

OVR: A NOVEL ARCHITECTURE FOR
VOICE-BASED APPLICATIONS

Submitted in fulfilment
for the requirements of the degree of

MASTER OF SCIENCE

at Rhodes University

Mathe Maema

Grahamstown, South Africa

April 2011

To all my family and friends

– because of your willingness to always march with me to Zion!

Abstract

Despite the inherent limitation of accessing information serially, voice applications are increasingly growing in popularity as computing technologies advance. This is a positive development, because voice communication offers a number of benefits over other forms of communication. For example, voice may be better for delivering services to users whose eyes and hands may be engaged in other activities (e.g. driving) or to semi-literate or illiterate users. This thesis proposes a knowledge based architecture for building voice applications to help reduce the limitations of serial access to information.

The proposed architecture, called OVR (Ontologies, VoiceXML and Reasoners), uses a rich backend that represents knowledge via ontologies and utilises reasoning engines to reason with it, in order to generate intelligent behaviour. Ontologies were chosen over other knowledge representation formalisms because of their expressivity and executable format, and because current trends suggest a general shift towards the use of ontologies in many systems used for information storing and sharing.

For the frontend, this architecture uses VoiceXML, the emerging, and de facto standard for voice automated applications.

A functional prototype was built for an initial validation of the architecture. The system is a simple voice application to help locate information about service providers that offer HIV (Human Immunodeficiency Virus) testing. We called this implementation HTLS (HIV Testing Locator System).

The functional prototype was implemented using a number of technologies. OWL API, a Java interface designed to facilitate manipulation of ontologies authored in OWL was used to build a customised query interface for HTLS. Pellet reasoner was used for supporting queries to the knowledge base and Drools (JBoss rule engine) was used for processing dialog rules. VXI was used as the VoiceXML browser and an experimental softswitch called iLanga as the bridge to the telephony system. (At the heart of iLanga is Asterisk, a well known PBX-in-a-box.)

HTLS behaved properly under system testing, providing the sought initial validation of OVR.

Acknowledgements

“Knowledge is in the end based on acknowledgement.”

Ludwig Wittgenstein

I would like to acknowledge the financial support of Telkom SA, Bright ideas Projects 39, Comverse SA, Stortech, Tellabs, Amatole, Mars Technologies, openVOICE and THRIP through the Telkom Centre of Excellence in the Department of Computer Science at Rhodes University. I would also like to express my gratitude to the Government of Lesotho for offering me an interest-free study loan through the National Manpower Development Secretariat. I am truly thankful and hope that next generations will be as fortunate as I have been.

The overall support I received in my quest to complete this thesis is analogous to the legs of the three-legged pot. Finances represent but one leg of this pot, which stands upright only if none of the legs is missing or broken. The second leg is the mentorship and guidance I received from members of the convergence research group and my (official and non-official) supervisors. I am truly grateful to all of you for your contribution and influence in shaping this beautiful and ever eager-to-learn soul. Alfredo you were the anchor, even when I was in doubt you kept me grounded both technically and otherwise; with you it was so easy to always maintain a healthy sense of humour. Lorenzo as ironic as this may sound, you were the eyes; for you often helped me to see the forest for the trees as I navigated my way to the destination. Denis though on paper you were not my supervisor, you were the reasoning mind that patiently helped me to make sense of the vast field of knowledge representation and management. To everyone else on the research group, you were my ears; for on many occasions you made me feel like finding my voice was important through your willingness to listen to my rants and raves. Thank you all so much.

To my family and friends, you were undoubtedly the third leg of this magnificent pot. Without you guys, the journey would have not been worthwhile! Your love, support and prayers have done so much for me that no words in this world can ever describe. I love you guys - and yes I am not naming you, for you know who you are! Thank you so much for everything. I am truly blessed to have you in my life. I LOVE YOU!

Related Publications

The work that appears in this thesis has been presented in the following conference papers:

1. Mathe Maema, Alfredo Terzoli, and Lorenzo Dalvit. An Ontology-based Telephony Service for the Provision of HIV/AIDS Information. In *Southern Africa Telecommunication Networks and Applications Conference (SATNAC)*, 2008.
2. Mathe Maema, Alfredo Terzoli, and Lorenzo Dalvit. HIV Testing Site (HTS) Locator: Applying Current Computing Trends to Voice Applications. In *Southern Africa Telecommunication Networks and Applications Conference (SATNAC)*, 2010.

Table of Contents

1	Introduction	1
1.1	Motivation and Problem Statement	3
1.2	Goals for the Thesis	4
1.3	Thesis Organisation	5
2	Related Work	6
2.1	Knowledge and Related Concepts	6
2.1.1	What is Knowledge?	6
2.1.2	Types of Knowledge	8
2.1.3	Knowledge Representation and Reasoning	8
2.1.3.1	Knowledge Representation	9
2.1.3.2	Reasoning	10
2.1.4	Understanding Ontologies	11
2.1.4.1	Definitions	12
2.1.4.2	Types of Ontologies	13
2.1.4.3	Uses of Ontologies	14
2.1.4.4	Theoretical Comparison: Databases vs. Ontologies	15

2.1.4.5	Methods and/or Methodologies for Ontology Construction	16
2.1.4.6	Design Criteria for Ontologies	17
2.1.5	Understanding Reasoners	18
2.1.5.1	What is a Reasoner?	18
2.1.5.2	Uses of a Reasoner	19
2.1.5.3	Selection Criteria for a Reasoner	20
2.2	Human-Machine Dialog Systems	20
2.2.1	Types of Dialog Systems	21
2.2.2	Dialog Management	22
2.2.3	Dialog Systems Architectures	22
2.3	VoiceXML	24
2.3.1	What is VoiceXML?	24
2.3.2	VoiceXML Architectural Model	26
2.4	Information and Communication Technologies in Health	27
2.4.1	Healthcare Knowledge Management	27
2.4.2	IVR in Health	29
2.5	Summary	30
3	Software Components	31
3.1	Information Flow	32
3.2	Main Features of the Architecture	33
3.3	Components for Application Development	34
3.3.1	Ontology Authoring Tool	34

3.3.2	Ontology APIs	35
3.3.3	Reasoning Engine	35
3.3.4	Rule Engine	36
3.4	Components for the Deployment Environment	36
3.4.1	Application Servers	38
3.4.2	Asterisk	38
3.4.3	VoiceXML Browser	38
3.4.4	Text-to-Speech	40
3.5	Summary	40
4	Ontology Construction for Developing the Prototype System	41
4.1	Ontology Development Life Cycle	41
4.1.1	Development Life Cycle and the Final Product	43
4.1.2	Development Life Cycle and the Methodologies	44
4.1.2.1	Uschold and King's Method	46
4.1.2.2	Grüninger and Fox's Methodology	47
4.1.2.3	METHONTOLOGY	49
4.2	Building the HIV and AIDS Service Provider Ontology	50
4.2.1	Specification: Identifying the Purpose & Scope	54
4.2.1.1	Competency questions	54
4.2.1.2	Enumeration of important terms	55
4.2.2	Conceptualisation	56
4.2.3	Formalisation and Implementation	64

4.2.3.1	Ontology Language Selection	64
4.2.3.2	Tool Selection	65
4.2.4	Evaluation, Documentation & Maintenance	66
4.3	Summary	68
5	Basic Implementation of the Prototype System	69
5.1	Implementation Strategy	70
5.2	First Group of Iterations for the Implementation	72
5.2.1	View Implementation	72
5.2.2	Controller Implementation	77
5.3	Summary	82
6	Enhanced Implementation of the Prototype System	83
6.1	Second Group of Iterations for the Implementation	84
6.1.1	Dialog Processing with Rules	84
6.1.2	Dynamic Page Rendering	90
6.2	Summary	93
7	System Testing and Discussion	94
7.1	Proposed Architecture: Implementation View	94
7.2	System Testing	97
7.3	Integration and Implementation Experience	99
7.4	Potential Barriers for Proposed Architecture Adoption	100
7.4.1	Demand for Wide Range of Competencies	101
7.4.2	Availability of Ontologies	101
7.5	Summary	102

8 Conclusion	103
8.1 Thesis Synopsis	103
8.2 Achievements	104
8.3 Future Work	105
8.4 Summary	106
List of References	107
A Snapshot from Protege	118
B A Snippet XML File for the Service Provider Ontology	119
C Example Scripts	121
Call Script 1	121
Call Script 2	122
D Rule Engine Class	123
E Accompanying CD-ROM	124
F Deployment Guide	125

List of Figures

1.1	Overall Architecture	4
2.1	Snapshot of Areas of Interest to the Thesis	7
2.2	Guarino [60] Ontology Categorisation	13
2.3	Evolution of Representation of Backend Data	16
2.4	Reasoner Basic Functionality	18
2.5	Self-Service Model	20
2.6	Classification of Dialog Systems	21
2.7	Reference Architectures for Dialog Systems	23
2.8	VoiceXML Architecture (from [50])	26
2.9	Pyramid of HKM services	28
3.1	Categorisation for Utilised Components	31
3.2	Overall Architecture	32
3.3	Components to Support Core VoiceXML Functionality	37
3.4	iLanga Main Components (from [96])	37
4.1	Ontology Development Life Cycle	42
4.2	Decision Tree for Selecting Ontology Development Life Cycle (from [106])	45

4.3	Spiral Model	45
4.4	A Comprehensive Service for HIV and AIDS Information Dissemination . .	52
4.5	Process Overview for Building the HIV and AIDS Service Provider Ontology	53
4.6	Tasks for Conceptualisation Activity According to METHONTOLOGY . .	57
4.7	Excerpt of Concept Taxonomy from the HIV and AIDS Provider Ontology	58
4.8	Ad hoc Binary Relation Diagram Example for the HIV and AIDS Provider Ontology	58
4.9	High-level Conceptual Model of the HIV and AIDS Provider Ontology . . .	59
5.1	Use Case Diagram for HIV Testing Site Locator	69
5.2	Model View Controller Architectural Design Pattern	70
5.3	Implementation Breakdown based on MVC	70
5.4	Spiral Model for Implementing the Prototype System	71
5.5	High Level Flowchart Capturing Prototype Functionality	74
5.6	Listing of VoiceXML Page for the Initial Dialog (i.e. Welcome Page)	75
5.7	Events for Basic VoiceXML Page Rendering	76
5.8	Basic Components in a Description Logic based System	77
5.9	Basic Controller's Task	78
5.10	Visual Implementation of the Prototype Service Rendered in HTML	79
5.11	Listing for Processing and Presenting Query Results	80
5.12	Listing for Query Generation	80
5.13	Taxonomy Tree for HIV Test Providers	81
5.14	Listing for Preference Configurations	82
6.1	Transformation Process: Basic to Enhanced Prototype	83

6.2	Servlet Chaining	84
6.3	Flowchart for Presenting Query Results	85
6.4	Listing for Results Filtering Using <i>If-Else</i> Statements	86
6.5	Listing of a Sample Drools Rule for Performing Alphabetic Sort	87
6.6	Overview of Processing of Query Results by the Rule Engine	88
6.7	Listing of Code Snippet for Results Filtering Using the Rule Engine	88
6.8	Events for Interacting with the Backend	89
6.9	Listing for Creating a VoiceXML Form for Presenting Provider Details	90
6.10	Listing of VoiceXML Template Page for Results Presentation	92
6.11	Listing for Initialising Template Variables	92
7.1	Complete Architectural View of HTLS	95
7.2	Requirements Classification	97
A.1	Snapshot from Protege of the Service Provider ontology	118

List of Tables

2.1	Explicit vs. Tacit Knowledge	8
2.2	Interpretations of “Ontology” in Computing (adapted from Guarino and Giaretta [62])	12
2.3	Databases vs. Ontologies (from [71])	15
2.4	VUI vs. GUI Development	25
4.1	Comparison of Life Cycles and their Final Product	43
4.2	Classification of Activities in Enterprise Methodology	47
4.3	Classification of Activities in Grüninger and Fox’s Methodology	48
4.4	Classification of Activities in METHONTOLOGY	49
4.5	Example of Glossary of Terms for the HIV and AIDS Service Provider Ontology	58
4.6	Excerpt of Concept Dictionary from the HIV and AIDS Service Provider Ontology	60
4.7	Excerpt of Ad hoc Binary Relation Table from the HIV and AIDS Service Provider Ontology	61
4.8	Excerpt of Instance Attribute Table from the HIV and AIDS Service Provider Ontology	61
4.9	Example Formal Axiom from the HIV and AIDS Service Provider Ontology	62

4.10	Example Rule from the HIV and AIDS Service Provider Ontology	63
4.11	Excerpt of Instance Table from the HIV and AIDS Service Provider Ontology	64
4.12	A Brief Comparison of OWL Sublanguages	65
4.14	Comparison of FaCT++ and Pellet Reasoners	67
7.1	Decision Table Used for Testing	98
F.1	Deployment Locations and Context Names Used for Testing	126

Glossary of Terms

API Application Programming Interface

AI Artificial Intelligence

ASR Speech Recognition

DL Description Logic

DTMF Dual Tone Multiple Frequency

FOL First Order Logic

GUI Graphical User Interface

IVR Interactive Voice Response

KM Knowledge Management

MVC Model-View-Controller

OVR Ontologies, VoiceXML and Reasoners

OWL Web Ontology Language

RDF Resource Description Framework

TTS Text-to-Speech

VUI Voice User Interface

W3C World Wide Web Consortium

XML Extensible Markup Language

Chapter 1

Introduction

The convergence of various modes of communication, supported by ever less expensive visualisation technologies, provides many opportunities for creating non voice based services that can be used for both business and social innovation. Notwithstanding this, voice remains a powerful mode of communication. This means that development of voice applications remains important in delivering services to users. In part, this is because for certain tasks, such as simple information retrieval while driving or engaged in any other activity, access using voice is better (and generally requires less costly terminals). Further, access with voice has the advantage of allowing services to be availed to a wider community of users that includes visually impaired and semi literate individuals. In fact, a look into the trends suggests that the opportunities of voice as a medium of communication, influenced by advances in speech recognition technologies, is revolutionising the commercial sector [85, 102].

Driving this revolution is a trend capturing a switch “from e-everything to u-everything” [73], (i.e., electronic to ubiquitous). This trend consists of systematically digitising resources to enable access to information anywhere, anytime and (hopefully) to every person [102]. This trend is fuelled and underpinned by numerous factors. These include the use of the converged network and the ever increasing computational power for creating and delivering services to users.

This thesis proposes an architecture for building voice applications i.e. applications or services¹ that allow users to interact with systems using the acoustic interface. Interactive

¹It is important to note that unless otherwise stated, service in this thesis shall refer to a telephony service and will be regarded as a catch all term that encompasses the view of application as a service.

Voice Response (IVR) based applications including those that use Dual Tone Multiple Frequency (DTMF)² are types of voice applications.

The impetus behind this proposal is to encourage use of voice applications because of their potential to provide a truly ubiquitous solution for various users, across different socio-economic backgrounds. But most importantly, the intent is to add value, in terms of promoting and encouraging self service by users. This is important because self-service offers numerous benefits. For example, from the point of view of businesses or service providers, self service offers a cost effective solution for supporting customer care operations. Conversely, from the point of view customers or users, self service offers the convenience of anytime, anywhere access to information.

The proposed architecture, named Ontologies, VoiceXML and Reasoners (OVR), uses knowledge management and representation techniques in the backend instead of traditional databases to structure and manage information (consumed by a developed service). This allows one to build voice applications that can infer new information through reasoning. This is a major benefit, since databases lack the intrinsic capability to reason and infer new information. This has long been recognised by the Information and Communication Technology (ICT) industry, but has until recently received little attention.

A reasoner, albeit a more specialised one, is also used in the proposed architecture to help with dialog generation. Overall, the use of reasoners in the architecture provides the backend with some degree of intelligence that can be harnessed to create a next generation of voice applications. Applications that can use the power of inference to adapt interaction will hopefully transform the user experience in voice based self-service contexts, making it ‘short, sweet and painless’.

The proposed architecture was functionally validated through the implementation of a prototype service. This service focuses on increasing access to HIV and AIDS testing information. The aim of the service is to contribute to ongoing efforts for reducing rates of infection. The requirements of this service were determined from information gathered in interviews and relevant online resources.

²In this thesis, *touch tone* will be used instead of DTMF because of growing acceptance of the term within the telephony industry.

1.1 Motivation and Problem Statement

Through the years, the market has continued to grow for voice applications as self-service systems [85]. In part, this is due to the ubiquity of phones and the potential they offer for creating services that are not dependent on the visual modality.

Many of these voice self-service systems have been implemented as IVR based applications. Unfortunately, both anecdotal and empirical evidence suggest that users find IVR based applications confusing and frustrating [42, 107, 108]. Thus, given an opportunity, users tend to opt for a human agent and this leads to low usage of self-service [37, 42, 107, 108].

There are a number of factors discussed in literature to explain or justify this pattern of behaviour. One such factor, which is considered in this thesis, is the lack of flexibility of IVR based systems. According to Dean [42], the preference that users have for human agents is rooted in the perception that IVR transactions take longer to complete and are less customisable. (Customisability, in this case, is tied to flexibility and/or ability to be adaptive - one of the defining attributes of intelligence, see for example, Landauer and Bellman [79].)

The perception, justified or not, raises the question: should users expect more from IVR based systems? That is, should users expect IVR systems to treat them intelligently and in a timely manner? Assuming the answer is affirmative for these questions, another question may be asked from an implementation point of view: how can intelligence be introduced structurally into IVR systems in order to gain intelligent behaviour akin to that of a human agent?

Artificial Intelligence (AI), a branch of computer science dedicated to machine intelligence, has identified Knowledge Representation (KR) as a key area of study for understanding and generating intelligent behaviour. In particular, ontologies have received special attention within AI and other disciplines of computing [43]. Thus, as reflected by the trends ontologies might in the future be used by any system that enables knowledge and/or information exchange.

This thesis proposes an architecture that uses ontologies instead of traditional databases for structuring and managing information. The use of ontologies accentuates the importance of providing a shared and common understanding of the domain of discourse. As articulated by Fensel [46], this makes ontologies a key asset for application systems that support the exchange of data, information, and knowledge. The proposed architecture

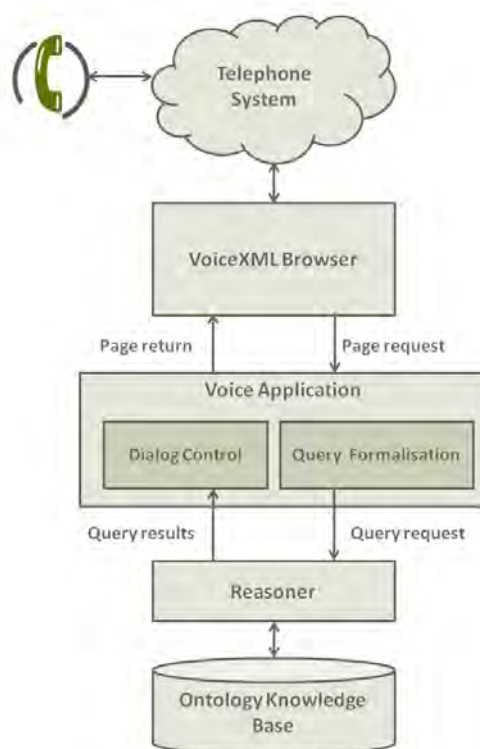


Figure 1.1: Overall Architecture

also uses rules for dialog processing and VoiceXML for rendering the dialogs to the user. An initial, high level view of this architecture is presented in Figure 1.1.

1.2 Goals for the Thesis

The two main goals for this thesis are as follows:

1. To propose an architecture for voice applications with reasoning mechanisms for inferring new information. The objective is to contribute to a move towards building intelligence into voice services to improve the user experience.
2. To build a functional prototype that will allow validation of the proposed architecture, as well as generate immediate spin off benefits for the community. A service directory for locating providers of HIV and AIDS specific services in Grahamstown was selected for the prototype implementation.

These two goals effectively provide the scope for this thesis, which is limited to demonstrating functional validity of the proposed architecture.

1.3 Thesis Organisation

The remainder of this thesis is organised as follows.

- ⇒ Chapter 2 discusses related work, organised under four main themes to ease presentation. It should also convey the message that work done in this thesis lies somewhere in the intersection of these themes.
- ⇒ Chapter 3 presents the various components of the proposed architecture and discusses their integration.
- ⇒ Chapter 4 discusses the construction of the ontology for the prototype system. This chapter outlines the importance of conceptualisation in constructing a functional and useful ontology.
- ⇒ Chapter 5 discusses the first ‘group’ of iterations for the implementation of the prototype system. The chapter is preambled by a discussion on the overall implementation strategy followed.
- ⇒ Chapter 6 discusses the second ‘group’ of iterations for the implementation of the prototype system.
- ⇒ Chapter 7 provides a summary of the system testing process and an analysis of the proposed architecture. The chapter discusses the theoretical and practical implications derived from the implementation of the prototype system.
- ⇒ Chapter 8 provides concluding remarks for the thesis and provides some directions for future work.

Chapter 2

Related Work

There is a lot of literature that is directly and indirectly relevant to the work done in this thesis. This chapter will review concisely some of this literature to provide an overall picture of how different ideas from different sources can be woven together to support a sound architectural design.

As a preamble to how the discussion will proceed, Figure 2.1 provides a snapshot of areas that will be explored. As seen in the figure, these areas are organised into four themes. The first theme discusses knowledge and key concepts surrounding its representation and processing. The second theme focuses on dialog systems and architectures used to build them while the third theme focuses on VoiceXML and its use in developing voice applications. Finally, the fourth theme aggregates (in a contextualised manner) topics relevant to health, the domain of the service used as a case study for the thesis.

2.1 Knowledge and Related Concepts

This section will focus on knowledge and its categorisation. Further, it will explore the meaning of knowledge representation and demonstrate how this fits with the concept of ontologies.

2.1.1 What is Knowledge?

The old adage '*knowledge is power*' is truer than ever in today's society, which recognises knowledge as a critical resource. This can be gleaned anecdotally from the pivotal role

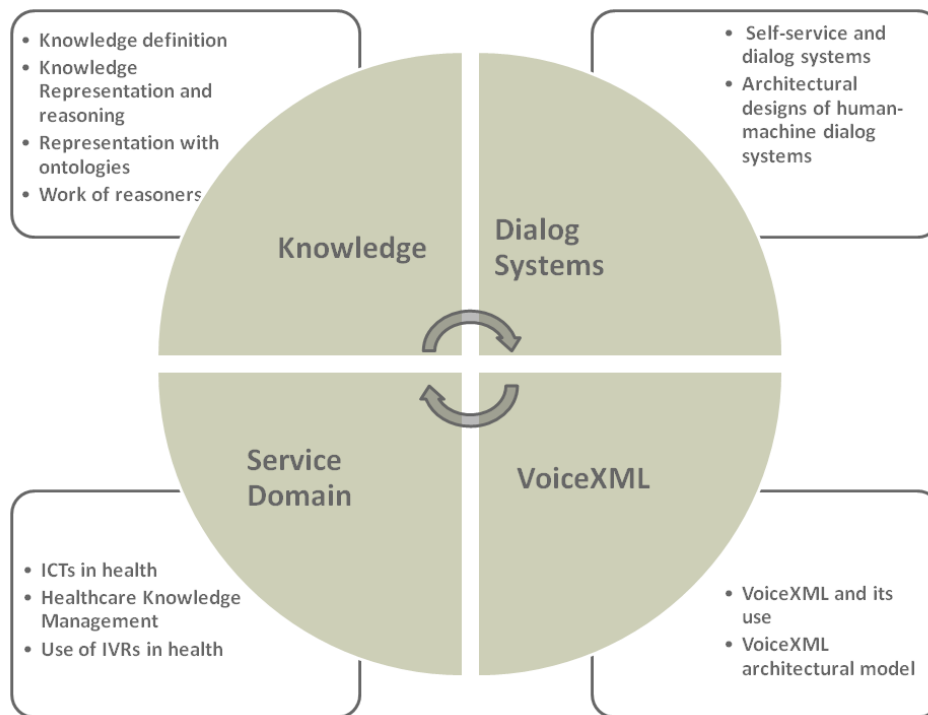


Figure 2.1: Snapshot of Areas of Interest to the Thesis

that knowledge and its management are believed to play in the decision making process of any organisation¹.

Otherwise, verification of the above fact can be sought from Knowledge Management (KM) literature. As aptly captured in the name, KM is about managing knowledge. The management involves, for example, processes and techniques for knowledge generation, knowledge codification, knowledge transfer and knowledge application [67]. A question to ask is: what exactly is knowledge or what constitutes knowledge? Unfortunately, there is no easy answer for this question. According to Nickols [92], knowledge is:

“A very slippery concept with many different variations and definitions. The nature of knowledge and what it means to know something are epistemological questions that have perplexed philosophers for centuries and no resolution looms on the horizon.”

In light of the above, no attempt will be made to explore the various definitions that exist for knowledge. Webster’s dictionary [10] defines knowledge within computing context as:

¹The deduction can also be made by looking at the role of information in decision making, since knowledge and information are often used interchangeably even though they are not synonyms.

Explicit Knowledge	Tacit Knowledge
Easy to code (e.g. document, identify, articulate)	Difficult to code (e.g. document, identify, articulate)
Easy to communicate	Difficult to communicate
Cannot exist independent of tacit knowledge	Can exist by itself

Table 2.1: Explicit vs. Tacit Knowledge

“objects, concepts and relationships that are assumed to exist in some area of interest”. For purposes of this thesis, this definition will be deemed as sufficient since it is consistent with ontologies and their modelling. This will become more apparent in the next chapter when implementation details of the service ontology are discussed.

2.1.2 Types of Knowledge

Although there are a number of ways to categorise knowledge, in this thesis it will be categorised as either explicit or tacit. Explicit knowledge refers to codified knowledge, articulated in some language or format. Conversely, tacit knowledge refers to knowledge that cannot be articulated [47, 92].

Table 2.1 provides a quick comparison of these two types of knowledge, whose categorisation stems from seminal work by Polanyi [99, 100]. According to Polanyi, “we can know more than we can tell”. Therefore, all knowledge is either tacit or rooted in tacit knowledge. As he argues, “a wholly explicit knowledge is unthinkable” (as cited in [67]). Further, it is not possible to have explicit knowledge (whether expressed in words, numbers, or any other format) without tacit understanding.

Given that part of the work in this thesis is to codify knowledge, Polanyi’s argument is relevant. As it will be seen in later discussions, it provides context for understanding why knowledge representation is, for example, a set of ontological commitments (i.e. a set of decisions on what to keep in focus and what to blur). In this particular case, an inference from his argument would be that a decision on what to focus and blur in representing knowledge is indeed dependent on tacit understanding. Further, such a decision is necessary because not everything can be represented.

2.1.3 Knowledge Representation and Reasoning

It is often stated that knowledge representation and reasoning are intertwined [31, 41, 43]. Thus, a discussion of one without the other would lead to incomplete understanding. In

order to prevent this incompleteness, this section will discuss these two concepts together. The section will show how representation of knowledge is connected to reasoning, and how reasoning over knowledge is impacted by its representation.

2.1.3.1 Knowledge Representation

An important question underpinning the work in this thesis is: “what is knowledge representation?” This question is not new and has been a subject of interest in the field of AI for decades.

Davis, Shrobe and Szolovits [41], in a paper dedicated to answering this seemingly challenging question, argue that knowledge representation can be understood in terms of five roles it plays. A brief summary of these roles is outlined below.

Role 1: A Knowledge Representation is a Surrogate

In this role, knowledge representation serves as a substitute or place holder (i.e. surrogate) for things that exist in the world under discussion. Hence representation functions as a mapping between two domains: the machine world and the real world. This mapping allows operations on and with surrogates to substitute those that are possible on the real thing. This means reasoning itself can be viewed as surrogate for action in the real world.

Role 2: A Knowledge Representation is a Set of Ontological Commitments

By virtue of being a surrogate, representations cannot be entirely accurate. This, aptly stated by Davis *et al.* [41], is because “the only completely accurate representation of an object is the object itself”. For this reason, the goal of representation is not to attain perfect fidelity. The goal is to approximate the real world as close as possible by capturing things that are deemed important to tasks to be performed.

Implicitly, this means that what is represented results from a set of decisions about what needs to be included and excluded. Hence, knowledge representation can be regarded as set of ontological commitments. These commitments serve as strong pair of glasses that bring into sharp focus some parts of the world whilst blurring other parts. This focusing effect may be due to a constraint imposed by representation technologies or tools used. Hence, it is important to note that the commitments begin with the choice of representation technology to use.

Role 3: A Knowledge Representation is a Fragmentary Theory of Intelligent Reasoning

This role captures the strong tie that exists between representation and reasoning. Representation at its core is about how people can reason intelligently with represented information to obtain sound results as fast as possible. For this reason, a theory of what constitutes intelligent reasoning is implicitly embedded within a representation. As argued by Davis *et al.* [41], this theory has three components to it: (i) the representation's fundamental conception of intelligent inference; (ii) the set of inferences the representation sanctions (i.e. permissible inferences); and (iii) the set of inferences it recommends.

Role 4: A Knowledge Representation is a Medium of Efficient Computation

This role serves as a general reminder of the importance of computational efficiency. The ultimate goal for using representation is to compute with it. To this end, efficiency is important and cannot be divorced from representation.

Role 5: A Knowledge Representation is a Medium of Human Expression

This role communicates the idea that representations are similar to metaphors in speech. Thus, as people use representations, they are essentially expressing how they see the world. In this role, key questions to ask include: How general is the representation? How precise is it? How expressive is it? How well does it support communication? And any other question that can be linked to how easy representation as a language contributes to communication.

2.1.3.2 Reasoning

As previously stated, knowledge representation and reasoning are inextricably entwined together. However, it is important to note that the two concepts are not the same. Reasoning refers specifically to the computation process of deriving conclusions that are sanctioned by the represented knowledge [41, 56]. This computation is also referred to as *derivation, deduction or inference* [56].

Many applications that use a knowledge base (repository for represented knowledge) also make use of a reasoning system (reasoner). The reasoner helps to harness the full potential of the knowledge base by computing, as needed, the conclusions from the encoded knowledge. Without the reasoner, the alternative would be to explicitly code these conclusions [31, 43, 56]. This is not usually ideal because, on the one hand, there is potentially an

infinite set of possible conclusions that can be derived from a knowledge base [56]. On the other hand, the encoding reduces the overall flexibility of the application.

Based on the above, the benefits of using a reasoner are evident. However, for practical and efficient use in any system, there are two important considerations that need to be made. The first is how to reduce the set of sanctioned inferences without making unsuitable recommendations. As stated by Davis *et al.* [41], representation must provide intelligent indication on which inferences to recommend. For this reason, it is prudent to choose a representation technology well; since this choice is inevitably a choice in the conception of what the chosen technology deems as intelligent reasoning [41].

The second consideration pertains to the task of selecting a reasoner. According to Greiner, Darken and Santoso [56], an ideal reasoner is one that “always returns all-and-only the correct and precise answers, immediately, to arbitrary queries” [56]. Unfortunately, as indicated by some researchers, it is not always possible to find such a reasoner, especially for reasoning over expressive representation [31, 56]. They argue that while increased expressivity makes a system more correct and precise compared to the world being represented, it also increases the inefficiency of reasoning, which in turn, may cause the reasoner to not be decidable. As such, there is a trade off between expressiveness of representation and efficiency of a reasoning system. Understanding this trade off is important for selecting both the representation language and the reasoner to use in any knowledge based system.

2.1.4 Understanding Ontologies

In recent years, ontologies have become the knowledge representation medium of choice in many computing disciplines [32]. Although fairly new to computing, ontologies have existed for centuries in the area of philosophy.

In literature, there are multiple definitions provided for what ontology is in computing, and unlike in the area of philosophy, there is still no single definition that is used. However, this does not suggest that the definitions are in conflict with each other. The definitions do reflect a general consensus that ontology is a description of concepts and their relations within a given domain. The difference in the definitions lies mainly in the interpretation of words such as ‘*conceptualisation*’ and ‘*formal*’.

In this subsection, some of the definitions will be reviewed, but not for the purpose of deciding whether or not one is better than the other. The objective for the discussion is to

Interpretation	Context of Use
1. Ontology as an informal conceptual system	Conceptual framework
2. Ontology as a formal semantic account	Conceptual framework
3. Ontology as a specification of a “conceptualisation”	Concrete artefact
4. Ontology as a representation of a conceptual system via a logical theory 4.1 characterised by specific formal properties 4.2 characterised only by its specific purposes	Concrete artefact
5. Ontology as the vocabulary used by a logical theory	Concrete artefact
6. Ontology as a (meta-level) specification of a logical theory	Concrete artefact

Table 2.2: Interpretations of “Ontology” in Computing (adapted from Guarino and Giaretta [62])

provide understanding on ontologies and how their use fits with knowledge representation and creation of telephony services.

