A semiotic approach to ad-hoc networked environments

This item was submitted to Loughborough University's Institutional Repository by the/an author.

Additional Information:

- A Doctoral Thesis. Submitted in partial fulfilment of the requirements for the award of Doctor of Philosophy of Loughborough University.

Metadata Record: https://dspace.lboro.ac.uk/2134/3146

Please cite the published version.
This item was submitted to Loughborough’s Institutional Repository by the author and is made available under the following Creative Commons Licence conditions.

For the full text of this licence, please go to:
http://creativecommons.org/licenses/by-nc-nd/2.5/
A Semiotic Approach to Ad-Hoc Networked Environments

by

Lezan Hawizy

A Doctoral Thesis

Submitted in partial fulfilment of the requirements for the award of

Doctor of Philosophy

of

Loughborough University

30th October 2007

Copyright 2007 Lezan Hawizy
Abstract

The aim of the work in this thesis is to develop a new approach of interacting with ad-hoc networked environments. These are networks where devices connect on demand with no underlying network infrastructure. The intention of this work is to develop these environments so that devices and services on these networks can publish their services, query for other services and connect with each other when required. The devices need to be able to perform these actions without prior knowledge of each other, therefore a theory of communication, semiotics, is presented. Ad-hoc networks provide an appropriate test-bed for this application of semiotics as they allow services to ‘know’ about each other and communicate with one another. By using semiotics, we aim to create a representation of communication that allows a system to communicate within the networked environment and ask for services and connections as well as interact with users and provide. This way a user can demand something from the surrounding environment and the elements within this environment can communicate with each other to provide the service the user required. To create an effective model for this representation, various research areas will be discussed such as smart environments, natural language processing, multicast environments and human computer interaction. Principles will be used from all these areas to implement an approach of interacting with smart environments. Different types of smart environments, such as smart homes and m-commerce environments, will be used to observe how different contexts affect communication. A prototype system was realised for proof of concept and evaluated by subjects. This work highlighted the feasibility of this approach and opened a new area worthwhile of further research.
Acknowledgements

Without sounding too much like a telephone directory, I would like to heartily thank the following people:

Baba and Daya for everything especially for allowing me to mooch off them till the ripe old age of 25 (and continuing). My brothers Haloon, Salan and Lawan. My grandparents, my aunt: Pla Soza, my uncles: Sarbast, Sarcote, Sarchill, Azad, Nawzad and especially Mam Azad for driving me to Loughborough in the first place and being my secondary ATM throughout these 3 years.

My uber-helpful supervisors Iain Phillips and John Connolly for making this process as smooth as possible with their constant support and gentle prodding in the right direction. My DOR Chris Hinde for being so helpful and incredibly random.

The best housemates that anyone could ask for: Alexandra Alecu, Pinar Boyraz, Tracey McCauley and Laura Gramaglia.

Mark Withall and John Whitley for helping me out since day one in just about everything, Matthew Atkinson for his help and enthusiasm that could only be described as ‘on crack’.

Lenna Saleh for creating THE list.

Long distance and long term buddies Farah Masri and Reema Abbasi.

The lovely WAM ladies: Sara Tedmori, Jatinder Gill, Rajveen Dhiensa, Senay Mihecin, Laween Meran and Pinar Kafali.

Scott Culcheth, Becky Yates, Martin Sykora, Khushvinder Gill, Yuming Yang, Karen Yunqiu Li and Jia Ren for being great company and an outlet of entertainment, support, lifts and the works.

Sirwa Jamal and Fuad Khoshnaw for the constant supply of yaprax and kufta and other gluttonous home comforts.

Café Direct for keeping me alive over the past 3 years.

Everyone at Holywell Park for providing such a fantastic environment as well as anyone I have met during my time here, if I have met you than I most likely have something to thank you for.

Finally I would like to dedicate this thesis to Sargoll and Polad Hawizy. Thank you.
Contents

1 Introduction .................................................................................................................. 6
  1.1 Background ................................................................................................................. 6
  1.2 Ad-Hoc Networks and Intelligent Ubiquitous Environments .............................. 7
  1.3 The Case for Semiotics ............................................................................................... 8
  1.4 Motivation for the Work ........................................................................................... 9
  1.5 Aims of the Work ..................................................................................................... 9

2 Semiotics ......................................................................................................................... 11
  2.1 Introduction ................................................................................................................ 11
  2.2 History of Semiotics .................................................................................................. 11
    2.2.1 Brief History of Sign Analysis .............................................................................. 12
  2.3 Notable Names in Modern Day Semiotics ............................................................... 14
    2.3.1 Ferdinand De Saussure (1857 - 1913) ................................................................. 14
    2.3.2 Charles Sanders Peirce (1839 - 1914) ................................................................. 14
    2.3.3 Roland Barthes (1915-1980) ............................................................................. 14
    2.3.4 Louis Hjelmslev (1899 - 1965) ......................................................................... 15
    2.3.5 Charles Morris (1901-1979) ............................................................................. 15
    2.3.6 Thomas Albert Sebeok (1920 - 2001) ................................................................. 15
    2.3.7 Umberto Eco (1932-) ......................................................................................... 16
    2.3.8 Peter Bøgh Andersen (1945-) .......................................................................... 16
    2.3.9 Ronald Stamper (1934-) ................................................................................... 16
  2.4 Key Distinctions in Semiotics ................................................................................... 17
    2.4.1 Types of Sign ...................................................................................................... 17
<table>
<thead>
<tr>
<th>Section</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>2.4.2 Encoding and Decoding</td>
<td>18</td>
</tr>
<tr>
<td>2.4.3 Denotation and Connotation</td>
<td>19</td>
</tr>
<tr>
<td>2.5 Areas where Semiotics has been applied</td>
<td>19</td>
</tr>
<tr>
<td>2.5.1 Algebraic Semiotics</td>
<td>19</td>
</tr>
<tr>
<td>2.5.2 BioSemiotics</td>
<td>19</td>
</tr>
<tr>
<td>2.5.3 Computer Semiotics</td>
<td>20</td>
</tr>
<tr>
<td>2.5.4 Organisational Semiotics</td>
<td>20</td>
</tr>
<tr>
<td>2.5.5 Semiotic Engineering</td>
<td>21</td>
</tr>
<tr>
<td>2.6 Semiotic Analysis</td>
<td>21</td>
</tr>
<tr>
<td>2.7 Contribution of Semiotics to the Present Study</td>
<td>23</td>
</tr>
<tr>
<td>2.8 Final Remarks</td>
<td>24</td>
</tr>
<tr>
<td>3 Smart Environments</td>
<td>25</td>
</tr>
<tr>
<td>3.1 Smart Home Technologies</td>
<td>26</td>
</tr>
<tr>
<td>3.1.1 Live-in Labs</td>
<td>26</td>
</tr>
<tr>
<td>3.1.2 Software Architectures</td>
<td>27</td>
</tr>
<tr>
<td>3.1.3 Context and Information</td>
<td>28</td>
</tr>
<tr>
<td>3.1.4 Resource Co-ordination</td>
<td>30</td>
</tr>
<tr>
<td>3.1.5 Natural Interaction in Computerised Home Environments</td>
<td>31</td>
</tr>
<tr>
<td>3.1.6 Assistive Technologies</td>
<td>32</td>
</tr>
<tr>
<td>3.2 Smart workspaces</td>
<td>33</td>
</tr>
<tr>
<td>3.2.1 Stanford iRoom</td>
<td>33</td>
</tr>
<tr>
<td>3.2.2 Obje Project</td>
<td>34</td>
</tr>
<tr>
<td>3.3 Current Challenges of Smart Environments</td>
<td>35</td>
</tr>
<tr>
<td>3.4 Final Remarks</td>
<td>38</td>
</tr>
<tr>
<td>4 Technologies</td>
<td>39</td>
</tr>
<tr>
<td>4.1 Network Technologies</td>
<td>39</td>
</tr>
<tr>
<td>4.1.1 Ad-Hoc Networks</td>
<td>39</td>
</tr>
<tr>
<td>4.1.2 M-Commerce Environments</td>
<td>41</td>
</tr>
<tr>
<td>4.1.3 Multicast</td>
<td>43</td>
</tr>
</tbody>
</table>
Chapter 1

Introduction

1.1 Background

The last twenty years have witnessed an enormous growth in the use of computer networks. The development of the global Internet and the world wide web have enabled users to access information services on a scale that was hardly imagined a few decades ago.

However, the Internet and the web are not the only areas of network-based technology that have been extensively researched. Another very interesting topic that has developed is that of ‘intelligent ubiquitous environments’, in which computer-based devices are unobtrusively embedded in their surroundings. The devices in such an environment are able to connect to one another on demand (rather than requiring a pre-existing network infrastructure, as is typically needed at present), in order to provide users with services when required.

The work described in this thesis is primarily concerned with the process of interacting with intelligent environments. These include the ‘smart’ homes, ‘smart’ offices and mobile commerce (m-commerce) shopping environments of the future.

At present, of course, we are faced with an abundance of electronic devices working separately from each other. While a lot of work has been done in creating intelligent devices, much less has been done in developing an environment that enables interaction between them. Currently a printer and a computer can be in the same room,
yet it would be nearly impossible to make them work together if an installation CD went missing or a driver was corrupted. A simple communication between these devices could enable each of them to establish the nature of the other device and its services, and to perform the right actions and download the appropriate drivers to help them inter-operate. Now as the environment gets more complex, such as in the example of an inter-operable house with many devices working together, the need for communication gets bigger.

Intelligent ubiquitous environments demand a complex multi-device network which involves communication between all devices and users. However, adequate communication between users and networked computer systems demand a novel way of allowing human control of systems where people specify what they want to happen. In current systems, human users are essentially being given choices from what the computer dictates can happen. However, a more adequate approach would be to make systems more user-centred, so that they can adapt and provide services based on the users’ needs. This gives the user more control regarding what to expect from the system. It is the goal of the present research to develop such an approach.

Behind our understanding of communicative interaction lies the theory of Semiotics. A semiotic model of the device to device and user to system communication is presented here, in relation to environments where we have not only multiple devices but also multiple modes of communication, such as text, speech or visual communication. Interacting with these environments also include configuring their underlying networks on demand, in order to facilitate the achievement of the user’s goals and to enhance the user’s experience.

1.2 Ad-Hoc Networks and Intelligent Ubiquitous Environments

Intelligent ubiquitous environments are based on ‘ad-hoc’ networks, which comprise a set of devices that can interact without the need for a fixed network infrastructure. The interaction is normally wireless, being carried out by means of radio signals.
The ad-hoc networks investigated in the research described in this thesis are intelligent home environments and intelligent office environments, where a number of different devices collaborate when needed by the user to provide a desired service, and m-Commerce shopping environments, such as may be found in supermarkets or shopping centres of the future, in which electronic commerce is carried out via mobile devices, such as cellphones, PDAs (personal digital assistants) or laptop computers, which can connect in an ad-hoc manner to services broadcast by shops or other mobile devices to get information about certain products and services.

As well as providing labour-free solutions to everyday activities, intelligent environments are also beneficial for providing care to the elderly or those in need of assistance. Examples of that include the Elite Care Environment[67] and the smart medical fridge[46].

1.3 The Case for Semiotics

Semiotics is concerned with the study of signs (representations, such as text, speech, gesture or visual depiction) and their effect on social life, through their use in communication.

Semiotics plays a vital part in the research described in this thesis. The semiotic framework facilitates the creation of a model of communication between the user and the network. Part of this model is the expression of the user’s goal when interacting with the network. The interpretation of the users’ goal by the system triggers it to provide the desired services to the users.

For a system to be intelligent, it should be able to correctly interpret what a user wants, regardless of how the user wishes to express an intention. A semiotic framework enables all forms of input to be processed in a consistent manner.

To achieve this we need to tap into various research areas to create a seamless model that encompasses not only interaction between the devices in an intelligent environment, but also interaction with the user.
1.4 Motivation for the Work

Ad-hoc environments present various opportunities for the application of semiotics:

- The semiotic framework accommodates any mode of communication whether it be speech-based, action-based or command-line.

- The framework encompasses all the different facets of communication (including the context) and not just the physical hardware aspect, and it covers all forms of language (natural, formal or nonverbal).

- The framework not only applies to user-system interaction but also to communication inside the system (including communication across a network).

Because semiotics has not been applied to ad-hoc networks before, it will be interesting to see the benefits of this novel internally consistent approach to communication.

1.5 Aims of the Work

In the course of the present research project, it is our intention to further this work by developing semiotically inspired methods to support the actual implementation of user-friendly systems in ad-hoc networks. Semiotic theories will be discussed in the next section and a framework will be selected to represent communication in our environment. The framework will split communication across separate levels to take account of all aspects that affect communication. Ad-hoc networks, will in turn help the semiotic framework by providing contextual information about the surrounding environment. The system that will be developed using these principles will have the following characteristics:

- Multi-modal: operates different semiotic systems such as natural language or gestures.

- Multimedia: works with visual, auditory or haptic (tactile) input.

- Represents the user’s goals semiotically.
• Represents structures in a network semiotically.

• Adapts to different environments, and can re-configure to provide services whenever required.

The structure of this thesis is as follows: chapters two, three and four discuss related literature to the work in this thesis; chapter five illustrates applying semiotics to networks; chapter six is an overview of the prototype system; chapter seven includes a series of tests run on the system and finally chapter eight is the conclusion and discusses how this work could be further extended.
Chapter 2

Semiotics

2.1 Introduction

Semiotics is a discipline that is very difficult to classify as it is rarely studied as a field of its own; its theories are usually applied to other fields, which can be anything from biology to media. The most concise definition of semiotics is that it is the study of signs. A sign in this sense can be seen as anything that designates something other than itself and can be used to communicate meaning. So here a language text (with its syntactic and semantic structures) is a sign, as also is a gesture such as waving to signal goodbye. This view of semiotics, described as cultural or linguistic semiotics, can be seen as a branch of communication theory.

2.2 History of Semiotics

Even though the term semiotics was not coined until a few centuries ago, analysing of signs that produce communication has been around for at least two thousand years\(^1\). The following sections will discuss the prominent names in sign analysis.

\(^1\)The earliest known work on general linguistics is thought to be that of Panini on Sanskrit Grammar, around the 5th Century BC.
2.2.1 Brief History of Sign Analysis

The ancient Greeks had a fascination with languages, and in particular sophists\(^2\) elaborated on the art of language.

Prodicus of Ceos, one of the early Sophists, based his teachings on the practical idea that properly chosen words are fundamental to effective communication. This was particularly important to the latter-day sophists as they were interested in rhetoric and logical fallacies.

Plato, a critic of sophism, and Prodicus, in particular, was also interested in the separation between words and objects. His most famous work, Cratylus [56], is often quoted by linguists. The main synopsis is a dialogue between the three main characters, Socrates, Cratylus and Hermogenes. They discuss whether the linguistic sign is determined by the nature of an object or by convention. Hermogenes declares

Any name which you give, in my opinion, is the right one, and if you change that and give another, the new name is as correct as the old.

On a similar note, Aristotle noted that

There can be no natural connection between the sound of any language and the things signified \([40]\).

This arbitrary nature of a sign is quite important to semiotics as it will be discussed by others semioticians as well.

Roger Bacon was one of the most prominent semioticians in the middle ages. One of his works entitled ‘De Signis’ provides a detailed analysis of signs by combining sign theories from Aristotle and Augustine. In it he classifies signs into two main categories: natural and given (using ideas from Augustine) and probable and necessary; further details of this can be found in appendix A. Italian theologian Bonaventura had this to say about the nature of a sign [75]:

\(^2\)A sophist was a term used for the member of a respected group of philosophy teachers in ancient Greece at the end of the 5th century. However the term ‘sophistry’ gradually acquired the negative connotation of cleverness not restrained by ethics and, the sophists were subsequently disbanded as they were accused of immorality by the state. http://www.iep.utm.edu/p/protagor.htm

12
A sign has a twofold comparison: both to that which it signifies, and to that to which it signifies; and the first is essential and the sign always has it in act, but the second it has in habit; and it is from the first that it is called a sign, not from the second. Whence a circle above a tavern is always a sign, even if no one looks at it.

William of Ockham was another known philosopher in the middle ages, also developed some ideas behind sign analysis like signification in his work ‘Summa of Logic’. His view on language as a sign system is as follows [76].

The function of language, therefore, is not so much to communicate thoughts from one mind to another, but to convey information about the world.

The first time the word semiotics was used was in medical article by Henry Stubbes, an English physician and mathematician in the 17th century, to denote the branch of medical science relating to the interpretation of signs.

John Locke, an English philosopher, used the word ‘Semeiotike’ in Book 4 (entitled Knowledge and Probability), Chapter 21, of An Essay Concerning Human Understanding [51], where he divides science into three parts which he called: Physica, Practica and Semeiotike. His description of semiotics is as follows:

Semeiotike. Thirdly, the third branch may be called Semeiotike, or the doctrine of signs; the most usual whereof being words, it is aptly enough termed also Logike, logic: the business whereof is to consider the nature of signs, the mind makes use of for the understanding of things, or conveying its knowledge to others.

The semiotics we know today was created independently by a Swiss Linguist, Ferdinand De Saussure, who called it ‘semiology’ and an American logician, Charles Peirce, who called it ‘semeiotic’[sic].
2.3 Notable Names in Modern Day Semiotics

2.3.1 Ferdinand De Saussure (1857 - 1913)

Ferdinand De Saussure’s most famous work is the ‘Course in General Linguistics’ [19]. In this book he describes the foundation of semiotics, where he names the science of studying signs semiology and describes it as ‘a science that studies the life of signs within society.’ He also sees language as ‘a system of signs that express ideas’. The ‘Course in General Linguistics’ also discusses the basic concepts in semiotics such as arbitrariness and the duality of a sign model (signifier and signified).

2.3.2 Charles Sanders Peirce (1839 - 1914)

Charles Peirce was an American logician and mathematician as well as the co-founder of modern day semiotics. He developed the idea of the ‘pragmatic maxim’, developed from William James’ work, in semiotics. This maxim can be seen as a method of clarifying difficult ideas by linking their meaning to their practical consequences. He also developed the idea of the triadic model for a sign (Representamen, Interpretant, Object), and the three classes of signs (iconic, symbolic and indexical). The other main area he was active in was formal logic. His main contributions in this area were devising existential graphs (diagramatic notations for predicate calculus), Boolean algebra and the classification of logical concepts. Being a logician, Charles Peirce was interested in combining logic with signs, and he made two attempts at creating a complete system for first order logic. The first was a algebraic notation for predicate calculus [11, 12], the most commonly used version of logic, and the second was existential graphs, a less mathematical approach to logic systems.

2.3.3 Roland Barthes (1915-1980)

Roland Barthes was a French literary critic and semiotician. His works in semiotics are mainly on communication through, and interpretation of, photographic images. His best known work in these areas include ‘Camera Lucida’ [6] and ‘Image, Music, Text’ [7]. In his work he uses the concepts of studium and punctum which are similar...
to the denotation and connotation described in the next section.

2.3.4 Louis Hjelmslev (1899 - 1965)

Louis Hjelmslev was a Danish Linguist. He coined the term Glossematics along with Hans Jørgen Uldal, based on a combination of glossary and mathematics, to produce a structuralist study of language. A glosseme is the basic unit of language that could carry a meaning. He agreed with Saussure that from the point of view of language use, language was made up of signs, but being a structuralist he saw that linguists should view language as a system of smaller units. He also viewed signs as a structure made up of two forms, a content form and an expression form.

2.3.5 Charles Morris (1901-1979)

Charles Morris was an American semiotician and philosopher. He also worked on pragmatics, as well as proposing a threefold division of a sign (but dividing it into a sign vehicle, designatum, and interpreter). His work was mainly in behaviourist semiotics.

2.3.6 Thomas Albert Sebeok (1920 - 2001)

Thomas Sebeok was a Hungarian-born semiotician who wrote extensively about the topic (his work is made up of five hundred and seventy nine publications [23]). He was heavily influenced by the works of Charles Morris. He extended semiotics to also include non-human communication. He coined the term ‘Zoosemiotics’, and was one of the creators, the other being Friedrich Salomon Rothschild, of ‘Biosemiotics’, as well as a significant contributor to it. In his work on zoosemiotics he distinguishes between communication and language, he sees language as a subclass of communication specific to humans. He is also the founder of the Semiotica Journal.
2.3.7 Umberto Eco (1932-)

An Italian novelist, philosopher and semiotician, his most famous work is ‘The name of the Rose’ [27]. His first work on semiotics was entitled ‘Opera Aperta’ (Open Text). This is an analysis of literary text. It highlighted the importance of context and showed how the meaning of a text is not dependent on its lexical structure alone, i.e. the same text could produce different meanings depending on what the reader brings to it. His next work, a Theory of Semiotics, criticises the notion that the meaning of a sign is determined by the object it refers to, therefore refuting the proposition that icons should share likeness with the object.

2.3.8 Peter Bøgh Andersen (1945-)

Peter B. Andersen is a Danish linguist. He pioneered the application of semiotics to computer systems in his book ‘A Theory of Computer Semiotics’ [1]. Based on the notion that programs can represent and manipulate storage cells, he sees programming as ‘sign creation’ and computer systems as ‘sign-vehicles’ [2]. He refers to this programming concept as sign-oriented programming.

2.3.9 Ronald Stamper (1934-)

Ronald Stamper is an English semiotician well known for his work on organisational semiotics and relational databases. He expands Peirce’s and Morris’s work by creating a six-level semiotic framework for analysing a sign, and emphasises the importance of norms in the application of semiotics in organisations [60]. A good summary of Stamper’s theories can be found in Kecheng Liu’s ‘Semiotics in information systems engineering’ [49].
2.4 Key Distinctions in Semiotics

2.4.1 Types of Sign

As mentioned before, a sign stands for something other than itself and is thereby capable of conveying meaning. Therefore anything from a simple gesture to a well structured language text can be considered a sign. Saussure believed in a dyadic model of a sign, whereby a sign is composed of two parts. A signifier and a signified.

- The signifier: The form that a sign takes.
- The signified: The concept it represents.

Examples of this are traffic cones (signifier) and road works (signified) or a flashing green light on an electronic device (signifier) and the power in the electronic device being on (signified). Using this model a linguistic sign is divided into the word and the thing it represents, for example the word ‘desk’ and the actual desk that is signified. In computer semiotics variables and data structures can be seen as signifiers that represent the memory storage in the hard disk of a computer system (signified). Hjelmslev also believed in the dyadic model but referred to signifier and signified as the expression and content respectively.

According to Saussure the nature of the sign is arbitrary, i.e. there is no natural connection between the signified concept and the sign. So for example the word ‘desk’ has little to do with an actual desk and could, in principle, be replaced by any other combination of characters.

Peirce however believed in a triadic model for the sign:

- The Representamen: The form which the sign takes.
- An Interpretant: The sense made of the sign.
- An Object: The entity that the sign refers to.

So for example, in the traffic cone example, the representamen is the traffic cones and the object is the road works, the Interpretant here is that the cones represent road works.
Peirce liked classifications. He predicted that all signs can be grouped into 50,049 classes [10]. But realising that this was impractical, he reduced them to 66, and a further reduction narrowed them down into three classes:

- Symbolic: A mode where the signifier does not resemble the signified (such as language and numbers).

- Iconic: A mode where the signifier resembles or imitates the signified (such as photographs or items on a desktop in a GUI). The issue of icons has always been a topic of debate in the semiotic community, as most semioticians maintain that there are no ‘pure’ icons there is always an element of cultural convention involved [10] and generally only see the resemblance between an icon and the actual object when we know the meaning of it.

- Indexical: A mode where the signifier and the signified are associated in some way and there is a direct connection between them. In short an index indicated something, for example, clouds indicate rain.

Peirce also mapped these types of signs were mapped on to what he described as the Firstness, Secondness and Thirdness of signs (adapted from De Souza [20]):

- Firstness [icon]: The category of undifferentiated qualitative experience.

- Secondness [index]: The category of strict associations between two phenomena.

- Thirdness [symbol]: The category of mediated relations.

2.4.2 Encoding and Decoding

Encoding and Decoding are semiotic terms for producing and understanding signs, respectively. However, understanding in this sense does not simply stop at the literal comprehension of text but also the interpretation and evaluation of its meaning. The encoding-decoding theory has its supporters and critics. The most common criticism is that this theory does not cover human communication because language is only part of it. Much can be inferred using non-verbal cues, as explained here by Daniel Sperber [64]:

18
True, we have our rich languages and many minor codes too, but - and this is where the old story breaks down - we manage to communicate much more than we encode and decode, and not just occasionally, but all the time. So, our having language is, at best, a mere part of the true story.

2.4.3 Denotation and Connotation

Denotation is the meaning of a sign, in so far as this is systematized within the mode of communication concerned (e.g. language). For example, colour is an infinitely dense category, but language systematizes colour into six different categories which simplify understanding. Connotation is the socio-cultural and ‘personal’ associations (ideological, emotional etc) of the sign. Using the colour example again, the colour red can connote anger while red flags could connote communism. Connotations can be inferred either by choosing certain words in text or by focusing and blurring images in photographs. Barthes used studium and punctum to describe his interpretation of connotation and denotation in images [6]. In his view studium referred to the cultural, linguistic, and political interpretation of a photograph, while punctum referred to the personal, touching detail an image can bring.

2.5 Areas where Semiotics has been applied

2.5.1 Algebraic Semiotics

Algebraic semiotics is a combination of algebraic specification and social semiotics. It considers the interactions between the human use of information and its processing by computers. Applied work in this area include semiotic morphisms and design theory in user interfaces [35].

2.5.2 BioSemiotics

Biosemiotics is a growing field that studies communication in living systems. The term ‘biosemiotic’ was first used by F.S. Rothschild. As mentioned earlier in 2.3.6
this was an area that Thomas Sebeok extensively work on.

2.5.3 Computer Semiotics

Computer Semiotics is an interdisciplinary field coined by Peter Andersen in the mid-nineties. It is a combination of computer, logic, mathematics and natural language studies. It can be seen as a natural extension to algebraic semiotics. It is a slowly emerging field and there has been recent work on using semiotics on information systems [49] as well as using semiotics in a guideline on how to enable human computer interaction [20].

2.5.4 Organisational Semiotics

Organisational Semiotics is concerned with the study of communication and it’s effects on organisations. It was first introduced by Stamper [65]. It overlaps with computer semiotics, but while computer semiotics focuses more on the IT platform the latter gives equal awareness to all other levels. One of the main themes of Organisation Semiotics is an information systems methodology called MEASUR (A Method for Eliciting, Analysing, and Specifying User Requirements) [66]. It is a suite of methods based on organisational semiotics, used to address the IT and business requirements. It is made up of three parts:

- Problem Articulation: Scoping, making ill-defined, vague problems structured and manageable.

- Semantic Analysis: Elicit and represent requirements precisely.

- Norm Analysis: Dynamic aspects: identify norms governing organisations.

These techniques help understand the business problem. They capture the semantics and intentions of users in requirement models, and also implements technical information systems that are flexible and adaptable to the organisational change. We tried using MEASUR for the user requirements of the system we intended to create (as can be seen in Chapter 6). But these techniques were more appropriate for business and
organisation solutions. Also, we only intended to create a demo system as proof of concept therefore detailed user requirements were not needed.

2.5.5 Semiotic Engineering

Semiotic engineering is a relatively new area of semiotics concerned with Human Computer Interaction (HCI) [20]. On a theoretical level it is similar to Organisational Semiotics but it is solely concerned with the HCI level. It views HCI as a metacommunication between the users and the designers [21] and depends on user feedback for effective design. This view on HCI helps develop user-centered systems and this will influence the design of our prototype system in chapter 6.

2.6 Semiotic Analysis

The semiotic levels will be discussed below. There seems to be some minor conflict as to who first distinguished the semiotics levels, some attribute it to Peirce while other’s attribute it to Morris, as they both worked on it at the same time. The first three levels discussed were: Semantics, Syntactics and Pragmatics. These were later further extended by Ronald Stamper [60]. He added three more levels to create the six level framework of semiotics:

1. Physical: The physical support or basis for the communication. In the case of interpersonal (human-human) communication this is the anatomy and the physiology of the speakers and the air between them. In the case of human-system communication this extends to the hardware of the networked system.

2. Empirics: The physically observable activity. In the case of interpersonal communication this consists of the movements of speech organs, the acoustic patterns in the speech signals and the neural activity in the auditory system. In the case of computer based communication this extends to the movements of the fingers in activating the keyboard and the electrical signals that flow within the system and across the network.
3. Syntactics: Elements, structures and rules that form and govern expressions.


5. Pragmatics: The resultant achievement of the user’s intentions that lie behind and motivate the communicative activity.


To demonstrate, here is an example [14] of a semiotic analysis of the diagrammatic representation of a network using Stamper’s six-level framework:

1. Physical world: The paper or screen on which the diagram is manifested.

2. Empirics: The observable, perceptible manifestation of the diagram itself.

3. Syntactics: Under this heading there are three semiotic considerations:
   (a) The basic elements of the diagram, namely the nodes and arcs of the network diagram.
   (b) Any labels serving as annotations to the network diagram. These may well be in the form of natural language.
   (c) Any constraints imposed upon the combination of elements.


5. Pragmatics: The communication of the network configuration from the originator of the diagram to the reader (which could be the same person at a later time, in which case we speak of self-directed or reflexive communication).

6. Social world: The accomplishment of tasks that human beings such as engineers (considered as members of social communities and organisations) want done, for instance the (re)configuration of a network inside a spacecraft.
2.7 Contribution of Semiotics to the Present Study

As mentioned in the introduction, the advantages of using semiotics in an ad-hoc networked environment can be seen as follows:

- The semiotic framework accommodates any mode of communication whether it be speech-based, action-based or command-line.
- The framework applies equally well to user-system interaction as to communication inside the system (including communication across a network).
- The framework encompasses all the different facets of communication (including the physical and social context) and not just the physical hardware aspect, and it covers all forms of language (natural, formal or nonverbal).
- Semiotics accords with the holistic, systems approach to problems and their solutions. A holistic approach views a system as a whole taking into account every component within a system and the communication and linkages between them. Semiotics is similar in that every level is important and for a communication to be clear all the levels should be accounted for.

There are many benefits of using semiotics in an ad-hoc environment, as the transmission of intentions into a system is a form of communication, and many methods of human communication are applicable to computer interaction. The other aspect where semiotics can be useful is regarding communication within the system, where the exchange of messages in accordance with the appropriate protocols (norms) can be seen as a form of communication. For example, in a smart home environment, since connections between devices are dependent on the features of a device, we can exploit the notion of an arbitrary sign. For example, we do not need to know that the name of a device is a TV for it to be receiving a video signal; all we would need to know is that it is a device capable of receiving a video signal at a set bandwidth. Semiotics can be seen as a new approach for handling information within a system as well as during interaction with a system.
2.8 Final Remarks

This chapter was a brief history of semiotics and the areas it has been applied in. The main points behind the theory of semiotics will be carried onto our research and influence the design of the prototype system which will be discussed in chapters 5 and 6. These main points include: Stamper’s six-level framework, Peirce’s classification of signs and theories from semiotic engineering. Semiotic theories will influence the structure of the interface, the communication between the user and the system and inter-network communication. As mentioned in the introduction semiotics have not been previously applied to ad-hoc networks. Ad-hoc networks will be used here to assist in clarifying communication. Within the six-level framework, communication over ad-hoc networks can contribute to the pragmatics level as information about services and their locations are able to be transferred between different devices on demand. Once the system has information about the context of the communication and is ‘aware’ of its surrounding environment as well as the elements in it, it can make more ‘intelligent’ inferences from commands given to it by the user.
Chapter 3

Smart Environments

Environments where humans seamlessly interact with computers have been one of the main goals of AI-driven work for the past few decades. The smart home is a classic example of this case. The first smart house TRON [61] was created in 1989 in Nishi Azabu, Japan, in a collaboration between 16 Japanese companies. It was considered a technological feat at its conception, but it soon faced harsh criticism from the Japanese press which, along with its spiralling maintenance cost, led to its closure three years later. It was however useful in sparking interest in that area and companies, universities and research centres soon were all trying to achieve the same goal. This field became even more active from the mid-90’s onward, mainly due to the emergence of the web and the rapid advancement and cheaper costs of hardware technology. Also, it became more apparent that the technology behind these environments could be put to better use than simple automating household chores. This enabled more approaches concerned with moving the computer to the background and terms like ‘ambient intelligence’, ‘ubiquitous computing’ and ‘pervasive computing’ emerged and the area expanded to include intelligent business centres, workspaces and healthcare systems. Also as accessibility and inclusiveness began to play a larger role in systems design, the target audience widened and services were produced that could benefit and assist the elderly as well as people with disabilities. This section will discuss the different approaches taken to creating these environments as well as the many obstacles that have to be overcome.
3.1 Smart Home Technologies

Smart home technologies have been evolving over the past 20 years. It is important to make a distinction between the hardware and software concepts of smart home technologies. The hardware concept of a smart home is mainly concerned with what is called home automation or domotics. This is a subfield within building automation that deals with the wiring of the electronic components within a home and specialises in the application of automation techniques for the comfort and security of its residents. It uses a set of protocols, such as the X10 protocol\textsuperscript{1}, to control different devices in the environment. Each device in the environment reacts to commands specifically addressed to it, or possibly to several broadcast commands. This form of communication is highly developed but it is only hardware related, whereas in the present research we are concerned with the software side. Ideally, a software equivalent of this technology is desired that would take into account the inhabitants’ needs, perform ‘intelligent’ processing of information, and learn and predict inhabitants’ behaviours. This section will demonstrate some projects that have experimented with these ideas.

3.1.1 Live-in Labs

Live-in labs are home like environments. They are ideal testing areas as they allow for more natural behavioural observation and data collection on everyday activities such as cooking, socializing, sleeping, cleaning, working from home and relaxing than can be obtained from short laboratory visits \cite{39}.

MIT PlaceLab

PlaceLab is an MIT project and one of the more thoroughly researched applications of pervasive and ubiquitous computing. PlaceLab is a 1000 square foot flat located in a residential building in Massachusetts. It consists of a living room, dining area, kitchen, small office, bedroom and bathroom. It has micro controllers, networked sensors, actuators and wired switches that are discreetly embedded in their surroundings. The sensors monitor the conditions in the house such as temperature,
humidity, light and water flow. PlaceLab is optimized to perform multi-day and multi-week observations of individuals living alone.

**Microsoft EasyLiving Lab**

The EasyLiving lab [37] is a mock up of a small living room on the Microsoft campus using various non-traditional mechanisms, such as controlling them by speech, gesture, touching or using a wall display.

**Phillips HomeLab**

HomeLab is a house located adjacent to the Philips High Tech Campus in Eindhoven, The Netherlands. It is a complete home environment equipped with an advanced observation infrastructure for conducting feasibility and usability research.

**AwareHome**

The awareHome [45] is a project by Georgia Institute of Technology interested in creating an environment that is capable of knowing information about itself. The laboratory itself is a three-story, 5,040-square-foot house located on the campus at Georgia Tech. The house is currently configured with two apartments (one which will house actual residents in the near future) and a basement. The house is equipped with a number of sensors to record activity in the living space.

**3.1.2 Software Architectures**

MavHome (managing an intelligent versatile home) [15] is a project by the University of Texas, Arlington, interested in creating a house that acts as an intelligent agent. The main goal for this agent is to maximise the comfort and productivity of its inhabitants and minimise operation costs. To accomplish all this the house must be able to predict, reason about and adapt to its inhabitants.

To help achieve this the project was distributed among the following four layer agent architecture:
• Physical: Network hardware needed to gather information like: sensors, actuators, networks and agents.

• Information: Gathers, stores and generates knowledge for decision making through data-mining, action prediction and mobility prediction.

• Communication: Routes and formats information between agents.

• Decision: Selects which actions to execute, based on information from other layers.

Using these layers the mavHome is able to predict when and where to set alarms, switch on heaters, sprinklers, TV’s and other devices in the home. The strength of mavHome is in its sophisticated decision-making that helps it predict the inhabitants’ next move. It uses a combination of prediction algorithms (SHIP, Active Lezi, TMM and Episode discovery), discussed below, to intelligently select its next move.

Microsoft’s EasyLiving Lab [37] came up with the following list of requirements, that it referred to as grating factors, that are needed for smart homes to properly take off:

• Location Technology: Sensors that detect movement still only cover a low-range, are quite expensive and are still only good for monitoring one person at a time.

• Standard Device Protocols: Standard protocols that make appliances computer controllable are not uniform yet, as no major players are involved yet. So currently devices working on different protocols.

• Abstraction across devices: Get different devices to communicate in the same language regardless of their architecture.

3.1.3 Context and Information

Information is an important factor in these environments, and everything is measured and taken into account. The experiments in PlaceLab [39], for example, collected 20-25 GB of data per person per day and monitored dietary intake, daily activity and behaviours.
The awareHome [45] also focused on the importance of context. Information about
the users were gathered using sensors (such as wearable computing or ‘smart floors’
that tracked users’ movements), signal processing or by direct interaction through
LCD touch screens. The system then used this information to understand contextual
cue of the occupant.

A suitable use for all this information is in prediction and learning algorithms, the
mavHome [15] project, for example, used a set of prediction algorithms, which will be
briefly described below:

- **SHIP algorithm**: SHIP stands for Smart Home Inhabitant Prediction. It matches
  the most recent sequence of events with sequences in collected histories. The
  key disadvantage to this method is that it collects a large amount of data that
  make it difficult to give real-time response.

- **Active Lezi(ALZ)**: ALZ is based on the incremental LZ78 technique, a lossless
data algorithm created in the 70’s by Lempel and Ziv. Data compression enables
ALZ to process all the information online, giving it an advantage over the SHIP
algorithm.

- **Task-based Markov Models(TMM)**: This technique identifies high-level tasks
  in action sequences to help created Markov Models for action prediction. The
  Markov Models created from the action sequences and current state of the agent
  are used to predict the next action.

- **Episode Discovery**: This is a data-mining algorithm that identifies significant
  episodes within an inhabitant event history. A significant episode is a related
  set of device events that occur at regular intervals. A lot of the episodes in
  the environment occur daily or weekly. Activities such as switching one and off
  the coffee maker, the alarm, shower, and the lights are seen as daily, whereas
  activities like online shopping or turning the sprinklers on are seen as weekly.
  Activities that don’t happen frequently are not recognised.

- **Meta-Prediction**: This is where the prediction algorithms are combined to
  achieve the final result. The meta-predictor used by the mavHome is called
Predict. It takes in all the information and features it finds when an event happens and passes it through the four prediction algorithms, which in turn produce a confidence value. A voting scheme using each of these values produce the final prediction prediction.

3.1.4 Resource Co-ordination

The Intelligent Home [73] is a project by the University of Massachusetts that focused on the importance of resource coordination. The devices are deployed as distributed autonomous agents working in a simulated environment. The main focus of this project is resource coordination. These resources (mainly hot water and electricity) are also used as constraints on the system. This is a more realistic approach as the users of these homes would rather have products that are more efficient with resources, as well as the growing demand of having environment friendly products.

The system is given a map of the house and information on where these agents are located. The user provides a scenario on what is needed from the iHome and adds additional constraints to the ones already built into the iHome. It deals with goals on a priority level and has to make sure that the agents do not use the shared resources at the same time. To best explain the architecture of iHome, here is an example [73] of scenarios and constraints given to the iHome.

Consider a complex situation. It is 4 PM now, and the party starts at 6p.m. Before the party:

- the dishes have to be cleaned;
- the dining table cloth has to be washed;
- the house needs to be cleaned;
- room temperature needs to be set before the guests arrive;

Constraints:

- all the appliances cannot run at the same time due to resource constraints;
• some appliances have to be used concurrently, (dishwasher + dryer, washing-machine + vacuum cleaner)

• Additional constraint: The noise level should be low between 5 and 5:30 since some one wants to watch a TV show during that slot.

The system takes in these constraints and starts time planning the scenario as a problem solving process. It also has to coordinate its actions with that of the occupant. For example if the occupant is taking a shower the system knows that it can not use the dishwasher or the washing machine because the water resource is being used.

3.1.5 Natural Interaction in Computerised Home Environments

Natural Interaction in Computerised Home Environment (NICHE)[32] was a research project by the ISTI-CNR in Pisa in collaboration with the Domotics Lab of the Institute, the Dylan Lab in the Institute for Computational Linguistics and the W3C Italian Office in 2004. The main goal of this project was to deal with interoperability and integration of home appliances with next generation smart devices, while taking into account the impact of new emerging technologies on our daily habits.

NICHE focuses on the idea of using human natural language as a mode of interacting with devices in the networked home- a theme developed also in the research described in this thesis. It will allow users to communicate through natural language to control, query and program devices. The different components in this environment communicate with each other through DomoML. DomoML is an XML-based mark up language used by this system to define interoperability standards between the different resources. The mark-up language simply acts as a ‘glue’ between devices that do not have prior knowledge of each other’s services.

DomoML is split between three different layers:

1. DomoML-env(environment): A markup language that describes the spatial and behavioural properties of resources and their relationships with each other.
2. DomoML-fun(ctional): A markup language that describes the functional aggregation of resources. It defines the sequences of operations that carry out composed behaviours. This allows for the assignment of Natural Language descriptions of certain sets of actions.

3. DomoML-com(munication): A markup language for the communication between the different resources. Aims to create a common language between the different components based on information from DomoML-env.

The NICHE system depends on DomoML to make use of the information provided by various components. Information about the locations and attributes of components in the home environment are attained through DomoML-env, which gives the system an accurate representation of the home environment. DomoML-fun will define the functionalities of the services provided while DomoML defines the communication between the devices.

### 3.1.6 Assistive Technologies

Integrated systems based on Home Systems technology offer improved access and control of the domestic environment for the elderly who require assistance. Most elderly people prefer to live in their own homes as long as possible and be in control of their own lifestyle.

Projects such as EIAAL (Embedded Intelligence for Ambient Assisted Living) and CAALYX (Complete Ambient Assisted Living Experiment) are examples of how information could be gathered in elderly people’s homes in an un-intrusive manner. In the EIAAL information is gathered from wireless connected sensors distributed throughout the home and are used for fall detection, location tracking and drink monitoring, while in the CAALYX project, a lightweight unobtrusive device able to measure specific vital signs and to detect falls in the elderly. It also includes gadgets like geo-location systems to track movement and locations. Other forms of unobtrusive monitoring included embedding sensors in furniture that the elderly are most likely to use, such as sofas [38]. The aim of these technologies is to assist the elderly stay in
their own homes as most are forced to move out of their homes if they have difficulty living on their own and have no one to continually support them.

3.2 Smart workspaces

3.2.1 Stanford iRoom

iRoom was a project by Stanford in 1999 to investigate human interaction with large high-resolution displays. The work slowly evolved to interactive workspaces and Ubiquitous computing. Many prototypes for this workspace were created, but they were mainly made of two parts:

- iRos: a software infrastructure for this environment.
- iRoom: The room iRos was implemented.

iRoom was also used as the main project room for the group. The iRoom allowed users to adjust the environment instead of having the room react to the user. The idea for the iRoom centred around three main characteristics:

1. Moving data: The users in the room were allowed to transfer data between all electronic devices available in the room such as laptops and PDA’s.
2. Moving control: Any user can control any device in the iRoom.
3. Dynamic application coordination: There are many different software tools running in the iRoom; the activities of these tools should be able to co-ordinate with other tools whenever appropriate.

iRos was designed to include subsystems to match each of the iRoom characteristics:

1. Data Heap (moving data)
2. iCrafter (moving control)
3. Event Heap (dynamic application co-ordinator)
Unlike a standard PC, it had multiple users, devices and applications running at the same time. On short-time scales individual devices might be turned off and wireless devices enter and exit the room.

It worked only for room-based devices, but also allowed project teams access in remotely located workspaces.

### 3.2.2 Obje Project

The Obje (formerly Speakeasy) Project was a project that began in 2000 at XEROX’s Palo Alto Research Centre (PARC), exploring the idea of impromptu interoperability.

PARC have also coined the term ‘recombinant computing’. The idea behind recombinant computing is to setup an environment where different devices and services on a network can connect with little or no prior knowledge of each other. It also dictates that computing environments should be created from the bottom up.

Even though the devices do not have knowledge about other devices surrounding them, they must be able to communicate with them when required. This may sound implausible but it can be done using the idea behind the web. A web server and a web client communicate using a small set of protocols (HTTP, FTP) as well as the format of how this data is transferred (HTML, XML). Using these rules different programs are able to communicate with each other. Obje was an infrastructure that was created to exemplify this type of interoperability.

It is based on three premises:

1. Agreement on a small fixed set of universally known interfaces which can be applied to multiple purposes: Here we have a choice between domain dependent interfaces, where the protocols are particular to the domain of functionality of the device or service, and domain independent interfaces where all interactions are guided over a fixed set of protocols. The web is an example of this; different web pages do not use different protocols, but in most cases a generic interface is generated, capable of many uses.

2. Mobile code: allows parties to acquire new behaviour as appropriate. While
fixed laws are useful for communication, they also tend to be restrictive. Adding mobile code helps add dynamism to the rigid laws.

3. Users are ultimate arbitrators, as they are able to understand the semantics of these entities. Semantic arbitration happens at user level not development.

All Obje components and clients are expected to understand:

- Data transfer: How components exchange information with one another.
- Collection: How to classify devices into groups, making it easier for discovery.
- Metadata: How components use contextual information about themselves.
- Control: How components allow users and other components effect change in them.

### 3.3 Current Challenges of Smart Environments

Hardware and software difficulties are not the only issues that are standing in the way of creating these environments; there are also social aspects to consider. PARC, using the example of a smart home, summed up the main seven challenges faced [29]:

- The Accidentally smart home: Most of the work done in smart homes tend to assume that real smart homes will operate in a similar fashion to lab environments and that houses will be built from the start ready to accommodate new technology. It is true that in the future there will be purpose-built smart homes widely available but in the early stages consumers are more likely to gradually add smart devices to their homes, hence the term ‘accidentally’ smart home. How would these different technologies work together without the house owning any underlying technology to connect them together?

- Impromptu Interoperability: Continuing from the previous point, it would then be assumed that devices are not bought at the same time and the most likely case would be to have a cocktail of different devices from different brand. How would these devices operate with each other?
• No System administrator: If a simple home computer network involving at least two computers needs some system administration, then one can imagine how difficult the task would be when more devices enter the house. Especially in regards to how the operate with each other and with the inhabitants. These can easily put off consumers who have little or no computer skills. However, there are various solutions to this problem:

  – Having an on-site maintenance engineer who can be called whenever there are problems.
  – Keeping the devices appliance-centred so the faults can be easily traced back.
  – Using very simple easy-to-use front-end interfaces for the inhabitants, leaving all the intricate issues running the background

• Designing for domestic use: How would users adapt to new technology? Studies examining domestic settings to see how certain devices can be used have only recently been put forward. The paper also refers to studies by Hughes et al-Hughes, done regarding when shared devices such as the TV, Internet can be used in a household and the rules that govern their use. Therefore, before any new device enters the home, studies should be run to how occupants would use it on a long term basis.

• Social Implications of home-aware technologies: A smart home can make life easier but at the expense of privacy. The idea of an aware house may seem very intrusive to some people. To have a house aware enough to monitor your daily activities and predict your needs could be a security threat, especially if that type of information gets into the wrong hands.

• Reliability: Another main issue for consumers, the fear of futuristic devices crashing as frequently as our computers do. Of course domestic appliances are designed differently from desktop appliances. They do not have the luxury of receiving updates the way software does, so the devices have to make sure that these items are absolutely reliable before they are distributed on the market.
And then you also have the difference in technological approach; a faulty land phone will almost always be related to a bad network. However the same can not be said for mobile phones; the fault can be anywhere in the phone.

• Inference in the presence of ambiguity: Not all information can be solely detected by sensors or other input-detecting devices. There would be certain tasks, such as user preferences, that require human interaction. The main problem with human interaction is that human communication heavily relies on context. The usual problem lies in finding a midpoint between asking too many questions from the user (which decreases its usability) and processing incomplete or incorrect information.

The AMIGO project\textsuperscript{2} AMIGO(Ambient Intelligence for the Networked Home Environment) is a cross European project involving 15 companies and universities, namely Phillips, Microsoft, France Telecom and the Fraunhofer Institute. They conducted a cross cultural user study for 6 sites in 5 different European countries. The study used four different scenarios to show the different services provided by AMIGO. The scenarios ranged from home caring (such as automating chores), sharing ambiance (such as constantly on video conference to speak to family and friends), being followed by content (summarising favourite news stories, playing favourite music) and playing games (recognising and integrating game devices, asking for permission from parents and downloading game play lists). Participants were told to view these different scenarios and rank them. The results showed that people were outspoken in their desire to maintain control over their environments and like the automation of home chores (Home Caring). They also wanted to protect their children from inappropriate information (Game Playing), and liked the idea of personalising and summarising their favourite news stories (Being Followed by Content). They did not, however, rank the Sharing Ambiance scenario as highly as the others as they felt it invaded their privacy. They liked the idea of speaking to people through video but were not comfortable with the idea of the house being aware when users entered or left rooms.

\textsuperscript{2}http://www.hitech-projects.com/euprojects/amigo/summary.htm
3.4 Final Remarks

This chapter showed an overview of intelligent environments, the services they are able to provide and the challenges they currently face. This thesis is not about creating intelligent environments but more about interacting with them and exploiting their services to create a user-centered system. Representing communication between users and devices, as well as between devices, in this environment is important.

The ideas from this chapter that will influence the design of our prototype system are the importance of context, natural language interaction, mobile data and interfaces for inter-operability between different devices. Machine learning was discussed here, but it will not be used, as it is a deep field on its own. It was mentioned to highlight the advantages it brings to a system and can be seen as a possible extension to this research. It is worth noting that home environments, such as MavHome and PlaceLab, concentrated more on providing services and comfort to users, whereas the work environments, such as Obje and its focus on recombinant computing, were more interested in the inter-operability of different components. By using a semiotic approach to communication, as will be seen in chapters 5 and 6, we will create a framework that can combine the providing of services with interoperability.
Chapter 4

Technologies

4.1 Network Technologies

The previous two chapters separately discussed the areas of semiotics and intelligent environments respectively. This research aims to use the theories behind semiotics and apply them to ad-hoc networked environments. An application will be created as a proof of concept. It will combine different research areas such as ad-hoc networks, multicasting, human computer interaction (HCI) and natural language processing. These areas will discussed in the following sections.

4.1.1 Ad-Hoc Networks

As the name suggests, ad-hoc networks are networks that are configured on demand, as and when required, rather than pre-installed in the way that, for instance, a wired LAN network in an office would be.

Most ad-hoc networks are wireless, which means that they operate on the basis of radio frequency communication links. Each device on the network is seen as a node and is able to forward data to other nodes. The network is generated by the nodes, so there is no specific network hardware. Work on the wireless ad-hoc networks started in 1970s with ‘packet radio’ networks. Under the sponsorship of DARPA [34], two companies, BBN Technologies and SRI International, designed and tested these networks. Wireless ad-hoc networks were primarily used for military purposes, but
are now part of mainstream technology and are embedded in environments such as smart homes and m-commerce shopping environments.

Within the environment of our research, a number of wireless services offer information and knowledge to all devices that are present. In order for an ad-hoc network to appear, a set of devices must come close enough together in order to communicate. We further define ad-hoc communication to include situations where small devices connect over a wireless link to a fixed infrastructure. Examples of ad-hoc network components include:

- Bluetooth and short-range radio hardware, which have a range of up to 10 metres. These will support device to device communication.
- WiFi or 802.11, which have a similar range to Bluetooth, but more often support device to base-station communication. Where a base system is installed in a store, for example.
- DAB (Digital Audio Broadcasting) or DVB-H (Digital Video Broadcasting for Handheld Devices), for delivery of information through broadcast networks.
- GPRS or 3G, for connections over mobile telephone networks.

In these environments each device is independently purchased but they can be used together in collaboration.

The Roles of Service Discovery Protocols

Service Discovery Protocols (SDPs) are often used in ad-hoc networks; they allow for automatic detection of devices on a network. These technologies are still fairly new but they will provide the backbone for most future ad-hoc networked environments. Some examples of SDP’s are as follows:

1. Service Location Protocol (SLP): This protocol is used by devices to announce services on a local network. It has a URL that is used to locate its services and a number of pairings called attributes that describe its services.
2. Simple Service Discovery Protocol (SSDP): This protocol is the basis of the discovery protocol for Universal Plug and Play (UPnP) devices. They require little or no static configuration and provide multicast discovery support and discovery routing.

3. Zeroconf: A networking technique that can create usable IP addresses without prior configuration.

The protocols based on the Zeroconf Technique are Apple’s Bonjour Protocol and Linux’s Avahi Protocol. Bonjour can discover services on a local area network, and allow users to set up networks without any configuration. This way users can connect to a printer that is Bonjour enabled and connect to other computers to share files and services. Avahi is a free Zeroconf implementation for Linux. It includes a system for multicast DNS/DNS-SD service discovery. It allows programs to publish and discover services and hosts running on a local network with no specific configuration. Avahi will be used as the simulated wireless environment for our prototype system. The Avahi Daemon will be used to publish and browser services in the environment. Further explanation on its use and examples will be discussed in 6.2.2.

4.1.2 M-Commerce Environments

The concept of electronic payment started to take shape in the late 1960’s with the introduction of technology such as EDI, electronic data interchange, to exchange structured data between computers while adhering to an agreed messaging standard. The most common use for EDI was business transactions between companies. This was a time and money saving technology that facilitated and sped up the business process. It was first used by the transport industry and later evolved to support banking operations, and it remains the backbone of most e-commerce applications we know today. The advent of the web in 1994 expanded the market for companies by providing a front-end for consumers to directly purchase goods and services and eventually spawned online companies that did not require physical stores to sell products but solely depended on the web as their marketing tool. With the web access
becoming cheaper and accessible to more parts of the world, many web-only companies have been transformed into multi-billion dollar businesses. The introduction of new technology such as wireless networks and mobile devices, as well as interoperability between different devices, provided further evolution for the commerce industry such as u-commerce (ubiquitous commerce) t-commerce (television commerce) and m-commerce (mobile commerce). The current research work, however, is related to m-commerce. Further t-commerce reading can be found in [41, 55] and u-commerce in [17, 74].

M-commerce can be briefly summarised as electronic commerce on mobile devices. The latter, such as palm pilots, mobile phones, PDA’s, internet tablets and other handheld devices, can be used to communicate, purchase products or provide entertainment. M-commerce relies heavily on wireless technology and, to some extent, on ad-hoc networks, as these devices are expected to connect anywhere and anytime. As stated above, ad-hoc networks are networks where devices connect on demand and when required. The example we are interested in is similar to m-Mall [33] by Reinoso et al, where users can pick up information from other users who could be shopping in stores nearby, or they could interact with the stands in an exhibition. It is also worth noting that most m-commerce environments depend on impulse shopping. The idea that a certain product is just one click away makes it much more tempting and easy to buy than having to stand in a queue to pay for your product. The latter also gives you more time to ‘cool down’ and rethink your purchase. A study conducted by O’Hara and Perry regarding impulse shopping and m-commerce devices can be found here [54].

Advantages:

1. Ubiquity: instant access to information wherever a network is available.

2. Ability to compare prices between shops, as well to compare them with web-based shops.

3. Secure purchasing. Since most mobile phones have an embedded chip that stores personal information, this provides secure authorization and identification that
are not available in other devices. This leads some to believe [24] that mobile phones could replace smart cards as methods of payment.

Disadvantages:

1. Slow connections and limited and unattractive WAP sites.

2. Limited memory.

3. Accessibility issues, though this is related more to the size of the devices than to the environments.

4. Security issues. Even though purchasing through mobile devices is considered safe, the main problem is security on the network and keeping your device safe. Security issues will not be discussed in this paper as they are a topic on their own.

Semiotics is particularly relevant to M-commerce, where we have not only multiple devices but also multiple modes of communicating with them, such as textual input, speech, gesture or visual communication.

4.1.3 Multicast

Communication over the internet adheres to a set of rules, named Internet Protocols (IP), the best known of which are Transmission Control Protocol (TCP) and User Datagram Protocol (UDP). With regard to IP, multicast refers to a connection between a single sender and multiple receivers. It conserves bandwidth by sending a single stream of packets to multiple clients that have registered for this service. These packets are then replicated through routers. Multicast is very suitable for environments where real-time delivery is required, such as live TV and radio broadcasts and video conferencing, as well as for environments where users in different locations use shared resources such as whiteboard collaborations. The network is managed using the Internet Group Management Protocol (IGMP) \(^1\) that dynamically registers individual hosts in a multicast group on a particular LAN. Hosts identify group membership by sending IGMP messages to their local multicast router.

\[^1\text{www.faqs.org/rfc/rfc2236.html}\]
Advantages and Disadvantages of Multicast

Multicast has the following qualities that make it a suitable transmission tool for m-commerce devices:

1. Information is only delivered to groups that request it, unlike broadcast, where information is sent to the entire subnet, and unicast, where information is only sent to a single address.
2. Operation over wireless networks.
3. Efficient bandwidth usage, as only one stream is sent to users.
4. Support for distributed applications. This makes it very useful for m-commerce environments where different devices co-exist.

However, it does have the following disadvantages:

1. It depends on UDP which is not reliable.
2. All the routers in the network need to be enabled to multicast as well as be able to relay packets.
3. It cannot be applied to TCP communications, and therefore lacks mechanisms such as congestion avoidance.

4.2 RDF and Semantic Web Activity

4.2.1 Semantic Web

It is our aim that our work should be compatible with developments in web technology, and in particular the advent of the Semantic Web. According to the Semantic Web Activity Statement\(^2\), the goal of the Semantic Web initiative is to create a universal medium for the exchange of data. It is envisaged to smoothly interconnect personal information management, enterprise application integration, and the global sharing of

\(^2\)http://www.w3.org/2001/sw/Activity}
commercial, scientific and cultural data. In short, the Semantic Web is simply an extension of the current Web. It aims to make contents of the Web machine-readable as well as human-readable, using methods already available such as tagging and using them for new concepts such as ontologies.

A tag is a label that describes an object. Currently tagging is used in web development tools such as HTML (Hyper-Text Markup Language) for display purposes only (e.g. specifying the text’s colour, font and position on the screen) without giving any extra information about the content. Examples of HTML tagging are the following:

<font colour="blue">This tag shows blue text</font>
<p>This is the start of a paragraph</p>
<h1>This is a new header</h1>

Tags are enclosed by the “<>” brackets. Most tags occur in pairs the second member of which begins with “/”.

In order to facilitate the interpretation of data within the Semantic Web, the tags need to be able to represent information about the data that they enclose (see example below). Another example of a tag-based language is XML (Extensible Markup Language). In XML, unlike HTML, users are free to create their own tags. Here is an example of how XML can be used to describe different types of shops:

<shop_list>
  <Grocery_Store>
    <Name>MiniMart</Name>
    <address>Union_Street</address>
  </Grocery_Store>

  <Grocery_Store>
    <Name>Food_Market</Name>
    <address>King_Street</address>
  </Grocery_Store>

  ......
</shop_list>
Using these tags the machine can find out for itself information about the name and address of the grocery stores. Once such information has been determined it can easily be further processed and manipulated.

Tags can also be used to support the representation of ontologies. Ontologies describe the semantic structure of particular domains, including the relations between objects in those domains.

4.2.2 Motivation for RDF

In the previous section, we discussed the use of tagging and ontologies for representing information. In the Semantic Web, RDF is a highly suitable language for the representation of semantic information. RDF stands for Resource Description Framework and it is based on the XML language. It is a highly suitable language for the representation of semantic information. It stores information as a collection of triples. Triples are made up of three terms, generally known as Subject, Predicate and Object or Resource, Property and Value (though these labels are not ideal). The subject is the resource that is being described. The predicate is a property or trait of that resource, and usually expresses a relationship between the subject and the object. The object is the object of the relationship or value of that trait. We propose using RDF for the additional representation of information within the system.

The following example will demonstrate how RDF is used:

The user is located in (the fictional shop) Minimart.

This can be represented as the triple:

\[
\text{(user located Minimart)}
\]

where:

\begin{align*}
\text{Resource} & : \text{user} \\
\text{Property} & : \text{located} \\
\text{Value} & : \text{Minimart}
\end{align*}

It can be presented in RDF as:
As can be seen from the example, RDF uses the XML language to group properties about a certain subject (in this case the user). The “http://context” is referred to as the Uniform Resource Identifier (URI), and it is considered the address of the resource, so in this example the location of the user is part of the context of the system.

### 4.3 Multi-Modal Input

#### 4.3.1 Natural Language Processing

Natural Language Processing (NLP) is a research area that combines artificial intelligence (A.I.) with linguistics. It is concerned with the understanding and generation of human languages. In NLP, the term understanding is used to describe the analysis and interpretation of input by the computer without implying that it is equivalent to the comprehension of language by human beings. NLP garnered intensive research interest after the publication of Noam Chomsky’s *Syntactic Structures* [13] in 1957, in which he introduced the first method of formalizing linguistics.

Natural Language Understanding involves four steps:

1. **Word analysis:** This step breaks text down into individual tokens. The tokens can be words or non-words such as punctuation marks. This also includes morphological analysis, the purpose of which is to identify the grammatical components of words such as plural endings or suffixes.
2. Syntactic analysis: In this step, syntax rules are employed. The syntax is a set of grammatical rules that govern how structures such as sentences should be formed.

3. Semantic analysis: In this step structures are assigned meaning. Here, sentences that are grammatically sound like “Lead desks eat cake” can be rejected as they are semantically wrong. This requires a knowledge base with objects and their meanings.

4. Discourse integration: This step decides on overall coherence. The meaning of individual sentences depends on sentences that precede them. This step decides whether and how a stream of sentences are connected at some level, and if so then in what way.

These steps are in place to resolve ambiguity in text. Many words have different meanings and parts of speech depending on the context and on where they are placed in the sentence. A common example of ambiguity could come from homographs such as the word *bow*, which could either be a noun as in a *bow tie* or a verb *to bow*.

Another aspect of NLP is Natural Language Generation (NLG). This is the process of producing meaningful texts in natural language.

There are three forms of NLG:

1. Canned NLG: This is the simplest form of NLG where text is hard-coded into the system and can be used in limited domains like email business letters and spam.

2. Deep NLG: This can be seen as the inverse of natural language understanding, where the four steps discussed above can be done in reverse order.

3. Shallow Template-Based NLG: This method extracts what it sees as interesting information and fills a template to send to the user, the template here being based on context.

Due to the nature of NLP it can be used in many interesting areas, such as robotics, information retrieval, machine translation, text summarising, data mining and chat
bots, to name a few.

Natural Language can be manifested in two distinct media, text and speech. In this research The Natural Language Toolkit (NLTK) and Wordnet will be used to assist in parsing text. The Natural Language Toolkit (NLTK)\(^3\) is a suite of open source modules, data sets and tutorials supporting research and development in natural language processing. Wordnet\(^4\) is a large lexical database of English, developed in Princeton University. Nouns, verbs, adjectives and adverbs are grouped into sets of cognitive synonyms (synsets). Synsets are either linked lexically or through conceptual semantics. NLTK will be used to get the parts of speech of the individual words and using a *Chunking* procedure (discussed below) to parse phrases. Wordnet will be used to find synonyms and hypernyms to clarify the meaning of ambiguous word tokens.

The following is useful terminology that will be used in our system with regards to NLP.

- **Hypernym**: A word that encompasses the meaning of another word. For example ‘plant’ is hypernym of ‘rose’.

- **Deictics**: Characteristic of a word whose reference depends on the circumstances of its use. For example, in the sentence ‘I want to eat here and now’, the words ‘here’ and ‘now’ are deictics as they are particular to the circumstances of the speaker.

- **Stemming**: Is an attempt to reduce a word to its stem or base form. For example the stem of the words ‘worker’ and ‘working’ are all work.

- **Chunk Parsing**: Is a form of shallow parsing that makes subdivisions of a given input but does not specify their internal structure or their role in the main sentence. The output of chunk parsing produces noun-phrases, verb-phrases etc.

Speech technology represents a further aim of research. This encompasses the analysis of speech input (known as speech recognition and *understanding*), which is natural

\(^3\)http://nltk.sourceforge.net

\(^4\)http://wordnet.princeton.edu/
language in an auditory medium. However, it does have more complications than textual natural language:

1. Noise Elimination.

2. Accommodation to different accents and pronunciations.

3. The fact that spoken sentences are often not syntactically correct.

4. Homophones: How to differentiate between to, two and too, for example.

In the present investigation it is not our intention to advance the state of the art in NLP, but rather apply existing techniques to the problem in hand (similar to how natural language was applied in the NICHE project in 3.1.5). DragonSpeak/Sphinx will be used for speech recognition. Additional information about NLP can be found in Jurafsky and Martin’s *Speech and Language Processing* [44].

### 4.4 Human Computer Interaction

Human Computer Interaction (HCI) is an area concerned with the study, design and evaluation of human-centred interactive computer-based systems. The first notable device that allowed direct interaction and manipulation between a user and a computer was the drawing program Sketchpad [68] in a thesis by Ivan Sutherland in 1963. However it did not become established as a professional community until about twenty years later in 1982 at the inaugural conference of Human Factors in Computers Systems in Gaithersburg, Maryland. This was also around the same time that personal computers emerged, which pushed computers out of labs and encouraged the computer industry to create interfaces that were easy to use by consumers.

In order to support interactive computing a combination of input and output devices are required, and communication can be achieved through different modes (e.g. natural language, speech or gesture recognition) and through different media (i.e. auditory, visual or haptic). Users can communicate with computers using input devices such as keyboards, mice, cameras, scanners, eye-gaze detectors, styluses and microphones, which can be used in connection with a variety of different modes such
command-line, natural language and WIMP (windows, icons, menus and pointers), while the computer interacts with users through output devices such as printers, screens and speakers. The role of HCI is to provide users with easy to use input media and provide informative feedback quickly.

It is also important to note that the choice of input and output modes varies between different users. While some users prefer highly graphical interfaces; others find it much quicker scripting and using command-line functions. Therefore these interfaces need to be adaptable to different tastes.

HCI is a large area and covers User Interface Design, Interaction Design, Usability and Accessibility. Further information on HCI can be found in Dix et al [25].

4.4.1 Usability

Usability is a very important feature in HCI. While some degree of human interaction is needed to communicate the user’s goal to the system, no one would like to go through long command dialogues every time they want to perform a certain task. In user interfaces, usability refers to the ease of achieving a particular goal employing a particular tool. In industry, companies are willing to spend a lot of time and effort on developing usable interfaces, this is mainly because usability:

1. Promotes productivity: Awkward interfaces can discourage users, while usable interfaces accelerate the accomplishment of tasks.

2. Provides an edge over competition: Systems with usable interfaces can overtake other systems with similar features but less developed interfaces.

Usable systems should be efficient in the resources they utilize, effective in achieving the desired task, safe to use and satisfying to employ. There are no clear-cut rules that can explain how these characteristics can be obtained, but there are design guidelines such as Shneiderman’s [62] eight golden rules that can assist good design:

1. Strive for consistency: Allow for consistent naming structures and action sequences through the system.
2. Enable frequent users to use shortcuts: Cut down number of interaction to increase pace.

3. Offer informative feedback: System feedback proportional to the actions, for example frequent minor actions should receive minor feedback while larger more infrequent actions should receive more.

4. Design dialogue to yield closure: Divide interactions into chunks so users know one interaction ends and where another begins.

5. Offer simple error handling: Provide easy to read mechanisms for error handling.

6. Permit easy reversal of actions: Undo errors easily.

7. Support internal locus of control: Make users feel in control of the system not the other way around.

8. Reduce short-term memory load: Design simple displays and provide adequate time for training and in between sequence of actions.

Usually there are two different approaches for designing usable systems [25]:

1. Paradigms: Use the ideas behind a successful system as an example for future designs.

2. Principles: Consider factors like psychological and social aspects of a domain (such as Shneiderman’s guidelines) and derive design theories through them.

Examples of successful paradigms are timesharing [48] which allowed resources to be shared by different people on the same computer (instead of using different computers for different resources), Visual Display Units (VDUs) which were preferable to printouts, the first drawing tool the Sketchpad [68] program mentioned in the introduction, Windows WIMP features, and currently Ubiquitous Computing, where computers blend into our everyday lives.

Principles that support usability are Learnability, how easy and quick a system is to learn, Flexibility, how flexible it is with regards to the modes of information exchange, and Robustness, how quickly it achieves users’ goals.
Evaluating Usability

One method of evaluating usability is to use finite state modelling. A finite state models behaviour of a number of states and the transition and action between them.

One example here is Thimbleby's work on Analysis and Simulation of User Interface [70]. The scope of the work for this method was mainly push button devices and websites. Each model was a finite state machine where key press’s were state transitions, and probabilities of key presses were provided. Usability was measured as the time (transitions) it took to reach a state. The probabilities were the likelihood of next button press. Other works of Thimbleby can be found here [69] [71]. The more traditional route for evaluating usability is to get a representative set of users to test the application and use their feedback to improve the system. The following methods are used to perform evaluation:

- Observation: This is where the user interaction is observed and recorded and later evaluated.
- Experiment: This is a scientific procedure where a hypothesis is set up and tested.
- Query: Using questionnaires and interviews to elicit users options.

User evaluation will be used for the outcome of this research. It will combine elements from experiment and query evaluation. A set of simple walk-through experiments will be given to the user followed by a questionnaire to fill on their experience with the system. The evaluation and results will be discussed in Chapter 7.

4.5 Final Remarks

This chapter presented an overview of the ideas and tools that will be used to develop the prototype system. Using a semiotic approach, we will create a prototype system that will work in an ad-hoc network in both the home and shopping environments. Information in both these environments will be multicast through the service discovery protocol, Avahi. Information regarding elements in the environment will be stored
in RDF format. Parsers will be used to process information from the user as well as within the system. We will also take a semiotic engineering approach to HCI and include users in the design of the system. The following two chapters will discuss how these ideas will be implemented.
Chapter 5

Applying Semiotics to the Representation of Networks

5.1 Introduction

As implied in the introduction, semiotics can be applied to three aspects of ad-hoc networks:

1. Communication between the user and the system.
2. Communication among devices within the system.
3. Representation of the network within the system.

None of these areas has been thoroughly researched and so it is the purpose of this thesis to explore them. This chapter is concerned with the representation of the network within the system. The other two aspects are dealt with in subsequent chapters.

5.2 Representing the Network

In its simplest form, a network is a set of items, called nodes, and the connections between them, called arcs. These networks can be defined as graphs, where Graph = (Node, Arc), and displayed in figure 5.1.
Figure 5.1: Diagrammatic Representation of a Network

Where:

- Circles represent nodes.
- Lines represent arcs.
- Numbers on the nodes represent node id.
- Numbers next to the arcs represent arc id.

These are the basic elements of a network. Complex networks require other features, such as information about each node and rules/protocols regarding how they connect.

Figure 5.1 is a diagrammatic representation of a network on paper. As we have seen in chapter two, using the six-level semiotic framework, the networked environment can be represented as follows [14]:

1. Physical World: The paper on which the diagram is manifested.
2. Empirics: The observable, perceptible manifestation of the diagram itself.
3. Syntactics: The basic elements of the diagram, namely the nodes and arcs of the network diagram.

5. Pragmatics: The communication of the network configuration from the originator of the diagram to the reader.


This diagram could also be described in a logic formulation:

\[(1)\]
\[
\text{Node}[N1] + \text{Arc}[A1] + \text{Node}[N2]
\]
\[
\text{Node}[N1] + \text{Arc}[A2] + \text{Node}[N3]
\]
\[
\text{Node}[N2] + \text{Arc}[A3] + \text{Node}[N4]
\]

In terms of the application of Stamper’s framework, this formulation differs essentially only at the syntactical level. The syntax here involves letters, brackets and mathematical signs instead of nodes and arcs. If this formal representation was to be stored in a computer then it would only differ from the paper-based version on the physical and empiric levels. The physical level would be the hardware where representation is stored and the empirics would be the internal state of the storage device.

We are going to use the term _plex_ to denote a network at the syntactical level (this is not the same as 'plex' in database theory). The formal representation of a plex can be generated by a grammar:

\[(2)\]
\[
\text{Plex} \quad \rightarrow \quad \text{Link}[n]^+
\]
\[
\text{Link}[n] \quad \rightarrow \quad \text{Node}[n] + \text{Arc}[n] + \text{Node}[n]
\]

The superscript $^+$ refers to one or more elements. The $n$ between the square brackets is shorthand for features such as $[\text{id} = n]$, and we can have more than one feature for example $\text{Node}[3,\text{TV}]$ which would be shorthand for $\text{Node}[\text{id} = 3, \text{Name} = 3]$.

On the basis of the above grammar, a description corresponding to diagram 5.1 can be represented as follows:

\[(3)\]
In this particular area of application, unlike natural language, there is a one to one mapping between the syntax and semantics. So the syntactic elements can be mapped onto the semantics elements as follows:

(4)
\[
\begin{align*}
\text{Plex} & \rightarrow \text{Network} \\
\text{Link} & \rightarrow \text{Connection} \\
\text{Node} & \rightarrow \text{Device} \\
\text{Arc} & \rightarrow \text{Via}
\end{align*}
\]

Therefore the syntactical grammar given in (2) is in effect equivalent to this semantic grammar:

(5)
\[
\begin{align*}
\text{Network} & \rightarrow \text{Connection}[n]^+ \\
\text{Connection}[n] & \rightarrow \text{Device}[n] + \text{Via}[n] + \text{Device}[n]
\end{align*}
\]

This approach to characterizing networks is novel and brings several benefits. Firstly this grammar is economical in representing networks of any scale. Secondly, grammars open up the possibility of parsing representations of networks, and because parsing techniques are widely recognised, we can exploit and adapt some of their features. Using a parser here is economical because parsing is already used in the system (for
the user to network communication, discussed in the next chapter), therefore the same parser, via a different grammar, can be used again for the network representation. This is also useful because grammars set out the rules governing well-formed structures, such as networks, and a parser can use a grammar to check the integrity of a network.

5.3 Overview of a Generic Grammar

The networks used here are ad-hoc computer networks. These provide a suitable illustration of the application of semiotics as they relay large amounts of information from various sources. A grammar is generated to check for valid connections within a network. This grammar is generated every time a new service appears or disappears on the network as well as when the RDF information about a service is edited. The ideal environment for this is the office and home environments where services need to connect to each other to provide what a user requires. For example, laptop computers connecting to printers on demand or a TV connecting to a security camera when the doorbell rings. In order to get the right connections, the system must know in advance which services are capable of being connected to one another. The following sections will elaborate how this theory can be applied on our prototype, but the general structure of parsing a network using a generated grammar is as follows:

1. Start with a simple rule for the grammar required: For example we can state that a ‘A connection occurs between services that send and accept the same type of data’.

2. Expand the rule by information from the RDF files of the services: For example if a service sends stream X, create a rule that states ‘X-Sending Service can connect to an X-Accepting Service’, these rules will be written in short hand, of course, as will be seen in the following sections.

3. Store these generated rules in a file: This will be used by the parser later as a ‘grammar’.

4. Tag the services in the ad-hoc environment: This is similar to the part-of-speech
tagging in natural language, here a service is tagged by what it can provide. For example a printer can be tagged as a ‘binary stream accepting device’ while a TV can be tagged as a ‘video stream accepting device’. Again, these tags will be written in shorthand. The tagging provides us with more information than the name or the brand of a service would.

5. Feed the ‘grammar’ into the parser and run the parser on the tagged network: This is similar to natural language in the way individual tokens are ‘chunked’ together to form phrases and phrases to form sentences. Here, complimentary services form a connection and connections form a network. The output of this stage is a ‘Network Parse Tree’

6. Store the Parse Tree in RDF format.

The first three steps in this procedure generate the grammar while the last three steps perform the actual parsing. The parser can be used to check for the integrity of a network and it can provide contextual information about the network that can be understood by a machine.

5.4 Illustrative Scenario

The previous section provided a general overview of how to parse a network. This section will elaborate on the details discussed above by walking through an example. The ad-hoc environment used for this example is the house environment, where a number of devices are co-located together and connected to a common networking medium, which may be either wired or wireless. Within such a network the devices communicate to determine the services that each can offer; and then the collaboration offers a set of combined services to the user or users.

Whenever a device enters a network, whether through a physical connection (being plugged in) or coming into the range of a wireless connection, it must communicate its presence to the other devices and the services it provides. In addition, devices require the ability to enquire about the services that other devices offer and to respond to such enquiries themselves. Devices therefore provide services to other devices; and the
combinations of these services (super-services) provide benefits to the user or users of the network.

The novel method of representing a network, as described in the previous section, will provide useful information in assisting the network to self-configure when required (such as when new devices are introduced to the environment).

The following scenario will be used as a walk-through for the next section: We have a house environment made up of three areas: lounge, kitchen and garden. Devices that are available in the house (e.g. the lounge and kitchen) are on a wired connection, which has a high bandwidth, while devices outside the house are on a wireless connection, which has a low bandwidth. The devices in this environment and their attributes are as follows:

(6)

Tuner: Transmits video and audio streams. It is located in the lounge and therefore can be connected to a wired network and sends a high bandwidth connection.

TV: Receives video stream. It is located in the lounge and therefore can be connected to a wired network and receives a high bandwidth connection.

Speakers: Receive audio stream. They are located in the lounge and therefore can be connected to a wired network and receive a high bandwidth connection.

Transcoder: Receives video and audio streams. It converts high bandwidth connections to low bandwidth connections. It is located in the lounge and therefore can be connected to a wired network and receives a high bandwidth connection.

All information regarding the environment and the devices in it are stored in a dynamic RDF file, that adapts quickly to network re-configuration.
5.5 Parsing of Network Representation

Using the information above, the system should find all the valid connections between the various devices. Figure 5.2 demonstrates how we can represent the valid connections on paper.

- HighVideo Connection means: A connection that involves high bandwidth video streams.
- HighAudio Connection means: A connection that involves high bandwidth audio streams.

The description of this environment is as follows:

- The basic element in this environment is a device. (e.g. TV, Tuner, Speaker or Transcoder).
- A pair of devices that communicate with each other form a connection.
- A set of connections form a network.

In order to represent this within a computer, simple techniques drawn from natural language technology will be used. This involves two steps: the first step is to generate
a grammar dynamically from information about devices and the second step is to use this grammar to parse a network to check if it is well-structured.

5.5.1 Generating the grammar

The first step here is to create a grammar for the parser. The grammar is generated dynamically, based on the information in the RDF files. These rules are generated every time a device goes offline or a new device comes online. The RDF files need to be as flexible as possible (i.e. their authors should be able to put in any information they want), but at the same time the files need to have a consistent format. As a compromise, a few keywords are selected to always be present in every RDF file, such as bandSends and bandAccepts, as the bandwidth is an important factor in connectivity between devices, and keywords ending with Sends and Accepts suffixes, representing the type of data that will be sent or received (e.g. streamSends and streamAccepts). These keywords are used in generating the grammar (as will be shown below). Since this is a simple scenario, only a small number of keywords are needed. So for this environment, devices are deemed compatible if they communicate in the same bandwidth type and send and accept the same datatype. For example, the following are snippets from the RDF files of a Tuner device:

```
<?xml version="1.0"?>
<rdf:Description rdf:about="http://Context/TUNER">
  <ns0:bandSends
    xmlns:ns0="http://Context/">high
  </ns0:bandSends>
</rdf:Description>

<rdf:Description rdf:about="http://Context/TUNER">
  <ns0:streamSends
    xmlns:ns0="http://Context/">video
  </ns0:streamSends>
</rdf:Description>
```
and a TV device:

```xml
<rdf:Description rdf:about="http://Context/TV">
  <ns0:bandAccepts
    xmlns:ns0=":http://Context/">high
  </ns0:bandAccepts>
</rdf:Description>
```

```xml
<rdf:Description rdf:about="http://Context/TV">
  <ns0:streamAccepts
    xmlns:ns0=":http://Context/">video
  </ns0:streamAccepts>
</rdf:Description>
```

From the previous chapter, we can translate these RDF files into:

<table>
<thead>
<tr>
<th>Resource</th>
<th>Property</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>Tuner</td>
<td>bandSends</td>
<td>high</td>
</tr>
<tr>
<td>Tuner</td>
<td>streamSends</td>
<td>video</td>
</tr>
<tr>
<td>TV</td>
<td>bandAccepts</td>
<td>high</td>
</tr>
<tr>
<td>TV</td>
<td>streamAccepts</td>
<td>video</td>
</tr>
</tbody>
</table>

The first step is to generate a rule that captures the notion that a network is made up of a number of connections. This rule consists of a ‘Network’ symbol followed by an arrow (meaning ‘consists of’) and a symbol representing one or more connections.

(7.1)

\[ \text{Network} \rightarrow \text{Connection}[\ast]^+ \]

The * stands for one or more, as yet uninstantiated, features.

Then, the grammar builder takes in the RDF files as input and searches for the type of bandwidth that each device sends. In this example we have a tuner that sends high bandwidth. It assumes by default that a Sends device must have a complementary Accepts device. To do this, the system generates a rule which consists of a ‘Connection’ symbol with the feature ‘high’ (to represent the bandwidth) followed by an arrow and
a sequence of three symbols. The first of these symbols consists of the ‘Device’ symbol with the feature ‘highBandSends’, followed by the ‘Via’ symbol with the feature ‘high’ (again representing the bandwidth) and then another ‘Device’ symbol with the feature ‘highBandAccepts’.

(7.2)

Because the feature ‘high’ is duplicated in the ‘Connection’ and ‘Via’ symbols it is feasible to simplify (7.2) by omitting the ‘Via’ symbol resulting in:

(7.3)
Connection[high] → Device[highBandSends] Device[highBandAccepts]

(7.3) would therefore be the basis for the computer implementation of the rule generating process.

Matching bandwidth is not the only criteria for a valid connection here. The devices should also send and receive the same type of data. The next step is to look for Properties that have the Sends suffix. Again in this example, we see that the tuner sends video streams, and like the previous step it is assumed by default that the Sends device must have a complementary Accepts device. So the rule now becomes:

(7.4)
Connection[highVideo]
→
Device[highBandSends,videoStreamSends]
Device[highBandAccepts,videoStreamAccepts]

So putting it all that together we get the following parsing rules:

(7.5)
(1) Network → Connection[*]⁺
(2) Connection[HighVideo]
→
Device[highBandSends,videoStreamSends]
Device[highBandAccepts,videoStreamAccepts]
Within the system, we use abbreviations to describe the properties of a device, so for example:

(8)

<table>
<thead>
<tr>
<th>Property</th>
<th>Abbreviation</th>
</tr>
</thead>
<tbody>
<tr>
<td>High</td>
<td>hi</td>
</tr>
<tr>
<td>Low</td>
<td>lo</td>
</tr>
<tr>
<td>Video</td>
<td>vi</td>
</tr>
<tr>
<td>Audio</td>
<td>au</td>
</tr>
<tr>
<td>StreamSends</td>
<td>SS</td>
</tr>
<tr>
<td>StreamAccepts</td>
<td>SA</td>
</tr>
<tr>
<td>BandSends</td>
<td>BS</td>
</tr>
<tr>
<td>BandAccepts</td>
<td>BA</td>
</tr>
</tbody>
</table>

Using description (8), rule (7.3) will look like this within the system:

(9)

Connection[HighVideo] → Device[(hiBS)(viSS)] + Device[(hiBA)(viSA)]

Which is a shorthand way of saying highbandwidth videostream sending devices and highbandwidth videostream accepting devices form HighVideo Connections.

The following rules are generated for the environment based on information from (6):

(10)

(1) Network → Connection[*]^+
(2) Connection[HighVideo] → Device[(hiBS)(viSS)] Device[(hiBA)(viSA)]
(3) Connection[HighAudio] → Device[(hiBS)(auSS)] Device[(hiBA)(auSA)]
(4) Connection[LowVideo] → Device[(loBS)(viSS)] Device[(loBA)(viSA)]
(5) Connection[LowAudio] → Device[(loBS)(auSS)] Device[(loBA)(auSA)]

These dynamically generated rules are stored in a file so that they can be used by the parser.

### 5.5.2 Parsing the Network

This stage involves two steps:
The first step is to take each element in the network and create a tag for it based on its features (i.e. services it provides), the features we are concerned with here being the bandwidth and the type of data the device sends. So for example a Speaker device in a wired area can be described as *High Bandwidth Accepts, Audio Stream Accepts*. From (8) we know that this can be abbreviated to *hiBAauSA* and that will be the Speaker device’s tag.

Second, once the network is tagged properly, the elements are ready to be parsed. Using data from the example the input is passed to the parser (with its associated semantic grammar from the previous section) and the output (valid connections) is as follows:

(11)
(Network:
 (Connection:
   (HighVideo:
     ('Tuner', 'hiBSauSSviSS')
     ('TV', 'hiBAviSA')
   )
   (Connection:
     (HighAudio:
       ('Tuner', 'hiBSauSSviSS')
       ('Speaker', 'HiBAAuSA')
     )
   )
   (Connection:
     (HighVideoAudio:
       ('Tuner', 'hiBSauSSviSS')
       ('Transcoder', 'hiBAloBSauSAviSAauSSviSS')
     )
   )
)

This translates to:

- Network with the following connections:
High Bandwidth Video Connection between Tuner and TV.
High Bandwidth Audio Connection between Tuner and Speakers.
High Bandwidth Video and Audio Connection between Tuner and Transcoder.

Now if, for example, a user were to enter the lounge, unplug the TV and carry it outside to garden, what steps would the system take? In this situation the TV and Speaker, as they are both in the same device but represent two different entities on a network, will accept low bandwidth streams as the diagrammatic representation 5.3 shows.

![Diagram](image)

Figure 5.3: House Environment after TV is moved outside

Because the device isn’t a *Sending* device, no new rule will be generated, so the system will have to go ahead with the rules from the previous example. The parser will however give a different result:

(12)
(Network:
(Connection:
  (HighVideoAudio:
    ('Tuner', 'hiBSauSSviSS'))

68
Which translates to:

- Network with the following connections:
  - High Bandwidth Video and Audio Connection between Tuner and Transcoder.
  - Low Bandwidth Video Connection between Transcoder and TV.
  - Low Bandwidth Audio Connection between Tuner and Speakers.

As can be seen from the above scenario a new semiotically-based form of representation has been used and deployed successfully. The fact that the network is parse-able indicates that it is well-formed, and the parser is also useful in automating representations of the network which, in turn, keep up with changes within it.

5.6 Final Remarks

Ad-hoc networked environments provide a facet of retrieving information about the environment. In these networks devices and services that have no prior knowledge of
each other can query each other for services and give out information about themselves when required. We exploit this ability to retrieve the services and try to make valid suggestions on the possible connections between these different services based on the information from the RDF files. Starting with a simple rule a grammar could be generated and applied to a network. This is useful in providing contextual information that aids the pragmatics level of communication according to the six-level semiotic framework (discussed when designing the prototype in the next chapter). It also saves on the decision making aspect of the system in that the system does not need to infer about connections from each service or device because that information will already be provided.
Chapter 6

System Overview

In the previous chapter, we discussed that semiotics can be applied to three areas of ad-hoc networks:

1. Communication between the user and the system.

2. Communication among devices within the system.

3. Representation of the network within the system.

This chapter will show the implementation of a demonstration system that will incorporate these three application areas to create a multi-modal multimedia application.

6.1 Requirements

In order to investigate the application of semiotics to self-configuring networks (SCN), it is appropriate to develop a prototype system in order to try out and illustrate the ideas. The environments this system will be developed for are the home and shopping environments. This section will describe the functional and non-functional requirements needed to implement such a system. Because it is a demo system, sophisticated requirement analysis was not necessary.
6.1.1 Functional Requirements

The prototype system will behave as an Intelligent Remote Control (IRC) that will interface between the users and their environments and allow them to manipulate devices and services in those environments. The IRC device will be designed with the following requirements:

**Operate in a Networked Environment**

For the prototype to operate, a wired and/or wireless network is needed. Within this network, devices will be able to register their presence as well as their services every time they come online, and devices will also notify the network if they go off-line. The prototype system will use this network to connect to the other networks available and browse through the devices and services available online as well as pass commands to them.

**Provide an Interface between the Users and the System**

The prototype will have a multi-modal interface that will allow for graphical textual and spoken input which users will employ to interact with their environment. The user interface will provide a simple visualisation of the networked environment that the users are currently in. It will provide tools that allow the user to manipulate information to provide the services they require. The visualisation of the network is directly connected to the network, so all actions performed on the visualisation will directly affect the actual devices and services in the network, and conversely any activity in the network will be directly represented in the user interface.

**Parse Information in the Environment**

The prototype will have a parser that is able to process and analyse all forms of input passed to it. The parser will be economical and extendible so that the same parser will be used for three different types of input, namely:

- Network Representation (discussed earlier in chapter 5).
- Natural Language Input.

- Graphical Input: This will be achieved via mouse gesture and therefore be referred to as gesture input.

The parser will be designed semiotically so as to give a uniform representation of input. Parser input can come from two sources, the network and the user. Network input will parsed based on pre-defined rules of how a well-formed network is represented. User input is processed using the same parsing algorithms, but of course with different grammars, regardless of the mode (natural language, gesture or speech) or medium (auditory, visual or haptic) it is expressed in.

**Incorporate Decision Making**

One of the components of the system is the decision making whose role is to combine a rule-base with the information coming from different sources to achieve a desired goal. Information is received from the network, the user as well as the devices in the network. The decision making system will process all these inputs in the same manner and match them against its rule-base to find the best method of achieving a task.

**6.1.2 Non-functional Requirements**

The main idea behind creating this prototype is to demonstrate a proof of concept. Because a readily available physical smart environment is not available, a simulation is used. Devices and their services are also simulated. The two environments simulated here are the house and shopping environments. The environment is simulated by running processes on laptops that can send information across a network and produce similar actions to real physical devices when called.

The prototype system will be built on the Linux Ubuntu Platform. The final prototype will work as two different versions:

- As a standalone program on the Nokia 770 internet tablet.

- As an emulator on the Ubuntu Linux Platform.
The Nokia 770 adds further constraints on the system, mainly memory usage, as it has only 128 MB (split between 64 MB RAM and 64 MB flash card).

Data Storage

A lot of information is passed across this environment such as:

- Service Information.
- Parser Output.
- Network Representation.
- Contextual Information.

Because of the nature of this environment we require data that is not expensive on memory (unlike a database), that is well-structured (unlike a text file) and portable. Based on these features, the RDF data model is chosen, as it takes up a similar space to a text file and it is formatted as well as being portable. In addition it has APIs (application programming interfaces) that allow it to connect to many programming languages, making it easy to transfer information across.

As discussed in chapter 4 the RDF triplets will be renamed as resource, property, value (RPV) instead of subject, property and object, and the format of an RDF model is as follows:

```xml
<rdf:RDF
...

<rdf:Description rdf:about='http://Display/SonyTV'>
<ns0:FriendlyName xmlns:ns0='http://Display/'>MyTV<ns0:FriendlyName>
</rdf:Description>
...
<rdf:RDF>
```

The italicized characters are the main elements in the RDF data model. The first element:
http://Display/SonyTV

is the Uri(Uniform Resource Identifier) of the RDF. The first half (http://Display) refers to where the RDF is stored; in the current system this half of the Uri will include the filename. The second half (SonyTV) is the Resource in the RDF data model. The second italicized element:

*FriendlyName*

refers to the Property of the Resource SonyTV (Note: Friendlynames are editable by the user and help them avoid technical names such as ‘nntz150’ when referring to a device). and the third italicized element

*MyTV*

is the Value of the Resource SonyTV.

For readability the system will include a feature that will print out an RDF statement, in the format of:

```
----------------------------------------
Printing RDF:
----------------------------------------
| Resource | Property | Value |
----------------------------------------
| SonyTV   | FriendlyName | MyTV |
```

Other modules added to the RDF format to ease transfer of information between different elements. They include:

- Converting between RDF format and dictionary and vice versa.
- Converting between RDF format and tree and vice versa.
Brief Notes on the Programming Languages Used

The implementation here is not language specific. The only requirement is to find a language that is easy to use and seen as best fit for providing the solution. Python was selected because it was seen as the ‘glue’ that connects many different libraries together. Interesting python libraries include:

- PyGext: This stands for pygame extension, it is usually used to design games. This is one of the programming languages considered in the environment visualisation of the user interface.

- PyGlade: This is a rapid application development program in python. It is also one of the programming languages considered in the environment visualisation of the user interface.

- LibRDF: This library includes Python Bindings for the RDF library. This makes it easier to get data from and convert to RDF.

- NLTK: This stands for the Natural Language Toolkit\(^1\). It is a suite of open-source libraries and programs for symbolic and statistical natural language processing (NLP) for python. A version of it, known as NLTK-Lite, will be used for the parsers.

- