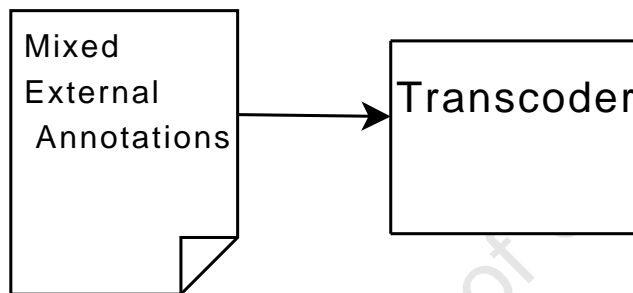


Figure 3.3: A transcoder which uses mixed annotations

3.1.2 Annotations about content

This section is organized as follows:

- Section 3.1.2.1 describes the annotations about content (AAC) in detail.
- Section 3.1.2.2 describes how to write AAC such that the following two objectives are achieved. The first objective is to create them in a manner that suits the process of creating HTML interfaces using the `XinterfaceDevMeth`. The second objective is to create them without experiencing similar disadvantages as when using HTML Dependent Annotations.
- Section 3.1.2.3 uses an example to demonstrate how to write AAC.

3.1.2.1 Introduction to annotations about content

The HTML interface is designed for a graphical interaction between the user and the user agent, usually a graphical web browser. In a graphical interaction, the main responsibility of the user agent is to display content on the screen of the user's communication device. As a result, for a graphical interaction, the HTML document becomes sufficient if it has enough information on how to display content on the screen. The user also takes certain responsibilities during the interaction. Some of these responsibilities are specified below:

- The user reads or views content on the screen of the communication device.
- The user formulates the meaning of content that he/she reads or views from the screen of the communication device.

In a speech interaction, TTS tools take some of the responsibilities which are done by the user in a graphical interaction. TTS tools are the ones which read content to the user. This usually requires them to formulate the meaning of content in order to be able to transform it from its written form into its spoken form. That is, TTS tools have to normalize content. These differences between a speech interaction and a graphical interaction makes it more important to specify semantics of content when creating a speech interface than when creating a graphical interface.

For example, there is more than one way of reading the number 1980. It can be read as “one thousand nine hundred and eighty” or as “nineteen eighty”. It is important for the speech interface to have a description of “1980” which will help TTS tools to translate “1980” into the desired spoken form. This description has the lower level of importance in a graphical interface. This is because the main aim of the graphical interface is to describe how to display content in the user's screen. The impact of this is that a graphical interface will usually not have the descriptions of text which are required when creating a speech interface.

Dinaco takes annotations about content as a separate document. Dinaco uses these annotations with an aim of converting HTML interfaces into usable speech interfaces. Dinaco uses these annotations for three purposes: The first is to do pre-normalization of non-standard words (NSWs). More details about pre-normalization

are covered in section 3.2.2. The second purpose is to append semantics of content in the speech interface that it creates. More details about appending semantics of content are covered in section 3.2.3.5.

3.1.2.2 How to write annotations about content

The aim of designing Dinaco such that it uses separated annotations was to make it possible to write AAC without using HTML elements. AAC are written in terms of the elements of XML documents.

AAC are written using the annotation language which this thesis refers to as the Content Annotation Language (CAL). CAL is an OWL based language which was designed to be used to annotate the elements of XML documents. The annotator who uses CAL to annotate the element of the XML document starts by creating an instance of the DinacoResource OWL class. This instance represents the XML element that is being annotated.

The DinacoResource class has properties which can be divided into two groups. The first group is composed of two properties which can be used to identify the XML element that is being annotated. These are the hasXPointer and the hasXPath properties. The hasXPointer property specifies the XPointer URI of the annotated XML document. The hasXPath property specifies the XPath URI of the annotated XML element. The annotator only has to specify one of the two properties.

The second group of the properties of the DinacoResource are those which are used to specify the properties of the contents of the XML element which Dinaco can use during transcoding. These properties are hasLanguage and hasType. The hasLanguage property is the International Standard Organization (ISO) 639-2 code [65] of the language which TTS tools should use when reading the contents of the annotated XML element. The hasType property is described in section 3.2.2.

This section has explained how to write AAC and introduced CAL. Section 3.1.2.3 uses an example to demonstrate how to write AAC using CAL. The details on how Dinaco uses AAC are covered in sections 3.2.2 and 3.2.3.5.

3.1.2.3 An example of annotations about content

This section uses an example to demonstrate how to use CAL to annotate the element of the XML document. The example demonstrates how to specify the `hasXPointer` and the `hasLanguage` properties of the XML element. Suppose Dinaco is required to transcode `results.html` of section 3.1.1 (see figure 3.1). In this case, AAC are written in terms of the elements of `source.xml` because it is the XML document that was transformed by XSLT to HTML. The content of `results.html` which comes from `source.xml` is the “Hello World” phrase. Figure 3.4 shows a simplified version of the OWL document which is written in CAL to specify the `hasXPointer` and the `hasLanguage` properties of the “Hello World” phrase.

In line 2 of figure 3.4, `source.xml#enGreeting` is the short way of writing the XPointer URI of the `<greeting id="enGreeting">` element of `source.xml`. In line 3, `en` is the ISO 639-2 code for the English language.

```
1 <dinaco:DinacoResource rdf:ID="Name2">
2   <dinaco:hasXPointer>source.xml#enGreeting</dinaco:hasXPointer>
3   <dinaco:hasLanguage>en</dinaco:hasLanguage>
4 </dinaco:DinacoResource>
```

Figure 3.4: Content Annotation Language

3.1.3 Annotations about the interface

3.1.3.1 Introduction to annotations about the interface

Annotations about the interface (AAI) are written to guide Dinaco on how to transcode contents of the HTML document which come from the XSLT stylesheet. This means that AAI are written from knowing the elements which define the structure of the HTML interface that Dinaco has to convert into a speech interface. AAI do not include specific information about the semantic descriptions of the contents of the HTML elements. The primary purpose of AAI is to guide Dinaco on how to transcode the HTML markup. Dinaco uses AAI for two purposes. The first purpose is to get guidelines on which elements of the transcoded interface it has to include in the resulting speech interface. The second purpose is to get guidelines on how to

create the navigation menu of the resulting speech interface.

AAC take precedence in this thesis over AAI, because the thesis contributes by describing how to transcode HTML interfaces which feature multilingual text and NSWs. AAI, are not used for the purposes of understanding content, they are used to understand how to transcode the HTML markup. As a result, the discussion about AAI will be made shorter. Annotations used by [20] and [26] are the examples of the AAI.

3.1.3.2 How to write AAI

AAI are written in terms of the HTML elements of the document that Dinaco has to transcode. AAI are written using the Interface Annotation Language (IAL). IAL is also an OWL-based annotation language. The annotator who uses CAL to annotate the HTML element of the transcoded interface starts by creating an instance of the Element OWL class. This instance represents the HTML element that is being annotated. Like the DinacoResource OWL class, the Element OWL class also has two groups of properties. The first group is composed of the hasXPointer and the hasXPath properties. These properties are used for similar purposes as in AAC.

The second group of properties of the Element OWL class are those properties which give Dinaco instructions on what it has to do with the annotated HTML element. The properties in the second group are the isRemovable and the startsMenuWithLevel. Dinaco does not include HTML elements which have the isRemovable property with the value of “yes”. Dinaco uses the startsMenuWithLevel property to get hints on how to create the navigation menu.

3.2 The architecture

This section is organized as follows:

- Section 3.2.1 describes the artifacts that Dinaco takes as inputs, before it begins its transcoding process. The details of how Dinaco uses these artifacts during its transcoding process are beyond the scope of this chapter. They are discussed in chapter 4.

- Section 3.2.2 discusses how Dinaco uses AAC to pre-normalize text and to help Dinaco to generate a speech interface which has semantic descriptions which help TTS tools to understand the text that they read.
- Section 3.2.3 discusses the interface that Dinaco outputs as a result of doing transcoding.

3.2.1 Inputs to Dinaco

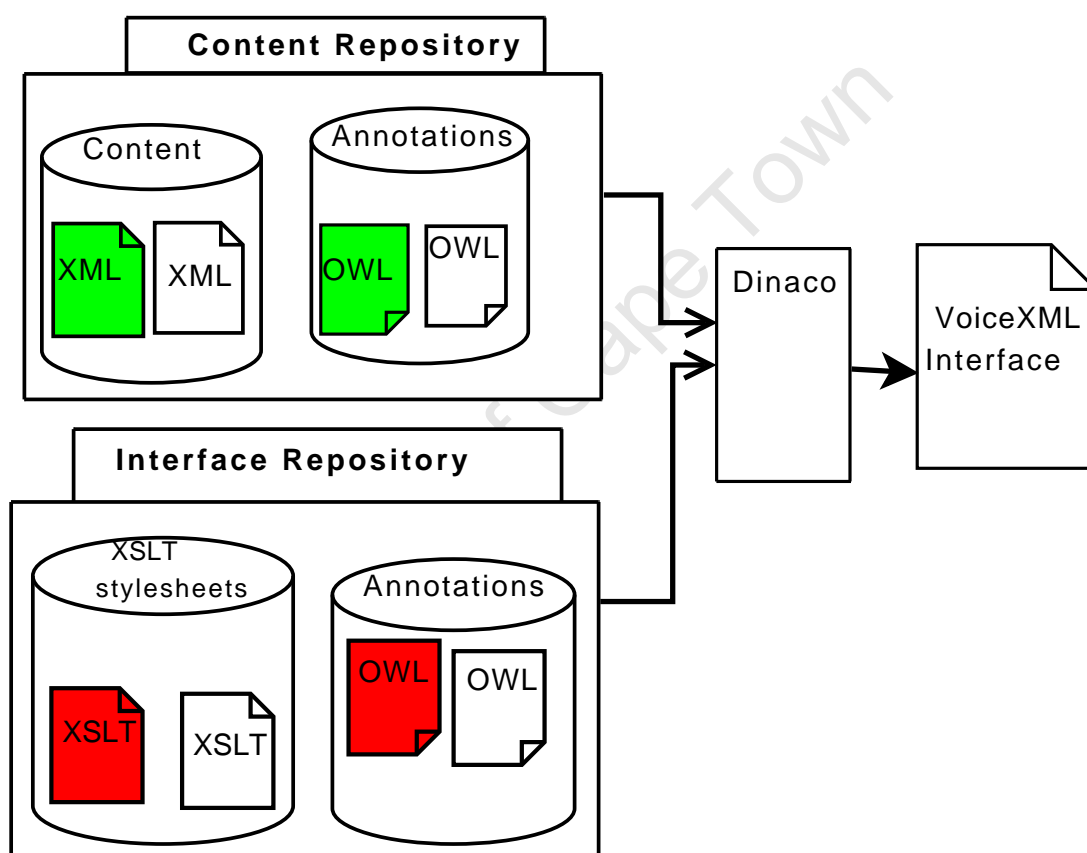


Figure 3.5: Overview of the inputs to Dinaco

The inputs taken by Dinaco are divided into two groups. The first group is composed of the artifacts which are produced by the interface development team. These artifacts are stored in the location called the Interface Repository. The second group is composed of the artifacts which are produced by the content creation team. These artifacts are stored in the Content Repository. Figure 3.5 shows an overview of

these inputs. More information about the Interface Repository and the Content Repository are in sections 3.2.1.1 and 3.2.1.2 respectively.

3.2.1.1 Interface Repository

An already existing HTML document does not have information about where the contents of its elements were taken from. This can be explained using the documents shown in figure 3.1. This figure shows the source XML document, `source.xml`, the XSLT transformations and the HTML document, `results.html`. The XSLT transformations take the phrase “Hello World” from the `<greeting id=“enGreeting”>` element of `source.xml` and make it to become the contents of the `<p id=“p1”>` element of `results.html`. `Results.html` does not have information which links the phrase “Hello World” with its source element: `<greeting id=“enGreeting”>` of `source.xml`. This makes it difficult to write a software system which can deduce that the contents of `<p id=“p1”>` were taken from `source.xml`. The phrase “Hello World” may have been taken from any other server side XML document, for example.

Dinaco would not be able to use AAC if it was designed such that its transcoding process begins after the HTML document to be transcoded has already been created. The reason for this is that it would not be able to associate AAC with the contents of the document it is transcoding. This is because AAC are written in terms of the elements of the XML documents and the HTML document does not have information which associates the contents of its elements with these XML documents. This led to a decision to not design Dinaco such that it takes HTML documents as one of its inputs.

Dinaco takes the XSLT stylesheet which was used to produce the HTML document from XML documents. The Interface Repository stores the XSLT stylesheets and the annotations (see figure 3.5). Each XSLT stylesheet in the Interface Repository is associated with an annotations document. This annotations document has annotations about the HTML document that the stylesheet produces. That is, they are the annotations about the interface (AAI).

3.2.1.2 Content Repository

The Content Repository stores the XML documents and the annotations, see figure 3.5. Each XML document in the Content Repository, is associated with an annotations document. These annotations are about its contents. That is, they are the annotations about the content (AAC).

3.2.2 Pre-normalization of non-standard words

One of the objectives of Dinaco is to convert HTML interfaces which feature NSWs into usable speech interfaces. This section describes how Dinaco pre-normalizes NSWs.

One of the challenges in web development is that the same HTML markup can be interpreted and rendered differently by different graphical web browsers. A similar problem occurs when the web is accessed through speech. Different users have different TTS tools installed in their machines. The problem is that the same piece of text can be read differently by different TTS tools. Sections 3.2.2.1 to 3.2.2.3 have tables which compare the outputs from the Festival [2] and the eSpeak [3] open source speech synthesis engines. There is also no guarantee that different TTS tools will be able to convert NSWs from its written form into its spoken form and read it correctly.

Dinaco has the capability to convert NSWs such that they become readable by TTS tools. Dinaco has transcoding modules called Normalizers. Dinaco uses Normalizers to pre-normalize NSWs and to append semantics of content in the speech interface that it creates. More details about appending semantics are covered in section 3.2.3.5.

Dinaco was designed such that it is easy to implement and plug in new Normalizers. Each Normalizer has a task of pre-normalizing specific type of text. Dinaco was designed this way so that it becomes easy for users of Dinaco to add new Normalizers. There are Normalizers which have already been implemented and plugged into Dinaco. These Normalizers are used to normalize references to Bible scriptures, dates and currency values.

Dinaco uses the value of the `hasType` property to identify NSWs and to delegate Normalizers. The range of the `hasType` property of the `DinacoResource` OWL class is made up of the instances of the `Type` OWL class. The `Type` class has the `hasName` and the `hasFormat` properties. The `hasName` property is the datatype of the contents of the annotated XML element. Dinaco uses the value of the `hasName` property to identify the Normalizer to be used to normalize a `DinacoResource`. The `hasFormat` property is the format of the datatype. The value of the `hasFormat` is only used by the Normalizer for dates. The Normalizers for the Bible scriptures and for the currency do not use it. The example in section 3.2.3.3 shows how to use the `hasType` property.