2.1.4.1 Definitions

There are many definitions for ontology in computing. One of the popular definitions describes ontology as: “an explicit specification of a conceptualisation” [58].

Table 2.2 provides a compilation by Guarino and Giaretta [62] of interpretations of other definitions that exist in literature including that by Gruber [58], provided above. As indicated in the table, each interpretation is valid within a specific context of use. According to Guarino and Giaretta [62], there are two contexts in which the term ‘ontology’ is used in computing; the first is ontology as a concrete artefact and the second is ontology as a conceptual framework. As a concrete artefact, ontology relates to “specific knowledge bases (or logical theories) designed with the purpose of expressing shared (or sharable) knowledge” [62] at a syntactic level. As a conceptual framework, ontology relates to a conceptualisation that reflects a particular view of the world being represented at a semantic level.

An important consequence of making a distinction between these two contexts of use is that ontology as an artefact can commit to different conceptualisations. Thus, from the point of view of knowledge representation as a set of ontological commitments, conceptualisation can be seen as commitments underlying the artefact. Therefore, depending on the commitments, ontologies in any given domain may vary significantly in their conceptualisation. This means a commitment in the selection of one or another ontology artefact within a domain can produce a very different view depending on the task to be completed [41].

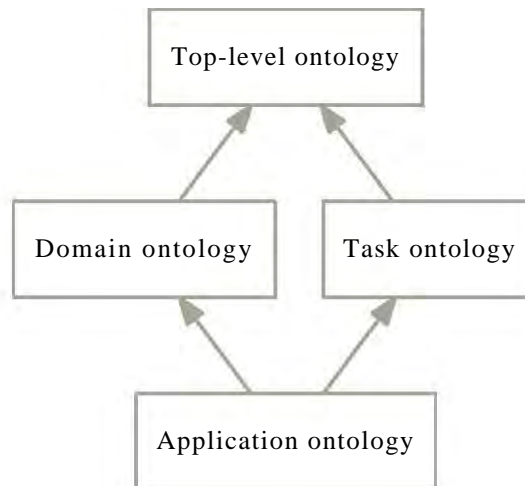


Figure 2.2: Guarino [60] Ontology Categorisation

2.1.4.2 Types of Ontologies

There are different types of ontologies to support various uses of ontologies. In literature, the categorisation is usually made using criteria that reflects the potential ontologies exhibit for reuse in engineering knowledge based systems [57].

One of the most notable categorisation has been provided by Guarino [60, 61]. Figure 2.2 provides a summary of this categorisation, which distinguishes four types of ontologies classified according to their level of dependence on a particular task or point of view. These are as follows:

- *Top-level Ontologies*

These ontologies describe very abstract and general concepts such as time and space in a manner that is not dependent on any particular problem or domain.

- *Domain and Task Ontologies*

These types of ontologies represent, respectively, knowledge about a specific domain (e.g. health), or knowledge about a specific task (e.g. scheduling). The representation is often achieved by specialising concepts introduced in top-level ontologies. This means that these ontologies although narrower than top-level ontologies, they offer more depth in their representation of the domain of discourse and a particular task.

- *Application Ontologies*

These ontologies serve as a medium for describing concepts relating to a task in a particular domain. They often make use of both domain and task ontologies to describe, for example, roles played by domain entities in performing a specified task. As such, they are much narrower in scope than domain and task ontologies.

In summarising Guarino's categorisation, Grimm, Hitzler and Abecker [57] say the lattice shown in Figure 2.2 "represents an inclusion hierarchy [where] the lower ontologies inherit and specialise concepts and relations from the upper ones." Thus, the upper ontologies tend to have a broader potential for reuse than the lower ones which have a much narrower scope.

2.1.4.3 Uses of Ontologies

Ontologies are used in many different ways in various fields of computing. For example, they are used for information retrieval, database design and integration, and semantically enhanced content management in specialised applications for bio-informatics, e-commerce and education [32, 57, 61, 63].

Despite the many ways in which ontologies can be used, authors like G. Dragan, D. Dragan and Vladan [43] and Fensel [46] contend that the main purpose of ontologies is to facilitate *knowledge sharing* and *knowledge reuse* by applications. The roots of this position can be traced back to early comments on use of ontologies by authors such as Davies, van Harmelen and Fensel [40] and Grimm *et al.* [57]. These authors attributed the increase in the momentum of use of ontologies to the promise they offer in providing a common and shared understanding of a domain, which can be communicated between people and machines.

Regardless of how they may be used, Baader, Horrocks and Sattler [24] assert that effective use of ontologies is underpinned by well designed and well defined ontology language that is supported by reasoning tools. According to them, reasoning is important for the creation of high quality ontologies, and for exploiting the rich structure of ontologies and ontology based information. This is because reasoning can be used to determine consistency of an ontology by, for example, checking for contradictions within concepts. Furthermore, reasoning can be used to derive implied relations and to answer queries based on both the implied and explicit knowledge captured in an ontology.

Database	Ontology
Closed World Assumption (CWA) – missing information is treated as false.	Open world assumption (OWA) – missing information is treated as unknown.
Unique Name Assumption (UNA) – each individual has a single unique name.	No UNA – individuals may have more than one name.
Schema use – schema behaves as constraints on structure of data to define legal database states.	Axiom use – axioms behave like implications (inference rules) that entail implicit information.
Query Answering – amounts to model checking i.e. a “look up” against the data. – can be very efficiently implemented to yield fast response times.	Query Answering – amounts to theorem proving i.e. logical entailment. – may have very high worst case complexity that may result in slow response times.
– schema plays no role; data must explicitly satisfy schema constraints.	– axioms play a crucial role to provide an answer that may include implicitly derived facts or an answer to a conceptual or extensional queries.

Table 2.3: Databases vs. Ontologies (from [71])

On the premise of the importance of ontology reasoning, Baader *et al.* [24] argue that Description Logics (DLs) are ideal candidates for ontology languages. DLs are but formal knowledge representation languages that model concepts and their relationships in a highly expressive manner. The argument for their use will further be explored in the next chapter when discussing the selection of the ontology language for the implementation of the ontologies underlying the prototype service.

2.1.4.4 Theoretical Comparison: Databases vs. Ontologies

As it might be deduced from some highlighted uses of ontologies, their use either replaces or enhances that of databases. A pertinent question that could be asked is: when to use ontologies or databases? The answer is, there are different motivations to justify either one of them. Ontologies have the ability to reason with the data they store. Databases are mature, stable and widely available. To gain more insight on why the motivations may differ, Table 2.3 compares the two technologies.



Figure 2.3: Evolution of Representation of Backend Data

As illustrated in the table, in some aspects, ontologies are analogous to databases. However, as noted in [71], there are important differences in semantics. For example, ontologies use ‘axioms’ and databases use ‘schema’; while these concepts are similar, the former involves implications and the latter constraints. Another notable difference lies in the potential response times for answering user queries, as shown in the table, databases are generally superior to ontologies and as it may be argued, this could be attributed to the maturity of databases.

As implied from above, ontologies are a relatively new technology. They are predicted to be next in the evolution of how backend data is represented, see Figure 2.3. Based on this prediction, ontologies can be viewed as the historical equivalent of relational databases when use of flat file (databases) was common. As reflected in literature, during that period, it was important to justify use of relational databases as opposed to flat files; unlike today when their use is accepted without much justification (precisely because the benefits of structure are well understood and the database technology has matured significantly). This means, with continued use, ontologies have the potential to be as widely accepted as relational databases are today.

2.1.4.5 Methods and/or Methodologies for Ontology Construction

There are a number of methods and/or methodologies for ontology construction. Examples include the Uschold and King’s method [117], the Grüninger and Fox’s methodology [59], METHONTOLOGY [49] and the 101 Method [93]. The suitability of each method and/or methodology is dependent on several factors. Some of these factors, as identified by Gómez-Pérez, Fernández-López and Corcho [63], include the development life cycle and strategies employed for identification of ontology concepts.

In comparing the above-mentioned methods (except the 101 Method), Gómez-Pérez *et al.* [63] conclude that each approach has some drawbacks. Thus, there is no single method

and/or methodology for ontology construction that is deemed as best [43, 63, 93, 98]. As a result, it is common, especially for the construction of domain ontologies, to merge different methods and/or methodologies together, so as to capitalise on their individual strength [34]. This approach of merging methods and/or methodologies for ontology construction is followed in this thesis as will be seen in the next chapter.

2.1.4.6 Design Criteria for Ontologies

As Gruber [58] aptly notes, the task of representing something in an ontology involves making decisions. These decisions inherently involve making trade offs. For this reason, Gruber argues that objective criteria is needed for evaluating ontology designs. To this end, he has proposed the following five general design criteria for ontologies, particularly those used in knowledge sharing and applications based on shared conceptualisation:

1. *Clarity*: The intended meaning of the defined terms should be captured objectively in the ontology, to enable effective communication. To this end, whenever possible definitions should be formalised (i.e. expressed using logical axioms). Further, definitions should be documented using natural language.
2. *Coherence*: Consistency should be achieved such that there are no contradictions between definitions, concepts and inferences made within the ontology. Without consistency, the ontology becomes incoherent.
3. *Extendibility*: An ontology should be designed to allow additions to be made without a need to revise existing definitions and concepts.
4. *Minimal encoding bias*: The conceptualisation of the ontology should be independent of the tools used for representing it. That is, “the representation choices [should not be] made purely for the convenience of notation or implementation” [58].
5. *Minimal ontological commitment*: An ontology should be modelled to serve its purpose but without making too many claims about the world being modelled so as to maintain a degree of generality for potential reuse. This means the ontological commitments made should be kept at a minimum. This can be achieved “by specifying the weakest theory (allowing the most models) and defining only those terms that are essential to the communication of knowledge consistent with that theory” [58].

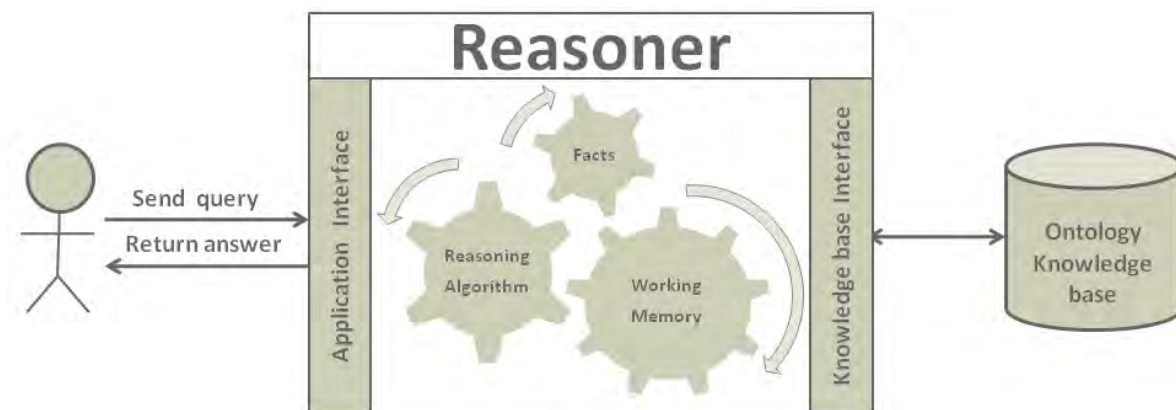


Figure 2.4: Reasoner Basic Functionality

2.1.5 Understanding Reasoners

The previous subsection has attempted to shed some light on the knowledge representation formalism used in this thesis. As noted in [56], for each formalism used to represent knowledge, there is an associated type of reasoning. In turn, this implies different types of reasoners are needed to support reasoning with different formalisms for representing knowledge. In this thesis, as it might have been deduced from page 14, interest is on reasoners that support DL formalism (i.e. DL reasoners).

Notwithstanding the interest of the thesis, this subsection will attempt to enhance overall understanding of what the task of reasoning entails. First, in greater detail than before, the subsection will define a reasoner and discuss qualities that can be used to select an ideal reasoner for use. Then, it will proceed to discuss the various functions or tasks that a reasoner is expected to perform.

2.1.5.1 What is a Reasoner?

A reasoner (also known as a reasoning engine) refers to a system that is used to perform computations for deriving conclusions or answers from a knowledge base. Figure 2.4 captures at a basic level how a reasoner works. As it can be seen from the figure, the user application sends through queries via the application interface to the reasoner. Before an answer can be returned, the reasoner performs necessary computations by consulting the knowledge base via an interface that enables communication between the two entities. These computations will not be discussed in detail within this thesis. In brief, the reasoner processes the facts it has in its working memory with the aid of a reasoning algorithm.

Essentially, this algorithm matches the facts to the represented knowledge in order to infer the appropriate answer to return to the user application.

As previously suggested, the processing done by a reasoner to infer an answer is also called inference (aside from being called reasoning). For this reason, sometimes a reasoner is called an inference engine. Technically, this is not accurate. A subsumption (*is-a*) relationship exists between the two, with the reasoner assuming the role of a superclass and the inference engine that of a subclass. Principally, as gleaned from [119], a reasoner generalises the concept of an inference engine.

In this thesis, both an inference engine and a reasoner are used. The inference engine (rule engine) is used to reason with knowledge organised as rules expressed in the form of *if...then* statements while the reasoner is used to reason with knowledge represented in a much richer format as ontologies.

2.1.5.2 Uses of a Reasoner

Reasoners offer other services aside from answering queries that define the scope of use. For example, reasoners offer a consistency checking service. This service is used to check that the knowledge reasoned with does not contain any contradictions. This verification is important because it ensures a degree of reliability before a reasoner can attempt to answer any queries presented to it.

Other services offered specifically by DL reasoners (as cited in [105, 119]) include:

- ➔ **Concept satisfiability:** This is very similar to consistency checking except focus is on verifying that any given class can have individuals that belong to it. If no individual can belong to a class, the class is unsatisfiable and defining any instance will lead to inconsistency.
- ➔ **Classification:** This computes the class hierarchy by determining subclass relationships between for named classes. The class hierarchy is important for answering queries such as obtaining all instances of a given subclass.
- ➔ **Realisation:** This helps to locate the most specific class that an individual belongs to. In principle, this can only be performed after classification.



Figure 2.5: Self-Service Model

2.1.5.3 Selection Criteria for a Reasoner

The factors that need to be considered for selecting an ideal reasoner have already been discussed. To reinforce what was discussed, a list of quality measures will be provided below. This list can effectively be used as criteria for selecting an appropriate reasoner to use. According to Greiner [56], there are four important measures that can be used to guide selection and these are outlined below:

1. *Correctness*: The reasoner returns the correct answer, preferably all the time. This is not possible but it remains important that the probability of providing the correct answer is high.
2. *Precision*: The reasoner returns the most specific answer from a possible list of inferences that it sanctions.
3. *Expressivity*: This refers to how well the representation formalism can express any possible piece of information without diminishing the ability for a reasoner to answer any possible question presented to it. In a way, this substantiates Davis *et al.* [41] assertion that knowledge representation and reasoning are inextricably entwined.
4. *Efficiency*: This quality measure is linked directly to the performance of the reasoner. Essentially, for a reasoner to be efficient, it must return its answers as fast as possible.

2.2 Human-Machine Dialog Systems

The practice of self-service is recognised by many sectors of the economy as beneficial. This practice operates on a simple model, shown in Figure 2.5. As shown in the figure, users send queries to self-service systems and after due processing, answers are returned.

The query and answer interactions embodied in the depicted self-service model represent human to machine communication. Thus, systems that emulate this type of communication are sometimes aptly called human-machine dialog systems. This section will review

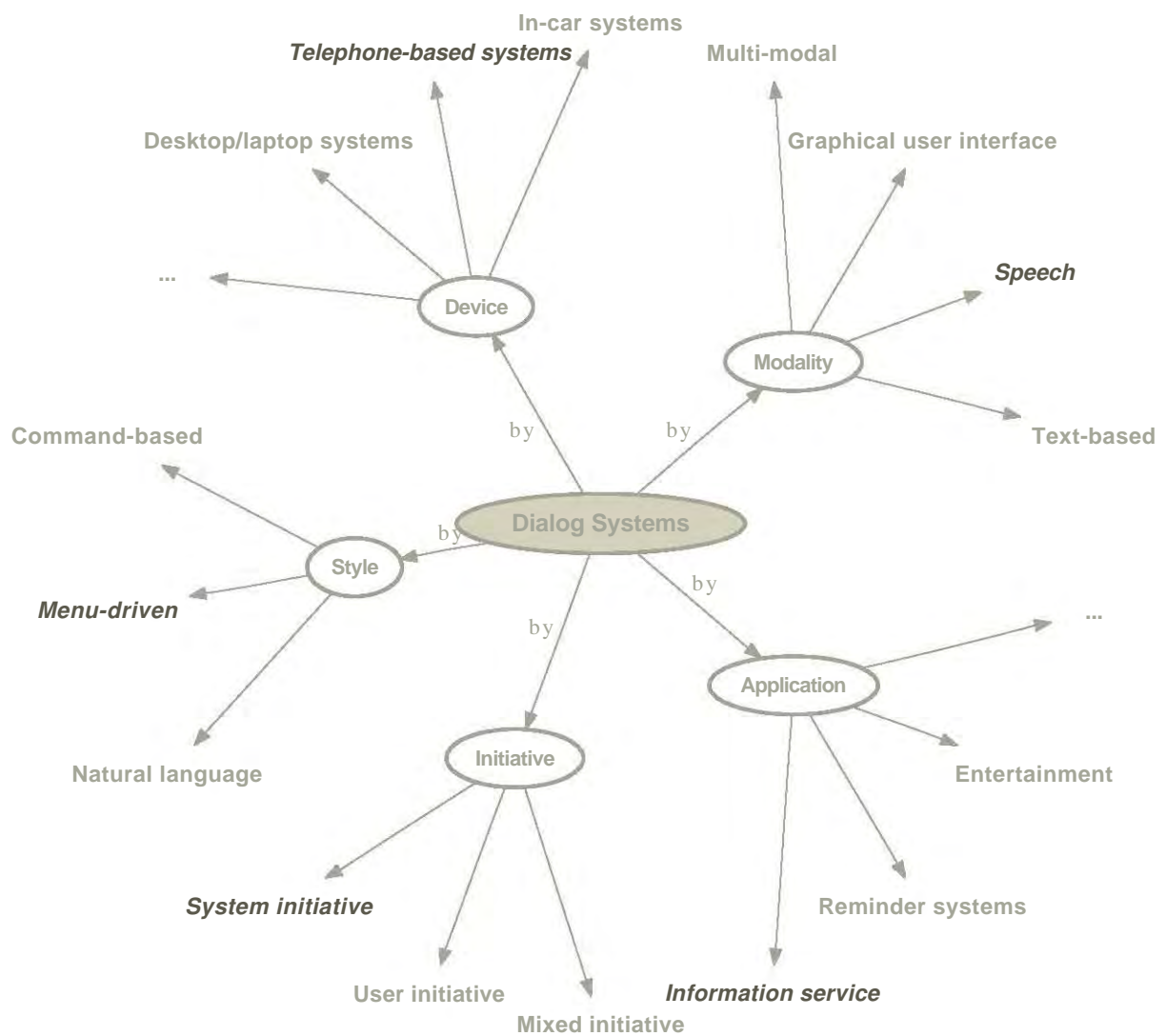


Figure 2.6: Classification of Dialog Systems

these types of systems and their architectures. However, primary focus will be on systems that use the acoustic interface for their interaction.

2.2.1 Types of Dialog Systems

There are a number of ways to categorise dialog systems. Figure 2.6 aggregates together some of the possible ways to make the classification. As it can be seen from the figure, categorisation can be done by device, modality, application, initiative, and style. However, as implied, for example, in [?, 97], there are many other dimensions or perspectives that could be used for the categorisation. Overall, as reflected from various literature sources, these perspectives coexist and overlap with each other. This means, for example, that

telephone based dialog systems can be seen as a major category as well as a subcategory of the other categorisations, without this being seen as a contradiction. In the figure, the bolded dialog systems reflect an overlap in how the prototype resulting from this thesis may be categorised.

2.2.2 Dialog Management

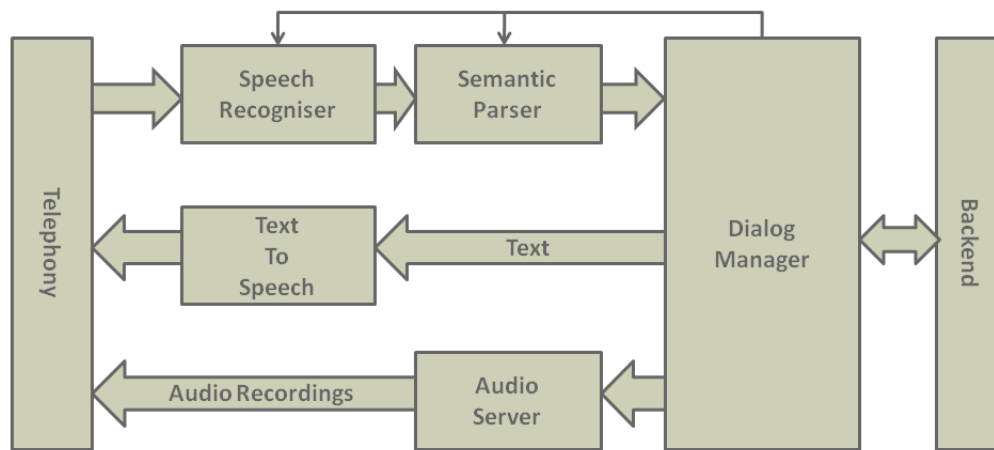
All dialog systems irrespective of categorisation require some dialog management mechanism. This mechanism is needed to ensure that interactions between the user and the system are seamless and efficient. A component aptly named dialog manager is used to implement this mechanism.

There are many definitions provided in literature for a dialog manager, but as noted by Pieraccini and Huerta [97], there is no consensus on the definition because “different systems [...] attribute different functions to it”. For purposes of this thesis, a dialog manager shall be defined as a component that is “responsible for controlling the human computer interaction by following predefined dialogue scenarios” [53]. This definition is sufficient because it is general and does not tie the dialog manager to specific functions. Nor does it embody a specific approach for carrying out dialog management.

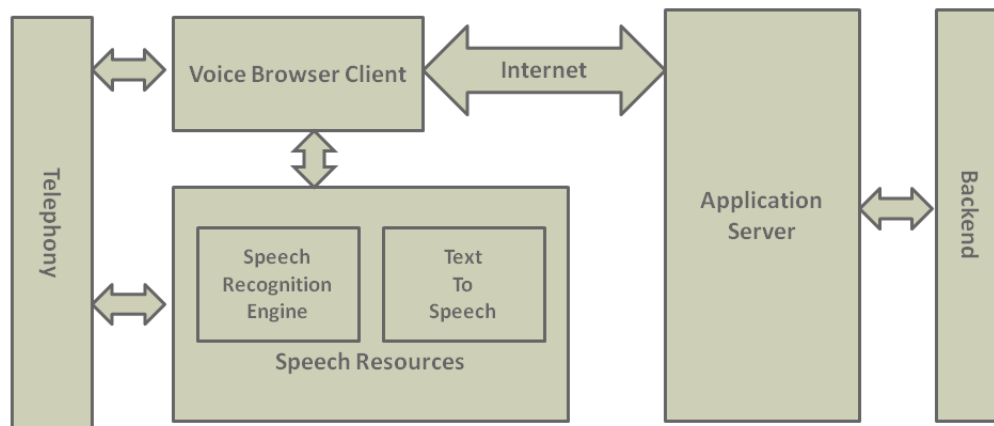
2.2.3 Dialog Systems Architectures

As already suggested, the focus of this thesis is on dialog systems that interact with users through the acoustic interface. Based on categorisation by modality, these systems are commonly called spoken dialog systems. This subsection will discuss in broad terms how these systems are architected. It should be noted, however, that the prototype system for this thesis qualifies as a subsystem of these systems, since users interact with the system using touch tone keys instead of voice.

There are many architectures that have been proposed and implemented for spoken dialog systems. These architectures have varying number of components arranged in different configurations. As a result, there are multiple ways in which these architectures can be classified. For example, Pieraccini and Huerta [97] provide what may be viewed as an oversimplified classification. According to them, these architectures may be classified based on their origin: as either from the commercial or research sectors.



(a) Typical Architecture for Research Prototypes (adapted from [97])



(b) Typical Architecture of Commercial Dialog Systems (from [97])

Figure 2.7: Reference Architectures for Dialog Systems

Figure 2.7a and Figure 2.7b depict typical architectures used to build research prototypes and commercial systems, respectively. As seen from the figures, there are some similarities and differences. In both architectures, input (e.g. speech) is received via the telephony platform and after due processing, requests for external resources may be made to the backend. The differences may be attributed mainly to what lies and happens between the telephony platform and the backend of these two types of architectures.

With architectures designed by the research community, the input goes via the speech recognition component. The recognition results are relayed to the semantic parser, which attempts to provide semantic understanding of what was said. After this, the dialog manager is contacted so that it can decide on the action to take. The action includes sending text the Text-to-Speech (TTS) engine, instructing the server to play an audio recording, or sending queries to the backend.

However, with commercial system architectures, the input is dispatched to the voice browser, which accepts documents written in a markup language for voice applications such as VoiceXML (which will be discussed in the next section). The browser may interact with the application server or gain access to speech resources, such as speech recognition or TTS engines. The determination is made based on what is embodied by the markup document.

An important observation to make from the comparison is that commercial architectures put little emphasis on semantic understanding. As Pieraccini and Huerta [97] explain, this is because commercial and research sectors have different goals and agendas, which lead to distinct implementation paths. The commercial sector follows a pragmatic path that prioritises usability and task completion. The research sector follows an idealistic path that aims for unconstrained natural language interactions. While it may seem like a contradiction, these authors also believe a ‘synergistic convergence’ between these paths is possible. The core of their argument hinges on the use of inference, which is very important for this thesis.

2.3 VoiceXML

As advances in fields like AI bring closer an era where users can verbally query systems using natural language without vocabulary restrictions, many organisations are increasingly moving towards voice self-service. Although this move is largely due to advances in technology, it does not mean that organisations were ignorant of the value of voice as a medium of interaction in implementing self-service.

According to Schoeller [104], before the introduction of VoiceXML, the architectural limitations of voice self-service acted as a barrier to broader adoption. In his opinion, “the movement to VoiceXML has revolutionised the voice self-service market” [104]. To understand this revolution, this section will provide an overview of VoiceXML and its benefits. Additionally, the section will discuss the architectural model of VoiceXML.

2.3.1 What is VoiceXML?

VoiceXML is a markup language for creating distributed voice applications [83]. It was designed to support creation of audio dialogs that feature synthesised speech, digitised

	Graphical User Interface	Voice User Interface
Language	HTML	VoiceXML
Browser Output	Markup formatted text and images	Markup formatted streaming audio, synthesised text or prerecorded audio
User Input	Keyboard or mouse	Touch tone keypad or voice
Field Navigation on a Form	Keyboard or mouse	Touch tone keypad or voice
Data Persistence	Screen display	User's short-term memory
Error Handling	Screen error message using reprompt	Audio error message using reprompt with cascading

Table 2.4: VUI vs. GUI Development

audio, recognition of spoken and touch tone input, recording of spoken input, telephony, and mixed initiative conversations [50].

VoiceXML was designed to be similar to HTML in order to blur the difference between Graphical User Interface (GUI) and Voice User Interface (VUI) development. Table 2.4 obtained from [104], provides a comparison between these two types of development. As it can be seen from the table, the two differ because of inherent differences between visual and voice interactions. Otherwise they are similar. This can be gleaned from the listed 'comparative criteria' in the first column of the table, which show, for example, that both need a browser and both use a concept of a form for accepting user input. Overall, the similarities align voice self-service application development with the web. As stated in [50, 75, 83, 104], this alignment brings about the following benefits:

- ➡ It allows developers to leverage on their Web development skills. Therefore, development can be done more easily.
- ➡ It allows application logic to be separated from the voice interface. This offers two main advantages. Firstly, there is the opportunity to bring the advantages of using design patterns such as Model-View-Controller (MVC) to the development of voice applications. (Although the MVC pattern does have drawbacks, it brings, for example, the ability to have code that is easier to port and reuse.) Secondly, organisations can outsource the VUI design and hosting while maintaining full control of the application logic.
- ➡ It enables integration of voice services with backend infrastructure (e.g. databases), using the familiar client/server paradigm.

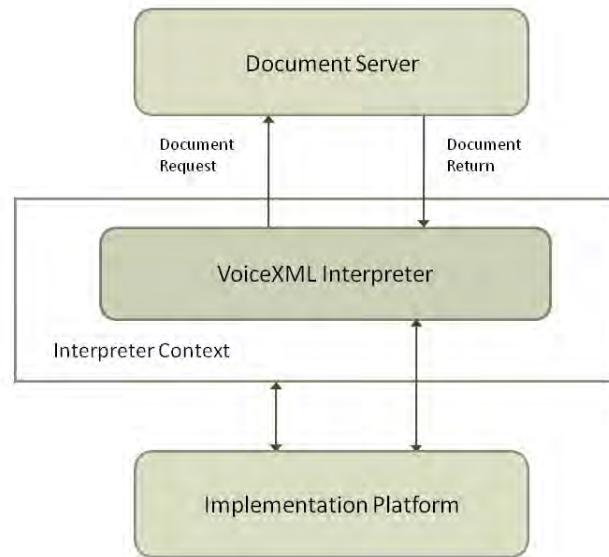


Figure 2.8: VoiceXML Architecture (from [50])

2.3.2 VoiceXML Architectural Model

The architectural model of VoiceXML, as shown in Figure 2.8, has three main components: 1) a document server (e.g. Web server), 2) an interpreter and associated interpreter context, and 3) an implementation platform. These map directly to components shown in Figure 2.7b. The document server maps to the application server. The VoiceXML interpreter and the associated context map to the Voice browser client. Finally, the implementation platform maps jointly to the telephony and speech resources in that it incorporates the both components in it.

As shown in the VoiceXML architectural model, the interpreter and the associated interpreter context constitute the heart of the architecture; as both play a critical role in managing all interactions between the document server and the implementation platform.

To capture the interactions within the architecture, a brief summary (from [50]) of the functions of the document server and the implementation platform is provided below.

- ➡ The document server processes requests made to it through the interpreter context and returns VoiceXML documents in response. The returned documents are processed by the VoiceXML interpreter.

- The implementation platform generates events under the control of the interpreter and its context. These events may be generated in response to user actions or events within the system.

2.4 Information and Communication Technologies in Health

In modern health care, the move “to equalise relationships between health professionals and lay people is gathering momentum” [38]. This move is due to the realisation that a shift from paternalism to partnership can increase the overall quality of health care [38]. This is underpinned by the principle that partners work together and share responsibilities.

As part of this shift, more emphasis has been placed on ensuring that lay people, just like health professionals, gain access to health information and knowledge. In turn, this emphasis has resulted in increased use of Information and Communication Technologies (ICTs) for processing and delivering health content to lay people [45, 115].

This section will explore this use from two perspectives. The first perspective will focus on the use of knowledge management and representation techniques. In theory, there are numerous areas of interest that could be explored under this perspective. However, only one area will be explored, namely, Healthcare Knowledge Management (HKM), which is a discipline that has emerged from knowledge management. Unlike with general knowledge management, HKM focuses specifically on the effective management of health information and knowledge to enable high quality and well informed decisions by all stakeholders [23]. The second perspective will focus on the experiences of lay people in using IVR systems to pursue health related goals. The objective is to demonstrate the potential for using the built IVR system for locating information about HIV testing facilities.

2.4.1 Healthcare Knowledge Management

HKM provides an approach for designing and deploying knowledge-driven services that demonstrate the value of knowledge as a key resource in decision making as well as a ‘service’ [23]. As a service, HKM caters to the needs of different stakeholders in a manner that allows the gap between knowledge and practice to be minimised. The reason, as aptly expressed by Abidi [23], is that HKM services provide a higher level of abstraction than

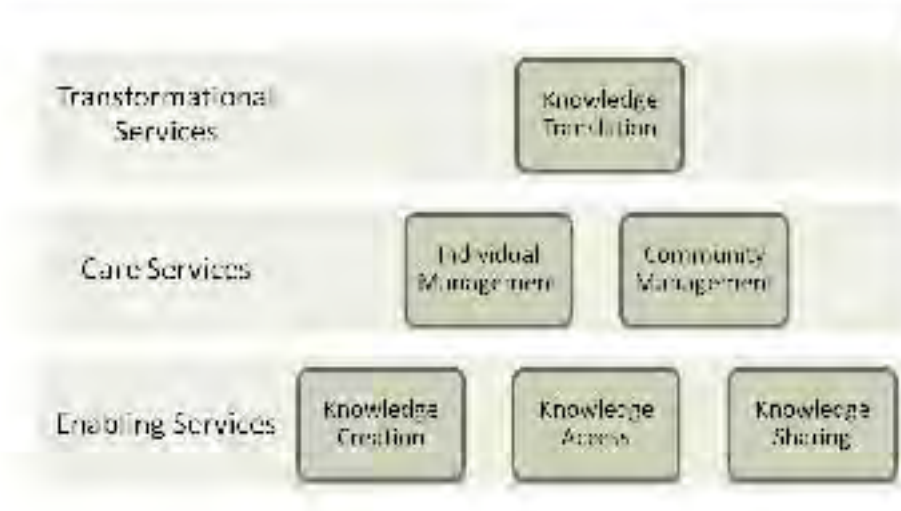


Figure 2.9: Pyramid of HKM services

data driven services. As such, they provide “semantic-validity and pragmatic-viability” in generating an overall service that is capable of improving health outcomes.

Figure 2.9, presents a pyramid of HKM services to capture this abstraction. As shown in the figure, HKM services can be categorised in three groups: 1) enabling, 2) care and 3) transformational services. In this thesis, focus will be on the enabling services. The enabling services provide the foundation for developing care and transformational services, which focus largely on the core objectives of public health and individual patient care.

The enabling services are constituted by three types of service activities: knowledge creation, knowledge access and knowledge sharing. As noted in [40], Semantic Web technologies, especially ontologies, support these activities in ways that promote synergies between them to be exploited for building sound knowledge driven services. This is reinforced by Abidi [23] who states that: ontologies “offer a semantically rich and executable knowledge representation formalism” that is highly practical for creating health oriented applications.

Overall, this subsection has highlighted one important point for this thesis. To translate information (from information rich disciplines like health)² into knowledge that can be shared, the foundation lies with use of knowledge representation formalisms. Although not explicitly stated, these formalisms contribute to the discipline of knowledge management: precisely because they act as the foundation for enabling service creation (Figure 2.9 captures this idea for health oriented services).

²A lot of information and/or knowledge is generated in the health sector as a result of active research on how to better treat and manage health related problems.

2.4.2 IVR in Health

IVR systems are pervasive in many sectors of the economy including the health sector. In health, Lee, Friedman, Cukor and Ahern [80] suggest that use of IVR systems can provide many benefits that include increased access to health care, reduced operating costs and increased staff efficacy. However, as they state, attaining these benefits is dependent on designing quality IVR systems.

In a study that sought to determine the feasibility and acceptability of using an IVR system for facilitating peer support among elderly diabetes patients, 79% of the participants stated that the IVR was easy to use [66]. This is one of the many examples that confirms the statement made in the previous chapter, that use of IVR systems in health can yield satisfactory outcomes. Other studies that can be used to substantiate this assertion include use of IVRs for chronic pain management as in [91], and their use as reminders for individuals to get preventative services such as mammograms, pap tests and influenza immunisations [39].

In a study very pertinent to this thesis by Frank, Wandell, Headings, Conant, Woody and Michel (as cited in [36]), 85% of participants chose to use an IVR session for HIV counselling instead of opting to talk to a counsellor over the telephone. While the result is impressive, it is very untypical for IVR applications. (As suggested before, typically people tend to opt for a human rather than go through an IVR session.)

There are possibly many reasons to explain the anomaly in Frank *et al.* study, such as stigma attached to HIV and AIDS. It can be argued that these reasons still apply, even though the study was done a few years ago (in another country). This is because stigma, for example, continues to be a problem in South Africa ³ [109]. The use of an IVR system to locate information on HIV testing facilities may help to encourage testing. Primarily, because an IVR dialog, affords individuals ‘uninhibited’ interactions that may propel them to actually go for testing, should they find a facility that meets their individual needs.

³The problem persists for a number of reasons that include poor leadership and media campaigns that have uncunningly exacerbated stigmatisation through *discourses* that reinforce views that only certain segments of the community face the risk of infection. For example, due to the use of BMW Z3 sports car in some media campaigns, “HIV infection is seen by ruralites as a modern disease ‘of the town’” [109].

2.5 Summary

This chapter focused on the core ‘pillars’ which constitute the foundation of the work carried out in this thesis. VoiceXML was identified as one of the underpinning pillars. For this reason, there was a discussion on VoiceXML which outlined some of its benefits as a standard language for creating audio dialogs. As stated by Schoeller [104], “standardisation on VoiceXML [has allowed] for closer alignment of content and business logic with web applications”. This alignment has revolutionised the voice self-service market and has consequently positioned VoiceXML as the technology standard to use now and in the future.

Arguably, one of the most important pillars under discussion was ontologies. Ontologies in computing can be defined in a number of ways, as indicated in Subsubsection 2.1.4.1. Without providing a definition, ontologies can simply be regarded as forms of knowledge representation. This means that they can be understood in terms of the five distinct roles played by knowledge representation, as discussed in Subsubsection 2.1.3.1. These roles as articulated by Davis et al. [41], capture the ‘spirit’ of representation and that of reasoning, since the two are intertwined. Although, as they admit, the roles may “create multiple, sometimes competing demands, requiring selective and intelligent trade offs” to be made on how to view and reason about the world under representation.

To deal effectively with some of these trade offs, Gruber [58] has proposed five design criteria, discussed in Subsubsection 2.1.4.6, to guide the evaluation of ontologies. Although the criteria are general, they were intended specifically for the design of ontologies used for knowledge sharing. In part, this is because ontologies are believed to be more suited for use in knowledge sharing; since they provide a shared and common understanding of a domain that can be communicated among people and machines [43, 46, 47].

In lieu of focusing only on the domain of the knowledge to be represented, this chapter discussed the domain of health as one of the pillars. However, because health is a knowledge rich discipline, the scope was purposefully confined to areas that capture the use of ICTs in health as they relate to this thesis in Section 2.4 on page 27.

The other pillar, which was also discussed in a purposeful manner pertained to human-machine dialog systems in Section 2.2 on page 20. The discussion centred on spoken dialog systems and their architectures. This helped to provide an overall context for understanding the work carried out in this thesis.

Chapter 3

Software Components

In the past, dialog systems were commonly built using proprietary architectures based on IVR platforms [97]. As documented widely in literature, the biggest problem with proprietary architectures is that they limit interoperability; and this impacts negatively on service creation, deployment, and general use. These problems (as also stated in the literature) can be ameliorated through standardisation.

OVR has been designed to take advantage of standards and to be in alignment with popular trends. One such trend is component reuse, which advocates for development of applications through simple or sophisticated building blocks. This chapter will give some background on the building ‘*blocks*’ for assembling the architecture. Additionally, it will provide an insight into how selection was done for components with multiple competing offerings. It will start with an overview of the information flow within the architecture, using as organising element the main use scenario: a user who calls to get service provider details.

Figure 3.1 provides a categorisation that will be used to present these components. As a deduction from the figure, there will be two main sections. One for components related to application development and the other essential components for assembling the deployment environment. There is a subcategorisation that can be made for components used



Figure 3.1: Categorisation for Utilised Components

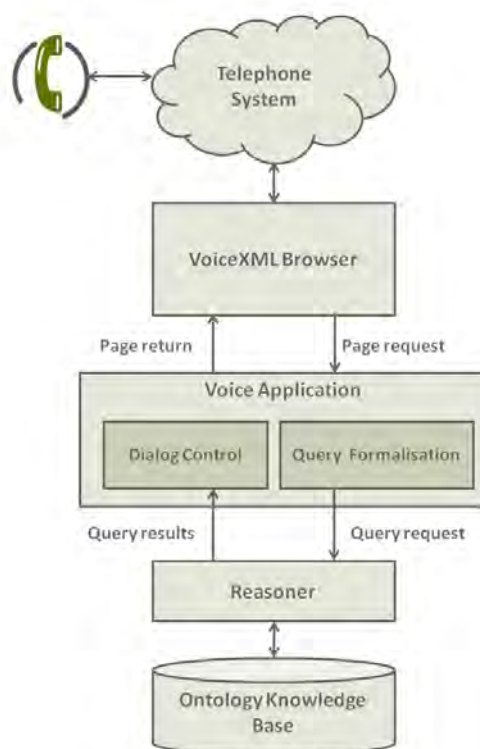


Figure 3.2: Overall Architecture

in development; they can be classified in terms of the role they play at runtime. However, in the presentation structure, this subcategorisation shall not be made explicit.

3.1 Information Flow

As a means of providing clarity on how the various components of the architecture interact with each other, an example of use is provided below. The example illustrates how a user's request for information may be handled within the architecture. Figure 3.2 will be used to aid the discussion. (This figure is the same as Figure 1.1 on page 4.)

- ➡ A user places a call to request information.
- ➡ The telephone system handles the processing details of the call. These may involve interactions with the telephone network (PSTN). After processing, the call is forwarded to the browser.
- ➡ The browser detects an incoming call and triggers events that cause the voice application to answer the call. These events can be viewed as examples of background

interactions between the browser and the telephone system.

- The voice application answers the call, interacts with the user and ultimately hangs the call. The interactions with the user are dependent on the type of service being offered by the voice application and user's actions. These interactions may result in:
 - The VoiceXML browser sending requests to the telephone system to send prompts, messages, requested information or audio material to the user. Alternatively, they may involve sending requests that cause the telephone system to accept user input and process it accordingly.
 - The voice application sending requests to the ontology knowledge base for information retrieval via the reasoner. These requests are handled by a query processing component designed to facilitate querying of the knowledge base.
 - The voice application sending requests to the rule engine to process dialogs that need to be generated as part of user's response. For example, given that presentation of fewer items has merits in terms of memorability in acoustic interfaces, the rule engine may issue a 'trim list' dialog if the list to be presented to the user is long.

3.2 Main Features of the Architecture

First, the architecture uses an ontology knowledge base instead of a traditional database. This brings in significant power to the architecture albeit at a potential increase in performance cost. The details will be explored in the next chapters. However, a notable contribution to this power lies in the fact that a reasoner is used, specifically Pellet [105]. Pellet is a popular DL reasoner that supports the variant of Web Ontology Language (OWL) based on Description Logic. OWLAPI ("a Java interface and implementation for World Wide Web Consortium (W3C) Web Ontology Language (OWL)" [17]) was used to build a customised query interface that facilitates access to the reasoner for querying the ontology knowledge base.

Second, the architecture has been designed to take advantage of open standards. Although Asterisk allows creation of IVR dialogs using dial plans or Asterisk Gateway Interface (AGI), the process is ad hoc. In the architecture, VoiceXML browser is included because VoiceXML is used to generate dialogs. VoiceXML is a W3C dialog markup language for

creating standard based voice applications [83]. The move to use standards, particularly aligned to the web, increases flexibility of the architecture. Further, it provides enhanced value proposition for created services.

Although not depicted, another notable feature is that the architecture has made an effort to combine two forms of representations. That is, in addition to ontologies, the architecture also uses rules to represent knowledge for driving dialogs that the system has with users. As a representation formalism, rules allow knowledge to be represented in the form of ‘*if-then*’ constructs, i.e. if this happens, then take the following action [101, 57]. In this thesis, the use of rules is two fold. First, they have been used as part of the implemented ontology to enhance reasoning over the knowledge base. Secondly, they have been used in the context of rule based programming to implement the voice application. Under this context, rules were used to increase the level of abstraction for the overall system and to bring emphasis on the importance of separation of concerns: unlike in the first instance where their use was primarily intended to support intelligent reasoning.

3.3 Components for Application Development

There are several components (tools and/or resources) that were vital for realising the goals for this thesis. This subsection will focus on those that were important for building the prototype application service. In some instances, it will be asserted that a component can be used as a third party library. What this means is that the component is required as a resource at runtime. Thus, it will be bundled in with the application.

3.3.1 Ontology Authoring Tool

The next chapter will discuss the process for building ontologies. As it will be seen, this process can be lengthy. Thus, having a tool to support authoring can be beneficial. *Protégé* was selected because of its reputation as a leading ontological engineering tool. Other reasons for selection have been articulated in the next chapter (4.2.3.2 on page 65).

Protégé can be downloaded from its website, [13]. There is an option to choose between a platform independent installer or a ZIP file that excludes Java Virtual Machine (JVM). In either case, installation is straight forward and documentation is available to clarify

any problems that may arise. Once installed, *Protégé* is ready for use; an ontology may be created and rendered in a number of formats, like XML as shown in Appendix B on page 119.

To install plugin extensions for *Protégé* to include, for example, reasoning services, there might be a need to set additional JVM parameters. *Protégé* (version 4 in particular) implements an automatic update mechanism that allows plugins to be downloaded and updated. This mechanism assumes that the user has direct Internet connection. Thus, if connection is through a proxy, the user has to provide proxy credentials. This can be done by adding the following parameters to the contents of the appropriate run file (*run.sh* for Linux or *run.bat* for Windows) found in *Protégé* home directory:

```
-Dhttp.proxySet=true -Dhttp.proxyHost=proxy_name -Dhttp.proxyPort=proxy_port
```

3.3.2 Ontology APIs

Application Programming Interfaces (APIs) are important for development of any software artefact. This is a well known and accepted fact that also holds true for ontology APIs, which aid with building ontology based applications. *OWL API* [68, 69], a high level Java API for working with OWL 2 ontologies was selected for use in this thesis. The alternative would have been to use *Jena* [74], a Java API designed for Resource Description Framework (RDF). *Jena* does provide support for OWL, but at a low level. It was deemed prudent to opt for a high level API in order to simplify the development process.

OWL API can be downloaded from SourceForge (an online repository for open source code), using the following URL [17]. Once downloaded, the API can be treated as a third party library. Before its use, its location must be added to the build path of the development project. This process varies depending on whether an integrated development environment (IDE) is used or not. And when an IDE is used, the process may vary depending on the chosen IDE. (The actual details are not pertinent for inclusion in this thesis.)

3.3.3 Reasoning Engine

The need for a reasoning engine (reasoner) has already been discussed in the previous chapter. *Pellet* reasoner was selected as the main reasoning engine for making inferences

on the domain knowledge, justification will be provided in the next chapter. The objective in this subsection is to provide briefly the details of how to incorporate a reasoning engine within an application. This can be done by first downloading *Pellet* from its website, [12]. After downloading, it can be used as a third party library (as discussed above) by specifying its location in the build path.

3.3.4 Rule Engine

A number of rule engines were surveyed: *Hammurapi* [5], *Jess* [8], *JLisa* [9], *SweetRules* [14], and *Drools (JBoss Rules)* [33]. *Pellet* reasoner was also considered for the dialog rules. It was eliminated in favour of using a more specialised engine for the task and (to a small extent) for purposes of demonstrating that multiple trend aligned technologies can be used together to provide a rich service.

Other candidate rule engines were eliminated using the following criteria: alignment with Java specification request for a Java based rule engine (JSR 94); latest release date (relevant for establishing status of maintenance); and evidence of community support. From the list of rule engines that remained, *Drools* was selected since it showed evidence of regular maintenance and strong support from its community of developers.

Drools, similar to *Pellet*, can be used as a third party library. It can be downloaded from its website, using the following URL [4].

3.4 Components for the Deployment Environment

The deployment environment for the prototype service is constituted by components that are essential for the core functionality of VoiceXML, see Figure 3.3. These components will be reviewed in this section. The choice for most of these components is rooted in the decision to use iLanga [96], a telephony platform developed at Rhodes University that comprises of three main components depicted in Figure 3.4.

There were a number of practical and theoretical reasons for using iLanga. Availability was one of the key reasons. As an already functional platform, iLanga provided a convenient telephony gateway (through use of *Asterisk*) for allowing users inside and outside campus access to the prototype service. Further, expertise was available to support the deployment of the service.

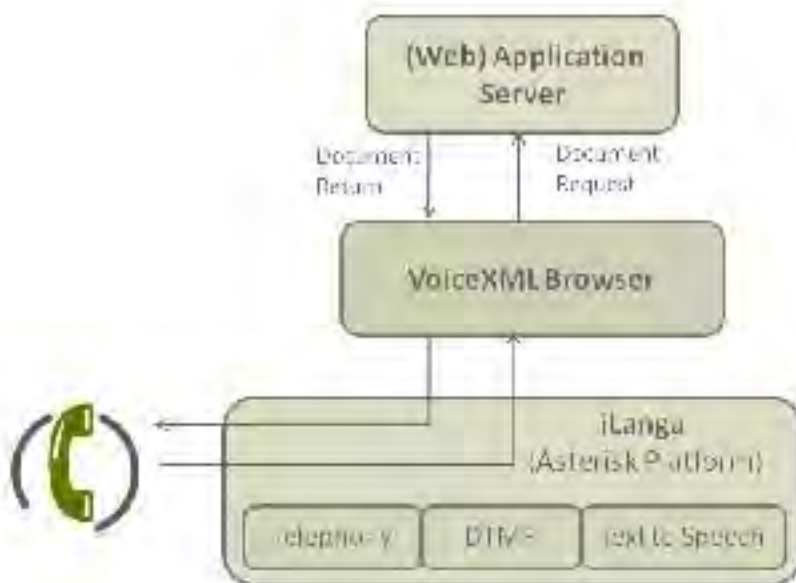


Figure 3.3: Components to Support Core VoiceXML Functionality

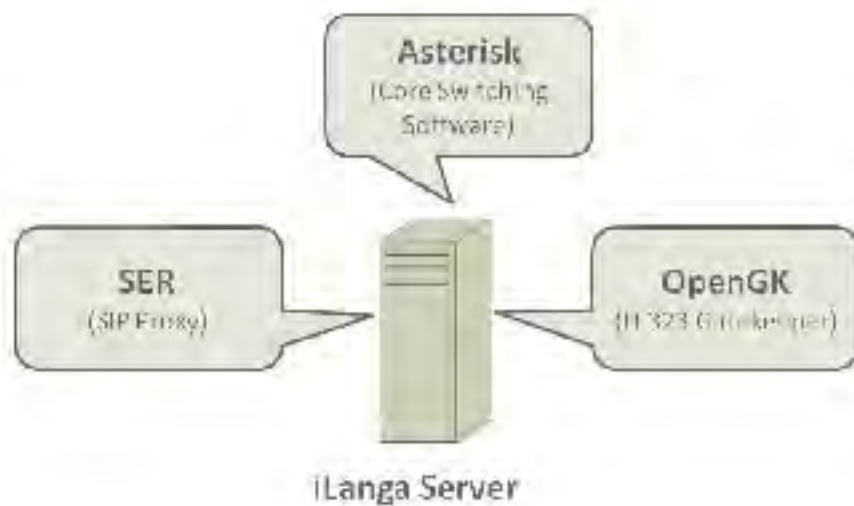


Figure 3.4: iLanga Main Components (from [96])

3.4.1 Application Servers

Tomcat [1], a popular open source server designed specifically to support Java servlets and Java Server Pages was selected for use. It was used in conjunction with *Apache* [16] web server. While indeed there were many application servers to choose from, no effort was made to review other existing alternatives. It was sufficient that *Tomcat* could support the goals for this thesis and that its sterling reputation could be leveraged to make its installation and configuration easy (enough to not warrant a discussion).

3.4.2 Asterisk

Asterisk is a very popular open source platform for creating communications solutions. As shown in Figure 3.4, *Asterisk* is one of the main components of iLanga. Thus, the decision to use iLanga came with the implicit use of *Asterisk*. The VoiceXML browser was tested with *Asterisk* 1.4.22. The package repository had a lower version; this meant installation in both the test and iLanga servers had to be done by compiling *Asterisk* from the source code. The process itself is fairly straight forward. It involved downloading the source from [2], running the configure script inside the source folder (after decompressing) and following the prompts.

After installation, one of the primary configurations to be made in *Asterisk* pertains to creating what is termed a dialplan. A dialplan consists of a set of instructions that tell *Asterisk* what to do [118]. These instructions are typically specified in *extensions.conf* file.

3.4.3 VoiceXML Browser

A VoiceXML browser by i6Net called *VXI** [21] was selected for use. This browser is specifically designed to work on the *Asterisk* platform and integrates easily with a number of widely used TTS and Speech Recognition (ASR) applications. Its implementation is based on *OpenVXI*, an old portable open source VoiceXML interpreter.

Prior to this work, an initial experimentation with integrating *OpenVXI* with iLanga was done and is reported in [75, 76]. This integration effort was considered; however, since the various components that were used to create a VoiceXML browser running on iLanga needed upgrading, a decision was made to explore other alternatives. This is

because the upgrading task also warranted a reimplementaion of the browser. In seeking alternatives, *VXI** was practically the only VoiceXML browser available for the *Asterisk* platform. Although this influenced the decision for its selection, the primary reason was that it demonstrated the potential to provide easy integration with iLanga.

The manual for *VXI** [21] provides a step by step guide on the installation and configuration process. Once *Asterisk* is installed, *VXI** can be installed by downloading the VoiceXML browser from i6Net's website at [20]. The downloaded package comes with an install script to automate the process. After installation, the commercial license may be activated to increase the number of simultaneous VoiceXML sessions.

As a next step, the VoiceXML interpreter and the *Asterisk* module need to be configured. The various parameters for their configuration are explained in the manual. The configuration of the interpreter is done through a text file (*/etc/openvxi/client.cfg*) that passes specified parameters to *OpenVXI*. To assist with configuration, the file includes a comment block that contains details of possible property names with example values for various components and settings.

The *Asterisk* application module is used by the browser to execute VoiceXML pages. Parameters for its configuration can be set inside the file, */etc/asterisk/vxml.conf*. These parameters include specifying default timeout values for DTMF recognition, codecs to be used and whether or not a VoiceXML application should automatically answer the *Asterisk* channel before starting a session.

Once configured, the *Asterisk* command line interface (CLI) can be used to interact with the application module to seek help or issue management commands like *vxml debug*, which enables VoiceXML debugging. To execute a VoiceXML session for any created voice application (say, with a root page *index.vxml*), a minimal dialplan like the one shown below is required. This could be added as a context within the *extensions.conf* file (or any included file).

```
exten => 448,1,Answer()  
exten => 448,n,Wait(3)  
exten => 448,n,Vxml(http://serverURL/voiceApp/index.vxml)  
exten => 448,n,Hangup ()
```

3.4.4 Text-to-Speech

As aptly captured by the name, a TTS converts text into speech. *VXI** integrates an HTTP client interface for communicating with a TTS server. As specified in [21], the advantage of using HTTP as opposed to MRCP (Media Resource Control Protocol) interface is that any generated speech can be cached by the VoiceXML browser.

A number of TTS engines are available both commercially and as freeware. However, as specified in [15], only the following are supported by *VXI**: *Acapela*, *Cepstral*, *eSpeak*, *Festival Lite (Flite)*, *Loquendo*, *Nuance*, *Verbio*, *Voiceinteraction* and *Yantra Software*. Using the principle that ‘free software is preferable to proprietary one’, all commercial TTS engines were eliminated from the list. This left *eSpeak* and *Flite* on the list of candidate engines to use. *Flite* was selected simply because configuration details for making it work with *VXI** were included in the manual [21].

Flite can be installed in a number ways. However, the easiest way is directly from the package repository. For a Debian distribution the command to use is:

```
sudo apt-get install flite
```

To use *Flite* a PHP script is provided to facilitate the conversion. This script needs to be placed in the designated TTS server (in this case it was *Apache* server). After its placement, the VoiceXML browser needs to be configured to be aware of the location (within the *OpenVXI* client configuration file). Once this is done, the browser will be able to execute the script, which in turn, will invoke *Flite* whenever a conversion is required.

3.5 Summary

OVR is constituted by several software components. This chapter provided a background on the key components. Further, through a brief discussion on the installation process, it was revealed how these components were selected and integrated together. This integration was paramount for the eventual deployment of services created for the architecture. The next chapter will provide implementation details of a prototype service built to validate the integration effort.

Chapter 4

Ontology Construction for Developing the Prototype System

There are many readily available ontologies that could have been used for testing the OVR. However, in order to gain more understanding about ontologies, the option to build an ontology from *scratch* was favoured. There was also another reason, at least during the early days of the project: to deploy a service within a specified domain that would benefit from a semantically rich representation. (However, as already explained, this was hampered by challenges in working collaboratively in a diverse team.)

It is important to note that building ontologies from scratch does not preclude use of other ontologies or resources; this is encouraged whenever possible. Hence, the word *scratch* is used in a qualified manner. This chapter will discuss the construction from scratch of the HIV and AIDS service provider ontology. This discussion will be preceded by sections covering theory for contextualising decisions that were made in building the ontology.

4.1 Ontology Development Life Cycle

In order to engage in the process of building any ontology, it is imperative to understand the various stages involved from start to end. These stages constitute what is called ontology development life cycle. In the literature, the commonly accepted stages of this cycle are specification, conceptualisation, formalisation, implementation and maintenance [49, 63, 98]. Figure 4.1 shows how these stages generally interact and feed into each other.

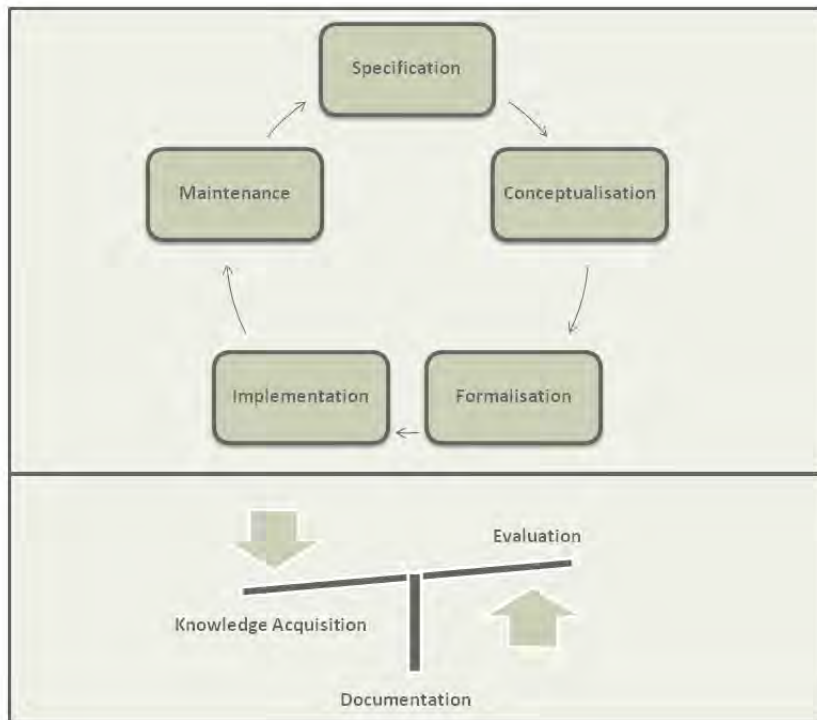


Figure 4.1: Ontology Development Life Cycle

Each stage of the ontology development life cycle is characterised by a set of activities (which will be discussed in Section 4.2). These activities are performed in conjunction with support activities namely: knowledge acquisition, evaluation and documentation.

It is important to emphasise that the support activities are carried throughout the whole ontology development life cycle. However, the degree to which knowledge acquisition and evaluation activities are performed varies at each stage. As shown in the lower portion of Figure 4.1, there is in fact a delicate balance between these two activities and documentation plays a pivotal role in supporting them. That is, documentation operates as a ‘pivot’ that helps to bring clarity at every stage on the *weighing of decisions*¹ that, for example, reflect how evaluation might have impacted on knowledge acquisition, or vice versa.

Another point to emphasise is that the ontology development life cycle models are comparable, in two important ways, to life cycle models in Software Engineering. Firstly, they identify stages which lead to the final product. Secondly, they provide a framework for defining methodologies that outline activities as well as guidelines on when to perform these activities in order to move from one stage to the next [49].

¹As noted in literature, weighing decisions to make informed trade offs is an integral part of designing and/or developing any artefact such as an ontology.

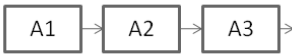
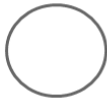
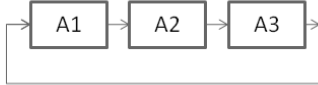



	Life Cycle	Final Product
Waterfall		
Iterative		
Evolving		

Table 4.1: Comparison of Life Cycles and their Final Product

4.1.1 Development Life Cycle and the Final Product

It has been suggested in the beginning of this section that understanding the development life cycle is important. A question that is yet to be answered directly, however, is: why?

A simple answer to this question is that a development life cycle provides a structured approach, which guides development. This helps to produce a final product of good quality in a timely and cost-effective manner. This justification was conveyed by Royce [103] in an article that provided the first formal description of the development life cycle model, now commonly known as the waterfall model. (In this article, the gist of Royce's argument was that the 'waterfall model' - a term he did not use - provided a "fundamentally sound" approach to development, albeit with risks that needed to be mitigated.)

In the waterfall model, each stage of the development proceeds in a strict order and the next stage cannot be initiated unless the preceding stage has been completed and perfected. Due to this strictness, the waterfall model has been criticised for its inability to allow refinements and mitigate against risk. To address this criticism, there are a number of adaptations that have been proposed for the waterfall model. Table 4.1 offers a comparison of the waterfall model together with two models adapted from it, namely: iterative² [25] and evolving prototyping³ [54] life cycle models. The waterfall life cycle

²Similar to incremental life cycle model. Hence, sometimes the terms are used interchangeably.

³Also called evolutionary.

model is a single build model whereas the iterative and the evolving life cycle models develop their final product in multiple builds.

A notable difference between these two multiple build life cycles is that with the evolving life cycle model, it is not imperative to have all requirements known upfront. This is reflected in the table by the haphazard looking final product that suggests a possible change in direction with every build milestone. In contrast to the iterative life cycle model, it can be seen that the final product grows in layers with each milestone build. This suggests that most of the requirements need to be established up front in order to progressively enhance a product from a previous build; otherwise it would be difficult to plan how to systematically build up the product to completion.

Single build development life models are usually not deemed appropriate for ontology construction. These models are ideal for non experimental settings and as alluded to, ontology construction is a non trivial process that requires several revisions to be made before producing the final product. Necessarily, this means the process for ontology construction is iterative and demands careful analysis, feedback and redesign throughout the entire life cycle [43, 49, 93, 98].

Notwithstanding the above, it is still important to know which life cycle model to use for any ontology development project. For this purpose, Suárez-Figueroa and Gómez-Pérez [106] propose a guideline for selection, which is captured as a decision tree shown in Figure 4.2. The decision tree includes another popularly recognised life cycle model, called the spiral life cycle model [29], shown in Figure 4.3.

The spiral model, shown in Figure 4.3, can be regarded as a generic model that “combines the iterative nature of prototyping with the controlled and systematic aspects of the waterfall model” [18]. An important feature of this model is that it explicitly focuses on minimising risk associated with each iteration of development. Hence, analysis of risk is done as soon as objectives for the iteration are determined and a solution that offers minimal risk is implemented and evaluated. The next iteration is planned and started to either resolve risk related problems or incrementally add features to the artefact produced from the implementation stage.

4.1.2 Development Life Cycle and the Methodologies

By virtue of creating structure, development life cycles provide a framework that underpins many methodologies used for the construction of any artefact. Methodologies

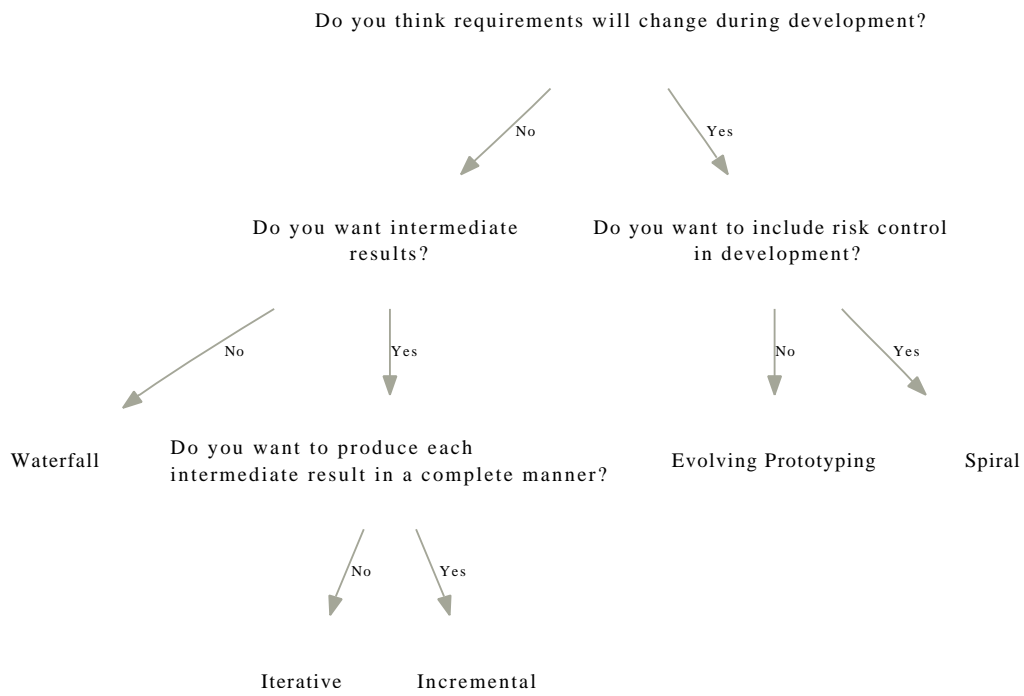


Figure 4.2: Decision Tree for Selecting Ontology Development Life Cycle (from [106])

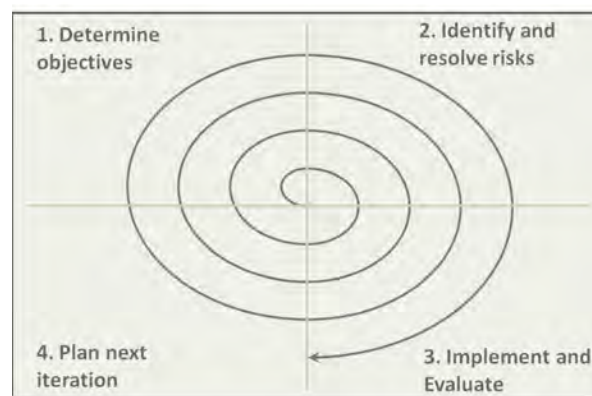


Figure 4.3: Spiral Model

identify activities that must be performed and to some extent their sequencing. While development life cycles determine when the identified activities should be carried out and the different stages that lead to completion [106].

To concretely show the relationship between development life cycles and methodologies, this subsection will outline activities defined in popular ontology methodologies. Further, it will discuss how these map to the overall stages of development life cycle as depicted in Figure 4.1.

The mapping and accompanying analysis will be guided largely by two sources: 1) Gómez-Pérez *et al.* [63] and 2) Pinto and Martins [98]. The reason is that these two sources provide an in depth analysis on methods and/or methodologies⁴ used to build ontologies from scratch. For consideration in this thesis, three methods and/or methodologies are discussed below.

4.1.2.1 Uschold and King's Method

As aptly reflected in the name, the method was proposed by Uschold and King [117] based on their experiences in developing the Enterprise ontology. It is due to this fact that sometimes the method is also referred to as the Enterprise methodology. A skeletal form of this methodology includes the following activities:

- ⇒ *Identify purpose and scope* to clarify why the ontology is being built and who the intended users are.
- ⇒ *Build the ontology* by breaking the activity into three tasks. These entail:
 1. Capturing knowledge by:
 - Identifying key concepts and relationships in the domain.
 - Producing precise, unambiguous text definitions for such concepts and relationships.
 - Identifying terms to refer to such concepts and relationships.
 - Agreeing on all of the above.
 2. Coding the knowledge using a formal language.

⁴There is a difference between terms *methods* and *methodologies*. However, in literature these words are often used interchangeably [63]. In this thesis, they will also be used interchangeably as dictated by quoted sources.

Enterprise Activity	Maps to
Identifying purpose and scope	Specification
Capturing knowledge	Knowledge Acquisition and Conceptualisation
Coding	Formalisation and Implementation
Evaluation	Evaluation
Documentation	Documentation

Table 4.2: Classification of Activities in Enterprise Methodology

3. Integrating knowledge from existing ontologies.

- ⇒ *Evaluate* the built ontology using Gómez-Pérez and Pazos [64] criteria.
- ⇒ *Document* all important assumptions to facilitate effective knowledge sharing.

Table 4.2 shows how the above outlined activities map to the ontology development life cycle stages. As it can be seen, knowledge acquisition and conceptualisation as well as formalisation and implementation are performed together. In the latter case, it is important to note that a formal language used for formalisation may not necessarily be the implementation language.

From the table, it can also be observed that maintenance has not been included. This is because the methodology lacks maintenance activities. As explained by Pinto and Martins [98], this is typical of first generation ontology construction methodologies. According to them, this can be attributed to the fact that these methodologies focused squarely on bringing understanding on how to build ontologies.

4.1.2.2 Grüninger and Fox's Methodology

The methodology by Grüninger and Fox [59] is based on the experience of the TOVE project; a project whose goal was to develop a set of ontologies for capturing knowledge that can be shared across enterprises. The methodology is also known as the TOVE methodology. Similar to the Enterprise methodology, the TOVE methodology is a first generation methodology.

As part of the process for building ontologies, this methodology stipulates use of First Order Logic (FOL) for formalisation and implementation stages. With regard to the whole process, the methodology recommends the following activities:

TOVE Activity	Maps to
Capture motivating scenarios and formulate informal competency questions	Specification
Specify terminology, formulate formal competency questions and specify axioms in FOL	Conceptualisation, Formalisation and Implementation
Evaluate competency and completeness	Evaluation

Table 4.3: Classification of Activities in Grüninger and Fox's Methodology

- ⇒ *Identify motivating scenarios* to capture the problems to be addressed and intuitively generate a set of possible solutions to these problems. These scenarios are typically presented in a story format.
- ⇒ *Formulate informal competency questions* to reflect the questions that the ontology must be able to answer based on the motivating scenarios.
- ⇒ *Specify the terminology in FOL* to produce formal definitions of concepts and relations.
- ⇒ *Formulate formal competency questions in FOL* from identified questions using formalised terminology from the previous activity.
- ⇒ *Specify axioms and definitions for the terms using FOL* to define the semantics, or meaning of terms in a manner that provides constraints on their interpretation.
- ⇒ *Evaluate competency and completeness* of the ontology by demonstrating that identified competency questions can be answered and depending on specific conditions, the answer can be deemed as complete.

Table 4.3, outlines the activities in the TOVE methodology and shows how they map to the ontology development life cycle stages. As it can be seen, conceptualisation, formalisation and implementation are not treated as different stages. Further, maintenance, knowledge acquisition and documentation are not explicitly defined as part of this methodology. Again, this can be attributed to the fact that this methodology belongs to a class of first generation methodologies.

METHONTOLOGY Activity	Maps to
Specification	Specification
Conceptualisation of domain knowledge	Conceptualisation
Formalisation of conceptual model	Formalisation
Implementation of formal model	Implementation
Maintenance	Maintenance
Knowledge Acquisition	Knowledge Acquisition
Documentation	Documentation
Evaluation	Evaluation

Table 4.4: Classification of Activities in METHONTOLOGY

4.1.2.3 METHONTOLOGY

METHONTOLOGY [49] unlike Enterprise and TOVE methodologies is not a first generation methodology. It is a second generation methodology that has gone through many refinements since the time it was proposed.

From Table 4.4, it can be seen that a one-to-one mapping exists between activities specified by METHONTOLOGY and the stages of the ontology development life cycle. Other activities that include integration, control and quality assurance activities are proposed by METHONTOLOGY but below only the tabulated activities will be elaborated on.

- ⇒ *Specification* entails stating the purpose of ontology and its scope as well as specifying competency questions. The product of specification is either an informal, semi-formal or formal document written in natural language.
- ⇒ *Knowledge acquisition* embodies a number of techniques and methods that enable knowledge to be acquired. These include brainstorming, interviews, formal and informal analysis of text (e.g. from books, handbooks, websites, etc). Most of the acquisition, as noted in [49], “is done simultaneously with the requirements specification phase, and decreases as the ontology development process moves forward”.
- ⇒ *Conceptualisation* structures the domain knowledge acquired during knowledge acquisition activity into a conceptual model that provides a picture of the overall problem and its solution. This model is built independent of how the ontology will be formalised and implemented. Therefore, knowledge representation paradigms and implementation languages are not considered in making any decisions that will yield the conceptual model.

- ⇒ *Formalisation* involves transforming the conceptual model to a formalised model that can be understood by the machine when implemented.
- ⇒ *Implementation* involves coding the formalised model using an ontology language in order to make the model executable.
- ⇒ *Maintenance* is carried out whenever there is a need to update or make corrections to the built ontology.
- ⇒ *Documentation* details each completed activity and captures assumptions made.
- ⇒ *Evaluation* includes the process of validating and verifying the built ontology. Gómez-Pérez and Pazos [64] evaluation criteria or any other suitable criteria is recommended.

4.2 Building the HIV and AIDS Service Provider Ontology

With the above in mind, the question is determining which service to create in order to build a real ontology to use in the validation of OVR. HIV and AIDS in South Africa, like in many other countries, is a public concern. There are over five million people living with HIV [112, 114]. In recent years, there has been a decline in the HIV prevalence among the youth [114]. This can be attributed to ongoing efforts and commitment to reduce and reverse the spread of HIV by all stakeholders including universities. Rhodes University like all concerned stakeholders has shown commitment to reducing rates of infection and increasing better management of the disease. To this end, it is involved in a lot of HIV and AIDS advocacy work.

At the time of finding a suitable project aligned to the goals of the thesis, Rhodes University was embarking on a project under the Higher Education HIV/AIDS (HEAIDS) Programme, funded by the European Union [22]. This project involved many parties across the university working on HIV and AIDS initiatives. Hence, it became clear that selecting this as a project, would allow a strategic synergy to be formed. On the one hand, the selection provided an assurance that there will be a collective in the construction of the ontology as it is recommended by best practice currently. On the other hand, the selection yielded an opportunity to work in a knowledge rich discipline. As stated in [23],

health care in broad terms is knowledge rich and the rate in which the knowledge grows is exponential.

Many implemented IVR systems are in the business domain. The last of the pertinent questions asked was: will health, as the selected domain of interest, play a role in reinforcing perceptions that users have of IVR applications? This question sought to determine whether users expectations differed significantly when using IVR applications for health motivated reasons than when using them for business related reasons.

According to Migneault, Farzanfar, Wright and Friedman [88], there is growing evidence that IVR applications can yield satisfactory health outcomes. They provide a two fold argument to support this view. First, they draw on their two decades of experience in designing and implementing automated telephone based conversational systems (i.e. IVRs) for improving health outcomes and the general delivery of health services. They present results from evaluation of some of their systems to argue the potential of IVRs in health, and in a sense, to demonstrate a growing trend in their use. Subsection 2.4.2 provides a brief review of other studies involving use of IVR in health to confirm the validity of the argument.

Secondly, they make a generic argument based on accessibility. They argue that any information delivered telephonically is easily accessed because “telephones are ubiquitous” [88] and familiar to most people across socio-economic lines. Overall, as deduced from their arguments, the domain itself is not important but the design of an IVR in meeting its objectives.

What should the service delivered be exactly?

Many HIV preventative measures are designed as initiatives that communicate tailored messages “for promoting and sustaining risk-reducing behaviour in individuals” [3]. The initial idea of the thesis, as indicated in [84], was to create a service that would be grounded by behavioural change theories to encourage, for example, use of condoms and testing. Figure 4.4 provides a snapshot of five key areas that were identified for inclusion in this service: 1) basic and general facts, 2) counselling and testing, 3) prevention and awareness, 4) rights and responsibilities, and 5) medication and treatment.

However, due to challenges experienced in working within the HEAIDS project, a decision was made to trim down the scope of the service. This was done by eliminating the need to use behavioural change theories. Following this decision, the requirements analysis phase was revisited. Interviews with clinicians and lay people, in particular, revealed a need for

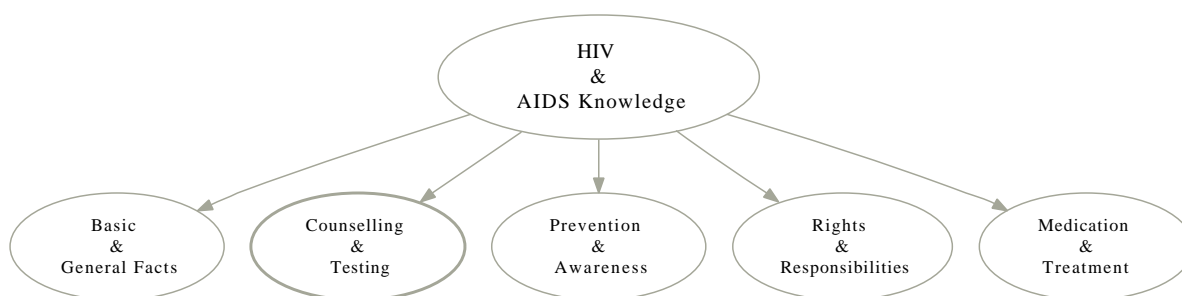


Figure 4.4: A Comprehensive Service for HIV and AIDS Information Dissemination

creating a service that would, at a minimum, help individuals locate service providers that offer HIV testing. This need was expressed mainly in the context of increasing the drive for testing; given that many individuals with HIV tend not to know their status early enough. As a result, they miss the opportunity to gain timely access to antiretroviral treatment, or the opportunity to change their behaviour to prevent re-infection and transmission to others [7, 113].

It is against the above background that a decision was taken to implement a service directory focusing only in the area of counselling and testing (indicated with a bold border in Figure 4.4). On the one hand, service directory is a classic telephony service that would allow the proposed architecture to be understood in context of service creation. On the other hand, the service itself has the potential to help individuals go for HIV testing. This, as suggested before, is beneficial to both the economy and the overall national health care system: assuming individuals with HIV can be helped to live long and productive lives.

This section will describe the development process for building the HIV and AIDS service provider ontology. The development essentially followed an *evolving prototyping life cycle*. As aptly articulated by Pinto and Martins [98]:

“In this life cycle, one can go back from any activity to any previous activity of the development process. As long as the ontology does not satisfy evaluation criteria and does not meet all requirements, the prototype is improved.”

The reason this life cycle allows moving back and forth to any activity as needs be, is that it inherently assumes that requirements for construction are ‘fluid’. As suggested in Chapter 1, the construction of the ontology for this thesis was to be driven by needs identified from an external project that provided an opportunity for a mutually beneficial partnership to be formed. For this reason, it was important to select a life cycle that provided flexibility in adapting to changes in the requirements at any time, and without

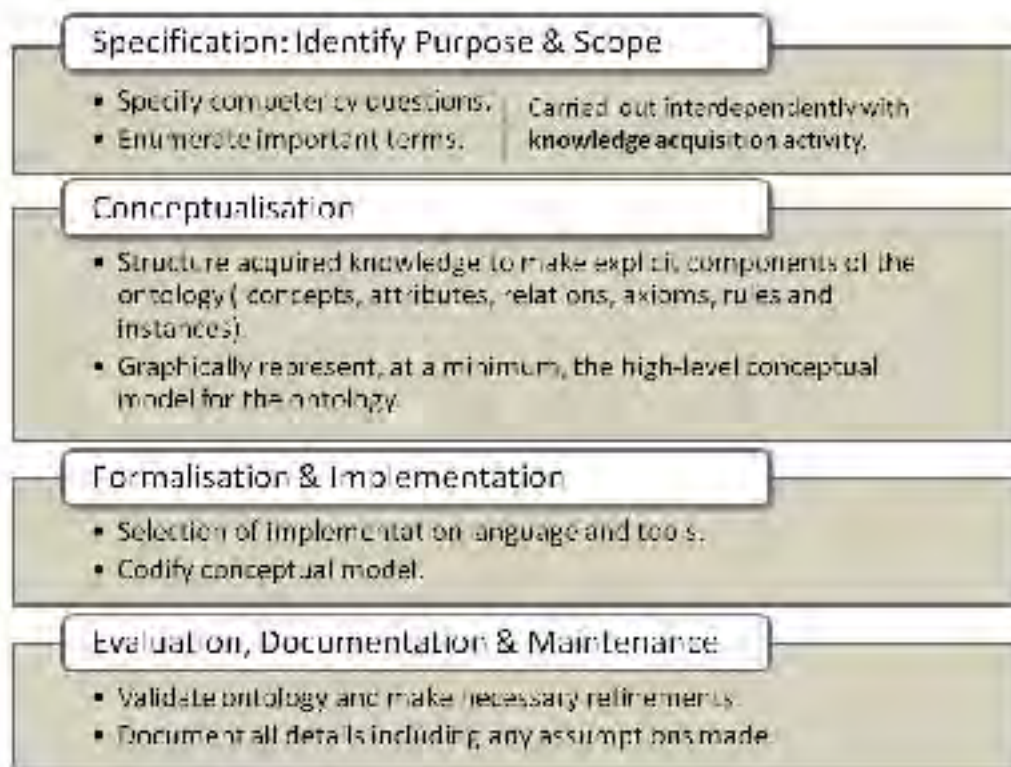


Figure 4.5: Process Overview for Building the HIV and AIDS Service Provider Ontology

necessarily worrying about the implications or risk factors associated with the changes. It is with this in mind and the aid of the decision tree shown in Figure 4.2 on page 45 that the evolving prototyping life cycle was selected.

Using this life cycle, four main stages as shown in Figure 4.5 were identified. The first stage is specification and its goal is to clarify the purpose and scope of the ontology through an informal process of knowledge acquisition about the domain to be modelled. The second stage is conceptualisation and its goal is to organise and structure the acquired knowledge in order to build a conceptual model for the ontology. The third stage formalises and implements the conceptual model to make the ontology executable. Finally, the fourth stage ties any loose ends from implementation. In this stage, the ontology is tested, refined and through documentation, an effort is made to ensure that any tacit form of knowledge that might have been used is articulated.

The above stages and activities form a hybrid methodology created by merging methodologies discussed in Subsection 4.1.2. According to a number of authors, for example, Brusa, Caliusco and Chiotti [34], depending on ontology goals and intended use: “it is common to merge different methodologies since each of them provides design ideas that

distinguish it from the others”. In the next subsections, more details will be provided on how different ideas from different methodologies were used to build the HIV and AIDS service provider ontology.

4.2.1 Specification: Identifying the Purpose & Scope

Although the terminology used may be different, the first stage for all methodologies is specification. In carrying out this stage, different strategies and activities are proposed. However, to bring increased clarity on the purpose and scope of the ontology, informal techniques for knowledge acquisition are preferred. For this reason, knowledge acquisition was done through brainstorm-type of meetings with members of the HEAIDS programme team, interviews with personnel at testing facilities and through analysis of content from relevant online resources.

In combining these three strategies, the purpose and scope of the ontology was clarified. Firstly, by identifying questions that can be answered by the ontology. Secondly, by listing important terms that needed to be captured within the ontology. The next subsections will provide more details on these two activities.

Before proceeding, it is worth mentioning that this stage would have included a review of candidate ontologies to reuse. However, it had already been established that no relevant ontologies existed for reuse when the project was selected. Furthermore, it was established that resources like HIV-911 Directory [6] would be useful to the development process. This directory provides a listing of organisations that offer services related to the management of HIV and AIDS in South Africa.

4.2.1.1 Competency questions

Competency questions, as suggested before, represent queries in a form of questions that can be used to test the validity of the ontology in serving its intended purpose [59]. Below is a listing of some of these questions for this ontology:

1. Where can I go for testing?
2. How far is the place from Rhodes University campus?
3. Where is the provider located at?
4. What are the opening hours?
5. What is the cost?
6. How long does one have to wait for their result?
7. How will the results be communicated?
8. Is counselling mandatory for one to be tested?
9. What other services are offered?
10. Is test confidentiality ensured?

The next step was to identify constraints and assumptions implicitly made in each question. Through this process, it was agreed that the mandate of this ontology was to offer information that enables a person to locate a place for HIV testing that is aligned to his/her needs but without delving deep into the testing details. This information can range from the duration of the test to other possible services offered by the place of interest.

For practical reasons, only places in Grahamstown were considered and Rhodes University has been used as a reference point for distance determination. The determination was done in approximate terms. That is, the actual distance measurement was not deemed to be as important as providing a broad indicator of how far or close the place is from the reference point. A similar argument was made for cost, which is given as either low cost or moderate cost.

4.2.1.2 Enumeration of important terms

The methodologies discussed in Subsection 4.1.2 advocate in varying degrees, for relevant terms about the ontology to be identified as part of specification. A methodology proposed by Noy and McGuinness [93] refers to this as enumeration (listing) of important terms.

The enumeration process requires engaging with questions such as: ‘*what terms are representative of statements that need to be captured in the ontology?*’, ‘*what is it that needs to be said about those terms?*’ and ‘*what properties are possessed by the terms?*’ [93]. By considering these questions, the scope is defined or further clarified. This in turn allows a transition to be made to the conceptualisation stage of the ontology building process. Below is an example of a listing from this process, shown as a “word cloud” to provide prominence to important terms:

Service provider, service, language used, telephone number, organisation type, result waiting duration, Cost, Healthcare provider, proximity

It is important to note that the enumeration process requires no effort to be put on pruning the list or classifying the terms in any meaningful way that may reveal relationships among the terms. This, as it will be seen shortly, is done as part of conceptualisation.

4.2.2 Conceptualisation

Conceptualisation, as noted by Gómez-Pérez *et al.* [63], is the most important activity of ontology construction. Its goal is to organise and structure knowledge acquired in the specification phase into taxonomies and vocabulary that is representative of the ontology. As noted by Dragan *et al.* [43], the taxonomies and the vocabulary provide a conceptual framework for the retrieval of information and most importantly facilitate knowledge sharing and knowledge reuse by applications.

METHONTOLOGY, due to its elaborateness, was used almost exclusively for guiding the process of conceptualisation. This methodology defines a set of ordered tasks to be performed as shown in Figure 4.6. Given that knowledge representation needs vary for different ontologies, it is not mandatory to perform all tasks or perform them in their stipulated order. However, according to Gómez-Pérez *et al.* [63], it is strongly advisable to adhere as much as possible to the order to ensure consistency and completeness in conceptualising the ontology.

A brief description of how each task was carried out will be provided below, following a format presented in [35, 63]. As it will be observed, the process is document intensive in that each task warrants some form of documentation to be done.

Task 1: Build glossary of terms

This was achieved by tabulating the enumerated terms to provide synonyms, acronyms, description based on natural language and a general classification for each term. Table 4.5 provides an excerpt of this tabulation.

Task 2: Build concept taxonomies

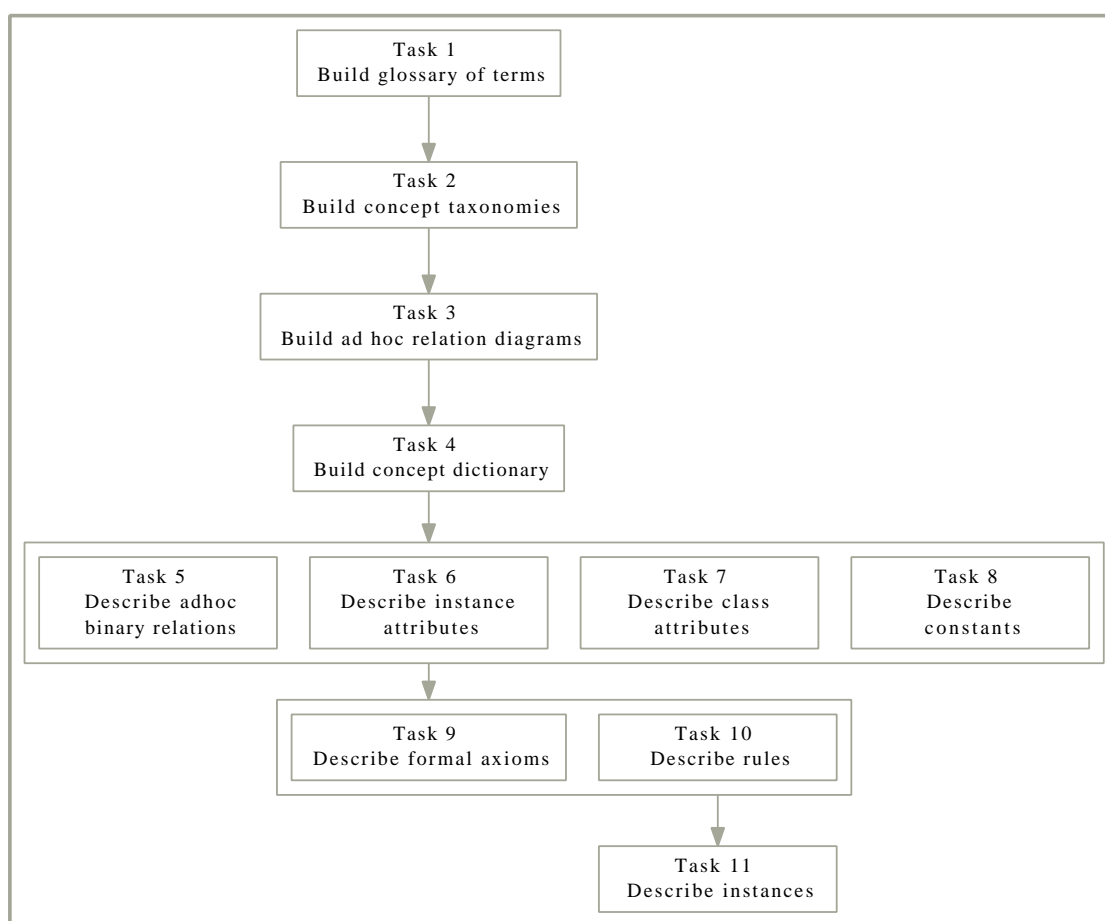


Figure 4.6: Tasks for Conceptualisation Activity According to METHONTOLOGY

To reveal parent-child relationships⁵ between concepts, concept taxonomies were built by selecting terms listed as concepts in the glossary of terms. A middle-out approach as recommended by Uschold and Grüninger [116] was used to guide the selection.

Using this approach, important concepts were identified first. Thereafter, the parents and then the children for those concepts were identified. As an example, to generate the taxonomy depicted in Figure 4.7 *Healthcare Provider* was selected first. The parent was identified as *Service Provider*. *Private Practice*, *Clinic*, *Hospital* and *Health Support Centre* were identified as children.

Task 3: Build ad hoc binary relation diagrams

Concepts of the same or different concept taxonomies may be related to each other through some ad hoc relations. Given that a picture is said to be worth a thousand words, the goal of this task was to visualise these relations. This was

⁵Formally known as superclass-subclass relationships or generalisation-specialisation relationships.

Term	Synonym	Acronym	Description	Type
Service provider	–	–	A provider of some service.	Concept
Non- Governmental Organisation	–	NGO	Organisations that offer services with the objective of not making profit.	Concept
Telephone number	Phone number	–	The telephone number of the service provider.	Instance Attribute
result waiting duration (HIV Test, Duration)	–	–	The time it takes for a person to receive results after testing.	Relation

Table 4.5: Example of Glossary of Terms for the HIV and AIDS Service Provider Ontology

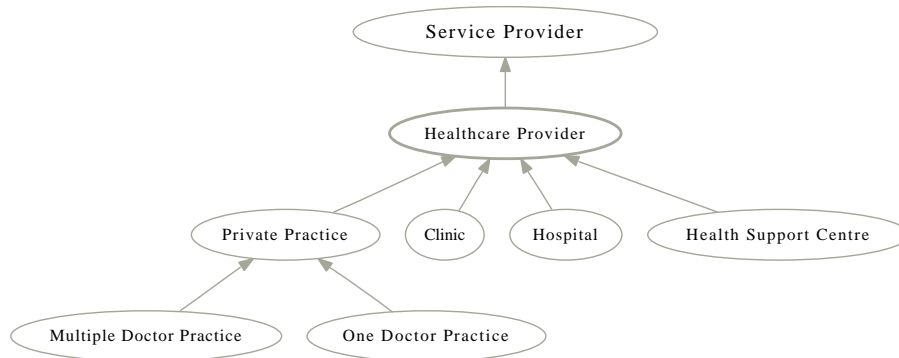


Figure 4.7: Excerpt of Concept Taxonomy from the HIV and AIDS Provider Ontology

achieved by generating ad hoc binary relation diagrams. In these diagrams, concepts are represented as labelled oval nodes and relations as arcs with lower camel case labels. An example of an ad hoc binary relation diagram is provided in Figure 4.8. In this figure, a binary relation *usesLanguage* connects *Service Provider* and *Language* concepts. This relation has an inverse relation *isLanguageUsedBy*, which is also shown in the figure.

The generated binary relation diagrams were then used as building blocks for generating the overall conceptual model diagram for the HIV and AIDS service ontology. Figure 4.9 shows a simplified conceptual model for this ontology. (This figure will be used extensively below to explain various concepts.)

In generating the conceptual model, a question arose as to how to represent n-ary relations, i.e. relations among more than two concepts. This question was

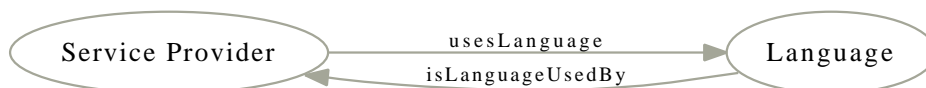


Figure 4.8: Ad hoc Binary Relation Diagram Example for the HIV and AIDS Provider Ontology

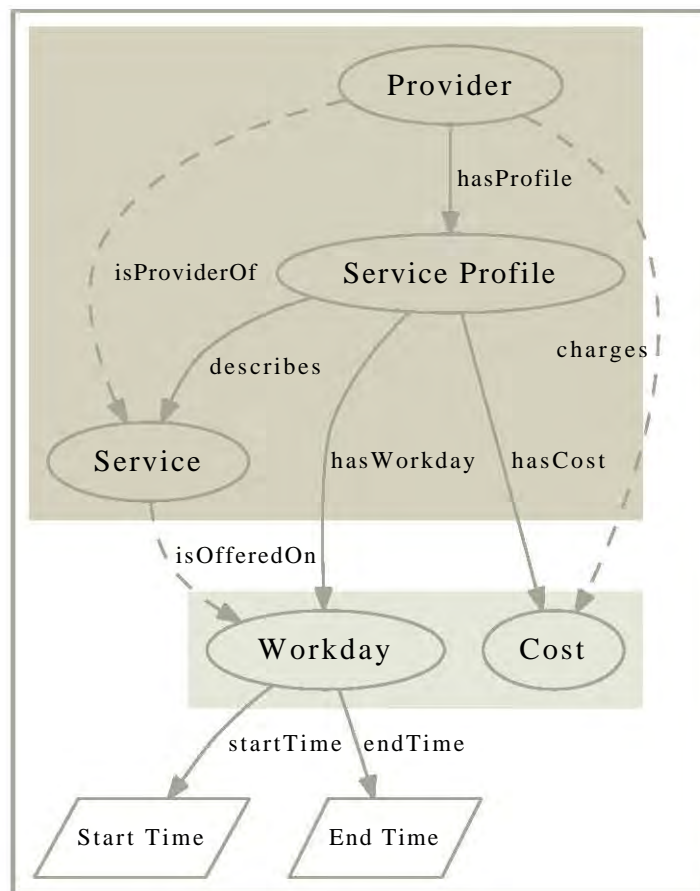


Figure 4.9: High-level Conceptual Model of the HIV and AIDS Provider Ontology

motivated by, for example, the need to assert that each service is associated with both cost and time, since some services are provided on specific days as opposed to every working day. In this instance, the problem translates to how to represent the following statement efficiently and accurately:

Service provider W offers service X on day Y at the cost of Z

Using guidelines offered in [94], the decision was to introduce a new concept that will group the n-ary relations together. For example, in order to capture the above, concept *Service Profile* was introduced and as it can be seen from Figure 4.9, it provides links to concepts *Service*, *Workday* and *Cost* via relations *describes*, *hasWorkday* and *hasCost* respectively.

Also shown in the figure, are dashed arcs which represent relations that indicate how inferences can be made. For example, if a service profile for a service provider has been fully defined with cost, service and day of offering stipulated, then the cost charged by the provider may be inferred via the relation *charges*. This can be done

Concept name	Instances	Class attributes	Instance attributes	Relations
Service profile	Joza profile 1	–	–	describe
	Raphael profile 1	–	–	hasCost
	Raphael profile 2	–	–	hasWorkday
Service provider	Fort England Hospital	–	telephoneNumber	isProviderOf
	Hospital Rhodes Health Care Centre	–	streetAddress	charges
	Raphael Centre	–	hasCloseRadius	hasWorkday
	Colcade Practice	–	email	hasProfile

Table 4.6: Excerpt of Concept Dictionary from the HIV and AIDS Service Provider Ontology

assuming the relation has been specified as a composition of relations *hasProfile* and *hasCost* (i.e. *hasProfile* o *hasCost*). Similarly, the day in which the service is offered on can be inferred via the *isOfferedOn* relation, a composition of the inverse relation of *describes* and *hasWorkday* (i.e. *inv(describes)* o *hasWorkday*).

As a final note on the discussion about Figure 4.9, the figure has been embellished to show that binary relations can also exist between concepts and attribute values. In the figure, this idea is represented by binary relations *startTime* and *endTime*, which link concept *Workday* to start time and end time values. These attribute values are represented in the figure using parallelograms.

Task 4: Build concept dictionary

For this task, the goal was to create a concept dictionary that lists all concepts together with their relations, their instances, and their class and instance attributes. Although not all details have been listed for selected concepts, Table 4.6 provides an excerpt of the concept dictionary from the HIV and AIDS service ontology.

Task 5: Describe ad hoc binary relations

The goal for this task was to describe in detail ad hoc binary relations listed in the concept dictionary. This was done by generating a table such as one shown in Table 4.7. This table specifies for each binary relation, a name, the names of source and target concepts, the source concept cardinality, the inverse relation, and mathematical properties, which indicate whether the relation is functional, symmetric, asymmetric, transitive, reflexive or irreflexive.

In the table, composing relations are also included if the relation was a composition i.e. a result of a sequence of relations. For example, *charges* is a composition of *hasProfile* and *hasCost* relations; since *hasProfile* followed by *hasCost* provides

Relation name	Source concept	Source card. (Max)	Target concept	Inverse relation	Mathematical properties	Relation composition
charges	Service Provider	1	Cost	isChargedBy	–	hasProfile o hasCost isProviderOf o hasCost
describes	Service Profile	1	Service	isDescribedBy	Functional	–
sharesPracticeWith	Doctor	N	Doctor	–	Symmetric	–

Table 4.7: Excerpt of Ad hoc Binary Relation Table from the HIV and AIDS Service Provider Ontology

Instance attribute name	Concept name	Value type	Cardinality	Default Value
hasCloseRadius (to campus)	Service Provider	boolean	(1,1)	–
startTime	Workday	time	(1,1)	08:00:00

Table 4.8: Excerpt of Instance Attribute Table from the HIV and AIDS Service Provider Ontology

the same source and target concepts as the *charges* relation. This can be confirmed quickly by referring to Figure 4.9 on page 59.

Task 6: Describe instance attributes

The goal for this task was to provide a detailed description of instance attributes listed in the concept dictionary. A table was used in which each row provided the details of an instance attribute. The table included the following fields: instance attribute name, the concept it belongs to, value type⁶, measurement unit, default values if they exist, allowed range of values for numerical values, and minimum and maximum cardinality. Table 4.8 shows a small section of the instance attribute table for the HIV and AIDS service provider ontology.

Additionally, the following could have been included in the table: instance attributes, class attributes and constants used to infer values for each instance attribute; formulae or rules that allow inferring values of the attribute; and attributes that can be inferred using values of this attribute. However, given the relative simplicity of the HIV and AIDS service provider ontology, these and some of the other descriptors previously listed were not required.

Task 7: Describe class attributes

The goal for this task was in many regards similar to that outlined in the previous task. The major difference is that class attributes describe concepts and take their values in the class where they are defined; unlike with instance attributes,

⁶These refer mainly to the numerous datatypes supported in XML schema.

Axiom name	HIV Test Provider Obligation
Description	Every HIV test provider must be a provider of either HIV testing or VCT.
Expression	for all(?X,?Y,?Z) (HIV Test Provider(?X) and HIV Testing(?Y) and isProviderOf(?X,?Y)) or (HIV Test Provider(?X) and VCT(?Z) and isProviderOf(?X,Z))
Concepts	HIV Test Provider HIV Testing VCT
Referred relations	isProviderOf
Referred attributes	–
Variables	?X ?Y ?Z

Table 4.9: Example Formal Axiom from the HIV and AIDS Service Provider Ontology

which describe concept instances and take their values in instances. Further, class attributes are neither inherited by the subclasses nor by the instances. In the concept dictionary, there were no listed class attributes otherwise a table similar to Table 4.8 would have been generated.

Task 8: Describe constants

The goal for this task was to provide a detailed description of constants listed in the glossary of terms by means of a table. There were no constants defined and so there was no need to carry out the task.

Task 9: Describe formal axioms

The goal of this task was to describe formal axioms, which are logical expressions for specifying constraints. It is important to ensure that each description is precise. Otherwise it is possible to sanction inferences that are inconsistent with defined definitions. In order to attain precision, METHONTOLOGY proposes inclusion of the following: name, description, logical expression that formally describes the axiom using first order logic, concepts, attributes and ad hoc relations as well as variables used.

Table 4.9 shows an example of a formal axiom defined within the HIV and AIDS service provider ontology. The axiom states that a service provider must provide HIV testing or VCT as services in order to qualify as an HIV test provider. Also included in the table are concepts and relations used in formulating the axiom.

Task 10: Describe rules

Rule name	Private practice doctors as contact persons
Description	A private practice doctor is as a contact person for the practice s/he works in.
Expression	if (Doctor(?D) and 'Private Practice'(?P) and isDoctorAt(?D, ?P) then hasContactPerson(?P, ?D)
Concepts	Private Practice Doctor Contact Person
Referred relations	hasContactPerson
Referred attributes	-
Variables	?D ?P

Table 4.10: Example Rule from the HIV and AIDS Service Provider Ontology

This task was very similar to the previous one. However, in this case the objective was to describe rules that may be used to infer knowledge within the ontology. The rules were used cautiously to avoid creating an ontology that is over committed, and also to minimise performance cost. In principle, rules increase expressivity of the ontology and as stated in the previous chapter, expressivity comes at a price.

The rules were presented in a form of *if-then* type of statements. Table 4.10 provides an example of one of the rules used in the HIV and AIDS service provider ontology. This rule states that private practice doctors can be regarded as contact persons for their practices. Also included in the table are concepts and relations used for formulating the rule.

Task 11: Describe instances

At this point, assuming the tasks were performed in some relative order, the conceptual model of the ontology is complete. There is the visualised and the textual part, which details all entities involved in the model. Therefore, all that is required in this task is to 'instantiate' the model. That is, define instances that appear in the concept dictionary inside an instance table. The instance table typically includes the following fields: instance name, name of concept the instance belongs to and the attribute values, if known. Table 4.11 on the following page presents a few instances from the HIV and AIDS service provider ontology.

Instance Name	Concept Name	Attribute	Values
Rhodes University Health Care Centre	Clinic	telephoneNumber hasCloseRadius streetAddress	6038093 true Lucas Avenue
RU Workday	Workday	startTime	08:30:00
Sister M V Nduna	Nurse	name	Mandisa V Nduna

Table 4.11: Excerpt of Instance Table from the HIV and AIDS Service Provider Ontology

4.2.3 Formalisation and Implementation

Similar to Enterprise and TOVE methodologies, formalisation and implementation were combined into a single stage. The goal of this stage was to transform the conceptual model to a model that is executable by a computer. To achieve this goal, two key questions needed to be asked. The first question was: which language to use for codifying the conceptual model? The second question was: which tools to use to support this dual process of formalisation and implementation? The next subsections will provide answers to these questions.

Once these answers were obtained, the only major requirement was effort to codify the conceptual model; then move to the next stage of the life cycle for rigorous testing and other related activities. For this reason, minute details of the formalisation and implementation process will not be discussed.

4.2.3.1 Ontology Language Selection

As already alluded to in Subsubsection 2.1.4.3 on page 14, the broad adoption of ontologies is dependent on the availability of well defined semantics and powerful reasoning tools. DLs provide both, and for this reason, a decision was made to select a DL based language. This decision was informed by nearly two decades of research in DLs, which according to Baader *et al.* [24] provides unequivocal evidence that DLs are “ideal candidates for ontology languages”. According to Baader *et al.* [24], the suitability of DLs has been highlighted by their role as the foundation for several web ontology languages that include OWL.

OWL is an ontology language standard developed by the W3C Web-Ontology Working Group. As reported in [87], compared to other W3C endorsed languages such as Extensible Markup Language (XML), RDF, and RDF Schema (RDF-S), OWL facilitates greater interpretation of content by machines. This is because in contrast to the other languages,

OWL sublanguages	Based on	Expressivity	Reasoning
OWL Lite	DL	Low	Decidable
OWL DL	DL	Medium	Mostly decidable
OWL Full	RDF-S	High	Mostly undecidable

Table 4.12: A Brief Comparison of OWL Sublanguages

OWL provides additional vocabulary together with formal semantics. Based on this, OWL was selected for use. Specifically, *OWL 1.1*⁷ was chosen even though at the time of construction that version was not yet endorsed.

There are three different sublanguages in OWL, namely OWL Full, OWL DL and OWL Lite. OWL Full is the superset of OWL DL and OWL Lite. OWL DL is based on DL and its subset OWL Lite is based on a least expressive form of DL. To further highlight the relationship between these sublanguages, Table 4.12 provides a brief summary and comparison. In particular, the table captures the relationship between expressivity and decidability in reasoning.

In the previous chapter, it was argued that reasoning and representation are inextricably intertwined. Further, that the role of representation, among other things, is to act as a medium for efficient computation. With these ideas in mind, OWL DL was selected as the best language to use for implementing the HIV and AIDS service provider ontology.

Essentially, OWL DL was selected because it offers a good middle ground between expressiveness and efficiency in reasoning. Again, by referring to Table 4.12, it can be seen that compared to OWL Lite, OWL DL offers more expressivity whilst retaining a high degree of decidability. Compared to OWL Full, however, it offers less expressivity but remains computationally superior. This is because OWL Full does not provide computational guarantees to enable useful or intelligent inferences to be made from an ontology [87].

4.2.3.2 Tool Selection

→ *Development Tool*

Protégé [13], a free and open source platform, is regarded as one of the leading ontological engineering tools [43, 93]. As Lam and Lee [78] elaborate, *Protégé* offers a “friendly user interface that provides a set of tools for constructing domain model and knowledge based applications with ontologies”.

⁷OWL 1.1 was developed informally by OWL practitioners in industry and academia as a revision of OWL-DL; and was submitted to W3C Working Group, which later decided to call it OWL 2 [55, 82].

At the time of construction, *Protégé-4* [70] was the only tool that provided solid support for constructing ontologies based on *OWL 1.1*. This, together with reputation for quality, made *Protégé-4* an ideal candidate for use. The alternative would have been to manually formalise and implement the ontology with the aid of a simple text editor. This would have worked, but would have taken longer. *Protégé* speeds up development substantially in that it provides an interface that obviates the need to learn low level syntax of *OWL 1.1*. Further, it allows, for example, for the ontology to be visualised like as shown in Appendix A on page 118.

→ *Reasoners*

There are a number of ontology reasoners available in the market. However, only *FaCT++* and *Pellet* [105] were considered. There are two interrelated reasons for this.

First, it was pragmatic to select a DL reasoner optimised for use with *Protégé*. Secondly, the reasoner had to have good support for *OWL 1.1*, the selected language for ontology authoring. Unfortunately, while reasoners such as *RacerPro* [65] and *HermiT* [90] could in practice be used with *Protégé*, at the time of construction, they did not offer much support for *OWL 1.1*. *FaCT++* and *Pellet*, however, showed a high degree of conformity with *OWL 1.1* and also came bundled in with *Protégé-4*.

Table 4.14, compares briefly these two reasoners. The lightly shaded rows represent comparison based on some relatively subjective criteria. For each criterion, a three point rating scale was used. The scores were awarded based on information gleaned informally from online resources and comparative studies (based on previous versions) such as those done by Sirin, Parsia, Grau, Kalyanpur and Katz [105], and Gardiner, Horrocks and Tsarkov [52].

Apart from the difference in the implementation language, it can be seen from the table that *FaCT++* and *Pellet* are almost on par with each other. However, by introducing a degree of subjectiveness, *Pellet* was deemed most suitable for use in that it offered rule and good technical support.

4.2.4 Evaluation, Documentation & Maintenance

Once implementation is done, there is a need to ensure that the quality of the resulting ontology meets expectations of its users. Further, that the captured knowledge can be shared without creating barriers in understanding. Given this dual mandate and the

	FaCT ++	Pellet
Language	C++ implementation but has Java wrapper	Pure Java implementation
Implementation algorithm	Tableaux	Tableaux
Licensing	Free	Dual licence – free for open source applications and fee based for closed source applications
Expressivity	SROIQ(D)	SROIQ(D)
Rule support	No	Yes (SWRL – DL Safe Rules)
Performance [Poor Average Good]	Good – but in few cases can be poor	Good – but in few cases can be poor
Release cycle / Maintenance activity [Irregular Semi regular Regular]	Semi regular – monitored but unless for example, a feature request or bug report is marked high priority, it may take time to implement or fix.	Regular
Technical support [Poor Average Good]	Average	Good

Table 4.14: Comparison of FaCT++ and Pellet Reasoners

relative simplicity in scope of the service provider ontology, evaluation, documentation and maintenance were combined into a single stage, which involved:

- ⇒ Validating the ontology by verifying that it can answer accurately the specified competency questions.
- ⇒ Using criteria recommended in literature (such as one discussed in 2.1.4.6 on page 17) to evaluate definitions and axioms within the ontology. For example, following a technical evaluation to determine how to reduce over commitment without decreasing performance, a decision was made to follow advice issued in [70]. The advice was to avoid explicit specification of source and target concepts of binary relations during the implementation stage; because it was possible to determine them during reasoning. This translated in the removal of asserted domains and ranges for many of the properties (binary relations) defined in the ontology.
- ⇒ Documenting the ontology by recording changes and adding content in the implementation code as comments, meta data, descriptions, etc. This also applies to documentation generated from specification and conceptualisation.

4.3 Summary

This chapter provided implementation details of the HIV and AIDS service provider ontology. This ontology captures knowledge pertaining to places that provide HIV testing.

In constructing the ontology, ideas from literature were used to inform the process as well as guide decisions for conceptual modelling and tool selection. One of the key ideas related to using a development life cycle. As explained by Fernández-López, Gómez-Pérez and Amaya [48], a life cycle outlines the various stages of ontology development, activities performed in each stage and how these stages are related to each other.

Although it may seem that a life cycle is prescriptive and rigid, according to Royce [103], it provides structure that transforms a risky ad hoc development process into one that produces the desired product in a timely and cost effective manner. As stated in this chapter, for the construction of the ontology an evolving prototyping life cycle model was used. This model was deemed more suitable in contrast to other presented models in that it provided the assurance that the final product could be declared as “reasonably modifiable and reliable” [25].

Chapter 5

Basic Implementation of the Prototype System

This chapter will discuss the implementation of the prototype system, an IVR system named HIV Testing Locator System (HTLS). This system has been designed to use the HIV and AIDS service directory realised through the ontology discussed in Chapter 4. The key objective of the system is to help users locate HIV testing sites aligned to their individual needs. Figure 5.1 provides the high level use case diagram for the system. Essentially, in order to locate a testing site, a search query is initiated to retrieve information from the ontology knowledge base. (As shown in the figure, a robot has been used to represent the knowledge base: the intention is to convey the future possibility of successfully averting the need for a human agent.) The query to the knowledge base is generated by obtaining search criteria that needs to be matched. In this particular case,

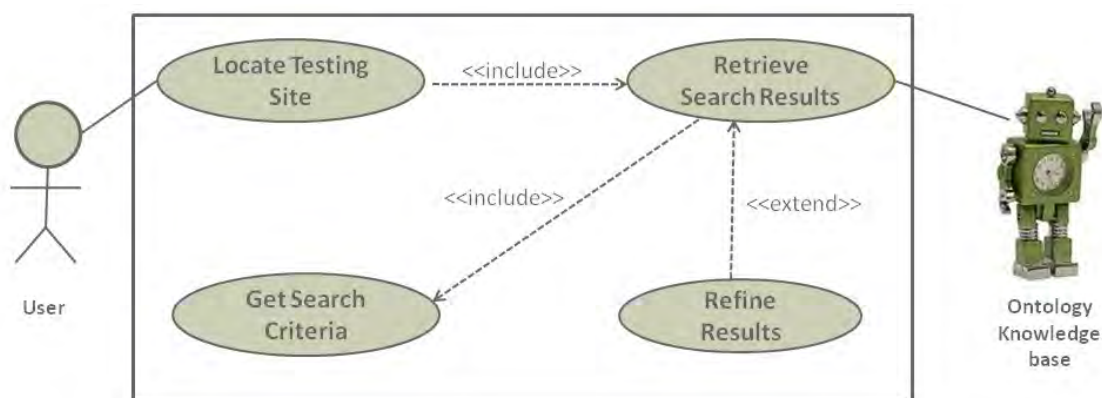


Figure 5.1: Use Case Diagram for HIV Testing Site Locator

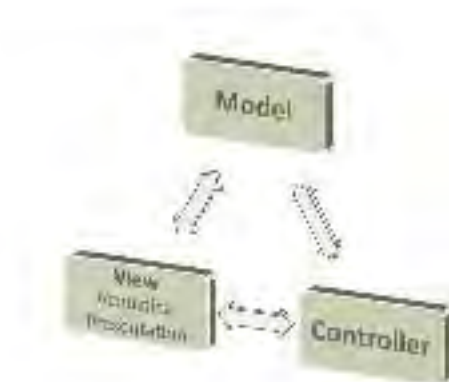


Figure 5.2: Model View Controller Architectural Design Pattern

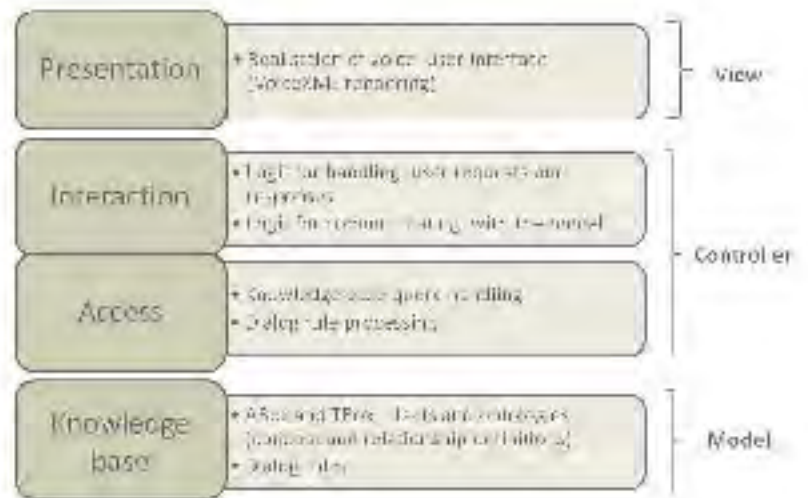


Figure 5.3: Implementation Breakdown based on MVC

the criteria is determined by the cost of testing and the proximity of a site to Rhodes University campus. Once retrieved, the results may be refined.

The refinement process will be discussed in Chapter 6 as part of the implementation of the enhanced prototype. This chapter will discuss the implementation of the initial, basic prototype. This discussion will be preceded by an outline of the overall strategy for implementing both the basic and the enhanced prototype systems.

5.1 Implementation Strategy

An important consideration for implementing the prototype system was maximising flexibility. As gleaned explicitly from computing fields like Software Engineering, this could be attained through separation of concerns, which effectively translates to modular system design.

There are a number of architectural models that have been proposed for achieving modularisation. The MVC [77] and three tier architecture are but examples of well known and widely used models for web oriented applications. Both models break down implementation into three parts. These parts are similar although their functionality may be categorised differently. In MVC, the parts are named: *model*, *view* and *controller*. In the three tier architecture, the parts are named: *presentation* layer, *model* layer and *persistance* layer. Conceptually, communication between these parts is triangular in the case of

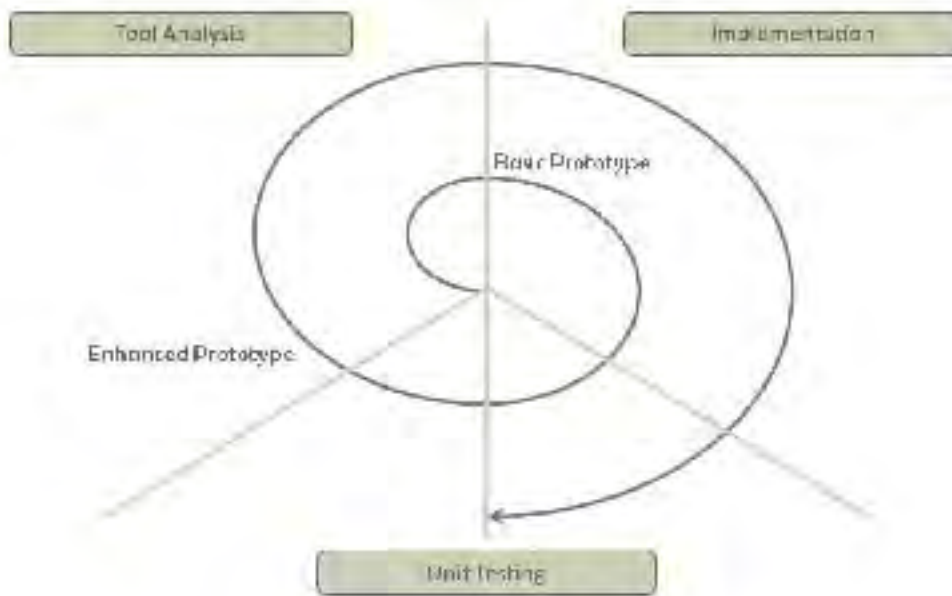


Figure 5.4: Spiral Model for Implementing the Prototype System

MVC (as shown in Figure 5.2) whereas in three tier architecture it is linear. This makes MVC model more flexible than the three tier architecture since all parts can communicate directly with each other. It is for this reason that MVC was selected for use over the three tier architecture.

The MVC parts focus, respectively on: representation of domain specific information, presentation, and executing control logic, which subject to user input may cause the model and the view to be modified. Although these are the key areas for MVC, it is important to note that a strict breakdown into three parts is not prescribed by this architectural model. Consequently, there are many variations of how the model is used. For example, in the case of this thesis, the breakdown is in four parts, as shown in Figure 5.3. Presentation maps to the view, the knowledge base maps to the model and the controller maps jointly to interaction and access components, with some elements of access belonging to the model.

Using the above breakdown, an implementation strategy was developed based on a simple question: what is needed to have an initial working prototype? Answering this question amounted to using an experimental approach, which involved two major phases of development, each comprising of multiple iterations. As expected, the first phase focused on implementing the initial prototype while the second focused on the enhancements of that prototype.

Inherent to the implementation strategy was the choice of a development life cycle model

to use. Several different development life cycle models were introduced in Chapter 4. As it might be deduced from the chapter, the suitability of each life cycle model is dependent on the needs and requirements of the artefact to be built. In this particular case, the spiral life cycle model was deemed suitable for use; it is iterative and accommodates for uncertainties inherent in using an experimental approach. This life cycle model as it pertains to the development of the prototype system is depicted in Figure 5.4 (albeit inaccurately, since individual iterations are not shown, instead iterations have been grouped under either basic or enhanced prototype).

5.2 First Group of Iterations for the Implementation

The ontology constitutes a large portion of what qualifies as the model component of the prototype system. Chapter 4 extensively discussed the implementation of the ontology for this thesis. This section will provide implementation details for the the view and the controller components. With these implemented, the initial working prototype will be realised.

Each component to be discussed in this subsection will in fact constitute a distinct iteration. For each iteration, the format of the discussion will be closely aligned to the steps of the spiral model shown in Figure 5.4. It will include a preamble capturing the design, followed by tool selection, implementation and unit testing segments.

5.2.1 View Implementation

The conversations the user holds with voice dialog systems qualify as the user interface for these systems. The realisation of the conversations is crucial to the overall implementation of these systems.

There are a number of strategies based on voice user interface design principles that can be followed to ensure that these conversations are indeed effective (i.e. lead to user satisfaction instead of frustration). These include: breaking content into clear small chunks to avoid memory overload; using quiz type questions; and/or making content conversational by introducing rhetorical questions as part of confirming user requests. These strategies are intended to help users perform their tasks quickly and without frustration.

With the above in mind, flowcharts were used to capture the order of events for various scenarios to be supported by the system. Then, conversations for different scenarios were written out in a drama script format to simulate communication by the user and the system (see Appendix C on page 121 for example scripts).

In creating the flowcharts (and subsequently writing out the scripts), caution was taken to limit options to be presented to the user to three. This decision was guided by anecdotal evidence which suggest that audio and serially presented information impose a limitation on the short term memory that is far less than ‘*Miller’s magical number seven, plus or minus two*’ [89]. (Miller argues that a person can remember between five and nine pieces of information presented at once visually).

On the basis of the flowcharts and produced scripts, pages based on VoiceXML [50] (version 2.1) were created. Details will be provided shortly below.

A) *Tool Selection*

Although development tools are available for the automated creation of VoiceXML pages, a decision was made not to use them. Instead, a simple text editor together with an online XML validating tool powered by Validome [19] were used.

Three reasons can be provided for this decision. The first reason is that comprehensive support for VoiceXML is not offered by many tools (i.e. GUI based tools) because of continuing amendments to the language as part of the ongoing standardisation process. Secondly, the learning curve for VoiceXML is low because as argued in 2.3.1 on page 24 its alignment to the web enables web development skills to be leveraged on. The third reason is in a sense interrelated to the other reasons. As a well known fact, a tool can enhance productivity and efficiency. This is dependent on the developer learning to use the tool properly, which is an investment if the application to be built is complex or there is a possibility of using the tool regularly in the future. Neither of these cases applied. Further, due to the ongoing standardisation process of VoiceXML, there would still be a need to learn VoiceXML in order to resolve any possible mismatches between properties of the language supported by the selected tool and VoiceXML browser. Weighing these together, learning the language as opposed to the tool seemed more worthwhile.

B) *Implementation*

Learning VoiceXML was crucial to the implementation of the view. Once this was done, VoiceXML pages for the prototype application were created. These pages rep-

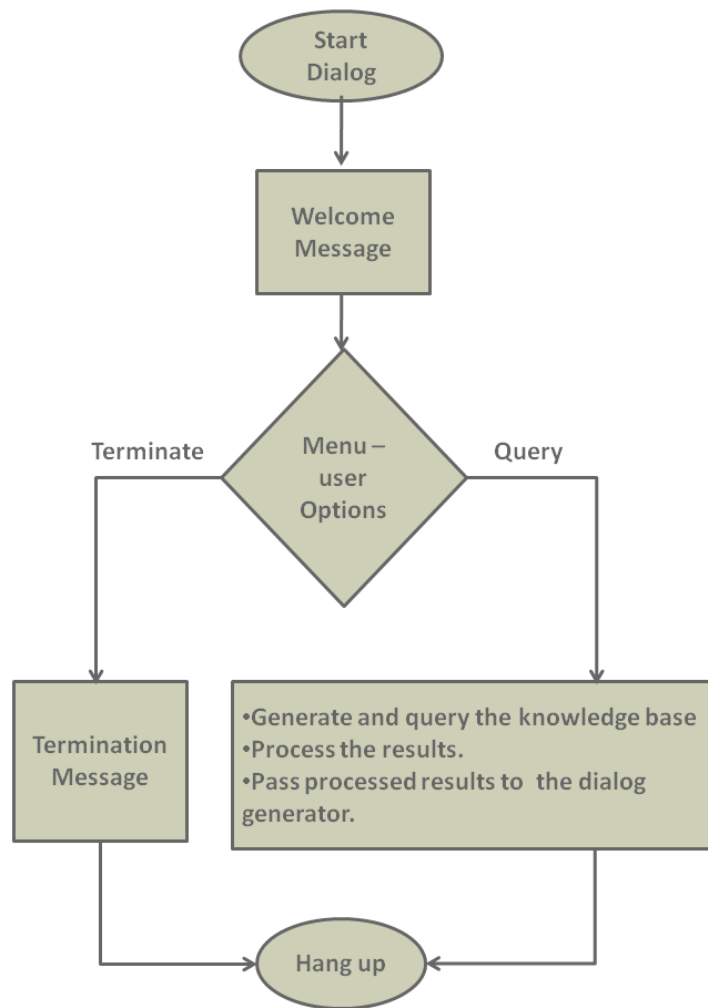


Figure 5.5: High Level Flowchart Capturing Prototype Functionality

resented dialogs using forms, which served to either collect user input or present information to the user.

As alluded to, the pages were authored using a simple text editor and an online tool for checking correctness of grammar and syntax. These pages were created based on flowcharts and scripted dialogs. Figure 5.5 shows a high level flowchart for generating the pages. For example, the initial page for the application, shown in Figure 5.6 was created. This page includes two main dialogs, with the logic for each enclosed in *form* tags. The first dialog, named *greetings*, greets the user and moves to the *options* dialog using the *goto* tag. Within the *options* (menu) dialog, the user is asked to make choices that would aid in locating a suitable provider. These choices are then submitted for further processing to the controller. The details of the processing will be described in the next subsection. Apart from selecting options for choosing a provider, the user has the option to choose to terminate the session (other than by

```

1 <?xml version="1.0"?>
2 <vxml version="2.1" xmlns="http://www.w3.org/2001/vxml">
3
4 <link dtmf="0" event="exit" />
5 <catch event="exit"> <goto next="terminate.vxml"/> </catch>
6 <!--Variable declaration omitted -->
7
8 <!-- Start initial dialog -->
9 <form id="greetings">
10 <!--The block element is needed for containing executable instructions -->
11 <block>
12 <audio src="">
13 <!-- Welcome message -->
14 </audio>
15 <goto next="#options"/>
16 </block>
17 </form>
18
19 <!--Present user with options -->
20 <form id = "options">
21 <!--Determine cost preference -->
22 <!--The field element declares an input field and prompts the user for values
23 that match a grammar. In this case, DTMF built-in grammar specific to the
24 VXL* platform is used.-->
25 <field name="costParam" type="boolean?y=1;n=2;">
26 <prompt>Are you interested in a free or low cost testing site? Press 1 for
27 yes and press 2 for no.</prompt>
28 <!-- Execute the filled segment to evaluate user response -->
29 <filled>
30 <if cond ="costParam=='true '">
31 <assign name = "cost" expr="'lowCost'"/>
32 <elseif cond="costParam =='false'"/>
33 <assign name ="cost" expr ="'moderateCost'"/>
34 </if>
35 </filled>
36 </field>
37 <!--Determine distance preference-->
38 <!--Submit user selection -->
39 <block>
40 <submit next="http://localhost/voiceLocApp/query" method="post" namelist="
distance cost"/>
41 </block>
42 </form>
43 </vxml>

```

Figure 5.6: Listing of VoiceXML Page for the Initial Dialog (i.e. Welcome Page)

just hanging up). If this decision is made, a dialog is initiated that thanks the user for calling. This dialog is captured in another VoiceXML page, *terminate.vxml*.

It is important to note that, as with the implementation of any application, there were multiple ways to get the desired results. Figure 5.6, for example, illustrates one possible implementation for realising the logic captured by the flowchart shown in Figure 5.5.

C) Unit Testing

For this iteration, testing focused mainly on validating the control flow and processing carried within the page. To perform this testing, the created pages were uploaded

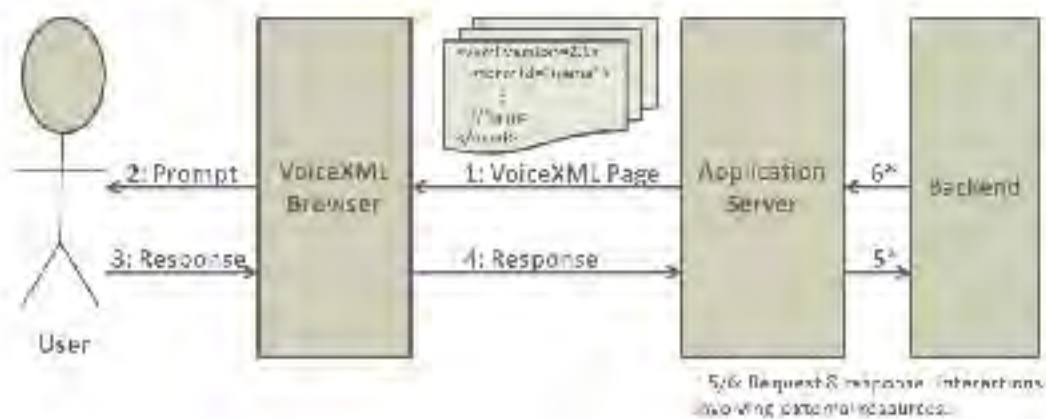


Figure 5.7: Events for Basic VoiceXML Page Rendering

to the (web) application server. Then through *extensions.conf* file (*Asterisk* file for dialplan configuration), a phone number (extension) was specified. This number linked to the URL of created VoiceXML pages as follows:

```
exten => 448,n,Vxml(http://serverURL/voiceApp/index.vxml)
```

Using the specified number (i.e. 448) a call was placed to verify the functionality of the pages. This entailed a process of validating communication between the user and the application server, as shown in Figure 5.7. That is, once the call has been answered the following happens as depicted:

- (1) The application server delivers the appropriate VoiceXML page to the browser.
- (2) The browser interprets the page and renders it appropriately. For interactions with the user, this may involve prompting (i.e. posing questions) to the user, which may be rendered by TTS engine or through an audio recording.
- (3) The user responds to the prompt. This could be through voice but the prototype was implemented specifically to use touch tone.
- (4) The response is forwarded to the application server. Another page may be delivered to the browser or a request for external resources may be made to the backend.
- (5) A request is placed to the backend, if at all necessary.
- (6) A response to the request is returned.

When the results were not as expected, debugging was performed in order to identify and fix the potential fault, then the verification process was repeated. The debugging and testing processes were repeated until the functionality of the prototype matched

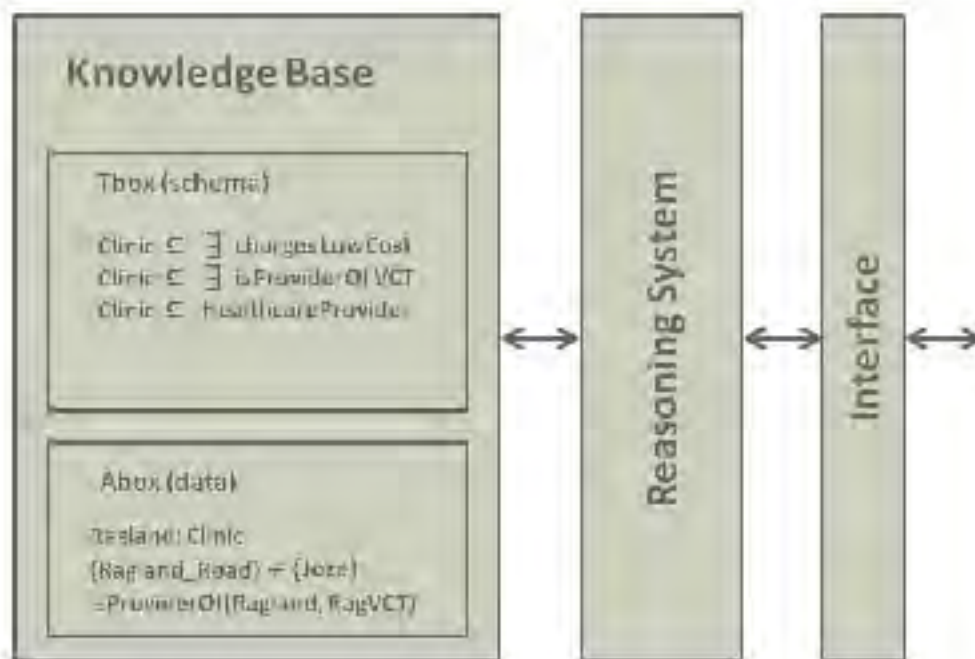


Figure 5.8: Basic Components in a Description Logic based System

the design requirements. (This type of testing is known as correctness testing because of its strong focus on ensuring correct behaviour of a software module.)

5.2.2 Controller Implementation

The *controller* as stated before refers to a component that processes and responds to events between the view and the model. As noted in the previous section, a VoiceXML based user interface was created using flowcharts and scripted dialogs. The process of generating these flowcharts and dialogs provided a ‘point of entry’ into understanding the interactions between the view and the model. This, in turn, helped in defining the behaviour of the controller to be built.

Figure 5.8 shows basic components in a DL based system and their interaction with each other. This figure effectively captures how communication with the model should be carried out. In order to access any data, an application must communicate with the knowledge base via an interface that connects directly to a reasoning system.

The primary task of a controller is to facilitate this communication with the model as well as communication with the view (see Figure 5.9 on the next page). As part of the analysis process, it was established that the controller would require implementing code

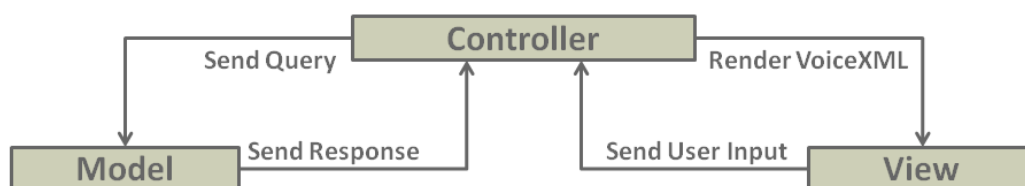


Figure 5.9: Basic Controller's Task

that handled requests and generated pages. Also required was code that handled the logic pertaining specifically to how reasoning services could be used to obtain information from the HIV and AIDS service provider ontology. This code was required to build an interface through which the ontology could be queried to retrieve relevant data from the knowledge base.

A) *Tool Selection*

An observation on current trends shows a steady move into a generation of Java based tools within the telephony sector. (This can be evidenced by the growing adoption of platforms like Mobicents, a popular open source telecommunications multi-service platform that is based on efforts by the Java community [110].) On the basis of this, Java was selected as the language of choice for implementing the controller (and any functionality that supports application integration on the server side). Implicitly, this meant grounding subsequent decisions on tools to use to Java oriented APIs, technologies, etc.

In line with the above, servlets were chosen for implementing the controller. *Pellet* was selected for offering reasoning services (for reasons provided in Subsubsection 4.2.3.2 on page 66). OWL API [68, 69], a high level Java API for working with OWL 2 ontologies was selected for implementing the interface for querying.

B) *Implementation*

Taking advantage of the similarity between VoiceXML and HTML, a decision was made to build a controller that would be tested using a view implemented in HTML. Fundamentally, the controller would work in the same way, irrespective of whether the user interface was created in VoiceXML or HTML. HTML was chosen simply because it was perceived that visual debugging would be faster, therefore productivity would be enhanced. Based on this argument, one other possibility would have been to render VoiceXML on the visual browser (i.e. display as an XML tree structure within the browser). This remained a viable option but for the initial demonstration of the basic functional prototype, HTML implemented view seemed more attractive.

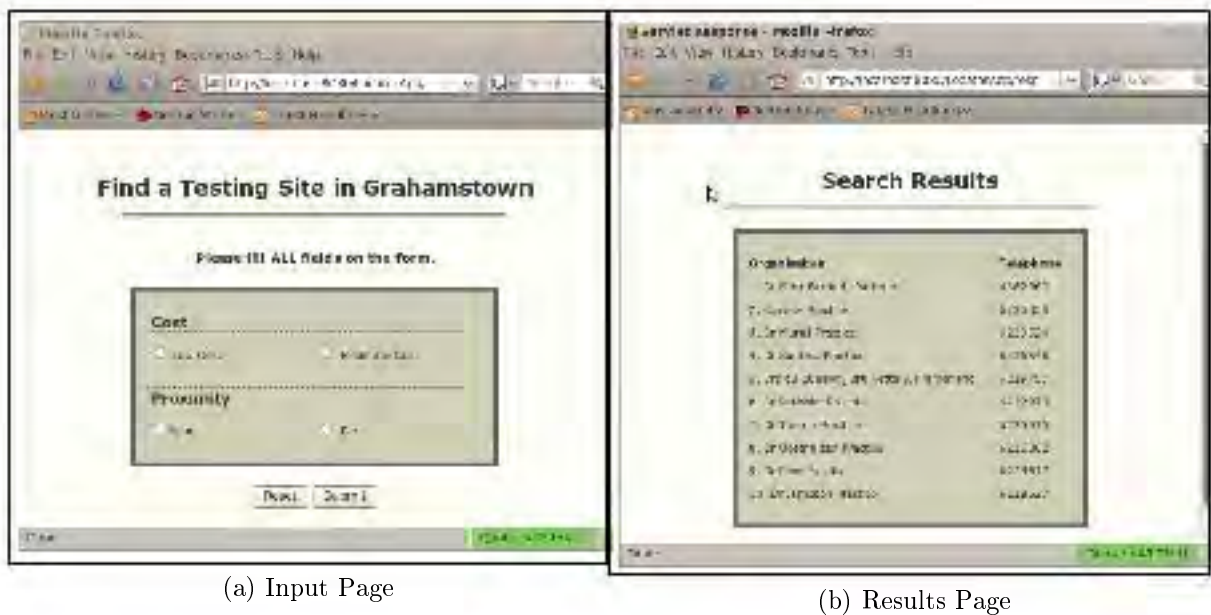


Figure 5.10: Visual Implementation of the Prototype Service Rendered in HTML

Figure 5.10 captures pages for this prototype: the input page (Figure 5.10a) and the results page (Figure 5.10b). In this case, the role of the controller is simply to process submissions from the input page and generate the results page.

Figure 5.11 provides snippet code of the servlet used to realise the controller. Essentially, what this code does is to retrieve user's input, generate a query to extract information from the ontology knowledge base, generate HTML page for presenting the results, and display the results from the query. These tasks (with the exception of query generation) have been expressed explicitly in the listing captured in the figure. The details of query generation have been abstracted away by implementing a class with the logic for performing the needed query.

This class was implemented using OWL API. It contains several methods that collectively serve to provide a convenient format for performing queries. For example, there are methods for loading the ontology, setting up the reasoner and retrieving various entities (e.g. classes, individuals and axioms) using a string as a keyword.

Figure 5.12 shows a listing that captures the methods responsible specifically for querying the ontology. For example, line 5 calls the method `getClassIndividuals(...)`, which returns individuals from the ontology based on a query that is passed to the reasoner. The query itself is generated by calling the `createQuery(...)` method, which uses cost and proximity parameters received from the user to create a query of the

```

1 protected void processRequest (HttpServletRequest request , HttpServletResponse
  response) throws ServletException , IOException {
2 //Declarations
3 try {
4 // Retrieve user input
5 String costOpt = request.getParameter("cost" ) ;
6 String proximityOpt = request.getParameter("proximity" ) ;
7 //Generate HTML page to present the results
8 out.println("<html>");
9 out.println("<body>");
10 out.println("<h1>Search Results</h1>");
11 out.println("<table id='table'>");
12 out.println("<tr><th>Organisation</th><th>Telephone<th></tr>");
13 // Generate a query using user's input and with display the results
14 results = directory.getClassIndividuals(costOpt , proximityOpt);
15 Iterator iterator = results.iterator();
16 int count = 0;
17 while(iterator.hasNext()){
18 String key = iterator.next().toString();
19 int num = ++count;
20 out.println("<tr><td>"+num+" "+
21 output.underscoreRemoval(key)+"</td><td>"+directory.getDataLiteral(key , pref.
  getValuesOfInterestPropName())+"</td></tr>");
22 }
23 out.println("</table>");
24 out.println("</body>");
25 out.println("</html>");
26 } finally {out.close();}
27 }

```

Figure 5.11: Listing for Processing and Presenting Query Results

```

1 public Set getClassIndividuals(String costOpt , String proximityOpt){
2 Set instances= null;
3 OWLDescription query = createQuery(costOpt , proximityOpt);
4 try{
5 instances = reasoner.getIndividuals(query , false);
6 }catch (OWLReasonerException ex){//handler code}
7 return instances;
8 }
9 //Ontology querying amounts to creating descriptions
10 private OWLDescription createQuery(String costOpt , String proximityOpt){
11 Set<OWLDescription> classes = new HashSet<OWLDescription>();
12 //Everything must belong to the rootCls i.e. HIVTestProvider
13 classes.add(getRootCls());
14 OWLObjectProperty prop = this.getNamedObjectProperty("charges");
15 if (costOpt.equalsIgnoreCase("lowCost")){
16 classes.add(manager.getOWLDataFactory().getOWLObjectSomeRestriction(prop ,
  getNamedClass("LowCost"));
17 }
18 if (costOpt.equalsIgnoreCase("ModerateCost")){
19 classes.add(manager.getOWLDataFactory().getOWLObjectSomeRestriction(prop ,
  getNamedClass("ModerateCost"));
20 }
21 if (proximityOpt.equalsIgnoreCase("near")){
22 classes.add(getBooleanDataValueDesc("hasCloseRadius","true"));
23 }
24 if (proximityOpt.equalsIgnoreCase("far")){
25 classes.add(getBooleanDataValueDesc("hasCloseRadius","false"));
26 }
27 return manager.getOWLDataFactory().getOWLObjectIntersectionOf(classes);
28 }

```

Figure 5.12: Listing for Query Generation

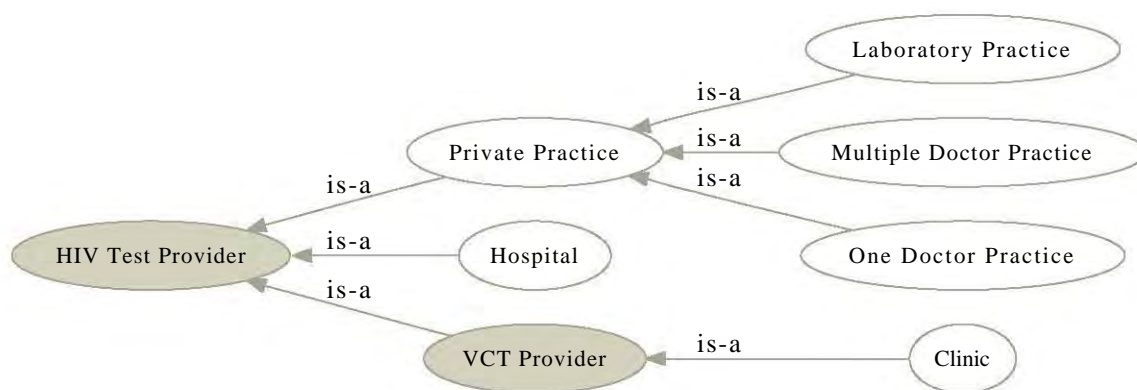


Figure 5.13: Taxonomy Tree for HIV Test Providers

form:

```

HIVTestProvider
and (charges some Cost)
and (hasCloseRadius value Boolean)
  
```

Note: Cost can either be LowCost or ModerateCost.

In essence, even though numerous queries could be made to the HIV and AIDS service provider ontology, the decision was to have an interface that would restrict the search space for the answer. This was achieved by specifying a ‘root node’ for a hierarchy of ontology classes likely to contain query answers.

In this particular case, the *HIVTestProvider* class was provided as the root class. Figure 5.13 shows a taxonomy generated from this class after the classification process by the reasoner. The shaded nodes represent ‘equivalent’ classes (i.e. classes defined using *equivalentTo* axioms to express necessary and sufficient conditions for instances to qualify as members). As already suggested, from the perspective of the interface, all the answers are captured within this tree. Hence, the act of answering any query, amounts to traversing the tree using algorithms provided by the reasoner and criteria that needs to be matched. The criteria consists of other classes and properties of interest from the ontology, specified within a configuration file, as shown in Figure 5.14.

For a simple system, limiting the queries as discussed above is sufficient. And as will be discussed in the next chapter, this can be enhanced with a simple rule based dialog driver. A further modification for a complete system would be to use an ontology together with rules to represent dialog interactions. Combined, these two representation formalisms would yield a better solution for deciding how ‘restrictions’ are done and, in general, how dialogs are managed.

```
1 <?xml version="1.0" encoding="utf-8"?>
2 <LocatorPrefs>
3   <OntologyLocation url="http://mathe.rucus.net/ontology/directory.owl"/>
4   <rootCls name="HIVTestProvider"/>
5   <ModerateCostCls name="ModerateCost"/>
6   <LowCostCls name="LowCost"/>
7   <menuOpt1Property name="charges"/>
8   <menuOpt2Property name="hasCloseRadius"/>
9   <valueOfInterestProperty name="hasTelephoneNumValue"/>
10 </LocatorPrefs>
```

Figure 5.14: Listing for Preference Configurations

C) *Unit Testing*

To carry out testing, the entire application was deployed on the application server. The deployment may vary slightly depending on the selected server, but otherwise the process itself is rudimentary, and will not be discussed further here.

In many ways, the testing was similar to the one done for the view. However, particular attention was devoted to the task of verifying that server side processing produced the expected results. That is, steps 5 and 6 captured in Figure 5.7 on page 76 yielded correct results.

As gleaned from the implementation discussion, the bulk of the testing was done using the visual version of the prototype (i.e. with user interface implemented in HTML). When this testing was completed and the controller was verified to work, the voice version of the prototype was implemented. The implementation focused on adapting code for page generation within the controller to render VoiceXML instead of HTML.

5.3 Summary

This chapter provided implementation details for building the basic prototype of HTLS. Similar to the process of building an ontology, a development life cycle model was used to guide the building process. Although the approach followed for building this prototype was experimental in nature, it was simple and systematic. Overall, this made the spiral life cycle model appropriate for use. The next chapter will continue to discuss the implementation of this system. Specifically, the chapter will provide implementation details of a second ‘group’ of iterations used to enhance the basic prototype system.

Chapter 6

Enhanced Implementation of the Prototype System

The previous chapter covered the first iterations in the implementation of the prototype system. For these iterations, the goal was to create a ‘vanilla’ implementation to act as a proof of concept that the user interface (view) and the model can communicate with each other as required. Alternatively put, the goal was to ensure that a user could send a query to the knowledge base, and after due processing receive an answer for their query.

This chapter will discuss the implementation of the ‘second’ group of iterations. For these iterations, the goal (as depicted in Figure 6.1) was to transform the previously implemented prototype into an enhanced final product. The presentation format for the implementation will be the same as the previous chapter; it will include a preamble capturing the design, followed by tool selection, implementation and unit testing segments.

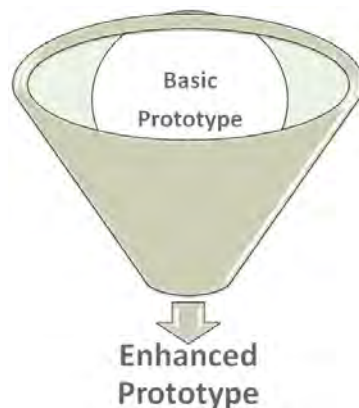


Figure 6.1: Transformation Process: Basic to Enhanced Prototype

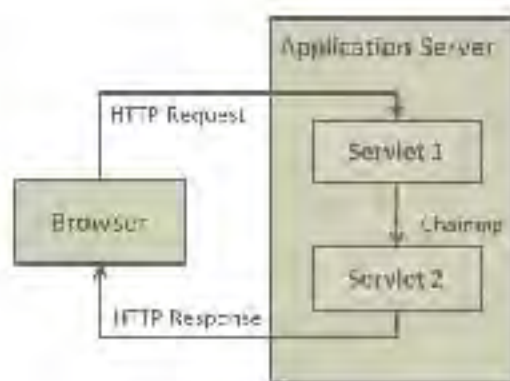


Figure 6.2: Servlet Chaining

6.1 Second Group of Iterations for the Implementation

Having produced a basic functional prototype, the iterations that followed focused on enhancing the overall system to make it more flexible and easy to maintain. The starting point was to have two servlets in lieu of a single servlet to carry out the functions of the controller.

These two servlets were chained together, as shown in Figure 6.2. The first servlet receives search parameters sent via an HTTP request, generates a query to the knowledge base, and forwards the returned results to the second servlet for generating output to the user. This section will focus primarily on the functions of the second servlet. Specific attention will be paid to dialog processing and page generation.

6.1.1 Dialog Processing with Rules

When a query is made, it is typically not known how many results will be returned. What is known is that, beyond a certain range, presenting the results serially to the user ceases to be useful¹. Given this, the question was: how best to present lengthy results to the user, factoring in the serial nature of presentation imposed by acoustical interfaces?

The answer was to either present the results in batches or find some criteria that could be used to shorten the list. The former option would simultaneously waste user's time and lead to short-term memory overload. For this reason, the latter option was preferred.

¹Using Miller's [89] hypothesis (the magical number seven, plus or minus two), the estimated range lies between zero and five.

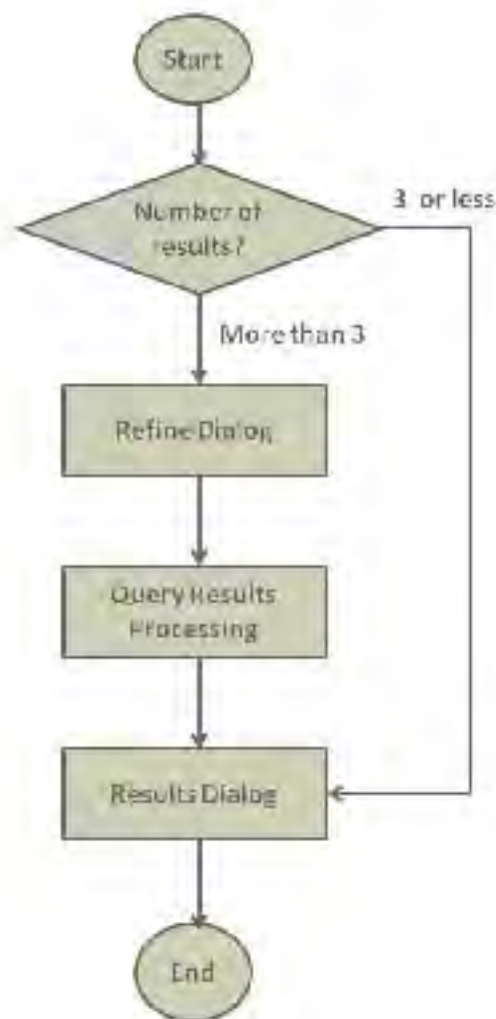


Figure 6.3: Flowchart for Presenting Query Results

Figure 6.3 depicts a flowchart for handling the presentation of query results. As shown in the figure, if the number of results is less than or equal to the threshold value, which is three in this case, the results are presented to the user, i.e. VoiceXML page for the results is generated and delivered to the user. Otherwise a VoiceXML page is generated to ask the user for criteria to be used to refine the results². After the user has provided an answer, the results are refined and presented to the user.

From an implementation point of view, matching criteria for refining the results could be achieved through use of *if...else* statements like as shown in Figure 6.4. However, current trends are moving away from this type of implementation, in favour of explicit rules residing outside the code. The former puts emphasis on ‘how’ to perform a task and

²With increased use, instead of asking the user how to refine the results, a determination could be made via an implementation of some learning mechanism.

```

1  if (request.getAttribute("resultsMap") != null){
2    map = (Map) request.getAttribute("resultsMap");
3    //Render results i.e. present results to user if list is short
4    if (map.size() <= maxOutput){
5      int arrSize = map.size();
6      refine = false;
7      vxmlPage.renderResults(map, out, templatePath, arrSize, refine)
8    }
9    //list is long, generate refinement list
10   else { //(map.size() > maxOutput)
11     dialogSession.setAttribute("storedResultsMap", map);
12     vxmlPage.renderRefine(map, out, templatePath, maxOutput);
13   }
14 }
15 else{ // resultsMap is null
16   if (dialogSession.getAttribute("storedResultsMap") != null){
17     map = (Map) dialogSession.getAttribute("storedResultsMap");
18     String criteriaSelection = request.getParameter("refineChoice");
19     //if-else statements to evaluate criteria for refinement
20     if (criteriaSelection.equalsIgnoreCase("alphabetic")){
21       //Dumping the Map into a TreeMap provides a map sorted automatically by key
22       Map sortedMap = new TreeMap(map);
23       vxmlPage.renderResults(sortedMap, out, templatePath, maxOutput, refine);
24     }
25     else if (criteriaSelection.equalsIgnoreCase("random")){
26       vxmlPage.renderResults(map, out, templatePath, maxOutput, refine);
27     }
28     //provide the unrefined list to user.
29     else {
30       refine = false;
31       vxmlPage.renderResults(map, out, templatePath, map.size(), refine);
32     }
33   }
34 }

```

Figure 6.4: Listing for Results Filtering Using *If-Else* Statements

uses *if* statements. While the latter puts emphasis on ‘what’ task to perform and uses *when* statements.

As cited in [33], this shift from *if* to *when* offers several benefits. The most important benefit is that knowledge can be represented in an easy readable format. As a result, non-technical users can make changes to the knowledge without necessarily engaging, say, a programmer. This is enhanced by the fact that knowledge (business logic) is separated from implementation, which means a change in the knowledge imposes no change in the source code. This separation brings other benefits such as reduced cost of maintenance and increased application adaptability.

In recognition of the above, dialog processing for the refinement of the results was implemented using rules. Implementation details are provided below.

A) Tool Selection

```
1  rule "Criteria 1 – Alphabetic sort "  
2      salience 3  
3      when  
4          $c: Criteria (userSelection == Criteria.CRITERIA_1)  
5          $q: QueryResultMap ($map: resultMap)  
6      then  
7          storeResultsMap = new TreeMap ($map);  
8          $q.setResultMap (storeResultsMap);  
9          $q.setRefined (true);  
10     end
```

Figure 6.5: Listing of a Sample Drools Rule for Performing Alphabetic Sort

Drools (JBoss Rules) was selected to act as the rule engine for driving dialog within the architecture. The reasons for the selection have been discussed in the previous chapter (Subsection 3.3.4).

B) *Implementation*

To replace *if...then* statements using Drools, three critical things were needed. Firstly, a fact model had to be built in order to provide the rule engine with ‘facts’ for making decisions. Technically, a fact model refers to a collection of ‘knowledge objects’ capturing information needed in decision making and the format for its storage [33]. Implicitly, this means that an analysis of required information had to be done as a prerequisite for building the fact model. When this analysis was completed, for the development of the prototype system, two Java classes were created: *Criteria* and *QueryResultMap* classes. These classes were implemented as plain old Java object (POJO) classes with getters and setters (i.e. methods used, respectively, to access and change data within an object).

Secondly, rules had to be authored and stored in some repository (such as the *Apache* web server). Figure 6.5 provides an example of a rule *Criteria 1- Alphabetic sort*. This rule is matched when the user selection is *CRITERIA_1* and the query result map (*QueryResultMap*) exists. The matching process is implemented within Drools and uses the Rete Algorithm. (This algorithm is efficient for comparing large collection of patterns to large collection of objects [51]. Due to its efficiency, this algorithm is used and continues to be used in the implementation of many rule engines.) Once the matching is completed, the rule fires and the query result map is put into a tree, which performs sorting, and the results are stored in the *storeResultsMap* variable. Within the engine, the instance of the query result map (*\$q*) is updated with stored results map and a boolean variable that indicates whether or not the results have been refined is set to true.

Thirdly, a Java class had to be implemented to set up the rule engine (see Appendix

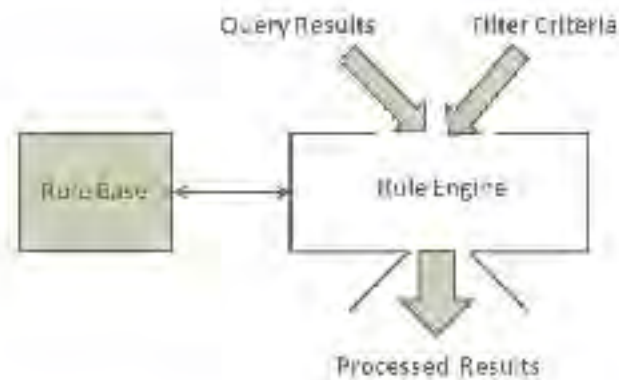


Figure 6.6: Overview of Processing of Query Results by the Rule Engine

```

1 try{
2   Object processedResults [] = runRules.criteriaProcessor(map, criteriaSelection);
3   Map processedMap = (Map) processedResults[0];
4   refined = Boolean.valueOf(processedResults[1].toString());
5   int size = refined ? maxOutput: map.size();
6   vxmlPage.renderResults(processedMap, out, templatePath, size, refined);
7 }
  
```

Figure 6.7: Listing of Code Snippet for Results Filtering Using the Rule Engine

D). This class includes methods that load and process resources needed for the rule engine to carry out its function. Thus, an instance of this class can be regarded as a wrapper object that contains logic for accessing Drools API classes. Implementing this class specifically for the prototype system was, for the most part, a simple exercise. This is because much of the code used was included as part Drools documentation, thus could be reused with few adjustments.

Once the three tasks were completed, all that was required was to implement the logic to activate the rule engine for the refinement of the results. The activation (as captured in the flowchart shown in Figure 6.3) would be triggered by having more than three results returned. As depicted in Figure 6.6, the rule engine would require the results from the query and filter criteria selected by the user as its input. The rule engine would also need to consult the rule base as part of processing. The output results from this processing are passed on as parameters for rendering. The actual code implementing all this is as shown in Figure 6.7. The next subsection will elaborate more on the task of rendering.

C) *Unit Testing*

Testing for this iteration involved two main phases. The first phase focused on validating independently the functionality of the new feature. This entailed verifying

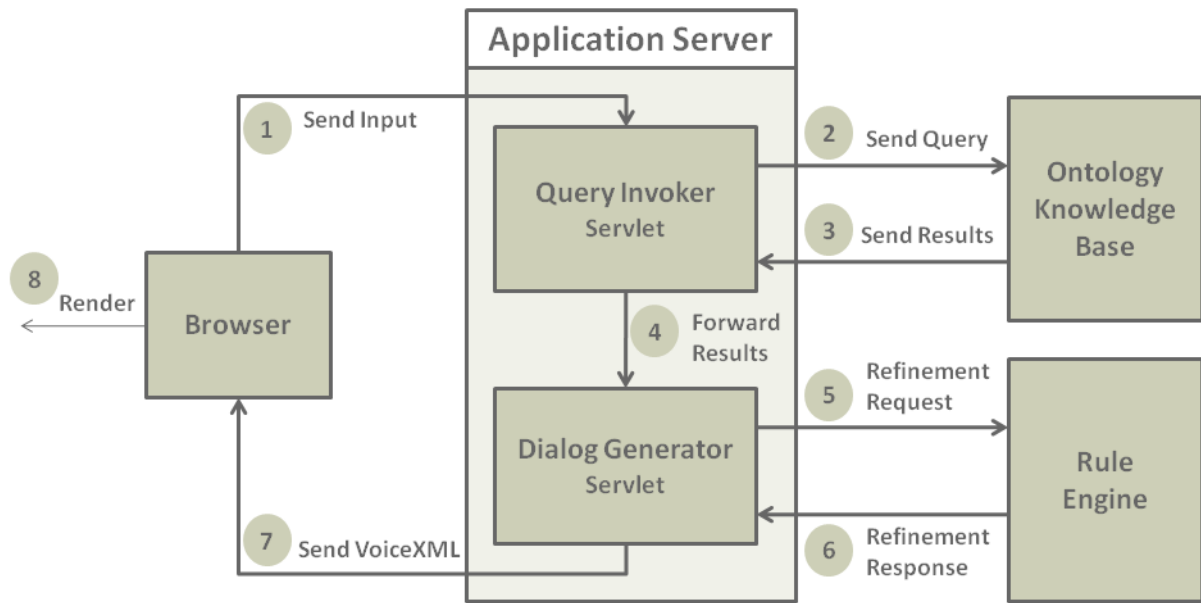


Figure 6.8: Events for Interacting with the Backend

that the authored rules behaved as expected.

The second phase of testing involved validating the added feature within the broad context of the application use. However, as a matter of preference and practicality, the rendering of VoiceXML pages for performing testing was again done using a visual browser (i.e. normal web browser). The preference, as previously suggested, was based on the fact that debugging the application would be faster.

The events underpinning the validation and verification process for the broad functionality of the prototype are captured in Figure 6.8. A brief description detailing each captured event is provided below.

- (1) The browser sends input received from the user for further processing. This input is validated locally before it is submitted using VoiceXML ‘filled’ element³.
- (2) The query invoker servlet receives the input and generates a query to be sent to the knowledge base.
- (3) The knowledge base sends its results back to the query invoker servlet.
- (4) The query invoker servlet forwards the query results to the dialog generator servlet.
- (5) Assuming the results need to be refined, the dialog generator servlet sends a request to the rule engine.

³As aptly stated by Lucas [83], this element “allows input validation code to gain control upon completion of any set of user inputs.”

- (6) The rule engine sends back its response to the dialog generator servlet.
- (7) The dialog generator servlet sends a VoiceXML page to the browser.
- (8) The browser renders the results of the query to user.

Naturally, steps 5-6 may be omitted if the number of returned results is equal or less than the threshold value, as suggested by the flowchart shown in Figure 6.3

6.1.2 Dynamic Page Rendering

In the basic prototype system, `out.println(...)` statements were used to print out each directive for rendering dynamic VoiceXML pages. That is, each VoiceXML statement was printed using `out.println(...)` as shown in Figure 6.9. Whilst this approach works, it blends the view with the controller, and as such, fails to bring about true separation of concerns.

```
1 out.println("<form>");
2 out.println("<block>");
3 out.println("<prompt>");
4 out.println("Provider <value expr = \"providerName\"/>");
5 out.println("Telephone number is <value expr = \"current_tel_number\"/> ");
6 out.println("</prompt>");
7 out.println("</block>");
8 out.println("</form>");
```

Figure 6.9: Listing for Creating a VoiceXML Form for Presenting Provider Details

In light of the above, an alternative approach for dynamically rendering pages was sought. As it will be explained below, this approach leveraged on the work discussed in Subsection 5.2.1.

A) Tool Selection

A common alternative to the use of `out.println(...)` statements for rendering the view when using servlets is to utilise Java Server Pages (JSP). Although JSP remained a viable candidate for use, an investigation into other possible alternatives was carried out.

Based on this investigation, *StringTemplate* [95] was selected. *StringTemplate* is a popular template engine that is able to enforce strict separation of generation logic (model/controller) and output format (view). It is the strict enforcement of the

model-view separation that distinguished *StringTemplate* from JSP and other template engines that were surveyed. (A compilation list of template engines found in [11] was used for the survey that informed the selection.)

B) *Implementation*

Using *StringTemplate* as the view engine allowed VoiceXML pages authored during the implementation of the view (as described in Section 5.2.1) to be utilised for creating generic templates for the dynamic rendering process.

Two main templates were created, one for querying how the results should be refined and the other for presenting the results. Figure 6.10 shows the template for the presentation of the results. The areas within the template surrounded with dollar signs are placeholders for content to be added by *StringTemplate* during application execution.

This content can be obtained by calling other templates as shown on line 26 and 54 of the listing. Alternatively, it can be obtained by writing code that will assign appropriate content to each placeholder (i.e. template variable) once the template has been loaded, like as shown in Figure 6.11. In this listing, the contents of variables *arrayString* and *boolMultiple* are assigned to template variables *populateArray* and *multipleItems* respectively, shown in line 9 and 53 of Figure 6.10.

C) *Unit Testing*

Testing for this iteration was fundamentally the same as discussed for dialog processing iteration. However, primary focus was on the validation of the rendered VoiceXML pages. These were confirmed to yield expected results.

```

1 <?xml version="1.0"?>
2 <vxml version="2.1" xmlns="http://www.w3.org/2001/vxml">
3   <link dtmf="0" event="exit" />
4   <catch event="exit"><goto next="terminate.jsp"/></catch>
5   <script><![CDATA[
6     var provider = new Array();
7     var tel_number = new Array();
8     $populateArray$
9     var listindex = 0;
10    var listlength = provider.length;
11    var current_provider = provider[listindex];
12    var current_tel_number = tel_number[listindex];
13  ]]></script>
14  <catch event="noinput nomatch">
15    <audio src="$audioCatchFile$">Please press * to repeat what was said ,
16    $audioCatchStr$ and Press 0 to exit at anytime</audio>
17  </catch>
18  <form id="start">
19    <block>
20      <prompt>
21        $if(refinePrompt)$
22        Your results have been refined
23        $else$
24        $result_prompt()$
25        $endif$
26      </prompt>
27      <audio src="$audioCatchFile$">Please press * to repeat what was said , $
28      audioCatchStr$ and Press 0 to exit at anytime
29      </audio>
30      <goto next="#navChoice"/>
31    </block>
32  </form>
33  <form id="navChoice">
34    <block><prompt>
35      Provider <value expr = "current_provider"/>
36      Telephone number is <value expr = "current_tel_number"/>
37    </prompt></block>
38    <field name="choice">
39      <grammar type="text/x-grammar-choice-dtmf" mode="dtmf">
40        * {repeat}|0 {exit} $grammarStr$
41      </grammar>
42      <filled>
43        <if cond="choice == 'repeat'">
44          <goto next="#navChoice"/>$fillStr$
45        <elseif cond ="choice =='exit'"/>
46          <goto next="terminate.jsp"/>
47        </if>
48      </filled>
49    </field>
50  </form>
51  $if(multipleItems)$
52    $prev_next_navigation()$
53  $endif$
54 </vxml>

```

Figure 6.10: Listing of VoiceXML Template Page for Results Presentation

```

1 StringTemplateGroup group = new StringTemplateGroup("templateFiles",templatePath);
2 StringTemplate page = group.getInstanceOf(templateName);
3 page.setAttribute("populateArray", arrayString);
4 page.setAttribute("multipleItems", boolMultiple);

```

Figure 6.11: Listing for Initialising Template Variables

6.2 Summary

The overall discussion for the implementation of HTLS was split into two by grouping the iterations of development. This chapter exclusively focused on the discussion of the second group of iterations. Through these iterations, the basic prototype developed from the first group of iterations was enhanced. As reported in the chapter, this was done by integrating two engines into the system: a rule engine to support use of rules for driving the dialog and a rendering engine to support the dynamic generation of VoiceXML pages. The overall integration and use of these engines was tested and confirmed to work as expected.

Chapter 7

System Testing and Discussion

The previous two chapters provided details of the implementation of HTLS; a functional prototype developed to illustrate OVR, the proposed knowledge-driven architecture for building voice applications. This chapter will examine the architecture before proceeding with the discussion on system testing. Additionally, the chapter will provide a critical review of OVR drawing on the integration and implementation experience for developing the prototype service.

7.1 Proposed Architecture: Implementation View

This section will review OVR from the implementation point of HTLS. Figure 7.1 provides an overview of the architecture under scrutiny. This architecture (albeit more elaborate) is the same as the one presented in Figure 1.1 on page 4. (System testing in the next subsection will be presented as proof for functional viability.)

The proposed architecture, OVR, now ‘grounded’ in a specific realisation, has three main components. These are discussed briefly below.

1) **Input/Output Component**

This component includes infrastructure that enables communication between the user and the system. It encompasses the components described in the VoiceXML architectural model, depicted in Figure 2.8 on page 26. As previously stated, these components are essential for deploying VoiceXML based applications. A brief description is provided below for two key components:

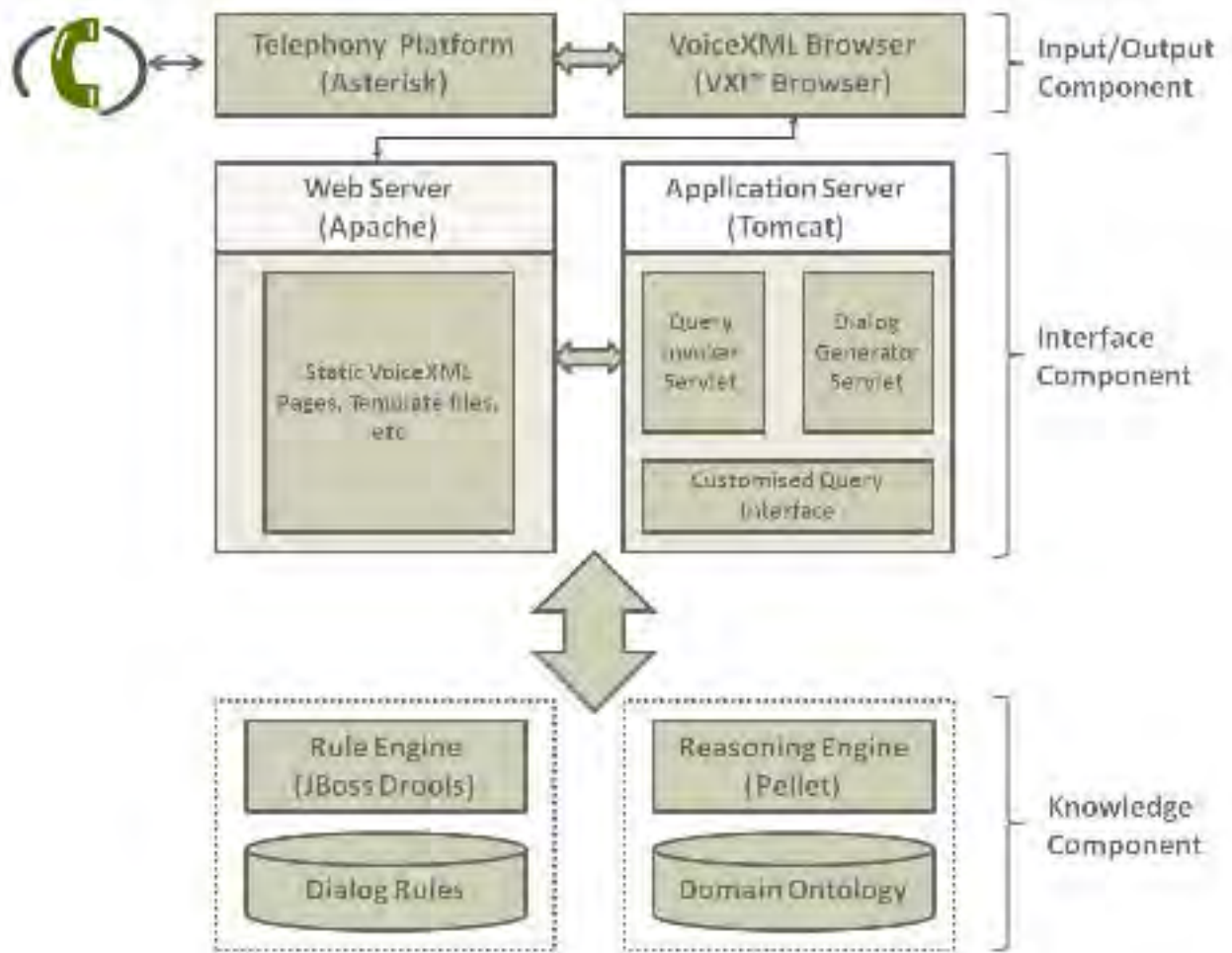


Figure 7.1: Complete Architectural View of HTLS

a) Telephony Platform

The platform is responsible for connecting to the telephony network, converting text to speech, playing audio files and performing other supporting functions for managing or enabling interactions that the user has with the system.

b) VoiceXML Browser

The browser is responsible mainly for facilitating the rendering of VoiceXML pages. It interprets and executes these pages appropriately in concert with the telephony platform.

2) Interface Component

Essentially, this component refers to the application server. The primary task of the application server is to run the voice application and effectively act as an intermediary that allows access to the knowledge. To support this task, the application server hosts the domain logic components, discussed briefly below. Additionally, it hosts resources

required during execution. Specifically, as shown in Figure 7.1, the hosting task for static content is carried out by the web server.

a) *Integration Logic*

The application integration logic layer acts as the glue that binds the view and model together. As reflected in the implementation of the prototype system, this layer was realised using servlets (i.e., query invoker and dialog generator servlets). These servlets embodied the logic that glued various components of the architecture together.

b) *Query Interface*

The query interface interacts directly with the reasoning engine and the knowledge base to facilitate the querying of the ontology. This interface can be customised via a configuration file to limit the scope of the queries to the ontology. (Other than setting the scope defining parameters, the configuration file is used for specifying the name and location of the ontology to be loaded. Effectively, this enables ‘plug and play’ of ontologies into the architecture.)

3) Knowledge Component

This component embodies the knowledge used by the system and mechanism for its access. To this end, it has two main components described below.

a) *Knowledge Bases*

As stressed already, a notable feature of OVR is that it uses knowledge, represented using either ontologies or rules. Accordingly, there are two distinct types of knowledge bases within the architecture. One used as a repository for the rules and the other for maintaining the instantiated ontology.

b) *Reasoning Engines*

The architecture uses reasoning engine(s)¹ to reason and infer new information from its knowledge bases. In the case of the built prototype service, the use supported two specialised tasks: query answering and processing of rules for driving the dialog.

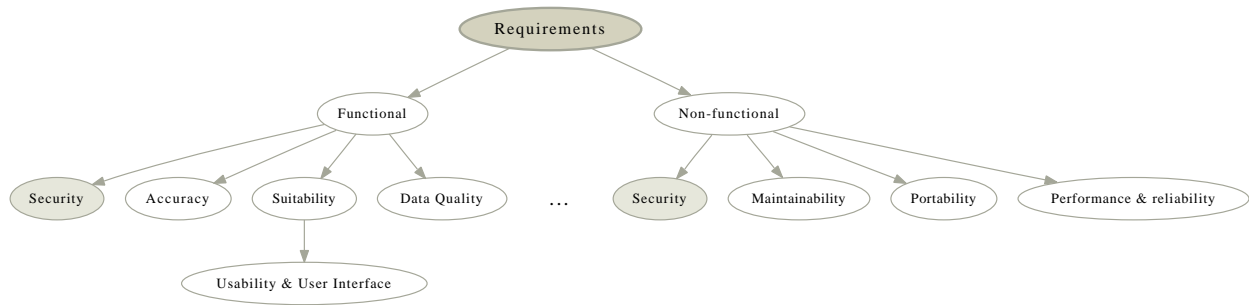


Figure 7.2: Requirements Classification

7.2 System Testing

As already stated, testing was performed for each module during each iteration in order to validate and verify that the individual components implemented were coded correctly and yielded the expected results.

Once all the development was completed, system testing was performed. As highlighted in literature [26, 28], the key objective of system testing is to verify that requirements are met. Figure 7.2 shows a possible classification² of the requirements (synthesised from sources: [26, 27, 28, 72, 81]) that are important to verify. As it can be seen, these requirements can be classified as either functional or non functional.

Functional requirements describe ‘what a system must do’ while non functional requirements describe ‘how well the system must carry out its tasks’ [44, 81]. As suggested, both types of requirements are important for a complete system test. However, for this thesis, system testing focused mainly on verifying compliance to functional requirements (even though testing researchers such as Beizer [26] and Black [28] state unequivocally that a single type of testing is not enough. This decision was based on pragmatic reasons).

Testing was done following a use case driven strategy. As aptly stated by Leffingwell and Widrig [81], “use cases carry the majority of the requirements for [any] system”. This strategy involved calling the service and using scenarios captured in the use case (shown in Figure 5.1) as tests for verifying that key functionality of the system worked.

Other testing techniques were used as part of the strategy. For example, outcome prediction was used to determine accuracy of returned query results. In the event of a mismatch

¹As stated before, it was possible to use just Pellet reasoner as the reasoning engine of choice for performing all required tasks.

²The levels of granularity and abstraction for specifying requirements may differ with context. This yields multiple classifications with nuances that may bring contradictions or overlaps in how some requirements are classified.

	Alternatives			
Input	1	2	3	4
1. Low or Moderate cost?	Low	Low	Moderate	Moderate
2. Near or Far ?	Near	Far	Near	Far
Expected Number of Results	2	3	10	1
Output	↓	↓	↓	↓
• Present results to user	Yes	Yes		Yes
• Refine results			Yes	
• Present results to user				

Table 7.1: Decision Table Used for Testing

between the actual and predicted outcome, error guessing was used. Further, as part of supporting the execution of this strategy, a decision table, as shown in Table 7.1, was used to capture the different combinations of input. Given the low complexity of the example service, all listed inputs from the table were used in the testing process.

The system testing helped to verify that the flow of ‘events’ happened as expected, from the beginning to the end of a transaction by the user yielding the correct results. With the aid of Figure 7.1, this entailed verifying that:

- ⇒ When a call is made, the telephony platform receives and handles the processing details for forwarding the call.
- ⇒ The call is forwarded to the VoiceXML browser, which maps the number called to the URI of the initial document to fetch.
- ⇒ A request is sent to the web server for the initial document (i.e., static VoiceXML page that qualifies as the “welcome” page).
- ⇒ The web server sends the document to the browser, which interprets the document and renders it appropriately using services offered by the telephony platform. Specifically, the result of the interpretation is a welcome message that prompts the user for relevant criteria for locating a suitable HIV testing site.
- ⇒ The user provides suitable criteria via touch tone input, which is forwarded from the telephony platform to the *QueryInvoker* servlet.
- ⇒ A query is generated using the received criteria and is sent to the knowledge base via the reasoning engine.

- ⇒ The results of the query are returned and forwarded to the *DialogGenerator* servlet. This servlet acts as the controller for the rule driven dialog generation process. Depending on the number of returned results, the results are presented to the user or the user may be asked to provide criteria to have the results refined before being presented.
- ⇒ An appropriate VoiceXML page is generated and rendered to the user.
- ⇒ The call is terminated by the user or by the telephony platform if no further requests are made by the user within a preset time.

Also verified was that error handling happened as expected. Further, in order to be reasonably certain about the functional viability of OVR, the verification process of the flow was repeated several times using both valid and invalid input. The repetition was done simply to gain a degree of confidence that behaviour was as expected.

Again, while it is important to heed the advice to perform more than one type of testing, the system developed in this thesis was tested only functionally. Usability testing would have required an amount of extra work that was not justifiable within the limits of a traditional thesis work for an MSc. As an aside, it could have been performed by the counterpart in the HEAIDS project, but as described earlier in this write-up, that cooperation had to come to an end for reasons extrinsic to the thesis.

7.3 Integration and Implementation Experience

This section will relate briefly the experience of integrating the various components together to implement the prototype. Fundamental to the integration process was the need to understand the functionality of each component used. Inherently, this means there was a learning curve associated with the use of these components.

As it would be expected, the learning curve differed from component to component. On the average, it was reasonable for many of the components. In part, this could be attributed to the actual choice of the components used (the selection was discussed in Chapter 3 on page 31). Each component, in its own right, provided impressive documentation and support resources to enable relatively easy use and to help with problem resolution in a timely manner.

The effort for implementing the logic for the prototype was also reasonable, albeit a bit tedious with regard to ontology construction. As stated before, for an artefact to qualify as an ontology, it takes more than just using an appropriate implementation language; the underlying conceptualisation has to be ratified as representative of the intended use. This takes time, patience (and to a degree, luck in coping with human dynamics).

It is also worth pointing out that the learning curve for ontology construction for anyone without a strong background in logic may be steep. Any misconceptions may lead to the implementation of ontologies with very high worst case complexity, which translate in slow response times and often high memory requirements.

Still, it is important to note that the gap between learning and applying what is learnt can be bridged through the use of design criteria such as provided in Subsubsection 2.1.4.6 on page 17 and simple practical guidelines like verifying that non overlapping concepts are explicitly declared as disjoint and different individuals are asserted as different.

In summarising all of the above, the effort required for implementing the prototype service using OVR architecture is reasonable, and by no means insurmountable. The availability of documentation and community support means developers can be assured of relatively fast assistance when a problem is encountered.

7.4 Potential Barriers for Proposed Architecture Adoption

Although the scope of this thesis did not include extensive testing of OVR, it can be argued that its immediate use lies within the realm of possibility. The components used are readily available, functional testing supports its validity and, as indicated, the overall integration and implementation experience is reasonable.

However, there are challenges that may act as barriers for the adoption of the proposed architecture. This subsection will discuss two major challenges that have been identified. On reflection, the first challenge is easily solvable, while the second is more serious but solvable hopefully over time.

7.4.1 Demand for Wide Range of Competencies

OVR draws on a wide range of competencies. This is a challenge because expecting one individual to possess all the required competencies is not realistic or practical. And having more individuals has cost implications, which for broad adoption may require justification. However, one should realise that in a real production environment this is not much of an issue since development is carried by a team of people.

7.4.2 Availability of Ontologies

The use of ontologies and reasoning tools plays a paramount role in the architecture. Ontologies provide a semantically rich formalism that can be harnessed by reasoning tools to finally help advance the idea of building inference based dialog systems. For example, as discussed in [30], these systems can be effective in managing the direction of how dialogs must proceed in that, through use of semantic representation, logical inferences can be drawn to determine: what action to perform, how to answer a question, and how to resolve any possible ambiguities between dialog participants.

Notwithstanding the above, it is important to point out the caveats associated with the use of ontologies. They require careful and meticulous design, which allows for intelligent reasoning without necessarily entailing strong ontological commitments. To attain this delicate balance is still a challenge. Thus, from a short term perspective, it is important to stress that until ontologies are readily available, their use has to be weighed with the effort (or cost) that goes into their development. In other words, one should stress that:

“To achieve wide-spread use of ontologies, they have to be *established* (emphasis added) as usable software artifacts [sic] that are interchanged and traded between parties, similar to computer programs or other forms of electronic content” [57].

In order to establish a “wide-spread use of ontologies”, development of ontologies has to be encouraged. On the one hand, to help overcome any challenges that make development ‘burdensome’. On the other hand, to increase their availability so that they eventually become trivially simple entities which can be plugged to backends of knowledge oriented systems.

7.5 Summary

This chapter reviewed the components of OVR. This was done as a preamble to the discussion on system testing, and the broad analysis of the architecture. Although OVR was deemed to be functionally viable, a few issues were identified that could impact its adoption.

A key issue for adoption is the availability of ontologies. As alluded to, the functionality of OVR is dependent on the quality of the ontology it uses. Given that ontologies are still being established for “wide-spread use”, this concern about quality will actually diminish as more and more work goes into improving development strategies and efficiency mechanisms for using ontologies. Anecdotally, this will happen as an imperative for their adoption.

Chapter 8

Conclusion

Many telephony services that exist today are database driven. However, with the increase in the adoption of ontologies and accuracy of speech recognition technologies, this is certain to change. In computing, the change is inextricably intertwined with the trends. The change is driving the trends, and at the same time, the trends are driving the change.

As part of anticipating change in the telephony industry, this thesis has interpreted trends in computing in order to propose an architecture that might be used to build voice applications. Instrumental to the proposal were trends that suggest a move from imperative to declarative programming, and those that signify a move towards broad adoption of ontologies in any system that enables knowledge and/or information exchange. The former is linked to the use of business rules as opposed to *if...else* statements in order to separate knowledge from implementation. The latter is tied to the shift from databases to knowledge bases augmented with reasoning engines. Overall, these trends acted as pillars for the proposed architecture, which delivers voice services to the users by also capitalising on the benefits of the converged network and the ever increasing computational power of devices.

8.1 Thesis Synopsis

This thesis presented an architecture that illustrates how next generation IVR systems might be implemented. Based on the current trends, the first point was that these systems are likely to use VoiceXML specification, due to the growing endorsement of VoiceXML as a ‘standard and de facto language’ for implementing voice applications. Further, these

systems might, at a minimum, use explicitly represented knowledge and reasoning engines in order to interpret and respond to user requests intelligently (i.e. through appropriate inferences). Ontologies, as reflected by the trends, are likely to be used for knowledge representation as they provide a semantically rich formalism that can be exploited with ease by many reasoning engines.

From these considerations, the core components of the architecture were identified. To illustrate how these components interact with each other and perform an initial validation of the architecture, a prototype service was implemented.

The implementation of the prototype service was done through two distinct phases, each with several iterations. The first phase focused on the construction of the domain ontology, the details are reported in Chapter 4. The second phase focused on the implementation of two main components: 1) the integration and application logic, and 2) the user interface. These components were implemented incrementally in an iterative manner, with the goal of generating a simple working prototype that could later be enhanced. In alignment with the goal, the discussion for this phase was split in this thesis by grouping iterations based on two key milestone achievements: the production of the initial and the enhanced prototypes.

Finally, system testing was performed on the enhanced prototype, confirming the functional validity of OVR.

8.2 Achievements

This thesis had two main objectives. The first was to propose an architecture for building voice applications that offered an improved user experience and interpreted current trends in computing. The second, was to build a functional prototype service to provide an initial validation of OVR, the proposed architecture.

Both objectives were met. For this reason, the thesis has indeed achieved its mandate. It has provided an architecture for voice services that are standard based, has the potential to improve the user experience and is aligned to current trends. Further, in developing a functional prototype, this thesis has helped Rhodes University in its effort to encourage members of its community to test for HIV. As reported in [109], when it comes to management of AIDS:

“... death is not seen as the problem: being seen to be sick or being observed going for voluntary testing and counseling is identified as the real problem”.

For this reason, any initiative that may help people to be comfortable with the idea of testing and knowing their status is significant.

8.3 Future Work

A number of suggestions can be made for future improvements of the proposed architecture. One suggestion would be to upgrade the view to use *VoiceXML 3.0* [86], which is the next major release of VoiceXML (to be finalised as a complete specification early in 2011). According to the latest specification draft document [86], *VoiceXML 3.0* is designed exclusively as a presentation language. Thus, the backend of the proposed architecture can still be used as is, because the backend from the specification point of view is seen a separate entity.

Another suggestion would be to validate the proposed architecture from other perspectives by performing, for example, usability and performance testing. In order to yield meaningful results from these tests, the following might be considered:

- ⇒ Building a voice application over the proposed architecture that uses speech recognition technologies and sophisticated rules together with a specialised ontology for user interactions. This ontology could also be instrumented using a learning formalism to make user interactions more intelligent. For example, Bayesian statistics, which provides a formalism for combining learning with explicit modelling could be used.
- ⇒ Distributing the various components of the backend to run on separate servers to determine how the distribution may impact performance and user experience.

To realise the value of these amendments, the complexity of such an application would need to be high to allow for example, dialogs to be adapted accordingly if it is detected that the user is getting upset. Essentially, the complexity should need to move beyond user requests that focus on simple information retrieval to requests that factor in multiple parameters and constraints in processing a response for the user.

8.4 Summary

As a conclusion to the work done in this thesis, this chapter has synthesised the important trends that have influenced the design of OVR, the proposed architecture for building knowledge rich voice applications. This architecture is rooted in the branch of AI that deals with expert systems in that it uses reasoning engines to derive answers from knowledge bases.

At the same time, the architecture is aligned to the Semantic Web approach of building services in that it uses ontologies to explicitly represent knowledge. As deduced from current trends, this is vital for achieving automation that may make human to computer communication as seamless as possible within a domain, taking into account Polanyi's argument that "a wholly explicit knowledge is unthinkable" (as cited in [67]). That is, whilst the gap in communication between the two may be reduced, it will continue to exist because complete semantic understanding is not possible without tacit knowledge. Hence, the only question that can ever be asked is: will the gap be reduced substantially enough to pass the definitive test¹ of intelligence proposed by Alan Turing [111]?

¹The test attempts to determine whether responses from a computer with intelligent behaviour are comparable to responses from a human.

List of References

- [1] Apache Tomcat. Online: <http://tomcat.apache.org/> [Last accessed: 8 April2008].
- [2] Asterisk. Online: <http://www.asterisk.org/> [Last accessed: 8 April 2008].
- [3] Behavior Change Communication for HIV/AIDS. Online: <http://www.fhi.org/en/hivaids/pub/fact/bcchiv.htm> [Last accessed: 8 April 2008].
- [4] Drools - Business Logic integration Platform. Online: <http://jboss.org/drools> [Last accessed: 1 April 2010].
- [5] Hammurapi Group. Online: <http://www.hammurapi.biz/hammurapi-biz/ef/xmenu/hammurapi-group/products/hammurapi-rules/index.html> [Last accessed: 17 Nov 2009].
- [6] HIV-911 Programme. Online: <http://www.hiv911.org.za/>[Last accessed: 16 June 2009].
- [7] HIV Prevention. Online: http://data.unaids.org/pub/FactSheet/2008/20080527_fastfacts_testing_en.pdf [Last accessed: 25 March 2009].
- [8] Jess. Online: <http://www.jessrules.com/> [Last accessed: 17 Nov 2009].
- [9] JLisa. Online: <http://jlisa.sourceforge.net/> [Last accessed: 17 Nov 2009].
- [10] Knowledge. Online: <http://www.webster-dictionary.org/definition/knowledge> [Last accessed: 2 Dec 2009] .
- [11] Open Source Template Engines in Java. Online: <http://java-source.net/open-source/template-engines> [Last accessed: 9 Nov 2010].
- [12] Pellet: OWL 2 Reasoner for Java. Online: <http://clarkparsia.com/pellet/> [Last accessed: 8 April2008].

-
- [13] Protégé. Online: <http://protege.stanford.edu/> [Last accessed: 8 April 2008].
- [14] SweetRules. Online: <http://sweetrules.projects.semwebcentral.org/> [Last accessed: 17 Nov 2009].
- [15] Text-to-Speech: Speech Synthesis. Online: <http://www.i6net.com/purchase/text-to-speech-tts/> [Last accessed: 23 Mar 2010].
- [16] The Apache Software Foundation. Online: <http://www.apache.org/> [Last accessed: 8 April 2008].
- [17] The OWL API. Online: <http://owlapi.sourceforge.net/> [Last accessed: 9 October 2009].
- [18] The Spiral Model. Online: <http://scitec.uwichill.edu.bb/cmp/online/cs221/spiralmodel.htm> [Last accessed: 10 February 2010]. Copyright © Adrian Als & Charles Greenidge, 2003.
- [19] Validator for XML Documents. Online: <http://www.validome.org/xml/> [Last accessed: 9 Feb 2008].
- [20] VXI* VoiceXML Browser. Online: <http://www.i6net.com/> [Last accessed: 23 Mar 2010].
- [21] VXI* VoiceXML Browser Manual. Online: <http://www.i6net.com/support/documents/> [Last accessed: 9 Feb 2008].
- [22] Higher Education HIV/AIDS (HEAIDS) Programme Proposal. Online: <http://campus.ru.ac.za/download.php?actionarg=8881> [Last accessed: 9 May 2008], 2007.
- [23] ABIDI, S. S. R. Healthcare Knowledge Management: The Art of the Possible. In *K4CARE* (2007), D. R. no, Ed., vol. 4924 of *Lecture Notes in Computer Science*, Springer, pp. 1–20.
- [24] BAADER, F., HORROCKS, I., AND SATTLER, U. Description Logics. In *Handbook of Knowledge Representation*, F. van Harmelen, V. Lifschitz, and B. Porter, Eds. Elsevier, 2007.
- [25] BASILI, V. R., AND TURNER, A. J. Iterative Enhancement: A Practical Technique for Software Development. *IEEE Trans. Software Eng.* 1, 4 (1975), 390–396.

- [26] BEIZER, B. *Black-Box Testing: Techniques for Functional Testing of Software and Systems*, 1 ed. Verlag John Wiley & Sons, Inc., New York, NY, USA, 1995.
- [27] BELL, D., MORREY, I., AND PUGH, J. *Software Engineering: A Programming Approach*. Prentice Hall International (UK) Ltd., Hertfordshire, UK, UK, 1987.
- [28] BLACK, R. *Pragmatic Software Testing: Becoming an Effective and Efficient Test Professional*. John Wiley & Sons, Inc., New York, NY, USA, 2007.
- [29] BOEHM, B. A Spiral Model of Software Development and Enhancement. *SIGSOFT Softw. Eng. Notes* 11, 4 (1986), 14–24.
- [30] BOS, J., AND OKA, T. An Inference-based Approach to Dialogue System Design. In *COLING* (2002).
- [31] BRACHMAN, R., AND LEVESQUE, H. *Knowledge Representation and Reasoning (The Morgan Kaufmann Series in Artificial Intelligence)*. Morgan Kaufmann, May 2004.
- [32] BREWSTER, C., AND O’HARA, K. Knowledge Representation with Ontologies: Present Challenges – Future Possibilities. *International Journal of Human-Computer Studies* 65, 7 (July 2007), 563–568.
- [33] BROWNE, P. *JBoss Drools Business Rules*. Packt Publishing, 2009.
- [34] BRUSA, G., CALIUSCO, M. L., AND CHIOTTI, O. A Process for Building a Domain Ontology: an Experience in Developing a Government Budgetary Ontology. In *AOW ’06: Proceedings of the second Australasian workshop on Advances in ontologies* (Darlinghurst, Australia, Australia, 2006), Australian Computer Society, Inc., pp. 7–15.
- [35] CORCHO, O., FERNÁNDEZ-LÓPEZ, M., GÓMEZ-PÉREZ, A., AND LÓPEZ-CIMA, A. Building Legal Ontologies with METHONTOLOGY and Webode. In *Law and the Semantic Web* (2005), no. 3369, Springer-Verlag, pp. 142–157.
- [36] CORKREY, R., AND PARKINSON, L. Interactive Voice Response: Review of Studies 1989-2000. *Behavior Research Methods, Instruments, & Computers* 34 (2002), 342–353.
- [37] CORKREY, R., PARKINSON, L., AND BATES, L. Pressing the Key Pad: Trial of a Novel Approach to Health Promotion Advice. *Preventive Medicine* 41 (August 2005), 657–666.

- [38] COULTER, A. Paternalism or Partnership? Patients have grown up—and there's no going back. *BMJ* 319 (1999), 719–720.
- [39] CRAWFORD, A. G., SIKIRICA, V., GOLDFARB, N., POPIEL, R. G., PATEL, M., WANG, C., CHU, J. B., AND NASH, D. B. Interactive Voice Response Reminder Effects on Preventive Service Utilization. *American Journal of Medical Quality* 20 (2005), 329–336.
- [40] DAVIES, J., VAN HARMELEN, F., AND FENSEL, D. *Towards the Semantic Web: Ontology-driven Knowledge Management*. John Wiley & Sons, Inc., New York, NY, USA, 2002.
- [41] DAVIS, R., SHROBE, H. E., AND SZOLOVITS, P. What is a Knowledge Representation? *AI Magazine* 14, 1 (1993), 17–33.
- [42] DEAN, D. H. What's Wrong with IVR Self-Service. *Managing Service Quality* 18, 6 (2008), 594–609.
- [43] DRAGAN, G., DRAGAN, D., AND VLADAN, D. *Model Driven Architecture and Ontology Development*. Springer, 2006.
- [44] EIDE, P. L. H. Quantification and Traceability of Requirements. Tech. rep., NTNU Norwegian University of Science and Technology, 2005.
- [45] EYSENBACH, G. Consumer Health Informatics. Online: <http://www.bmj.com/cgi/content/full/320/7251/1713> [Last accessed: 8 April 2008].
- [46] FENSEL, D. *Ontologies: A Silver Bullet for Knowledge Management and Electronic Commerce*. Springer-Verlag New York, Inc., Secaucus, NJ, USA, 2003.
- [47] FENSEL, D., LAUSEN, H., POLLERES, A., DE BRUIJN, J., STOLLBERG, M., ROMAN, D., AND DOMINGUE, J. *Enabling Semantic Web Services: The Web Service Modeling Ontology*. Springer, 2007.
- [48] FERNÁNDEZ-LÓPEZ, M., GÓMEZ-PÉREZ, A., AND AMAYA, M. D. R. Ontology's Crossed Life Cycles. In *EKAW '00: Proceedings of the 12th European Workshop on Knowledge Acquisition, Modeling and Management* (London, UK, 2000), Springer-Verlag, pp. 65–79.
- [49] FERNÁNDEZ-LÓPEZ, M., GÓMEZ-PÉREZ, A., AND JURISTO, N. METHONTOLOGY: From Ontological Art Towards Ontological Engineering. In *Proceedings of the AAAI97 Spring Symposium* (Stanford, USA, March 1997), pp. 33–40.

- [50] FERRANS, J., PORTER, B., TRYPHONAS, S., LUCAS, B., DANIELSEN, P., BURNETT, D. C., HUNT, A., MCGLASHAN, S., REHOR, K., AND CARTER, J. Voice Extensible Markup Language (VoiceXML) Version 2.0. W3C Recommendation, W3C, Mar. 2004. Online: <http://www.w3.org/TR/2004/REC-voicexml20-20040316/> [5 May 2008].
- [51] FORGY, C. L. Rete: A fast algorithm for the many pattern/many object pattern match problem. *Artificial Intelligence* 19, 1 (1982), 17 – 37.
- [52] GARDINER, T., HORROCKS, I., AND TSARKOV, D. Automated Benchmarking of Description Logic Reasoners. In *Description Logics* (2006).
- [53] GEORGILA, K., TSOPANOGLIOU, A., FAKOTAKIS, N., AND KOKKINAKIS, G. A dialogue system for telephone-based services integrating spoken and written language. In *Interactive Voice Technology for Telecommunications Applications, 1998. IVTTA '98. Proceedings. 1998 IEEE 4th Workshop* (Sept. 1998), pp. 55 –59.
- [54] GILB, T. *Principles of Software Engineering Management*. Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA, 1988.
- [55] GRAU, B. C., HORROCKS, I., MOTIK, B., PARSIA, B., PATEL-SCHNEIDER, P. F., AND SATTLER, U. OWL 2: The Next Step for OWL. *Journal of Web Semantics* 6, 4 (2008), 309–322.
- [56] GREINER, R., DARKEN, C., AND SANTOSO, N. I. Efficient Reasoning. *ACM Comput. Surv.* 33, 1 (2001), 1–30.
- [57] GRIMM, S., HITZLER, P., AND ABECKER, A. *Knowledge Representation and Ontologies Logic, Ontologies and Semantic Web Languages*. Springer-Verlag, 2007, ch. 3, pp. 51–106.
- [58] GRUBER, T. R. Towards Principles for the Design of Ontologies Used for Knowledge Sharing. In *Formal Ontology in Conceptual Analysis and Knowledge Representation* (Deventer, The Netherlands, 1993), N. Guarino and R. Poli, Eds., Kluwer Academic Publishers.
- [59] GRÜNINGER, M., AND FOX, M. S. Methodology for the Design and Evaluation of Ontologies. In *IJCAI95 Workshop on Basic Ontological issues in Knowledge Management Sharing* (1995), D. Skuce, Ed.

- [60] GUARINO, N. Semantic Matching: Formal Ontological Distinctions for Information Organization, Extraction, and Integration. In *Information Extraction: A Multidisciplinary Approach to an Emerging Information Technology* (1997), M. T. Paziienza, Ed., Springer Verlag, pp. 139 – 170.
- [61] GUARINO, N. Formal Ontology and Information Systems. In *Proceedings of the 1st International Conference on Formal Ontologies in Information Systems (FOIS'98)* (Trento, Italy, 1998), IOS Press, pp. 3–15.
- [62] GUARINO, N., AND GIARETTA, P. Ontologies and Knowledge Bases: Towards a Terminological Clarification. In *Towards Very Large Knowledge Bases: Knowledge Building and Knowledge Sharing* (1995), pp. 25–32.
- [63] GÓMEZ-PÉREZ, A., FERNÁNDEZ-LÓPEZ, M., AND CORCHO, O. *Ontological Engineering: With Examples from the Areas of Knowledge Management, e-Commerce and the Semantic Web*. Springer, 2004.
- [64] GÓMEZ-PÉREZ, N. J. A., AND PAZOS, J. Evaluation and Assessment of Knowledge Sharing Technology. In *Towards Very Large Knowledge Bases* (1995), N. J. Mars, Ed., IOS Press, pp. 289–296.
- [65] HAARSLEV, V., AND MÖLLER, R. RACER System Description. In *Automated Reasoning, First International Joint Conference, IJCAR 2001* (Siena, Italy, June 2001), R. Goré, A. Leitsch, and T. Nipkow, Eds., vol. 2083 of *Lecture Notes in Computer Science*, Springer.
- [66] HEISLER, M., AND PIETTE, J. D. "I Help You, and You Help Me": Facilitated Telephone Peer Support Among Patients With Diabetes. *The Diabetes Educator* 31 (2005), 869–879.
- [67] HOLSAPPLE, C. W. *Handbook on Knowledge Management 1: Knowledge Matters*, vol. 1. Springer-Verlag Berlin Heidelberg New York, Inc., 2003.
- [68] HORRIDGE, M., AND BECHHOFFER, S. Igniting the OWL 1.1 Touch Paper: The OWL API. In *OWLED* (2007), vol. 258 of *CEUR Workshop Proceedings*.
- [69] HORRIDGE, M., AND BECHHOFFER, S. The OWL API: A Java API for Working with OWL 2 Ontologies. In *OWLED* (2009).
- [70] HORRIDGE, M., JUPP, S., MOULTON, G., RECTOR, A., STEVENS, R., AND WROE, C. *A Practical Guide To Building OWL Ontologies Using Protegé 4 and CO-ODE Tools*, 1.1 ed. Manchester University, 2009. Edition 1.1.

- [71] HORROCKS, I. Ontologies and databases. *Semantic Days 2008*. Stavanger, Norway, April 2008. Online: <http://www.comlab.ox.ac.uk/ian.horrocks/Seminars/download/onto-db.ppt> [Last accessed: 17 August 2010].
- [72] ISO/IEC. *ISO/IEC 9126. Software engineering – Product quality*. ISO/IEC, 2001.
- [73] ITU. Ubiquitous Network Societies: Their Impact on the Telecommunication Industry. In *Background Paper, International Telecommunication* (2005).
- [74] JENA. Jena – a Semantic Web Framework for Java. Online: <http://jena.sourceforge.net/> [Last accessed: 8 April 2008].
- [75] KING, A. Constructing a Low-Cost, Open-Source, VoiceXML Gateway. Master’s thesis, Rhodes University, 2007.
- [76] KING, A., TERZOLI, A., AND CLAYTON, P. Creating a Low Cost VoiceXML Gateway to Replace IVR Systems for Rapid Deployment of Voice Applications. In *Southern African Telecommunications, Networks and Applications Conference (SATNAC)* (September 2006), pp. 131–136.
- [77] KRASNER, G., AND POPE, S. A Description of the Model-View-Controller User Interface Paradigm in the Smalltalk-80 System. *J. Object Oriented Programming* 3, 1 (August 1988), 26–49.
- [78] LAM, T. H. W., AND LEE, R. S. T. iJADE FreeWalker-An Intelligent Ontology Agent-based Tourist Guiding System. *Studies in Computational Intelligence (SCI)* 72 (2007), 103–125.
- [79] LANDAUER, C., AND BELLMAN, K. Some measurable characteristics of intelligent computing systems. In *Systems, Man, and Cybernetics, 2000 IEEE International Conference on* (2000), vol. 3, pp. 2086 –2091 vol.3.
- [80] LEE, H., FRIEDMAN, M. E., CUKOR, P., AND AHERN, D. Interactive Voice Response System (IVRS) in Health Care Services. *Nursing Outlook* 51 (2003), 277–283.
- [81] LEFFINGWELL, D., AND WIDRIG, D. *Managing Software Requirements: A Use Case Approach*. Pearson Education, 2003.
- [82] LIEBIG, T. Reasoning with OWL – System Support and Insights. Tech. Rep. TR-2006-04, Ulm University, Ulm, Germany, September 2006.

- [83] LUCAS, B. VoiceXML for Web-based Distributed Conversational Applications. *Communications of the ACM* 43, 9 (2000), 53–57.
- [84] MAEMA, M., TERZOLI, A., AND DALVIT, L. Ontologies for Provision of HIV/AIDS Information Using Telephony Infrastructure. In *Annual Conference of the South African Institute of Computer Scientists and Information Technologists (SAICSIT 2008)* (2008).
- [85] MASSIE, T., OBRST, L., AND WIJESSEKERA, D. TVIS: Tactical Voice Interaction Services for Dismounted Urban Operations. In *IEEE Military Communications Conference (MILCOM)* (2008).
- [86] MCGLASHAN, S., BURNETT, D. C., AKOLKAR, R., AUBURN, R., BAGGIA, P., BARNETT, J., BODELL, M., CARTER, J., OSHRY, M., REHOR, K., YOUNG, M., AND HOSN, R. Voice Extensible Markup Language (VoiceXML) 3.0. Online: <http://www.w3.org/TR/2010/WD-voicexml30-20100304/> [Last accessed: 2 May 2010].
- [87] MCGUINNESS, D. L., AND VAN HARMELEN, F. OWL Web Ontology Language Overview. Tech. Rep. REC-owl-features-20040210, W3C, 2004.
- [88] MIGNEAULT, J. P., FARZANFAR, R., WRIGHT, J. A., AND FRIEDMAN, R. H. How to Write Health Dialog for a Talking Computer. *Journal of Biomedical Informatics* 35, 5 (October 2006), 468–481.
- [89] MILLER, G. A. The Magical Number Seven, Plus or Minus Two: Some Limits on Our Capacity for Processing Information. *Psychological Review* 63 (1956), 81–97.
- [90] MOTIK, B., SHEARER, R., AND HORROCKS, I. Optimized Reasoning in Description Logics using Hypertableaux. In *Proc. of the 21st Conference on Automated Deduction (CADE-21)* (Bremen, Germany, July 17–20 2007), F. Pfenning, Ed., vol. 4603 of *LNAI*, Springer, pp. 67–83.
- [91] NAYLOR, M., KEEFE, F., BRIGIDI, B., NAUD, S., AND HELZER, J. Therapeutic interactive voice response for chronic pain reduction and relapse prevention. *Pain* 134 (2007), 335–345.
- [92] NICKOLS, F. W. The "Knowledge" in Knowledge Management. In *The knowledge management yearbook 2000-2001* (Boston, USA, 2000), J. Cortada and J. Woods, Eds., Butterworth-Heinemann, pp. 12–21.

- [93] NOY, N., AND MCGUINNESS, D. *Ontology Development 101: A Guide to Creating Your First Ontology*. Tech. rep., Stanford University, 2001.
- [94] NOY, N., AND RECTOR, A. *Defining N-ary Relations on the Semantic Web*. Tech. rep., W3C World Wide Web Consortium, 2006. W3C Working Group Note 12 April 2006, Online: <http://www.w3.org/TR/swbp-n-aryRelations/> [1 June 2009].
- [95] PARR, T. J. *A Functional Language For Generating Structured Text*. Tech. rep., University of San Francisco, 2006. Online: <http://www.cs.usfca.edu/~parrrt/papers/ST.pdf> [Last accessed: 5 March 2010].
- [96] PENTON, J., AND TERZOLI, A. iLanga: A Next Generation VOIP-based, TDM-enabled PBX. In *Southern African Telecommunications Networks and Applications Conference (SATNAC)* (2004).
- [97] PIERACCINI, R., AND HUERTA, J. Where do we go from here? Research and commercial spoken dialog systems. In *Proceedings of the 6th SIGdial Workshop on Discourse and Dialogue* (Lisbon, Portugal, September 2005).
- [98] PINTO, H. S., AND MARTINS, J. A. P. Ontologies: How can They be Built? *Knowledge and Information Systems* 6, 4 (July 2004), 441–464.
- [99] POLANYI, M. II. Knowing and Being. *Mind* LXX, 280 (1961), 458–470.
- [100] POLANYI, M. *The Tacit Dimension*. Doubleday, Garden City, NY, 1966.
- [101] PÉREZ-JIMÉNEZ, M. J., AND ROMERO-CAMPERO, F. A CLIPS Simulator for Recognizer P Systems with Active Membranes. In *Proceedings of the Brainstorming week on Membrane Computing* (Sevilla, España, 2004), G. P. A. R. N. Álvaro Romero Jiménez; Fernando Sancho Caparrini, Ed., pp. 387–413.
- [102] ROUILLARD, J. *Multimodality in Mobile Computing and Mobile Devices: Methods for Adaptable Usability*. IGI Global, 2009, ch. Multimodal and Multichannel Issues in Pervasive and Ubiquitous Computing, pp. 1–23.
- [103] ROYCE, W. W. Managing the Development of Large Software Systems: Concepts and Techniques. In *Proc. IEEE WESTCON* (August 1970), pp. 1–9. Reprinted in *Proceedings of the Ninth International Conference on Software Engineering*, March 1987, pp.328–338.
- [104] SCHOELLER, A. Voice Self-Service Aligns with Web Architectures to Reduce TCO, Increase Flexibility and Ensure Content Consistency. Tech. rep., YANKEE Group,

2006. Online: http://www.cisco.com/en/US/prod/collateral/voicesw/custcosw/ps5694/ps1006/prod_white_paper0900aecd8051711b.pdf [Last accessed: 7 May 2010].
- [105] SIRIN, E., PARSIA, B., GRAU, B. C., KALYANPUR, A., AND KATZ, Y. Pellet: A Practical OWL-DL Reasoner. *Journal of Web Semantics* 5, 2 (2007), 51–53.
- [106] SUÁREZ-FIGUEROA, M. C., AND GÓMEZ-PÉREZ, A. Building Ontology Networks: How to Obtain a Particular Ontology Network Life Cycle? In *Proceedings of the International Conference on Semantic Systems (ISEMANICS08)* (Graz, Austria, September 2008).
- [107] SUHM, B., BERS, J., MCCARTHY, D., FREEMAN, B., GETTY, D., GODFREY, K., AND PETERSON, P. A Comparative Study of Speech in the Call Center: Natural Language Call Routing vs. Touch-tone Menus. In *CHI '02: Proceedings of the SIGCHI conference on Human factors in computing systems* (New York, NY, USA, 2002), ACM, pp. 283–290.
- [108] SUHM, B., AND PETERSON, P. Call Browser: A System to Improve the Caller Experience by Analyzing Live Calls End-to-end. In *CHI '09: Proceedings of the 27th international conference on Human factors in computing systems* (New York, NY, USA, 2009), ACM, pp. 1313–1322.
- [109] TOMASELLI, K. G. (Re)Mediatizing HIV/AIDS in South Africa. *Cultural Studies <=> Critical Methodologies* 9, 4 (2009), 570–587.
- [110] TSIETSI, M., TERZOLI, A., AND WELLS, G. Mobicents as a Service Creation and Deployment Environment for the Open IMS Core. In *Southern African Telecommunications, Networks and Applications Conference (SATNAC)* (2009).
- [111] TURING, A. M. Computing Machinery and Intelligence. *Mind* LIX (1950), 433–460.
- [112] UNAIDS. HIV/AIDS Country Profile for South Africa. Online: http://www.unaids.org/en/CountryResponses/Countries/south_africa.asp [Last accessed: 8 April 2008].
- [113] UNAIDS. UNAIDS Practical Guidelines for Intensifying HIV Prevention: Towards Universal Access. Online: http://data.unaids.org/pub/Manual/2007/20070306_prevention_guidelines_towards_universal_access_en.pdf [Last accessed: 24 March 2009], 2007.

-
- [114] UNAIDS. South Africa - Country Situation. Online: http://data.unaids.org/pub/FactSheet/2008/sa08_soa_en.pdf [Last accessed: 24 March 2009], July 2008.
- [115] UNITED STATES GENERAL ACCOUNTING OFFICE. Consumer Health Informatics - Emerging Issues. Tech. rep., United States General Accounting Office, Washington, D.C. 20548, July 1996.
- [116] USCHOLD, M. Building Ontologies: Towards a Unified Methodology. In *16th Annual Conf. of the British Computer Society Specialist Group on Expert Systems* (1996), pp. 16–18.
- [117] USCHOLD, M., AND KING, M. Towards a Methodology for Building Ontologies. In *Workshop on Basic Ontological Issues in Knowledge Sharing, held in conjunction with IJCAI-95* (1995).
- [118] VAN MEGGELEN, J., SMITH, J., AND MADSEN, L. *Asterisk™: the future of telephony, 2nd edition*, second ed. O'Reilly, 2007.
- [119] VELÁSQUEZ, J. D., AND JAIN, L. C. *Advanced Techniques in Web Intelligence-1*. Springer, 2010.

Appendix A

Snapshot from Protege

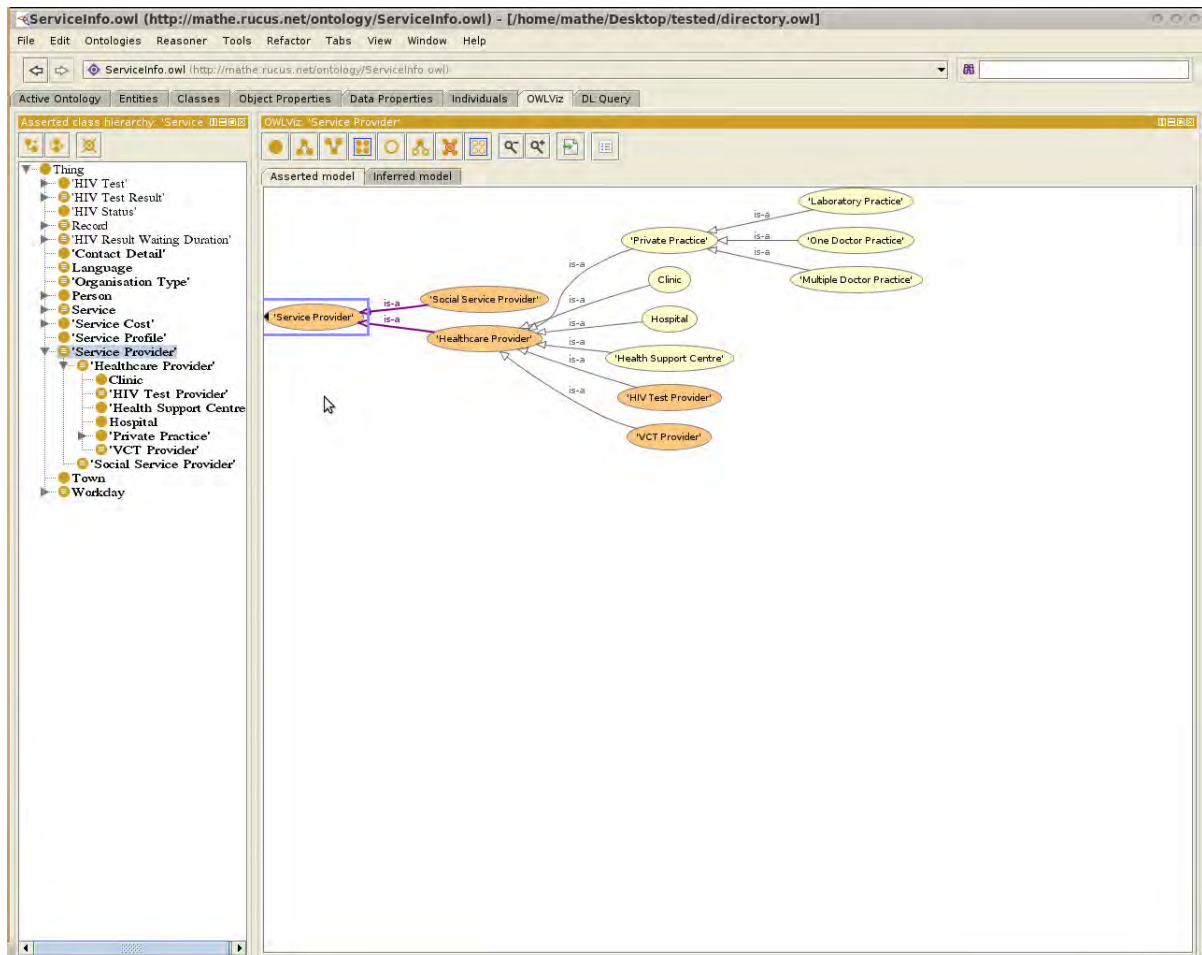


Figure A.1: Snapshot from Protege of the Service Provider ontology

Appendix B

A Snippet XML File for the Service Provider Ontology

```
1 <?xml version="1.0"?>
2 <!DOCTYPE rdf:RDF [
3   <!ENTITY time "http://www.w3.org/2006/time#" >
4   <!ENTITY owl "http://www.w3.org/2002/07/owl#">
5   <!ENTITY xsd "http://www.w3.org/2001/XMLSchema#">
6   <!ENTITY owl2xml "http://www.w3.org/2006/12/owl2-xml#">
7   <!ENTITY rdfs "http://www.w3.org/2000/01/rdf-schema#">
8   <!ENTITY rdf "http://www.w3.org/1999/02/22-rdf-syntax-ns#">
9   <!ENTITY ServiceInfo "http://mathe.rucus.net/ontology/ServiceInfo.owl#">
10  <!ENTITY HIVTesting "http://mathe.rucus.net/ontology/HIV/testing.owl#">
11 ]>
12 <rdf:RDF xmlns="http://mathe.rucus.net/ontology/HIV/testing.owl#"
13   xml:base="http://mathe.rucus.net/ontology/HIV/testing.owl"
14   xmlns:rdfs="http://www.w3.org/2000/01/rdf-schema#"
15   xmlns:time="http://www.w3.org/2006/time#"
16   xmlns:owl2xml="http://www.w3.org/2006/12/owl2-xml#"
17   xmlns:directory="http://mathe.rucus.net/ontology/Service/directory.owl#"
18   xmlns:owl="http://www.w3.org/2002/07/owl#"
19   xmlns:HIVTesting="http://mathe.rucus.net/ontology/HIV/testing.owl#"
20   xmlns:xsd="http://www.w3.org/2001/XMLSchema#"
21   xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
22   xmlns:ServiceInfo="http://mathe.rucus.net/ontology/ServiceInfo.owl#">
23   <owl:Ontology rdf:about="">
24     <owl:imports rdf:resource="http://mathe.rucus.net/ontology/HIV/testing.owl"/>
25   </owl:Ontology>
26
27   <!--***** Object Properties *****-->
28   <!-- http://mathe.rucus.net/ontology/ServiceInfo.owl#charges -->
29   <owl:ObjectProperty rdf:about="#charges">
30     <owl:propertyChainAxiom rdf:parseType="Collection">
31       <rdf:Description rdf:about="#hasProfile"/>
32       <rdf:Description rdf:about="#hasCost"/>
33     </owl:propertyChainAxiom>
34     <owl:propertyChainAxiom rdf:parseType="Collection">
35       <rdf:Description rdf:about="#isProviderOf"/>
```

```

36 <rdf:Description rdf:about="#hasCost"/>
37 </owl:propertyChainAxiom>
38 </owl:ObjectProperty>
39 <!--***** Data Properties *****-->
40 <!-- http://mathe.rucus.net/ontology/ServiceInfo.owl#hasCloseRadius -->
41 <owl:DatatypeProperty rdf:about="#hasCloseRadius">
42 <rdfs:comment>In this ontology we regard the centre to be the RU campus. Further,
    we have decided not to attach the actual distance value to the centre. Thus
    for each provider the question is: is the provider within close radius to
    campus or not?</rdfs:comment>
43 <rdfs:range rdf:resource="&xsd:boolean"/>
44 </owl:DatatypeProperty>
45 <!--***** Classes *****-->
46 <!-- http://mathe.rucus.net/ontology/ServiceInfo.owl#HIV_Test_Provider -->
47 <owl:Class rdf:about="#HIV_Test_Provider">
48 <rdfs:label>HIV Test Provider</rdfs:label>
49 <owl:equivalentClass>
50 <owl:Class>
51 <owl:unionOf rdf:parseType="Collection">
52 <owl:Restriction>
53 <owl:onProperty rdf:resource="#isProviderOf"/>
54 <owl:someValuesFrom rdf:resource="&testing;HIV_Testing"/>
55 </owl:Restriction>
56 <owl:Restriction>
57 <owl:onProperty rdf:resource="#isProviderOf"/>
58 <owl:someValuesFrom rdf:resource="#VCT"/>
59 </owl:Restriction>
60 </owl:unionOf></owl:Class>
61 </owl:equivalentClass>
62 <rdfs:subClassOf rdf:resource="#Healthcare_Provider"/>
63 </owl:Class>
64 <!--***** Individuals *****-->
65 <!-- http://mathe.rucus.net/ontology/ServiceInfo.owl#Raglan_Road_Clinic -->
66 <owl:Thing rdf:about="#Raglan_Road_Clinic"><rdf:type rdf:resource="#Clinic"/>
67 <rdfs:label>Raglan Road Clinic</rdfs:label>
68 <hasTelephoneNumValue rdf:datatype="&xsd:integer">6036084</hasTelephoneNumValue>
69 <hasStreetAddressValue rdf:datatype="&xsd:string">Raglan Road</
    hasStreetAddressValue>
70 <hasCloseRadius rdf:datatype="&xsd:boolean">>false</hasCloseRadius>
71 <locatedIn rdf:resource="#Grahamstown"/>
72 <hasOrganisationType rdf:resource="#Public_Sector"/>
73 <hasProfile rdf:resource="#RagProfile1"/>
74 <hasProfile rdf:resource="#RagProfile2"/>
75 <usesLanguage rdf:resource="#Sepedi"/>
76 <usesLanguage rdf:resource="#isiXhosa"/>
77 </owl:Thing>
78 <!--***** General Axioms *****-->
79 <rdf:Description>
80 <rdf:type rdf:resource="&owl;AllDifferent"/>
81 <owl:distinctMembers rdf:parseType="Collection">
82 <rdf:Description rdf:about="#Settlers_Hospital"/>
83 <rdf:Description rdf:about="#Fort_England_Hospital"/>
84 </owl:distinctMembers>
85 </rdf:Description>
86 <!-- Generated by the OWL API (version 2.2.1.1138) http://owlapi.sourceforge.net -->

```


Appendix C

Example Scripts

Please note that the scripts were designed specifically for DTMF-driven responses.

CALL SCRIPT - 1

SYSTEM Hello. Welcome to the telephone number listing service for locating HIV testing sites in Grahamstown. To provide you with an initial listing, please answer two yes or no questions for us. You may press 0 at anytime to exit.

(User is moved to another dialogue)

Should the testing site be within close proximity to Rhodes University?

Press 1 for Yes; Press 2, for No; And press 0, if you want to exit.

USER DTMF-2

SYSTEM OK. Are you interested in a free or low cost testing site?

Press 1 for Yes; Press 2, for No; And press 0, if you want to exit.

USER DTMF-2

SYSTEM There is one site. It is Dr Mackenzie Practise and the telephone number is

6 2 2 3 3 8 3.

Press *, if you would like to repeat the information just given.

Press 9, if you would like to return to the main menu.

Press 0, if you want to exit.

USER DTMF-0

SYSTEM Thanks for calling. Please complete the satisfaction survey for us.

(Five questions intended to improve service)

CALL SCRIPT - 2

SYSTEM Hello. Welcome to the telephone number listing service for locating HIV testing sites in Grahamstown. To provide you with an initial listing, please answer two yes or no questions for us. You may press 0 at anytime to exit.

(User is moved to another dialogue)

Should the testing site be within close proximity to Rhodes University?

Press 1 for Yes; Press 2, for No; And press 0, if you want to exit.

USER DTMF-1

SYSTEM OK. Are you interested in a free or low cost testing site?

Press 1 for Yes; Press 2, for No; And press 0, if you want to exit.

USER DTMF-1

SYSTEM There are two sites. The first is Rhodes Health Care Centre and the second is Raphael centre.

Press 1, to obtain Rhodes Health Care Centre telephone number

Press 2, to obtain Raphael centre

Press *, if you would like to repeat the information just given.

Press 9, if you would like to return to the main menu.

Press 0, if you want to exit.

USER DTMF-1

SYSTEM Rhodes Health Care Centre telephone number is 6 0 3 8 0 9 3.

Press *, if you would like to repeat the information just given.

Press 9, if you would like to return to the main menu.

Press 0, if you want to exit.

USER DTMF-0

SYSTEM Thanks for calling. Please complete the satisfaction survey for us.

(Five questions intended to improve service)

Appendix D

Rule Engine Class

```
1 public class RuleRunner {
2     // Declarations
3     public Object [] criteriaProcessor(Map resultMap, String criteriaSelection){
4     // Initialisations
5     try{
6         // Load up the knowledge base
7         KnowledgeBase kbase = readKnowledgeBase();
8         StatefulKnowledgeSession ksession = kbase.newStatefulKnowledgeSession();
9         KnowledgeRuntimeLogger logger = KnowledgeRuntimeLoggerFactory.newFileLogger(
10             ksession, "ruleSession");
11         ksession.insert(resMap); //insert query result map
12         ksession.insert(criteria); //insert criteria
13         ksession.fireAllRules(); // "Fire all rules"
14         logger.close();
15         ksession.dispose();
16     } catch (Throwable t) {t.printStackTrace();}
17 // Store the result map and the refined boolean value in the object array.
18 processedResults[0] = resultMap.getResultMap();
19 processedResults[1] = resultMap.isRefined();
20 return processedResults;
21 }
22 private static KnowledgeBase readKnowledgeBase() throws Exception {
23     KnowledgeBuilder kbuilder = KnowledgeBuilderFactory.newKnowledgeBuilder();
24     kbuilder.add(ResourceFactory.newUrlResource(ruleRepoURL+"/refine.drl"),
25         ResourceType.DRL);
26     KnowledgeBuilderErrors errors = kbuilder.getErrors();
27     if (errors.size() > 0) {
28         for (KnowledgeBuilderError error: errors) {
29             System.err.println(error);
30         }
31     }
32     throw new IllegalArgumentException("Could not parse knowledge.");
33 }
34 KnowledgeBase kbase = KnowledgeBaseFactory.newKnowledgeBase();
35 kbase.addKnowledgePackages(kbuilder.getKnowledgePackages());
36 return kbase;
37 }
```

Appendix E

Accompanying CD-ROM

The following are contained within the accompanying CD-ROM:

- ⇒ **Thesis Document** – The electronic copy of this thesis in pdf format.
- ⇒ **Readings** – A selection of electronic copies of some of the referenced papers within the thesis. The papers are in pdf format. The naming convention uses primary author and year of publication, separated by an underscore (e.g. XYZ_1980.pdf).
- ⇒ **Domain Ontology** – The electronic copy of the developed ontology in xml format.
- ⇒ **Source Code** – The source code for the following is included:
 - ➔ **Locator Application** – The HTML version of the HIV Testing Site Locator application. The WAR file has also been provided for deployment.
 - ➔ **Voice Locator Application** – The VoiceXML version of the HIV Testing Site Locator application. The WAR file has also been provided for deployment.
 - ➔ **‘Query Interface’ Manager** – This application embodies the logic for managing access to the domain knowledge. It is imported into both versions of the Locator applications. The configuration file (**config.xml**) has also been included to alter, for example, the location of the ontology. When alterations are made, this file may be placed in */tmp/voiceApp/config.xml* or *C:/tmp/voiceApp/config.xml* depending on whether Linux or Windows is used.

Appendix F

Deployment Guide

As stated in Chapter 3, deployment process to the application server is dependent on the server used and its configuration. Below is a brief guide for deploying the voice application using Tomcat in conjunction with Apache web server. There are two basic steps that need to be followed. The description for each is provided below, with Table F.1 providing specific detail for each deployed resource.

1. Deploy resource to a suitable location, */filesystem/path/to/resource*. For testing, all resources where deployed on */var/www/[resourceFolderName]*.
2. Create a context XML file similar to one shown below:

```
1 <?xml version="1.0" encoding="UTF-8"?>
2 <Context antiJARLocking="true" docBase="/filesystem/path/to/application" />
```

The above created file should be saved based on the URL path that will be used to access the application. Once saved the file must be deployed in the directory:

```
$CATALINA_HOME/conf/[enginename]/[hostname]
```

where *enginename* might be *Catalina* and *hostname* might be *localhost*. In fact, for testing purposes, it is assumed that the host shall be *localhost*. However, if say, the ontology was to be deployed in a remote server, its location would need to be specified in the configuration file (*/tmp/voiceApp/config.xml* or *C:/tmp/voiceApp/config.xml*).

	Deployment Location	Context File Names
TTS Script	/var/www/tts	tts.xml
Ontology Folder	/var/www/ontology	ontology.xml
Rules Folder	/var/www/rules	rules.xml
VoiceApplication Folder/WAR	/var/www/voiceLocatorApp	voiceLocatorApp.xml

Table F.1: Deployment Locations and Context Names Used for Testing