3.2.2.1 References to Bible scriptures

Dinaco has a `BibleVerseNormalizer`. The decision to add the `BibleVerseNormalizer` was taken after finding out that both the Festival and the eSpeak TTS tool are not able to read the references to Bible scriptures correctly. Table 3.1 shows outputs from Festival and eSpeak when they read “1 John 1:1-3”. It can be seen from this table that it is not easy to understand the outputs of the two TTS tools. It is difficult to tell whether its verse 13 or its verse 1 to 3. The `BibleVerseNormalizer` normalizes pieces of text which have been identified by AAC as of type “scripture” such that it becomes easy for TTS tools to read these pieces of text. For example the `BibleVerseNormalizer` converts “1 John 1:1-3 ” to “1 John , chapter 1 , verse 1 to verse 3”. Most TTS tools are likely to be able to read the normalized version of the references to scriptures.

	Output from Festival	Output from eSpeak
1 John 1:1-3	“one John one one three”	“one John one colon one three”

Table 3.1: Reading of the Bible verse by different speech synthesis tools

3.2.2.2 Dates

Dates are the other type of content which TTS tools find difficult to read fluently. Table 3.2 shows outputs from Festival and eSpeak when they read “09-04-2008”. Dinaco has a `DateNormalizer` which normalizes `DinacoResources` which are identified

August 19, 2010

by AAC as of type “date”. For example, the DateNormalizer converts “09-04-2008” (assuming that the value of the hasFormat property is dd-MM-yyyy) to “09 April 2008” . The DateNormalizer also appends semantics to help TTS tools to recognize dates.

Written form	Output from Festival	Output from eSpeak
09-04-2008	“zero nine zero four two zero zero eight”	“zero nine zero four two thousand and eight”

Table 3.2: Reading of the dates by different speech synthesis tools

3.2.2.3 Currency

TTS tools may also not be able to fluently read numeric values of currency.

Table 3.3 shows the outputs from Festival and eSpeak when they read “R300”. Dinaco has a ZACurrencyNormalizer which normalizes DinacoResources which are identified by AAC as of type “zacurrency”. For example, the ZACurrencyNormalizer converts “R300” to “Three hundred, Rands”.

Written form	Festival	eSpeak
R300	“R three zero zero”	“R three hundred”

Table 3.3: Reading of the South African currency

3.2.3 Outputs from Dinaco

This section discusses the outputs from Dinaco. Section 3.2.3.1 discusses the markup language of the interfaces that Dinaco generates. Section 3.2.3.4 discusses how Dinaco generates the navigation menu. Section 3.2.3.2 discusses the limitations of Dinaco with respect to the HTML tags that it is able to convert. Section 3.2.3.3 discusses an example which is meant to help the reader to understand the outputs from Dinaco and to understand how to use Dinaco.

3.2.3.1 Markup language

The speech interface that Dinaco produces as output after doing transcoding is written in VoiceXML. See chapter 2 for an introduction to VoiceXML.

VoiceXML was chosen because it is a markup language recommended by W3C for generating audio dialogues. This gives us the following benefits. Firstly, it makes Dinaco to produce an interface which is ready to be used without doing further transcoding. This makes it possible for users of Dinaco to have a testing phase, where they interact with the generated VoiceXML interface in a similar manner to which web users will. Secondly, VoiceXML has tags which make it possible to append semantics of content. Lastly, the VoiceXML interface can be deployed such that it becomes accessible to people who use different communication devices and who connect using different networks. The VoiceXML application can be accessed telephonically. It can also be downloaded to and be interacted with on the user's local work station.

3.2.3.2 Limitations

Dinaco has the capabilities which enable it to transcode web pages which give information to users. Dinaco does not have the capabilities to transcode all HTML tags. For example, it does not have capabilities to deal with web pages which feature forms. There has been research on how to transcode web pages which feature forms (see [29], for example). Another example is that Dinaco does not have the capabilities to deal with links.

Dinaco was designed to produce usable speech interfaces by getting hints from human beings/systems which have information about the content that is being transcoded. The contents and the structure of the VoiceXML interface that Dinaco generates depend on the annotations about content and the annotations about the interface which Dinaco takes as inputs, before the transcoding process begins. Without these annotations, Dinaco is not able to do pre-normalization, include language semantics in the VoiceXML interface and to generate the menu.

3.2.3.3 Example

This section shows an example of the inputs that Dinaco uses and the VoiceXML interface which it generates. In this example, Dinaco is used to transcode a Web page which displays information about the two former South African state presidents,

Nelson Mandela and Frederik de Klerk. The HTML markup of this Web page (profiles.html) is shown in figure 3.6. The annotations about content (AAC) which describe the contents of profiles.xml are shown in figure 3.9. The annotations about the interface (AAI) which describe how to convert the HTML markup to VoiceXML are shown in figure 3.7. The inputs that Dinaco takes are described below.

- The XML document (profiles.xml) which was transformed by profiles.xslt into profiles.html is shown in figure 3.8. The annotations which describe the contents of profiles.xml are shown in figure 3.9. These annotations indicate that “de Klerk” and “Frederik” are Afrikaans words and that Mandela is an isiXhosa word. These annotations also indicate that in profiles.xml, elements with the XPath expression `people/person/date` have dates and that the format of these dates is `dd/MM/yyyy`.
- The XSLT document (profiles.xslt) which was used to produce profiles.html is shown in figure 3.10. The annotations about the interface are in figure 3.7. These annotations instruct Dinaco to fragment profiles.html using the `h2` tag.

The VoiceXML interface which Dinaco generated using the above mentioned inputs is shown in figure 3.11. The dialog modelled by this interface is started by the system by reading: “Profiles of former presidents” .“For Mandela press 1, for de Klerk press 2, for an introduction to this website press 3”. If the user presses 1, the system reads “Full name: Nelson Mandela. Date of birth: 18 July 1918” and goes back to play the menu again. If the user presses 2, the system reads “Full name: Frederik de Klerk. Date of birth: 18 March 1936” and goes back to play the menu again. If the user presses 3, the system reads “Profiles of former presidents”. The system reads the “Mandela” text using the isiXhosa voice, if it exists. The system reads the “de Klerk” text using the Afrikaans voice, if it exists.

```

<?xml version="1.0" encoding="UTF-8"?>
<html>
  <body>
    <h1>Profiles of former presidents</h1>
    <h2>Mandela</h2>
    Full name: Nelson Mandela<br/>
    Date of birth: 18/07/1918
    <h2>de Klerk</h2>
    Full name: Frederik de Klerk<br/>
    Date of birth: 18/03/1936
  </body>
</html>

```

Figure 3.6: The profiles HTML document.

```

<!DOCTYPE rdf:RDF [
  <!ENTITY rdf "http://www.w3.org/1999/02/22-rdf-syntax-ns#">
  <!ENTITY rdfs "http://www.w3.org/2000/01/rdf-schema#">
  <!ENTITY owl "http://www.w3.org/2002/07/owl#">
  <!ENTITY xsd "http://www.w3.org/2001/XMLSchema#">
  <!ENTITY dinaco "http://star.uct.ac.za/dinaco/
    interfaceAnnotationsSchema#">
  <!ENTITY star "http://star.uct.ac.za/dinaco/
    interfaceAnnotations#">
  <!ENTITY owl11 "http://www.w3.org/2006/12/owl11#"> ]>
<rdf:RDF xmlns:rdf="&rdf;" xmlns:rdfs="&rdfs;" xmlns:owl="&
  owl;" xmlns:owl11="&owl11;" xmlns:dinaco="&dinaco;" xmlns
  :star="&star;" xml:base="&star;">
  <owl:Ontology rdf:about="interfaceAnnotations"/>
  <dinaco:Element rdf:ID="h2">
    <dinaco:startsMenuWithLevel rdf:datatype="http://www.w3.
      org/2001/XMLSchema#string">0
  </dinaco:startsMenuWithLevel>
  <dinaco:hasTagName rdf:datatype="http://www.w3.org/2001/
    XMLSchema#string">
    h2
  </dinaco:hasTagName>
</dinaco:Element>

```

Figure 3.7: Annotations about the profiles HTML interface

```
<?xml version="1.0" ?>
<people id="birthDays">
  <person id="mandela">
    <firstName id="Mname">Nelson</firstName>
    <lastName id="Msurname">Mandela</lastName>
    <dateOfBirth>18/07/1918</dateOfBirth>
  </person>
  <person id="fw">
    <firstName id="Fname">Frederik</firstName>
    <lastName id="Fsurname">de Klerk</lastName>
    <dateOfBirth>18/03/1936</dateOfBirth>
  </person>
</people>
```

Figure 3.8: The source XML document

```

1 <?xml version="1.0"?>
2 <!DOCTYPE rdf:RDF [
3 <!ENTITY rdf "http://www.w3.org/1999/02/22-rdf-syntax-ns#">
4 <!ENTITY rdfs "http://www.w3.org/2000/01/rdf-schema#">
5 <!ENTITY owl "http://www.w3.org/2002/07/owl#">
6 <!ENTITY xsd "http://www.w3.org/2001/XMLSchema#">
7 <!ENTITY dinaco "http://star.uct.ac.za/dinaco/
   contentAnnotationsSchema#">
8 <!ENTITY star "http://star.uct.ac.za/dinaco/
   contentAnnotations#">
9 <!ENTITY owl11 "http://www.w3.org/2006/12/owl11#"> ]>
10 <rdf:RDF xmlns:rdf="&rdf;" xmlns:rdfs="&rdfs;" xmlns:owl="&
   owl;" xmlns:owl11="&owl11;" xmlns:dinaco="&dinaco;" xmlns
   :star="&star;" xml:base="&star;">
11 <owl:Ontology rdf:about="ContentAnnotations"/>
12 <dinaco:Type rdf:ID="date">
13 <dinaco:hasName rdf:datatype="&xsd:string">date</dinaco:
   hasName>
14 <dinaco:hasFormat rdf:datatype="&xsd:string">dd/MM/yyyy</
   dinaco:hasFormat>
15 </dinaco:Type>
16 <dinaco:DinacoResource rdf:ID="dateForEvents">
17 <dinaco:hasXPath rdf:datatype="&xsd:string">people/person/
   dateOfBirth</dinaco:hasXPath>
18 <dinaco:hasType rdf:resource="#date"></dinaco:hasType>
19 </dinaco:DinacoResource>
20 <dinaco:DinacoResource rdf:ID="mandelaSurname">
21 <dinaco:hasXPath rdf:datatype="&xsd:string">birthDays#
   Msurname</dinaco:hasXPath>
22 <dinaco:hasLanguage rdf:datatype="&xsd:string">xh</dinaco:
   hasLanguage>
23 </dinaco:DinacoResource>
24 <dinaco:DinacoResource rdf:ID="FName">
25 <dinaco:hasXPath rdf:datatype="&xsd:string">birthDays#
   FName</dinaco:hasXPath>
26 <dinaco:hasLanguage rdf:datatype="&xsd:string">af</dinaco:
   hasLanguage>
27 </dinaco:DinacoResource>
28 <dinaco:DinacoResource rdf:ID="Fsurname">
29 <dinaco:hasXPath rdf:datatype="&xsd:string">birthDays#
   Fsurname</dinaco:hasXPath>
30 <dinaco:hasLanguage rdf:datatype="&xsd:string">af</dinaco:
   hasLanguage>
31 </dinaco:DinacoResource>
32 </rdf:RDF>

```

Figure 3.9: Annotations about content for the profiles XML document

August 19, 2010

```
<?xml version="1.0" encoding="UTF-8"?>
<xsl:stylesheet version="1.0" xmlns:xsl="http://www.w3.org
  /1999/XSL/Transform">
  <xsl:output indent="yes" method="xml"/>
  <xsl:template match="/"><xsl:apply-templates /></xsl:
    template>
  <xsl:template match="people">
    <html>
      <body>
        <h1>Profiles of former presidents</h1>
        <xsl:apply-templates select="person" />
      </body>
    </html>
  </xsl:template>
  <xsl:template match="person">
    <h2><xsl:value-of select="lastName" /></h2>
    Full name: <xsl:value-of select="firstName" /><xsl:text>
      </xsl:text>
    <xsl:value-of select="lastName" />
    <br/>Date of birth: <xsl:value-of select="dateOfBirth" />
  </xsl:template>
</xsl:stylesheet>
```

Figure 3.10: The XSLT document which generates the profiles HTML document

```

1 <vxml version=" 2.0 "
2 xmlns=" http://www.w3.org/2001/vxml"
3 xmlns:xsi=" http://www.w3.org/2001/XMLSchema-instance "
   xsi:schemaLocation=" http://www.w3.org/2001/vxml_ http://
   www.w3.org/TR/voicexml20/vxml.xsd">
4 <form id="form_0_0">
5   <block>
6     <prompt><s>Birth day events calendar</s></prompt>
7   </block><block><submit next="#Menu0" /></block>
8 </form>
9 <menu dtmf="true" id="Menu0">
10  <prompt>
11    <enumerate>
12      <break/>For<value expr="_prompt" />press<value expr="
   _dtmf" />
13    <break/>
14  </enumerate>
15 </prompt>
16 <choice next="#form_0_1">Mandela</choice>
17 <choice next="#form_0_2">de Klerk</choice>
18 <choice next="#form_0_0">
19   an introduction to this web site
20 </choice>
21 </menu>
22 <form id="form_0_1">
23   <block>
24     <prompt>
25       <s>Full name:</s><s>Nelson</s>
26       <s xml:lang="xh">Mandela</s><s>Date of birth:</s>
27       <s><say-as interpret-as="date">18 July 1918</say-as></s>
28     </prompt>
29   </block><block><submit next="#Menu0" /></block>
30 </form>
31 <form id="form_0_2">
32   <block>
33     <prompt>
34       <s>Full name:</s><s xml:lang="af">Frederik</s>
35       <s xml:lang="af">de Klerk</s><s>Date of birth:</s>
36       <s><say-as interpret-as="date" >18 March 1936</say-as><
   /s>
37     </prompt>
38   </block><block><submit next="#Menu0" /></block>
39 </form>
40 </vxml>

```

Figure 3.11: The profiles VoiceXML document which Dinaco outputs

3.2.3.4 Navigation menu

The generated VoiceXML interface has menus which help the user to navigate through different sections of the web page and to be able to move back and forth. Dinaco divides the Web page into VoiceXML forms. These forms divide the contents of the Web page into fragments. The menus guide the user on how to move from one fragment to the other. Dinaco is guided by the AAI on how to create these fragments. Each form that Dinaco creates ends with a block element which takes the user to the next menu, see an example in section 3.2.3.3.

The menus accept DTMF key inputs, not speech inputs. The decision to generate menus this way was to make Dinaco to generate simpler interfaces which can be used without installing speech recognition tools and without facing the usability problems which users face when using applications which feature speech recognition.

3.2.3.5 Appending semantics

Web pages can feature multilingual text which TTS tools cannot read fluently, because they are unable to identify the correct language which should be used when reading different parts of the sentence or of the paragraph. Web pages which feature proper nouns, like the names of people and of places are an example. The inability of TTS tools to read multilingual text fluently can make users who access the web through speech to not understand the contents of the web page. Users of speech can miss important information, like “who the story is about” or “where the event took place”.

Dinaco endeavours to use AAC to resolve the problem of reading multilingual text, by taking advantage of the speech synthesis markup language (SSML) and VoiceXML. Dinaco can be given AAC to make it to create VoiceXML interfaces which have language semantics. Language semantics are the semantic descriptions of content which can be used by TTS tools which understand SSML to choose the correct language to use when reading text.

During the transcoding process, Dinaco uses the values of the `hasLanguage` property (from AAC) to attain information about the language semantics of `DinacoResources`. For each `DinacoResource` which has a value for the `hasLanguage` property,

that value becomes the value of the `xml:lang` attributes of the VoiceXML sentences where the `DinacoResource` (the piece of text) features in. This can be seen in line 35 of figure 3.11. In this figure, the value of the `xml:lang` attribute of the sentence which has the “de Klerk” text is “af”. This “af” is the value of the `hasLanguage` property in line 30 of the AAC in figure 3.9.

The `DateNormalizer` also appends semantics on the date values. The `DateNormalizer` puts the normalized date into the `say-as` tag. As it can be seen in line 27 of figure 3.11, that “18 July 1918” is in the `say-as` tag which has a `interpret-as` attribute with the value of “date”. The `say-as` tag is meant to help TTS tools which understand SSML to understand that they should read "18 July 1918" as a date value.

Dinaco also has a `CurrencyNormalizer` which does not convert the numeric currency values into words, like it is done by the `ZACurrencyNormalizer`. The `CurrencyNormalizer` was implemented to further demonstrate how Dinaco can be used to create VoiceXML interfaces which have semantics about NSWs. These semantics can help TTS tools to correctly normalize NSWs. The `CurrencyNormalizer` makes Dinaco to put currency values inside the `say-as` tags. For example, Dinaco can use the `CurrencyNormalizer` to put “R300” into the `say-as` tag like “`<say-as interpret-as="currency">R300</say-as>`”. Dinaco uses the `CurrencyNormalizer` to create the `say-as` tag for the `DinacoResource`, if the value of its `hasType` property is a Type OWL class which has "currency" as the value of its `hasName` property.

3.2.4 Deployment

Previous transcoding systems were designed to transcode an already existing HTML document. These transcoding systems can do transcoding both on the client side and on the server side. Dinaco does not use an already existing HTML document during its transcoding process. It uses the artifacts which were used to create the HTML document, for example, the XSLT stylesheet and the XML documents. This makes Dinaco to be suitable for deployment on the server side, where it can easily have access to these artifacts.

3.3 The impact of separating annotations in the XinterfaceDevMeth

Section 3.1 introduced separation of annotations. Section 3.2 described how Dinaco uses separated annotations during its transcoding process. This section discusses the advantages of separation of annotations and it is organized as follows. Section 3.3.1 explains how separating annotations makes it easier to design the software components which are used when creating annotated HTML interfaces in a manner that separates concerns. The advantages of achieving this separation of concerns are discussed in sections 3.3.2 to 3.3.5 and can be summarized as follows:

- It becomes easier to divide a complex software system into distinct components which can be created and managed by different members of the software development team.
- It becomes easier to manage changes to the interfaces, on the XML documents and on the annotations.
- It makes it possible to write annotations about content without being limited by the HTML markup.
- It makes it possible to reuse both the annotations about the interface and about content.
- It makes it easier to interchange annotated XML documents.

3.3.1 Separation of annotations and separation of concerns

Separation of concerns (SoC) refers to the software design principle, where a complex problem of designing and developing a complex system is divided into smaller and distinct components which are easier to solve and to manage. See [66] for an introductory information on SoC. This section explains the two issues of SoC when designing software components which are used to create annotated HTML

interfaces in the XinterfaceDevMeth. Firstly, the section explains how mixing annotations leads to a lower degree of SoC. Secondly, the section explains how separating annotations makes it easier to achieve a higher degree of SoC.

The software components which are created in the XinterfaceDevMeth are the XSLT stylesheet, the XML documents and the annotations. These components cannot be easily created in a manner that separates concerns, if annotations are not separated. The reason for this is that the annotations component overlaps with the XSLT stylesheets and with the XML documents. This overlap makes the software components to be divided in a manner that mixes concerns, as illustrated in figure 3.12.

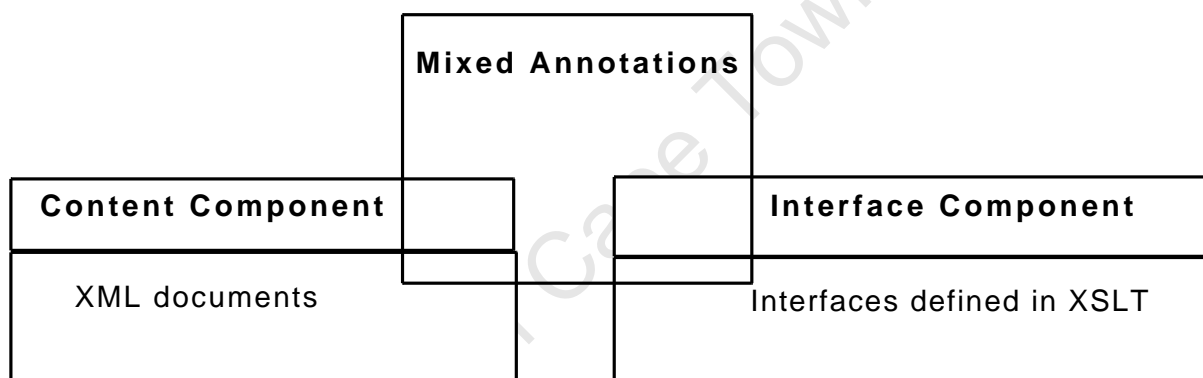


Figure 3.12: Mixed annotations overlap with the Interface Component and with the Content Component.

Mixed annotations, are created as described in chapter 2, section 2.3.1. Mixed annotations are written to describe how to transcode both the HTML elements and their contents. Mixed annotations are HTML dependent because both the annotations which describe how to transcode HTML elements and those which describe the contents of HTML elements are written using the HTML elements. This mixing of concerns has the following three disadvantages. The first disadvantage is that annotations cannot be created until both the XSLT stylesheets and the XML documents components have been created. The second disadvantage is that annotations are affected by the changes made on both the XSLT stylesheets and the XML documents. The third disadvantage is that the descriptions of contents of the HTML elements are limited to those which can be written using the elements of the HTML document that is being transcoded.

3.3. The impact of separating annotations in the XinterfaceDevMeth 69

Separating annotations makes it easier (in the XinterfaceDevMeth) to design and create annotated web interfaces in a manner that separates concerns. The annotated HTML interfaces are created using two components, which can be created and managed independently of each other. These components are the Content Component and the Interface Component. Each of these components is divided into two sub-components. The Content Component is divided into the XML documents and the annotations which describe the contents of these documents (annotations about content). The Interface Component is divided into the XSLT stylesheet and the annotations about the interface. These components are illustrated in figure 3.13.

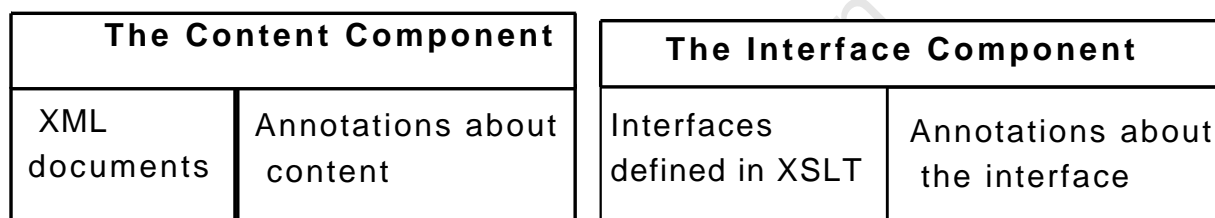


Figure 3.13: Separation of software components if annotations are separated.

3.3.2 It becomes easier to separate tasks

Separating annotations makes it possible to divide the task of creating annotated HTML interfaces into three sub-tasks: designing of the XML documents, creation of the Content Component and of the Interface Component. These tasks are shown in the process diagram of figure 3.14. The first sub-task is to design the XML schema of the XML documents which are to be published on the web. The second sub-task is to create the Content Component. The third sub-task is to create the Interface Component. The XML schema is required when creating both the Content Component and the Interface Component. However, the Content Component and the Interface Component can be created independent of each other (after the XML schema has been created).

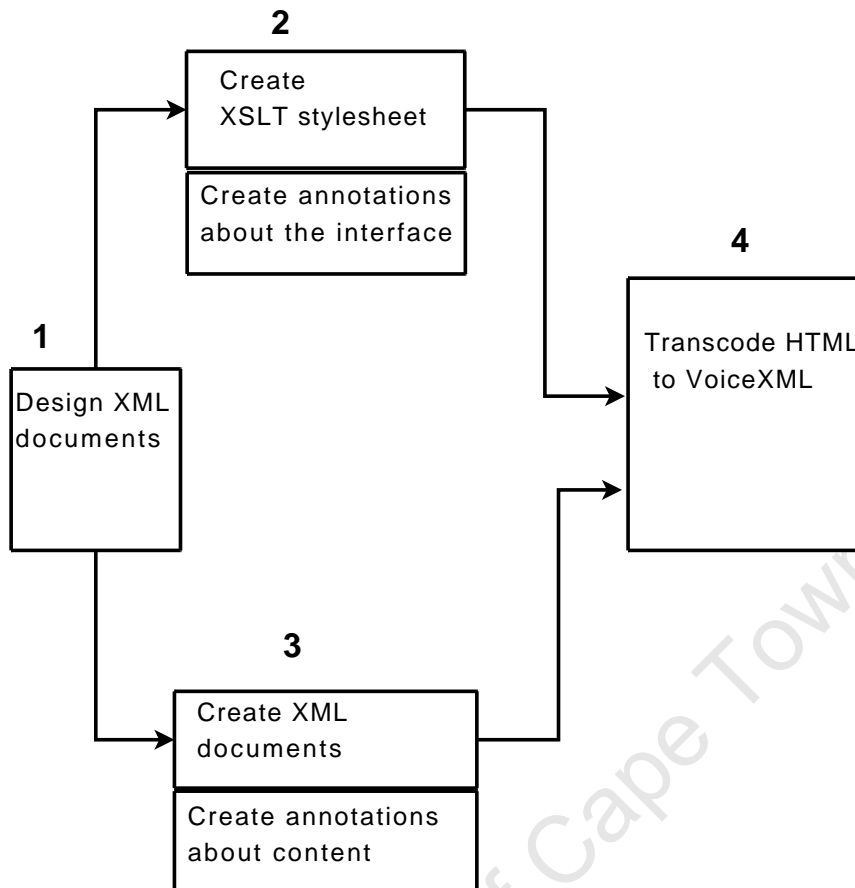


Figure 3.14: The process flow of the XinterfaceDevMeth if annotations are separated.

Separating annotations allows the developers to create and annotate the content and the interface separately from each other. This occurs because annotations are separated in a similar way to how the interface is separated from the content in the XinterfaceDevMeth. The annotations are separated such that the web interface development team and the web content creation team are able to work independently of each other. The web interface development team creates and maintains the interface and the annotations about it. The web content creation team creates and maintains the web content and the annotations about it.

This makes creating annotated HTML interfaces to be done in the way that matches the processes in the XinterfaceDevMeth. This makes it easier to allocate different tasks to teams which work independently of each other. Making it possible for different team members to create annotations was also proposed in [67].

3.3.3 Managing changes is easier

In separated annotations, it is easier to manage changes to the Content Component and to the Interface Component. This happens because the interface development team and the content creation team are able to continuously make changes to the component that they are responsible for, without making changes to the other team's component. This reduces the cost and the inconvenience of managing overlapping components.

The annotations about the interface are written in terms of the HTML elements of the interface that is to be transcoded. This makes changes to the interface to affect only the annotations about the interface, but not annotations about content. The impact of this is that the interface development team is able to make changes to the interface and on the annotations about it, without making changes to the Content Component.

The annotations about content are written in terms of the structural elements of the source XML documents. This makes changes to the XML documents to affect annotations about the content, but not annotations about the interface. The impact of this is that the content creation team is able to make changes to the content and on the annotations about it, without making changes to the Interface Component.

3.3.4 Writing annotations without being limited by the HTML markup

HTML dependent annotations limit the semantic descriptions which can be written about content to those which can be expressed in terms of the elements of the HTML interface that is to be transcoded (see section 2.3.1.2).

If annotations are separated, then annotations about content are written in terms of the elements of the source XML documents. They are written while content is still part of the source XML documents. That is, they are written before the XSLT stylesheet aggregates contents of different elements of XML documents to become the contents of HTML elements. This allows content annotators to write statements about segments of the annotated HTML element.

Separating annotations makes it possible to write external semantic descriptions of content which cannot be written externally if annotations are only written in terms of the HTML elements of the transcoded interface. Separating annotations makes it possible to write external semantic descriptions of content which cannot be written externally if annotations are only written in terms of the HTML elements of the transcoded interface. Section 2.3.1.2 discussed HTML dependent (mixed) annotations, where it was not possible to annotate the second paragraph of figure 2.11 as having the English words (“Nelson”, for example) and “Mandela” which is the isiXhosa word. Section 3.2.3.3 explained how it was possible to use separated annotations to write different language semantics of “Nelson” and of “Mandela” which appear in figure 3.6.

3.3.5 Annotations become reusable and sharable

In separated annotations, both the annotations about the interface and the annotations about content can be reused because they are not mixed. The content annotations language (CAL) is designed specifically to write annotations about the content. This makes annotations about content to be reusable. They can be used to make content accessible in speech, by giving it as input to Dinaco. They can also be used by other applications which also need to understand the contents of the source XML documents for different purposes.

Applications which receive annotated XML documents can be able to do intelligent processing on the contents of XML documents. For example, they can be able to integrate, search, translate and compare data which comes from different applications. Creating separated annotations makes it possible to create systems which interchange XML documents, together with annotations about those XML documents. This is a benefit because XML was designed to make it easier for Internet applications to interchange documents. An example of the interchange of XML documents between applications is that which happens between a web service and the applications which access it.

Creating semantic descriptions of web resources is also the goal of the Semantic Web. See [68] for the applications of Semantic Web in Biomedical computing and [69]

for the research which relates the Semantic Web with web services. See also [70] for a discussion on the importance of creating meta data which describe resources and for the framework which was proposed.

The interface annotations are also reusable. That is, they can be used as the annotations about different HTML documents which are created by the same XSLT stylesheet. This is made possible by not including semantic descriptions about content, because content from different XML documents is likely to have different descriptions. See section 2.1.2.5 for a discussion on how to reuse XSLT stylesheets.

3.4 Summary of chapter 3

This chapter introduced the concept of separation of annotations. It described how annotations which are used by the transcoder are separated into the annotations about content and the annotations about the interface. The annotations about the content help Dinaco to pre-normalize text and to create VoiceXML interfaces which have language semantics and semantic descriptions of non-standard words. The advantages of separated annotations were also discussed. Chapter three describes how Dinaco creates the VoiceXML interface.

Chapter 4

Implementation of the transcoding system

Dinaco is a transcoding system which converts HTML interfaces into VoiceXML. The HTML interfaces that Dinaco transcodes are those which were created using the XinterfaceDevMeth. Dinaco is written using the Java programming language. This chapter discusses how Dinaco converts HTML interfaces into VoiceXML interfaces. The chapter discusses the components and the technologies which Dinaco uses.

4.1 The Transcoding process

The transcoding process of Dinaco can be divided into 4 activities: Load inputs, Create the intermediate XSLT, Create the intermediate HTML and Create the VoiceXML interface. These activities are depicted in figure 4.1. Section 4.1.1 discusses the first activity: Load inputs. Section 4.1.2 discusses the third activity: Create the intermediate HTML. Section 4.1.3 discusses the second activity: Create the intermediate XSLT. The third activity is described before the second activity with an aim of making it easy for the reader to understand the purpose of both the first and the third activities. Section 4.1.4 discusses the fourth activity: Create the VoiceXML interface.

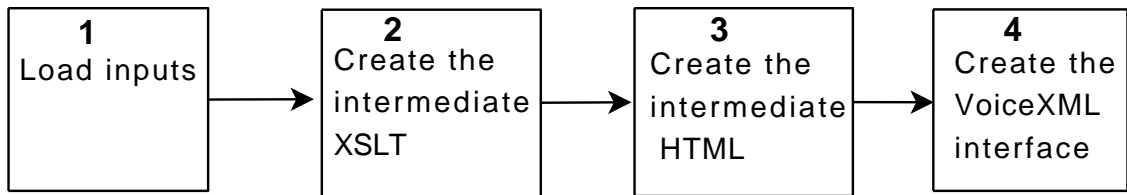


Figure 4.1: The Activities in Dinaco's transcoding process

4.1.1 Load inputs

Dinaco takes the following inputs:

1. The XSLT stylesheet which was used to create the HTML interface which Dinaco has to transcode.
2. The source XML document which was transformed by the XSLT stylesheet while creating the HTML interface which Dinaco has to transcode.
3. Annotations about content (AAC) which describe the contents of the source XML document.
4. Annotations about the interface which guide Dinaco on how to go about transcoding the HTML interface.

4.1.2 Create the intermediate HTML

Dinaco uses annotations which describe the content which is being transcoded and guide it on how to convert an HTML interface into a VoiceXML interface. AAC are written in terms of the XML documents which were transformed by the XSLT stylesheet to HTML.

Dinaco uses the XSLT stylesheet which it takes as input, in order to be able to use AAC. For purposes of clarity, this XSLT stylesheet is called the original stylesheet. Dinaco uses the original stylesheet to produce the new XSLT stylesheet, called the intermediate stylesheet. The output which is generated by the original stylesheet is different from that which is generated by the intermediate stylesheet.

The original stylesheet generates documents which have a correct HTML markup, such that they are ready to be rendered correctly by graphical web browsers. The

```
<?xml version="1.0" ?>
<people id="birthDays">
  <person id="mandela">
    <lastName id="Msurname">Mandela</lastName>
  </person>
</people>
```

Figure 4.2: The profiles source XML document

intermediate stylesheet does not produce a HTML document, it produces an intermediate HTML document. The intermediate document does not have HTML elements only. The intermediate HTML document has “anno” tags, in addition to HTML elements. The “anno” elements are the annotation elements which help Dinaco to be able to use AAC.

If there is an original stylesheet which transforms a simple XML document, profiles.xml which is shown in figure 4.2, to a simple HTML document: original.html which is shown in figure 4.4. Suppose that the “Mandela” text in original.html was extracted from the lastName element of profiles.xml by the original XSLT stylesheet. Dinaco can use annotations about the contents (AAC) of profiles.xml which are shown in figure 4.3 to produce intermediate.xslt. This intermediate.xslt transforms profiles.xml to intermediate.html which is shown in figure 4.5.

The main difference between original.html and intermediate.html is that intermediate.html has the “anno” element. The text “Mandela” which is a text content of the “p” element in original.html is the text content of the “anno” element in intermediate.html.

Each “anno” element in the intermediate HTML document has a “uri” attribute. The value of the “uri” attribute is the uri of the DinacoResource in AAC which describes the text content of the “anno” element. The value of the “uri” attribute of the “anno” element in figure 4.5 is “mandelaSurname”. This “mandelaSurname” is the value of the rdf:ID of the DinacoResource in the AAC (see line 12 in figure 4.3).

```

1 <?xml version="1.0"?>
2 <!DOCTYPE rdf:RDF [
3 <!ENTITY rdf "http://www.w3.org/1999/02/22-rdf-syntax-ns#">
4 <!ENTITY rdfs "http://www.w3.org/2000/01/rdf-schema#">
5 <!ENTITY owl "http://www.w3.org/2002/07/owl#">
6 <!ENTITY xsd "http://www.w3.org/2001/XMLSchema#">
7 <!ENTITY dinaco "http://star.uct.ac.za/dinaco/
   contentAnnotationsSchema#">
8 <!ENTITY star "http://star.uct.ac.za/dinaco/
   contentAnnotations#">
9 <!ENTITY owl11 "http://www.w3.org/2006/12/owl11#"> ]>
10 <rdf:RDF xmlns:rdf="&rdf;" xmlns:rdfs="&rdfs;" xmlns:owl="&
   owl;" xmlns:owl11="&owl11;" xmlns:dinaco="&dinaco;" xmlns
   :star="&star;" xml:base="&star;">
11 <owl:Ontology rdf:about="ContentAnnotations"/>
12 <dinaco:DinacoResource rdf:ID="mandelaSurname">
13 <dinaco:hasXPointer rdf:datatype="&xsd:string">birthDays#
   Msurname</dinaco:hasXPointer>
14 <dinaco:hasLanguage rdf:datatype="&xsd:string">xh</dinaco:
   hasLanguage>
15 </dinaco:DinacoResource>
16 </rdf:RDF>

```

Figure 4.3: Annotations about content for the profiles XML document

```

<html>
  <body>
    <p>Mandela</p>
  </body>
</html>

```

Figure 4.4: The original HTML document

```

1 <?xml version="1.0" encoding="UTF-8"?>
2 <html>
3 <body>
4 <p><anno uri="mandelaSurname">Mandela</anno></p>
5 </body>
6 </html>

```

Figure 4.5: An example of the HTML document which is produced by the intermediate XSLT stylesheet

```
1 <?xml version="1.0" encoding="UTF-8"?>
2 <xsl:stylesheet version="1.0" xmlns:xsl="http://www.w3.org
   /1999/XSL/Transform">
3 <xsl:output indent="yes" method="html"/>
4 <xsl:template match="/">
5   <xsl:apply-templates />
6 </xsl:template>
7 <xsl:template match="people">
8   <html>
9     <body>
10      <p><xsl:apply-templates select="person" /></p>
11    </body>
12  </html>
13 </xsl:template>
14 <xsl:template match="person">
15   <xsl:value-of select="lastName" />
16 </xsl:template>
17 </xsl:stylesheet>
```

Figure 4.6: An example of the original XSLT stylesheet

4.1.3 Create the intermediate XSLT

In step 2 of the transcoding process, Dinaco creates the intermediate XSLT stylesheet. The intermediate stylesheet is used to transform source XML documents to intermediate HTML documents. Dinaco has a transcoding module called the `Xslt2XSLTTranscoder` which it uses to create the intermediate XSLT stylesheet.

The `Xslt2XSLTTranscoder` begins the process of creating the intermediate XSLT by converting the original XSLT stylesheet into its document object model and creating an empty intermediate XSLT stylesheet. Sections 4.1.3.1 to 4.1.3.3 describe how the `Xslt2XSLTTranscoder` uses the original stylesheet to create the intermediate XSLT stylesheet. The XSLT elements which are not mentioned in this section are copied from the original XSLT straight to the intermediate XSLT as they are.

4.1.3.1 Creating the intermediate `xsl:stylesheet` and `xsl:output`

When the `Xslt2XSLTTranscoder` encounters the `xsl:stylesheet` element it takes three actions. The first action is to create a copy of the `xsl:stylesheet` which it has encountered and to add the `xmlns:trans` attribute to the created copy. The second

```
1 <xsl:stylesheet xmlns:trans="dinaco.transcoder.linker.  
    SourceAndResultsLinker" xmlns:xsl="http://www.w3.org  
    /1999/XSL/Transform" version="1.0">  
2 <xsl:output indent="yes" method="xml"/>  
3  
4 </xsl:stylesheet>
```

Figure 4.7: The xsl:stylesheet element in the intermediate XSLT

action is to put the created copy of the xsl:stylesheet into the intermediate XSLT stylesheet, as the root element. The third action is to create the xsl:output element and make it to be the child element of the xsl:stylesheet in the intermediate XSLT. After the third action has been processed the intermediate XSLT looks like the one in figure 4.7.

The class which is used to create the “anno” elements which are found in the intermediate HTML document is `dinaco.transcoder.linker.SourceAndResultsLinker`. The `SourceAndResultsLinker` requires access to the AAC and to the source XML document in order to be able to create the correct values of the uri attributes of the anno elements.

The `Xslt2XSLTTranscoder` ignores the xsl:output elements from the original stylesheet. This decision was taken to avoid errors which were occurring if the value of the method attribute of the xsl:output is “html”.

4.1.3.2 Creating the intermediate xsl:template

When the `Xslt2XSLTTranscoder` encounters the xsl:template element it takes the following four actions: The first action is to create a copy of this xsl:template and add the copy into the intermediate XSLT. The second action is to add the xsl:value-of element as the first child element of the created copy. This xsl:value-of element makes the `startTmplt` method of the `SourceAndResultsLinker` to be called when the created xsl:template is being processed. This method helps the `SourceAndResultsLinker` to keep track of which xsl:template is currently being processed. The `startTmplt` method takes the template ID as the parameter. The template IDs are created by the `Xslt2XSLTTranscoder`. Line 2 of figure 4.8 shows the xsl:value element which the

```
1 <xsl:template match="/">
2   <xsl:value-of select="trans:startTmplt(1)"/>
3   ...
4   ...
5   <xsl:value-of select="trans:endtmplt(1)"/>
6 </xsl:template>
```

Figure 4.8: The xsl:template element in the intermediate XSLT stylesheet

Xslt2XSLTTranscoder created when it encountered the xsl:template which begins in line 4 of the original stylesheet (see figure 4.6).

The third action is to add all the child elements of the xsl:template from the original XSLT stylesheet into the xsl:template which was created in the intermediate XSLT stylesheet. The fourth action is to add the xsl:value-of element as the last child element of the xsl:template of the intermediate XSLT. This xsl:value-of element makes the endtmplt method of the SourceAndResultsLinker to be called when the XSLT processor is about to finish processing the xsl:template. This method also helps the SourceAndResultsLinker to keep track of which xsl:template is currently being processed. It also takes the same parameter as the startTmplt. See, line 5 of figure 4.8 for the example on how the xsl:value child element is created.

4.1.3.3 Creating the intermediate xsl:value-of and the intermediate xsl:apply-templates

The xsl:value-of and the xsl:apply-templates elements can make the XSLT processor to copy text content from the source XML document into the resulting HTML document. The XSLT processor is able to identify the element of the source XML document (called the source element) where this text content is copied from. The aim of the Xslt2XSLTTranscoder is to add the original xsl:value-of or the xsl:apply-templates element into the intermediate stylesheet such that the following objective is achieved. This objective is to put the text content into the anno element if there is a DinacoResource which has semantic descriptions of the source element. See line 4 of figure 4.5, for an example.

The Xslt2XSLTTranscoder takes the following three actions when it encounters

```
1 <xsl:variable name="val_1">
2   <xsl:value-of select="lastName" />
3 </xsl:variable>
4 <xsl:copy-of select="trans:valueOf(lastName,$val_1)" />
```

Figure 4.9: An example which shows how to replace the `xsl:value-of select` element

the `xsl:value-of` element: The first action is to create the `xsl:variable` element in the intermediate XSLT stylesheet. The second action which the `Xslt2XSLTTranscoder` takes is to create a copy of the original `xsl:value-of` element and make the created copy to be the child element of the `xsl:variable`. Figure 4.9 shows the elements which can be created when the `<xsl:value-of select="lastName" />` (taken from line 15 of the original XSLT stylesheet) is added into the intermediate XSLT stylesheet.

The third action which the `Xslt2XSLTTranscoder` takes is to create a `xsl:copy-of` element in the intermediate XSLT, see line 4 of figure 4.9. This `xsl:copy-of` element calls the `valueOf` function of the `SourceAndResultsLinker` and copies its results into the resulting intermediate HTML document.

The `SourceAndResultsLinker`'s `valueOf` function creates the “anno” element which has the text content which was copied from the source XML element. The `valueOf` function only returns the “anno” element if there is a `DinacoResource` which has semantic descriptions of the source XML element, it returns the text content as it is if there is no such a `DinacoResource`.

The `valueOf` function takes two parameters. The first parameter is the `Node` object, which the function uses to compute the value of the “uri” attribute of the “anno” element. The `select` attribute of the `xsl:copy-of` element sets this parameter to be the value of the `select` attribute of the `xsl:value-of` element. In figure 4.6 line 15, the value of the `select` attribute of the `xsl:value-of` is “lastName”, and that is why “lastName” is used as the first parameter of the `valueOf` function in line 4 of figure 4.9. There are situations where the value of the `select` attribute of the `xsl:value-of` does not resolve into a `Node` object. As a result, the `SourceAndResultsLinker` has different versions of the `valueOf` function which avoid errors when this happens.

The second parameter of the `valueOf` function is the text content of the “anno” node. The `select` attribute of the `xsl:copy-of` element sets this value to be the `String`

```
1 <xsl:variable name="app_var0">
2 <xsl:apply-templates />
3 </xsl:variable>
4 <xsl:copy-of select="trans:endAtmplt(.,1,$app_var0,$app_var0)" />
```

Figure 4.10: An example which shows how to add the `xsl:apply-templates` element into the intermediate XSLT

value of the `xsl:attribute`, where the `xsl:value-of` element was copied to. In line 4 of figure 4.9, the second parameter of the `valueOf` function is “`$val_1`”. This is because “`val_1`” is the name of the `xsl:attribute` where the `xsl:value-of` element was copied to. See lines 1 to 3 in figure 4.9.

The `xsl:apply-templates` element is added into the intermediate XSLT in a similar way to the `xsl:value-of` element. An exception is that the `SourceAndResultsLinker`'s `endAtmplt` function is called instead of the `valueOf` function. Figure 4.10 shows the elements which can be used to add the `xsl:apply-templates` element in line 5 of the original XSLT stylesheet. The `endAtmplt` function takes four parameters. The first parameter is the reference of the source XML element. The value of this parameter becomes the value of the `select` attribute of the `xsl:apply-templates`, if the `xsl:apply-templates` element does have it. It changes to a dot (`.`) if the `xsl:apply-templates` does not have the `select` attribute. The second parameter of the `endAtmplt` function is the ID of the template which the `xsl:apply-templates` is in, this value is worked out by the `Xslt2XSLTTranscoder`. The third and the fourth parameters of the `endAtmplt` function are both the references of the `xsl:variable` where the `xsl:apply-templates` were copied to. The third parameter passes the reference `xsl:variable` as a `Node`, while the fourth parameter passes it as a `String`.

4.1.4 Create the VoiceXML interface

Dinaco has the `H2VTranscoder`. The `H2VTranscoder` is used to transcode the intermediate HTML document which was created by the intermediate XSLT document. When doing transcoding, the `H2VTranscoder` requires access to the `AAI` and the `AAC`. When `H2VTranscoder` encounters the “`anno`” element, it then uses the `uri` attribute of the “`anno`” element to identify the `DinacoResource` which describe the

text content which is found in the “anno” elements. It then uses the properties of the DinacoResource to pre-normalize text (using its Normalizers) and to append the language semantics of text content.

When the H2VTranscoder encounters any other element, other than the anno, it assumes that its the HTML element. It then uses AAI to make a decision on how to go about converting the HTML element to VoiceXML.

4.2 Summary of chapter 4

This chapter discussed the four step process which Dinaco follows when it converts HTML interfaces to VoiceXML. The chapter discussed how Dinaco uses the original stylesheet to create the intermediate stylesheet which is used to create annotated HTML interfaces. The annotated HTML interfaces are then transcoded to produce a semantically rich interface which has pre-normalized text. Chapter 5 discusses the experiments which were conducted during this research.

Chapter 5

Experimentation

This chapter is organized as follows:

- Section 5.1.1 states the objective of the experiments which were conducted during this research.
- Section 5.1.2 states the hypothesis and the null hypothesis which were used as the guidelines and which helped to design the experiments.
- Section 5.1.3 discusses the design of the experiment which was conducted in order to test the null hypothesis. It describes the HTML interface and the speech interfaces which were used during the experiments. It also explains how the speech interfaces were evaluated for usability.
- Section 5.1.4 describes the software and the hardware which were used while preparing for and during the experiments.

5.1 The objective and the design of the experiments

5.1.1 The objective of the experiments

Previous chapters of this report have introduced and described the transcoding system called Dinaco. This chapter discusses the experiments which were conducted after Dinaco was designed and implemented. The objective of conducting the experiments was to find out if Dinaco is able to transcode HTML web pages which

feature multilingual text and non-standard words (NSWs) into usable speech interfaces. Usable speech interfaces in the context of the stated objective are speech interfaces which are created such that TTS tools become able to read multilingual text and NSWs such that users are able to understand.

5.1.2 The hypothesis and the null hypothesis

5.1.2.1 Hypothesis

Dinaco is designed such that it can be able to transcode a web page which has multilingual text and NSWs into a more usable speech interface compared to previous transcoding systems, if the conditions which are listed below hold.

1. The web page was created using the XinterfaceWebDevMeth.
2. The web page features multilingual text and NSWs which were extracted from XML documents which were transformed by the XSLT stylesheet to HTML.
3. The annotations which guide the transcoding system are written externally from the resource that they describe and they include descriptions of multilingual text and NSWs.

5.1.2.2 Null hypothesis

There is no web page which features multilingual text and NSWs which Dinaco can transcode into a more usable speech interface compared to a speech interface which can be produced using previous transcoding systems.

5.1.3 Design of the experiments

The following steps were followed to setup experiments which were used to test the null hypothesis:

- A sample web page which featured multilingual text and NSWs was designed and created using the XinterfaceWebDevMeth. More details about the sample web page are covered in section 5.1.3.1.

- The sample web page was converted into two different VoiceXML interfaces. The first interface was called the control interface and is described in more detail in section 5.1.3.2. The second interface was called the experimental interface and is described in more detail in section 5.1.3.3.
- Two experiments which were going to be used to test the control and the experimental interfaces for usability were designed. More details about these experiments are discussed in section 5.1.3.4.

5.1.3.1 The sample web page

The sample web page was created using the XinterfaceWebDevMeth and it featured multilingual text and NSWs. It had a date value, a currency value, English words and Afrikaans words which were extracted from XML documents which were transformed by the XSLT stylesheet to HTML. The sample web page was created this way so that it meets the first two conditions which are mentioned in the hypothesis statement in section 5.1.2.1.

The sample web page was created to give users information about a dance competition. It featured information about the couple who won the competition, the place where it was held, the date when it was held and the prize which was won. This sample web page is shown in figure 5.1. The XML document which was transformed by the XSLT stylesheet while creating the sample web page is shown in figure 5.2. The XSLT which was used to transform the XML document is shown in figure 5.3.

The sample web page had a mixture of English and Afrikaans words. English and Afrikaans were chosen because of the following reasons: firstly, it was going to be easier to find people who understand these languages in South Africa. Secondly, a text-to-speech (TTS) tool which has both the English and the Afrikaans voices was found. The web page had 5 Afrikaans words: Pieter, van, der, Merwe and Welkom. The sample web page had seventeen English words, if the two NSWs which are mentioned next are not included.

The NSWs which were in the Web page are a date value: 2006/01/12, and a currency value: R200 000.

```

1 <html>
2 <body><h1>The last year's competition</h1>
3 <table>
4 <tr>
5 <td>Winning couple:</td>
6 <td>Pieter van der Merwe and Hope Jacobs</td>
7 </tr>
8 <tr>
9 <td>Location:</td>
10 <td>South Africa – Welkom</td>
11 </tr>
12 <tr>
13 <td>Details:</td>
14 <td>Price was R200 000, held on 2006/01/12</td>
15 </tr>
16 </table>
17 </body>
18 </html>

```

Figure 5.1: The sample HTML web page which was transcoded during experimentation

```

<?xml version="1.0" encoding="utf-8" ?>
<record id="profiles">
  <couple>
    <male>
      <firstName id="merweFname">Pieter</firstName>
      <surname id="merweSurname">van der Merwe</surname>
    </male>
    <female>
      <firstName id="hopeFname">Hope</firstName>
      <surname id="hopeSurname">Jacobs</surname>
    </female>
  </couple>
  <competition>
    <location>
      <suburb id="suburb">Welkom</suburb>
      <country>South Africa</country>
    </location>
    <date>2006/01/12</date>
    <price>R200 000</price>
  </competition>
</record>

```

Figure 5.2: The XML document which was transformed by XSLT to produce the sample page

```

<?xml version="1.0" encoding="UTF-8" ?>
<xsl:stylesheet version="1.0" xmlns:xsl="http://www.w3.org
  /1999/XSL/Transform">
  <xsl:output indent="yes" method="html"/>
  <xsl:template match="/">
    <xsl:apply-templates />
  </xsl:template>
  <xsl:template match="record">
    <html>
      <body>
        <h1>The last year's competition</h1>
        <table>
          <xsl:apply-templates select="couple" />
          <xsl:apply-templates select="location"/>
          <xsl:apply-templates select="competition"/>
        </table>
      </body>
    </html>
  </xsl:template>
  <xsl:template match="couple">
    <tr><td>Winning couple:</td>
    <td><xsl:value-of select="male/firstName" /><xsl:text> </
      xsl:text><xsl:value-of select="male/surname" /> and <
      xsl:value-of select="female/firstName" /><xsl:text> </
      xsl:text><xsl:value-of select="female/surname" />
    </td>
    </tr>
  </xsl:template>
  <xsl:template match="location">
    <tr><td>Location:</td>
    <td>
      <xsl:value-of select="country"/><xsl:text> - </xsl:text
      ><xsl:value-of select="suburb"/>
    </td>
    </tr>
  </xsl:template>
  <xsl:template match="competition">
    <xsl:apply-templates select="location"/>
    <tr><td>Details:</td>
    <td>Price was <xsl:value-of select="price"/>, held on <
      xsl:value-of select="date"/>
    </td>
    </tr>
  </xsl:template>
</xsl:stylesheet>

```

Figure 5.3: The XSLT stylesheet which was used to create the sample web page
August 19, 2010

5.1.3.2 The control interface

The intention of creating the control interface was to create an interface which represents the speech interface which was going to be produced by previous transcoding systems, if they were to transcode the sample page. However, it was going to be difficult to create such an interface because of the following obstacles: Firstly, no transcoding system which produces VoiceXML could be downloaded and be used for free. Secondly, even if all previous transcoding systems were available, they would not produce exactly the same speech interface because there is no standard which defines how to do transcoding.

The decision was made to use Dinaco itself to create the control interface. This is because the main difference between Dinaco and previous transcoding systems which use external annotations is that Dinaco takes separated annotations as input; while previous transcoding systems take mixed annotations which are written in terms of the elements of the transcoded HTML document as input.

If a previous transcoding system was going to be used to transcode the sample web page, it would take external annotations which are written in terms of the HTML elements of the sample web page. These external annotations would limit the semantic descriptions of the contents of each HTML element of the sample web page to those which are true about all the contents of that element.

It then follows that the annotations which were going to be used by a previous transcoding system would not have language semantics of the “td” elements of the sample web page (see figure 5.1) which appear in lines 6 and 10. The reason is that the contents of each of these elements are not all English and are not all Afrikaans. Each element has a mixture of English and Afrikaans words. It also follows that these annotations would not include the semantic descriptions of the date and of the currency values which are amongst the contents of the “td” element in line 14. The reason being that this element does not only have a date value and it also does not only have the currency value.

The decision was taken to use Dinaco to transcode the sample web page, without being given the annotations about content of the XML document (see figure 5.2) which was transformed by the XSLT stylesheet while creating the sample web page.

Dinaco was also not given annotations about the interface because the sample web page was designed to be simple. The VoiceXML interface which Dinaco created is shown in figure 5.4.

```
<?xml version="1.0" encoding="UTF-8"?>
<vxml
  xmlns="http://www.w3.org/2001/vxml"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  version="2.0"
  xsi:schemaLocation="http://www.w3.org/2001/vxml http://www.
    w3.org/TR/voicexml20/vxml.xsd">
  <property name="
    bargein" value="false"/>
  <form id="form_0_0">
    <block>
      <prompt>
        <s>The last year's competition</s>
        <s>Winning couple:</s><s>Pieter van der Merwe and Hope
          Jacobs</s>
        <s>Location:</s><s>South Africa – Welkom</s><s>Details :
          </s>
        <s>Price was R200 000, held on 2006/01/12</s>
      </prompt>
    </block>
  </form>
</vxml>
```

Figure 5.4: The VoiceXML interface for the control experiment

5.1.3.3 The experimental interface

Dinaco was made to transcode the sample web page into a VoiceXML interface. In this case, it was also given annotations about content (AAC) of figure 5.2. These AAC are shown in figure 5.5. The AAC meet the requirements specified in the hypothesis (section 5.1.2.1) because they have the following two characteristics. Firstly, they are written externally from the sample HTML interface. Secondly, they are written externally from the XML document which was transformed by the XSLT stylesheet while creating the sample web page. Dinaco used the AAC to deduce the following semantic descriptions about the contents of the XML document:

- The strings “Pieter van der Merwe” and “Welkom” are in Afrikaans.

- The string “2006/01/12” is a date value, with the format “yyyy/MM/dd”.
- The string “R200 000” is a South African currency, and its units are Rands.

The VoiceXML interface which Dinaco created using these annotations is shown in figure 5.6.

University of Cape Town

```

<dinaco:Type rdf:ID="date">
  <dinaco:hasName rdf:datatype="&xsd:string">date</
    dinaco:hasName>
  <dinaco:hasFormat rdf:datatype="&xsd:string">yyyy/MM/dd</
    dinaco:hasFormat>
</dinaco:Type>
<dinaco:Type rdf:ID="currency">
  <dinaco:hasName rdf:datatype="&xsd:string">zacurrency</
    dinaco:hasName>
</dinaco:Type>
<dinaco:DinacoResource rdf:ID="aMerweFirstName">
  <dinaco:hasXPath rdf:datatype="&xsd:string">profiles#
    merweFname</dinaco:hasXPath>
  <dinaco:hasLanguage rdf:datatype="&xsd:string">af</
    dinaco:hasLanguage>
</dinaco:DinacoResource>
<dinaco:DinacoResource rdf:ID="aMerweSurname">
<dinaco:hasXPath rdf:datatype="&xsd:string">profiles#
  merweSurname</dinaco:hasXPath>
<dinaco:hasLanguage rdf:datatype="&xsd:string">af</
  dinaco:hasLanguage>
</dinaco:DinacoResource>
<dinaco:DinacoResource rdf:ID="eventLocation">
  <dinaco:hasXPath rdf:datatype="&xsd:string">profiles#
    dpRivier</dinaco:hasXPath>
  <dinaco:hasLanguage rdf:datatype="&xsd:string">af</
    dinaco:hasLanguage>
</dinaco:DinacoResource>
<dinaco:DinacoResource rdf:ID="eventDate">
  <dinaco:hasXPath rdf:datatype="&xsd:string">record/
    competition/date</dinaco:hasXPath>
  <dinaco:hasType rdf:resource="#date"/>
</dinaco:DinacoResource>
<dinaco:DinacoResource rdf:ID="eventPrice">
  <dinaco:hasXPath rdf:datatype="&xsd:string">record/
    competition/price</dinaco:hasXPath>
  <
    dinaco:hasType rdf:resource="#currency"/>
</dinaco:DinacoResource>

```

Figure 5.5: Annotations about content which were used by Dinaco while creating the experimental interface

```

<?xml version="1.0" encoding="UTF-8"?>
<vxml
  xmlns="http://www.w3.org/2001/vxml"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  version="2.0"
  xsi:schemaLocation="http://www.w3.org/2001/vxml http://www.
    w3.org/TR/voicexml20/vxml.xsd">  <property name="
    bargein" value="false"/>
<form id="form_0_0">
  <block>
    <prompt>
      <s>The last year's competition</s>
      <s>Winning couple:</s>
      <s xml:lang="af">Pieter</s>
      <s xml:lang="af">van der Merwe</s>
      <s> and Hope Jacobs</s>
      <s>Location:</s>
      <s>South Africa - </s>
      <s xml:lang="af">Welkom</s>
      <s>Details: </s>
      <s>Price was </s>
      <s>Two hundred thousand , Rands</s>
      <s>, held on </s>
      <s><say-as interpret-as="date">12 January 2006</say-as
        ></s>
    </prompt>
  </block>
</form>
</vxml>

```

Figure 5.6: The VoiceXML interface for the experiment

5.1.3.4 Evaluation of the control and the experimental interfaces

Two experiments were conducted to evaluate the control interface and the experimental interface for usability. A voice browser which was used during experimentation was installed in the laptop as described in section 5.1.4. This voice browser was made to render either the control or the experimental interface based on which interface was being evaluated at the time.

Forty-four subjects were recruited to participate in the experiments. All subjects were students from the University of Pretoria. The requirement for participation was that subjects had to understand both English and Afrikaans. Subjects were

randomly assigned to evaluate either the control interface or the experimental interface, based on the order of their arrival. Twenty-two subjects were assigned to evaluate each of the two interfaces.

Each subject was given a chance to listen to the system to which he/she was assigned. The subject was given an evaluation sheet after listening. The only difference between the evaluation sheets were the letters which indicated the interface which was being evaluated. Evaluation sheets for the control interface were assigned a letter “B”. Evaluation sheets for the experimental interface were assigned a letter “A”. The sample evaluation sheet for system A (the experimental interface) is shown in appendix A.

Evaluation sheets were designed with an aim of achieving the following objectives: Firstly, to test if subjects were able to recognize the Afrikaans words which were read to them. Secondly, to test if subjects could still recall the date and the currency values which were read to them. Thirdly, to assess the impression that subjects had after an interaction with the system.

5.1.4 Setting up the environment

This section describes the software and the hardware which were used while conducting the experiments.

The real world scenario which was assumed while designing the experiments was as follows:

- Developers deploy Dinaco on the server side and use it to create speech interfaces.
- Speech users use a voice browser which is installed on their local workstations to access the created speech interface.

In order to model the above usage scenario, a laptop computer was used as both the machine where Dinaco was deployed and as the user’s local workstation. A voice browser and a text-to-speech (TTS) tool were installed on the laptop computer in order to make it suitable for use as a user’s local workstation. The details are covered in sections 5.1.4.1 and 5.1.4.2.

5.1.4.1 The voice browser

A voice browser which is capable of rendering VoiceXML interfaces was required. The voice browser was created by extending version 3.4 of the open source Vacalocity OpenVXI VoiceXML interpreter [71]. Vacalocity OpenVXI is an extension to the VoiceXML Interpreter which is described in [72]. Vacalocity OpenVXI can interpret up to version 2.1 of the VoiceXML specification. Vacalocity OpenVXI is written using the C programming language.

Vacalocity OpenVXI was extended such that it becomes able to do the following: Firstly, to read the DTMF inputs from the command line; this was done to make it to be able to render VoiceXML documents which receive DTMF inputs. Secondly, to use the TTS tool which was also installed in the laptop for the purposes of reading text to the user. This was achieved by making modifications in the implementations of the rec and the prompt Application Programming Interfaces of the interpreter. Please see the Vocality OpenVXI Architecture document which is downloaded together with the interpreter and also [71].

5.1.4.2 The TTS tool

The TTS tool which was used during the experiments had to meet the following requirements: Firstly, the TTS tool had to be able to interpret SSML inputs which the voice browser was going to pass to it. Secondly, the TTS tool had to be able to read at least two of the South African spoken languages. Thirdly, it had to be easy for the author of this thesis to install the TTS tool in the Linux operating system. The decision was made to use eSpeak because it met all the mentioned requirements. Festival could also have been used, but additional installations and setup are required in order to make it to be able to interpret SSML inputs.

5.2 Summary of chapter 5

Chapter 5 has discussed the design of the experiments which were used to test if Dinaco does achieve its first objective: to produce usable interfaces. Chapter 5 mentioned that the Vacalocity OpenVXI VoiceXML Interpreter was extended such

that it becomes a VoiceXML browser and that eSpeak is the TTS tool which was used during the experiments. Forty-four subjects were recruited to participate in the experiments. Twenty-two of the 44 subjects were assigned to evaluate the experimental interface and the other 22 to evaluate the control interface. Chapter 6 presents, analyses, interprets and compares the results for the experimental interface and for the control interface. Chapter 6 also makes concluding remarks about whether the results from experiments prove or disprove the null hypothesis.

University of Cape Town

Chapter 6

Analysis and interpretation of the results

This chapter is organized as follows:

- Section 6.1 analyzes and interprets the results from the experiments which were designed as described in chapter 5. This section also compares the results for the experimental interface with the results for the control interface.
- Section 6.2 uses the results from experiments to draw conclusions on whether the null hypothesis which was stated in chapter 5 was proved or disapproved.

6.1 Analysis and interpretation

This section analyzes, interprets and compares the results from experiments for the control interface and for the experimental interface. The results for reading multilingual text are discussed in section 6.1.1. The results for reading the date and the currency value are discussed in section 6.1.2. The results for the overall assessments by subjects are discussed in section 6.1.3. Section 6.2 provides the overall concluding remarks about the results from experiments.

6.1.1 Multilingual text

This section analyzes the results of section 1 of the evaluation sheet (reading of the Afrikaans words). The aim of including section 1 in the evaluation sheets was to find out if subjects could recognize the Afrikaans proper nouns which were read to them, during their interaction with the interface, which they were assigned to assess. These proper nouns were read at two different times. The first time was when reading “Pieter van der Merwe”, which is the full name of the first partner in the winning couple. The second time was when reading “Welkom”, which is the suburb where the last year’s competition took place.

The results for multilingual text are presented in section 6.1.1.1 and analyzed and interpreted in section 6.1.1.2. Section 6.1.1.3 uses the results from the experiments to make concluding remarks and also mentions the limitations of the experiment in the context of multilingual text.

6.1.1.1 The results

During the analysis of the results from the experiment, an average score was computed using the two values which were entered by subjects in table 2 of section 1 of the evaluation sheet. The average score was computed by obtaining the sum of the scores for “Pieter van der Merwe” and for “Welkom” and dividing this sum by two. These averages were used for two purposes: firstly, to compute the approximate percentage of the 44 subjects per average score, as shown in table 6.1, secondly, to plot the two graphs which are shown in figure 6.1.

Average score	1	2	3	4	5
The control interface	13.6%	27.2%	9.1%	0%	0%
The experimental interface	0%	15.9%	20.5%	11.4%	2.3%

Table 6.1: Approximate percentages of subjects per average score for Afrikaans words

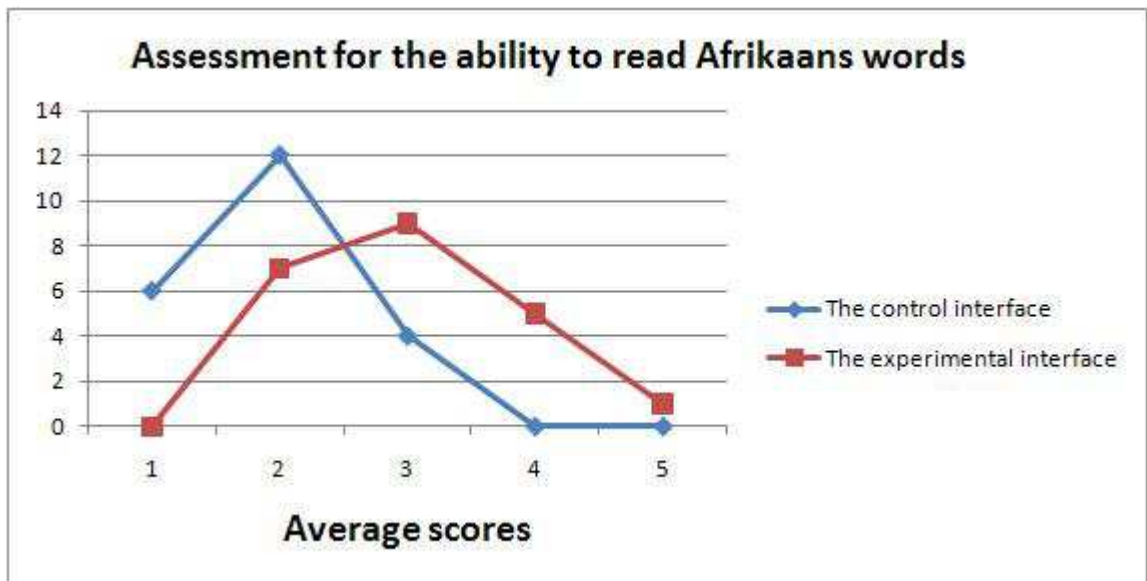


Figure 6.1: Comparing scores for the ability to read Afrikaans words

6.1.1.2 Analysis and interpretation for multilingual text

The results can be interpreted by looking at the graphs in figure 6.1 when the average score is less than or equal to 2 and when it is greater than 2. The graph for the control interface is above the graph for the experimental interface when the average score is less than or equal to 2. It can also be seen in table 6.1 that 1 was computed as the average score which was assigned to the control interface by 13.6% of the subjects and by 0% of the subjects to the experimental interface. It can also be seen in table 6.1 that 2 was computed as the average score which was assigned to the control interface by 27.2% of the subjects and by 15.9% of the subjects to the experimental interface. This means that there are more subjects who interacted with the control interface and indicated that they did not hear or they slightly heard the Afrikaans proper nouns being read to them compared to those who interacted with the experimental interface.

The graph for the experimental interface is above the graph for the control interface when the average score is above 2. This means that there are more subjects who gave the experimental interface an average score which is above 2, compared to those who gave it to the control interface. See table 6.1 for the actual figures. This means that there are more subjects who indicated that they heard the Afrikaans

proper nouns being read, while they were interacting with the experimental interface compared to those who interacted with the control interface.

6.1.1.3 Concluding remarks and limitations

It can be concluded that the results from experimentation indicate that: subjects who interacted with the experimental interface were in a better position to hear the Afrikaans proper nouns being read to them, compared to those who interacted with the control interface. Subjects who interacted with the experimental interface were in a better position to understand that Pieter van der Merwe was one of the partners who won the competition, and also that the competition was held in Welkom, compared to those who interacted with the control interface.

The presence of the language semantics in the experimental interface and not in the control interface is the likely cause which made the experimental interface to outperform the control interface when it comes to reading Afrikaans words. The VoiceXML markup for the control interface did not have semantic descriptions of content which could help eSpeak (the TTS tool) to identify Afrikaans words. As a result, eSpeak used the default English voice to read Afrikaans proper nouns. The VoiceXML markup for the experimental interface had semantic descriptions of content which helped eSpeak to identify the Afrikaans words and hence read them using its Afrikaans voice.

The Afrikaans and the English voices which were downloaded together with eSpeak were both male voices and sounded alike, as if they were recorded by the same person. The experiments did not test for what was going to happen if there was a noticeable difference between the two voices. It can then be said that the experiments tested for what happens when the possible impact of the differences between voices is ignored.

6.1.2 The date value and the currency value

This section analyzes and interprets the results of section 2 of the evaluation sheet (reading of the date and the currency values). The currency value was R200 000, which is the prize that was won by the couple. The date value was 2006/01/12,

August 19, 2010

which is the date of the competition.

The aim of including section 2 in the evaluation sheets was to achieve the following two objectives: The first objective was to find out whether the subjects could recall the date value and the currency value which were read to them. Section 6.1.2.1 of this chapter analyzes and interprets the results from sections of the evaluation sheets which were included with an aim of achieving the first objective. These sections are 2.1 to 2.2.

The second objective was to find out how the subjects assessed the overall ability of the system to read the date and the currency values such that they get to understand it. Section 6.1.2.2 of this chapter analyzes and interprets the results from section 2.3 of the evaluation sheets which was included with an aim of achieving the second objective. Section 6.1.2.3 makes concluding remarks based on the results from sections 6.1.2.1 and 6.1.2.2.

6.1.2.1 Recalling the currency and the date values

This section analyzes and interprets the results from sections 2.1 to 2.2 of the evaluation sheets. In section 2.1 of the evaluation sheets, subjects were required to fill in the year, the month and the day of the competition. In section 2.2 of the evaluation sheets, subjects were required to fill in the currency value which was read to them. During the analysis stage of this research, the score was assigned to each evaluation sheet based on the number of correct responses which it had. The maximum score for each evaluation was four (three values from section 2.1 and one value from section 2.2).

If all subjects filled in the correct responses, then the sum of the scores from 22 subjects was going to be 22×4 , which is 88. The sum of the scores for the subjects who evaluated the experimental interface was found to be 12 over 88. This means that approximately 13.6% of the responses were correct for the experimental interface. The sum of the scores for the subjects who evaluated the control interface was found to be 7 over 88. This means that approximately 8% of the responses were correct for the control interface.

It can be concluded that subjects who interacted with the experimental interface

were more likely to recall the actual values of the date and the currency compared to those who interacted with the control interface.

6.1.2.2 Overall assessment for non-standard words

This section analyzes and interprets the results from the overall assessments of the interfaces for the ability to read the date and the currency values (section 2.3 of the evaluation sheet). During the analysis of the results from the experiments, an average score was computed using the two values which were entered by subjects in section 2.3 of the evaluation sheet. The average score was computed by obtaining the sum of the scores for the date and for the currency value and dividing this sum by two. These averages were used for two purposes. The first purpose was to assign the approximate percentage of the 44 subjects for each average score, as shown in table 6.2. The second purpose was to plot the two graphs which are shown in figure 6.2.

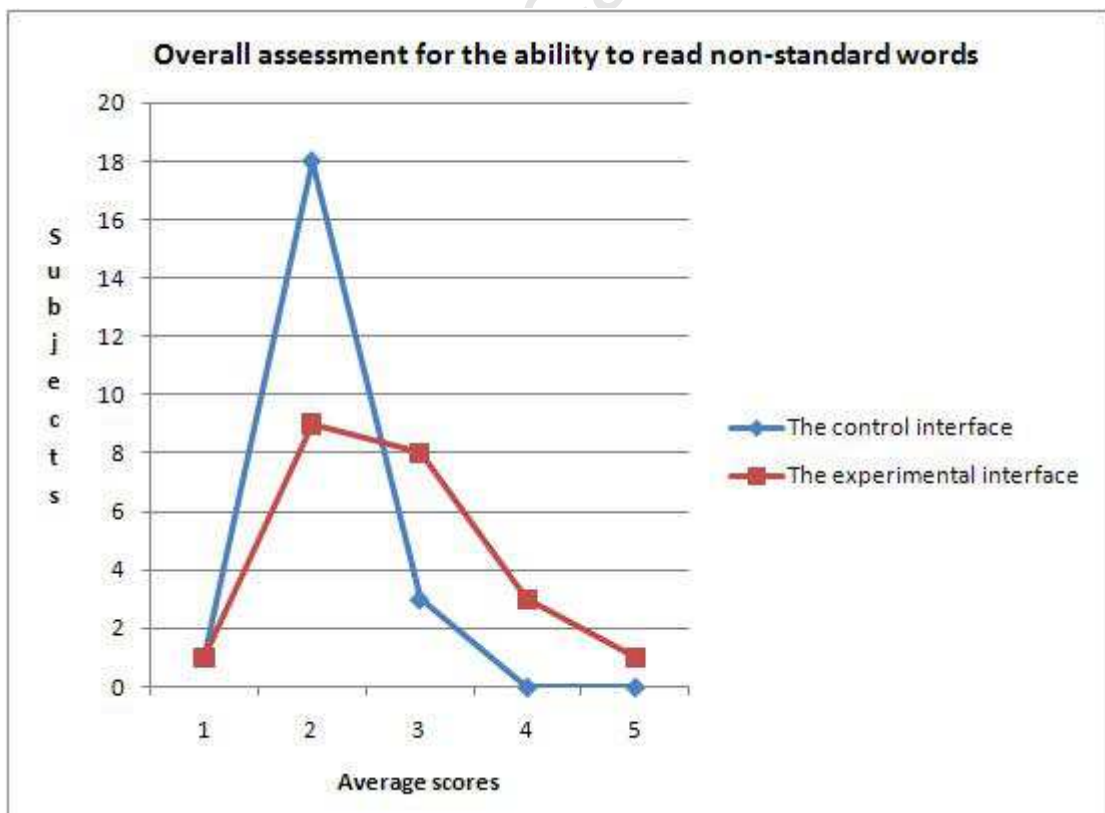


Figure 6.2: Comparing scores for the overall assessment of non-standard words

Average score	1	2	3	4	5
The control interface	2.3%	40.9%	6.8%	0%	0%
The experimental interface	2.3%	20.5%	18.1%	6.8%	2.3%

Table 6.2: Approximate percentages of subjects per average score for non-standard words

The meaning of the average scores can be seen in section 2.3 of the evaluation sheet. The average score of 1 means the interface was rated as horrible. The average score of 2 means the interface was rated as poor. The average score of 3 means the interface was rated as acceptable. The average score of 4 means the interface was rated as good and 5 means the interface was excellent.

The results can be interpreted by looking at the graphs in figure 6.2 when the average score is less than or equal to 2 and when it is greater than 2. The graphs are equal when the average score is equal to 1. This means that there was an equal number of subjects (2.3%) who believed that the interfaces were horrible.

The graph for the control interface is above the graph for the experimental interface when the average score is equal to 2. It can also be seen in table 6.2 that 40.9% of the subjects rated the control interface as poor and that 20.5% of the subjects had the same opinion about the experimental interface. This means that there were more subjects who rated the control interface as poor compared to those who rated the experimental interface the same way.

The graph for the experimental interface is above the graph for the control interface when the average score is greater than 2. It can also be seen in table 6.2 that 18.1% of the subjects rated the experimental interface as acceptable and that 6.8% of the subjects rated the control interface the same way. Table 6.2 also shows that 6.8% of the subjects rated the experimental interface as good and that no subjects rated the control interface the same way. 2.3% of the subjects rated the experimental interface as excellent and no subjects had the same opinion about the control interface. This means that there are more subjects who believed that the experimental interface was good or acceptable or excellent compared to those who had the same opinion about the control interface.

6.1.2.3 Concluding remarks for non-standard words

The results from section 6.1.2.1 indicate that approximately 13.6% of the responses from subjects who interacted with the experimental interface were correct and that approximately 8% were correct for the subjects who interacted with the control interface. The percentage of correct responses is noticeably low for both interfaces. However, there are more subjects who interacted with the experimental interface who filled in correct answers compared to those who interacted with the control interface. The experimental interface also outperformed the control interface when it comes to the overall assessments for the ability to read the date and the currency values.

The main difference between the control and the experimental interfaces is that the currency value and the date value were pre-normalized in the experimental interface and not in the control interface. Dinaco used annotations about content to attain the semantic descriptions of the date value and the currency value. Dinaco then used these semantic descriptions to pre-normalize the date value to “12 January 2006” and the currency value to “Two hundred thousand , Rands”.

It can be concluded that the results from experiments indicate that subjects who interacted with the experimental interface were in a better position to hear and understand the currency value and the date value while they were read to them, compared to those who interacted with the control interface. Subjects who interacted with the experimental interface were in a better position to understand that the prize was R200 000 and that the competition was held on 2006/01/12, compared to those who interacted with the control interface.

6.1.3 Overall assessment of the interfaces

This section analyzes and interprets the overall assessments of the interfaces as scored by the subjects after they interacted with the interface to which they were assigned. The results which are used in this section were obtained from the selections which were made by subjects in section 3 of the evaluation of sheet. The results from the experiments are first presented in section 6.1.3.1 and then analyzed and

interpreted in section 6.1.3.2.

6.1.3.1 The results

Table 6.3 shows the approximate percentages of the 44 subjects and their overall assessment scores for each interface. The scores in the table have the same meaning as average scores in section 6.1.2.2. The percentages were computed by adding together the number of people who assigned a score and then dividing that number by 44 and then multiplying it by 100. Figure 6.3 shows the graphs which compare the overall assessment scores of the interfaces.

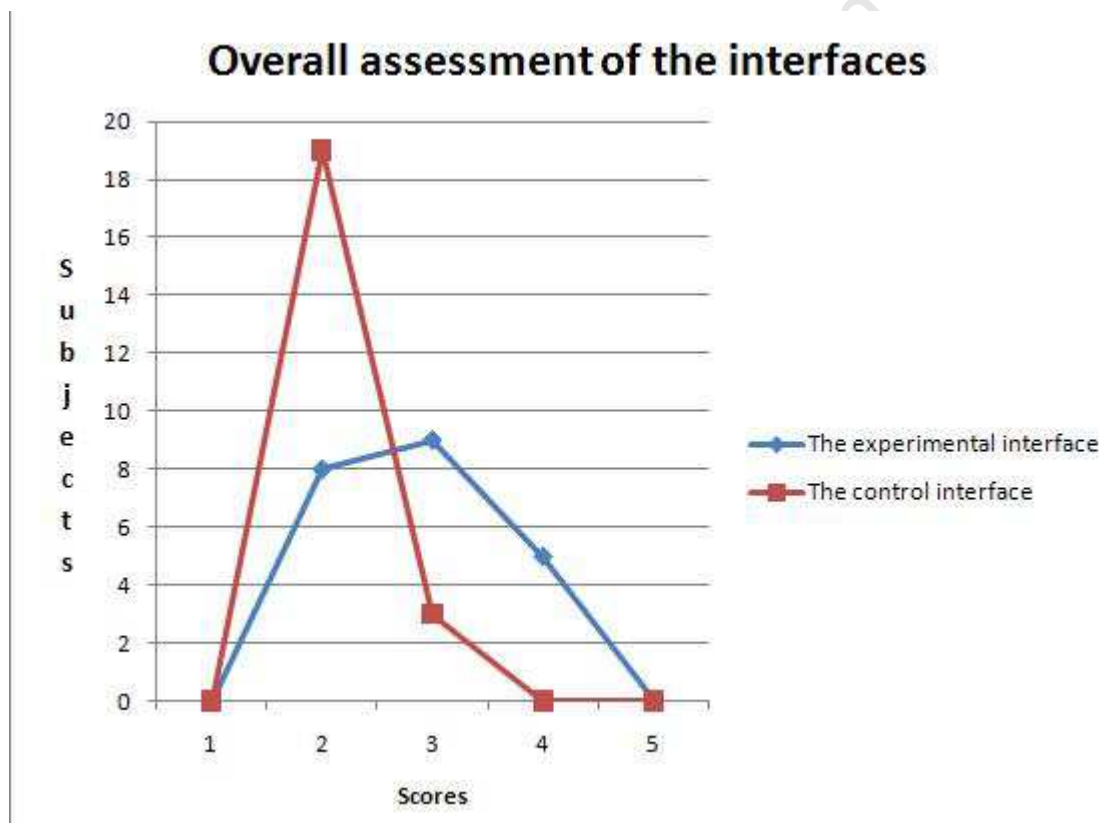


Figure 6.3: The results of the overall assessment of the interfaces by subjects

Score	1	2	3	4	5
The control interface	0%	43.2%	6.8%	0%	0%
The experimental interface	0%	18.1%	20.5%	11.4%	0%

Table 6.3: Approximate percentages of subjects per score for the overall assessment of the interfaces

6.1.3.2 Analysis and interpretation

The results can be interpreted by looking at the graphs in figure 6.3 when the average score is less than or equal to 2 and when it is greater than 2. The graph for the control interface is above the graph for the experimental interface when the score is greater than 1 but less or equal to 2. It can also be seen in table 6.3 and in the graphs that both the control interface and the experimental interface were not rated as horrible. Table 6.3 also shows that 43.2% of the subjects rated the control interface as poor and that 18.1% of the subjects rated the experimental interface as poor. This means that there are more subjects who believed that the control interface was poor compared to those who had the same opinion about the experimental interface.

The graph for the experimental interface is above the graph for the control interface when the score is greater than 2. It can also be seen in table 6.3 that 20.5% of the subjects rated the experimental interface as acceptable and that 6.8% of the subjects had the same opinion about the control interface. Table 6.3 also shows that 11.4% of the subjects rated the experimental interface as good and that no subjects had the same opinion about the control interface. This means that there are more subjects who rated the experimental interface as good or acceptable compared to those who rated the control interface the same way.

It can be concluded that the results from this section indicate that the experimental interface was rated by subjects as being more usable than the control interface.

6.2 Concluding remarks

Dinaco's ability to take separated annotations made it possible to write semantic descriptions of the contents of the sample web page which cannot be written if annotations are only written in terms of the HTML interface which is being transcoded. Dinaco used these semantic descriptions to produce an interface which had more language semantics of content than the interface which was going to be produced by previous transcoding systems. The interface which was produced by Dinaco also had pre-normalized values of the date and of the currency.

Appending language semantics and pre-normalizing the non-standard words made Dinaco to produce the experimental interface which received a better rating than the control interface. The experimental interface was rated as having a better ability to read web content which featured a mixture of English and Afrikaans words and which had a date value and a currency compared to the control interface.

It can then be concluded that the results from the experiment disproved the null hypothesis which was stated in section 5.1.2.2. This is because Dinaco's transcoding techniques were found to have produced a more usable interface compared to previous transcoding techniques. It can also be concluded that there are instances where Dinaco has the ability to produce more usable speech interfaces than previous transcoding systems.

6.3 Summary of chapter 6

Chapter 5 discussed the experiments which were designed to find out if Dinaco does have the ability to produce more usable interfaces compared to previous transcoding systems. This chapter analyzed the results from the experiments and used them to conclude that there is an HTML interface which Dinaco has the ability to convert into a more usable speech interface compared to previous transcoding systems. Chapter 7 uses the results from this chapter and the discussions in previous chapters to draw the overall conclusions for this thesis.

Chapter 7

Conclusions, future work and recommendations

In this thesis, the design of a transcoding system which transcodes HTML interfaces such that they become speech interfaces has been proposed and its implementation details discussed. This transcoding system was given a name: Dinaco. The characteristics of the HTML interfaces which Dinaco is targeted at transcoding are described in section 7.1. Section 7.2 summarises how Dinaco was designed such that it becomes able to transcode its target HTML interfaces. Section 7.3 gives the conclusions which are drawn by this thesis. Section 7.4 gives future work and recommendations.

7.1 Characteristics of HTML interfaces

Dinaco is targeted at transcoding HTML interfaces which are created using a method which this thesis calls the XML-Based User Interface Development Methodology (XinterfaceWebDevMeth) and which feature multilingual text and non-standard words. In XinterfaceWebDevMeth, the process of developing HTML interfaces consists of four tasks. The first task is to create the XML schema of the content that is to be published on the web. The second task is to create XML documents which have the content that is to be published on the web. These XML documents are created such that they conform to the XML schema which was created while doing

the first task. The third task is to create XSLT stylesheets which define how to transform XML documents to HTML. The fourth task is to use the XSLT processor to transform XML documents to HTML.

7.2 Summary of the solution

This thesis proposed the concept of separating annotations which are used by the transcoding system, into annotations about content (AAC) and annotations about the interface (AAI). AAC describe the contents of XML documents using the elements of the XML documents, without using the HTML elements of the HTML interface which is to be transcoded. AAI describe the HTML elements. The thesis described how Dinaco uses these annotations when it transcodes HTML interfaces to VoiceXML.

Dinaco uses AAC to create VoiceXML interfaces such that they have language semantics. These language semantics guide TTS tools on which language they should use to read text. Dinaco also has software components called Normalizers. Dinaco uses these Normalizers while creating the VoiceXML interface, to convert non-standard words into standard words which are readable by TTS tools. The Normalizers which were implemented can normalize dates, South African currency values and references to Bible scriptures. Dinaco is designed such that it becomes easy to add new or remove already existing Normalizers.

7.3 Conclusions

This thesis concludes that designing a transcoder such that it uses separated annotations offers the advantages which are listed below:

1. It becomes easier to divide a complex software system, when creating annotated HTML interfaces into distinct components which can be created and managed by different members of the software development team.
2. It makes it easier to manage changes in the interfaces, in the XML documents and in the annotations. This is because changes in the interfaces only affect

annotations about the interfaces and changes in the XML documents only affect annotations about content.

3. It makes it possible to reuse both the annotations about the interface and about XML documents. It also makes it easier for people and for systems who interchange XML documents to also interchange annotations about the contents of those documents.
4. It makes it possible to write external semantic descriptions of content which cannot be easily written externally if annotations are only written in terms of the HTML elements of the transcoded interface.

Because of advantages 1 to 3 above, this thesis concludes that Dinaco uses annotations which are easy to create, maintain, reuse and share. The results from the experiments which were conducted during this research, demonstrated by example that advantage number 4 of using separated annotations can give Dinaco the capability to transcode HTML interfaces which feature multilingual text and non-standard words into VoiceXML interfaces which are more usable than those which can be produced using external non-separated annotations.

7.4 Future work and recommendations

7.4.1 Future work

For future work, the following can be investigated:

- A graphical user interface which can make it easier to create both the annotations about content and the annotations about the interface which Dinaco uses.
- The differences between version 1 and version 2 of the XSLT specification and whether the same techniques which were used in this thesis to transcode version 1 can also be used to transcode version 2.

- The design and implementation of a XSLT processor which can be configured to create annotations when it transforms source XML documents from one form to the other. These annotations can be created such that they relate the contents of the document(s) which result from applying XSLT transformations to the contents of the source XML documents. The benefits of such a XSLT processor are twofold. Firstly, it will make it possible for its users to create annotated documents without downloading an annotation tool. Secondly, it will make it easy to create separated annotations.

7.4.2 Recommendations

Initiatives should be taken to create a common annotation language which can be used by transcoding systems. The need for such initiatives can be seen in the similarities between annotations which are used in [20], [21], [26], [27] and [73].

Bibliography

- [1] I. Jacobs, D. Raggett, and A. L. Hors, “HTML 4.01 specification,” W3C recommendation, W3C, December 1999. <http://www.w3.org/TR/1999/REC-html401-19991224>.
- [2] “The festival speech synthesis system.” Online [Last accessed 2009/02/10]. <http://www.cstr.ed.ac.uk/projects/festival/>.
- [3] “espeak text to speech.” Online [Last accessed 2009/02/10]. <http://espeak.sourceforge.net/>.
- [4] D. C. Burnett, A. Hunt, and M. R. Walker, “Speech synthesis markup language (SSML) version 1.0,” W3C recommendation, W3C, September 2004. <http://www.w3.org/TR/2004/REC-speech-synthesis-20040907/>.
- [5] J. Ferrans, B. Porter, S. Tryphonas, B. Lucas, P. Danielsen, D. C. Burnett, A. Hunt, S. McGlashan, K. Rehor, and J. Carter, “Voice extensible markup language (VoiceXML) version 2.0,” W3C recommendation, W3C, March 2004. <http://www.w3.org/TR/2004/REC-voicexml20-20040316/>.
- [6] B. Myers, S. E. Hudson, and R. Pausch, “Past, present, and future of user interface software tools,” *ACM Trans. Comput.-Hum. Interact.*, vol. 7, pp. 3–28, March 2000.
- [7] G. Mori, F. Paterno, and C. Santoro, “Design and development of multidevice user interfaces through multiple logical descriptions,” *IEEE Trans. Softw. Eng.*, vol. 30, pp. 507–520, August 2004.

- [8] C. Vassilakis, G. Lepouras, and C. Halatsis, "A knowledge-based approach for developing multi-channel e-government services," *Electronic Commerce Research and Applications*, vol. 6, p. 113–124, 2007.
- [9] G. Banavar, L. Bergman, R. Cardone, V. Chevalier, Y. Gaeremynck, F. Giraud, C. Halverson, S.-i. Hirose, M. Hori, F. Kitayama, G. Kondoh, A. Kundu, K. Ono, A. Schade, D. Soroker, and K. Winz, "An authoring technology for multidevice web applications," *IEEE Pervasive Computing*, vol. 03, no. 3, pp. 83–93, 2004.
- [10] W. Gu and A. S. Helal, "An xml based solution to delivering adaptive web content for mobile clients," in *International Symposium on performance Evaluation of Computer and Telecommunication Systems (SPECTS)*, (San Jose, CA), 2004.
- [11] "Xhtml+voice profile 1.0," W3C note, W3C, December 2001. <http://www.w3.org/TR/xhtml+voice/>.
- [12] Y. Chan and H. Suwanda, "Designing multinational online stores: challenges, implementation techniques and experience," in *CASCON '00: Proceedings of the 2000 conference of the Centre for Advanced Studies on Collaborative research*, p. 1, IBM Press, 2000.
- [13] D. Cyr, C. Bonanni, and J. ilsever, "Design and e-loyalty across cultures in electronic commerce," in *ICEC '04: Proceedings of the 6th international conference on Electronic commerce*, (New York, NY, USA), pp. 351–360, ACM, 2004.
- [14] S. He, "Interplay of language and culture in global e-commerce: a comparison of five companies' multilingual websites," in *SIGDOC '01: Proceedings of the 19th annual international conference on Computer documentation*, (New York, NY, USA), pp. 83–88, ACM, 2001.
- [15] T. Parr, "Web application internationalization and localization in action," in *ICWE '06: Proceedings of the 6th international conference on Web engineering*, (New York, NY, USA), pp. 64–70, ACM, 2006.

- [16] “Internet world users by language.” Online [Last accessed 2009/02/10]. <http://www.internetworldstats.com/stats7.htm>.
- [17] S. S, W. Martin, and P. Stephen, “Towards culture-centred design,” *Interact. Comput.*, vol. 18, no. 4, pp. 820–852, 2006.
- [18] C. Phanouriou, *UIML: A Device-Independent User Interface Markup Language*. Ph.d dissertation, Virginia Polytechnic Institute and State University, 2000.
- [19] M. Abrams, C. Phanouriou, B. Alan, S. M. Williams, and J. E. Shuster, “Uiml: an appliance-independent xml user interface language,” *Computer Networks*, vol. 31, pp. 1695–1708, 1999.
- [20] H. Masahiro, K. Goh, O. Kouichi, H. S, and S. Sandeep, “Annotation-based web content transcoding,” *Computer Networks*, vol. 33, p. 197–211, 2000.
- [21] H. Kim and K. Lee, “Device-independent web browsing based on cc/pp and annotation,” *Interact. Comput.*, vol. 18, no. 2, pp. 283–303, 2006.
- [22] B. Christos, K. Giorgos, and M. Ioannis, “A web content manipulation technique based on page fragmentation,” *J. Netw. Comput. Appl.*, vol. 30, pp. 563–585, April 2007.
- [23] S. G. J. Gabriel, V. J. S. Sosa, A. R. Montes, and y J. Carlos R. Olivares, “Multi-format web content transcoding for mobile devices,” *Mexican International Conference on Computer Science*, vol. 0, pp. 109–115, 2006.
- [24] K. Nagao, Y. Shirai, and K. Squire, “Semantic annotation and transcoding: Making web content more accessible,” *IEEE MultiMedia*, vol. 8, pp. 69–81, 2001.
- [25] M. D. Dikaiakos, “Intermediary infrastructures for the world wide web,” *Computer Networks*, vol. 45, no. 4, pp. 421–447, 2004.
- [26] S. Harper and S. Bechhofer, “Sadie: Structural semantics for accessibility and device independence,” *ACM Trans. Comput.-Hum. Interact.*, vol. 14, August 2007.

- [27] M. Lamb and B. Horowitz, "Guidelines for a voicexml solution using web-sphere transcoding publisher." Online [Last accessed 2010/02/10], 2001. <ftp://ftp.software.ibm.com/software/wtp/info/VxmlTranscodingGuide.pdf>.
- [28] Z. Shao, R. Capra, and M. A. Pérez-Quñones, "Annotations for html to voicexml transcoding: Producing voice webpages with usability in mind.," technical report cs.hc/0211037, Computing Research Repository (CoRR), 2002.
- [29] I. C. Hsu and S.-J. Kao, "An owl-based extensible transcoding system for mobile multi-devices," *J. Inf. Sci.*, vol. 31, pp. 178–195, June 2005.
- [30] M. A. Pérez-Quñones, N. Dannenberg, and R. Capra, "Voice navigation of structured web spaces," Technical Report TR-02-22, Department of Computer Science, Virginia Tech, Blacksburg, VA, 2002.
- [31] I. V. Ramakrishnan, J. Mahmud, Y. Borodin, M. A. Islam, and F. Ahmed, "Bridging the web accessibility divide," *Electron. Notes Theor. Comput. Sci.*, vol. 235, pp. 107–124, 2009.
- [32] H. Takagi and C. Asakawa, "Web content transcoding for voice output," in *Technology and Persons with Disabilities Conference*, 2002.
- [33] J. Clark, "XSL transformations (XSLT) version 1.0," W3C recommendation, W3C, November 1999. <http://www.w3.org/TR/1999/REC-xslt-19991116>.
- [34] B. Duggan and M. Deegan, "Considerations in the usage of text to speech (tts) in the creation of natural sounding voice enabled web systems," in *Proceedings of the 1st international symposium on Information and communication technologies*, pp. 433–438, Trinity College Dublin, 2003.
- [35] H. Romsdorfer and B. Pfister, "Text analysis and language identification for polyglot text-to-speech synthesis," *Speech Communication*, vol. 49, no. 9, pp. 697–724, 2007.
- [36] R. G. Gordon, *Ethnologue: Languages of the World*. SIL International, 2005.

- [37] S. Richard, W. B. Alan, C. Stanley, K. Shankar, O. Mari, and R. Christopher, "Normalization of non-standard words," *Computer Speech and Language*, vol. 15, no. 3, pp. 287–333, 2001.
- [38] G. K. Gerasimos Xydias and G. Kourouperoglou, "Text normalization for the pronunciation of non-standard words in an inflected language," in *3rd Hellenic Conference on Artificial Intelligence*, (Samos, Greece), 2004.
- [39] S. Pakhomov, "Semi-supervised maximum entropy based approach to acronym and abbreviation normalization in medical texts," in *Proceedings of the 40th Annual Meeting of the Association for Computational Linguistics (ACL)*.
- [40] E. R. Harold and W. S. Means, *XML in a nutshell*. O'Reilly Media, third edition ed., 2004.
- [41] International Organization for Standardization, "Standard Generalized Markup Language", ISO 8879, October 1986.
- [42] T. Bray, J. Paoli, E. Maler, F. Yergeau, and C. M. Sperberg-McQueen, "Extensible markup language (XML) 1.0 (fifth edition)," W3C recommendation, W3C, November 2008. <http://www.w3.org/TR/2008/REC-xml-20081126/>.
- [43] "Xpointer framework," W3C recommendation, W3C, March 2003. <http://www.w3.org/TR/2003/REC-xptr-framework-20030325/>.
- [44] J. Clark and S. DeRose, "XML path language (XPath) version 1.0," W3C recommendation, W3C, November 1999. <http://www.w3.org/TR/1999/REC-xpath-19991116>.
- [45] S. Pemberton, "XHTML™ 1.0 the extensible hypertext markup language (second edition)," W3C recommendation, W3C, August 2002. <http://www.w3.org/TR/2002/REC-xhtml1-20020801>.
- [46] C. Musciano and B. Kennedy, *HTML and XHTML: The Definitive Guide*. O'Reilly Media, fifth edition ed., 2004.

- [47] “Xhtml mobile profile,” tech. rep., Wireless Application Protocol Forum, January 2001.
- [48] “Xml schema part 1: Structures second edition,” W3C recommendation, 2004. <http://www.w3.org/TR/xmlschema-1/>.
- [49] “Xml schema part 2: Datatypes second edition,” W3C recommendation, W3C, October 2004. <http://www.w3.org/TR/xhtml+voice/>.
- [50] “Saxon: The xslt and xquery processor.” Online [Last accessed 2010/02/10]. <http://saxon.sourceforge.net/>.
- [51] J. M. Silva, A. S. M. Mahfujur Rahman, and A. El Saddik, “Web 3.0: a vision for bridging the gap between real and virtual,” in *CommunicabilityMS '08: Proceeding of the 1st ACM international workshop on Communicability design and evaluation in cultural and ecological multimedia system*, (New York, NY, USA), pp. 9–14, ACM, 2008.
- [52] G. Schreiber and M. Dean, “OWL web ontology language reference,” W3C recommendation, W3C, February 2004. <http://www.w3.org/TR/2004/REC-owl-ref-20040210/>.
- [53] P. Hayes, P. F. Patel-Schneider, and I. Horrocks, “OWL web ontology language semantics and abstract syntax,” W3C recommendation, W3C, Feb. 2004. <http://www.w3.org/TR/2004/REC-owl-semantics-20040210/>.
- [54] H. Takagi and C. Asakawa, “Transcoding proxy for nonvisual web access,” in *ASSETS'00*, (Arlington, Virginia.), 2000.
- [55] D. R. Lunn, “Sadie: Structural-semantics for accessibility and device independence,” master of science, University of Manchester, 2005.
- [56] M. Hori, K. Ono, M. Abe, and T. Koyanagi, “Generating transformational annotation for web document adaptation: Tool support and empirical evaluation,” *Web Semantics: Science, Services and Agents on the World Wide Web*, vol. 2, no. 1, pp. 1–18, 2005.

- [57] K. B. Richard and A. L. Marshall, "Deriving menu-based voice markup from visual markup," US Patent 7406658, IBM, July 2008. <http://www.freepatentsonline.com/7406658.html>.
- [58] H. W. Lie, T. Çelik, I. Hickson, and B. Bos, "Cascading style sheets level 2 revision 1 (CSS 2.1) specification," candidate recommendation, W3C, Apr. 2009. <http://www.w3.org/TR/2009/CR-CSS2-20090423>.
- [59] Z. Shao, R. Capra, and M. Pérez-Quiñones, "Transcoding html to voicexml using annotation," in *IEEE International Conference on Tools with Artificial Intelligence*, (Sacramento, California, USA.), 2003.
- [60] Y. Borodin, "Improved dialog strategies for non-visual web browsing - research proficiency examination." Online [Last accessed 2010/02/10], 2006. <http://www.cs.sunysb.edu/~sbhearsay/publications/borodinRPE.pdf>.
- [61] "Ibm websphere transcoding publisher for multiplatforms, version 4.0." Online [Last accessed 2010/02/10], 2001. <ftp://ftp.software.ibm.com/software/webserver/transcoding/brochures/transcoderspecv4.pdf>.
- [62] N. Annamalai, "An extensible transcoder for html to voicexml conversion," master of science, University of Texas at Dallas, 2002.
- [63] I. V. Ramakrishnan, A. Stent, and G. Yang, "Hearsay: Enabling audio browsing on hypertext content," in *13th international conference on World Wide Web*, (New York, USA), pp. 80–89, 2004.
- [64] G. Y. Saikat Mukherjee and I. V. Ramakrishnan, "Automatic annotation of content-rich html documents: Structural and semantic analysis," in *2nd International Semantic Web Conference*, October 2003.
- [65] International Organization for Standardization, "Codes for the representation of names of languages", ISO 639, July 2002.

- [66] M. Hafedh, E. Amel, and M. Hamid, "Understanding separation of concerns." Online [Last accessed 2010/02/10]. www.latece.uqam.ca/publications/milikharranz-mcheick.pdf.
- [67] C. Kouroupetroglou, M. Salampanis, and A. Manitsaris, "A semantic-web based framework for developing applications to improve accessibility in the www," in *W4A '06: Proceedings of the 2006 international cross-disciplinary workshop on Web accessibility (W4A)*, (New York, NY, USA), pp. 98–108, ACM, 2006.
- [68] C. Taswell, "Doors to the semantic web and grid with a portal for biomedical computing," *Information Technology in Biomedicine, IEEE Transactions on*, vol. 12, pp. 191–204, March 2008.
- [69] M. Hepp, "Semantic web and semantic web services: father and son or indivisible twins?," *Internet Computing, IEEE*, vol. 10, pp. 85–88, March-April 2006.
- [70] L. Nevile, "Adaptability and accessibility: a new framework," in *Proceedings of the 17th Australia conference on Computer-Human Interaction: Citizens Online: Considerations for Today and the Future*, (Narrabundah, Australia, Australia), pp. 1–10, Computer-Human Interaction Special Interest Group (CHISIG) of Australia, 2005.
- [71] "Open vxi voicexml interpreter." Online [Last accessed 2009/02/10]. <http://sourceforge.net/projects/openvxi/>.
- [72] B. Eberman, J. Carter, D. Meyer, and D. Goddeau, "Building voicexml browsers with openvxi," in *WWW '02: Proceedings of the 11th international conference on World Wide Web*, (New York, NY, USA), pp. 713–717, ACM, 2002.
- [73] Z. Fiala and G.-J. Houben, "A generic transcoding tool for making web applications adaptive.," in *17th Conference in Advanced Information Systems Engineering*, (Porto, Portugal), pp. 15–20, 2005.

Appendix A

Evaluation sheet for system A

1 Reading of Afrikaans words

Table 2 shows Afrikaans proper nouns, which the system read to you. Please use table 2 to score the system, by choosing a number in table 1, which appears next to the statement that you best agree with.

Table 1

Score	Statement
1	I completely did not hear this text being read to me.
2	I could guess that something like it was being read to me.
3	I heard this text being read, but I was not satisfied with the way it was read.
4	I heard this text being read, but I was not satisfied with the way it was read.
5	This text was clearly and fluently read to me.

Table 2

Afrikaans noun (s)	Score
Pieter van der Merwe	
Welkom	

2. Date and currency values

The system read a date value and a currency value to you. Please make an attempt to fill in the following information about these values.

2.1. **Date:** The year..... the month..... the day.....

2.2 **Currency:** The currency value is:

2.3. Your overall assessment of the system's ability to read date and currency

values such that you get to understand is:

		For the date value:	For the currency value:
1	Horrible		
2	Poor		
2	Acceptable		
4	Good		
5	Excellent		

3. Overall assessment of the system

What is your overall assessment of the system?

2	Poor	
3	Acceptable	
4	Good	
5	Excellent	

University of Cape Town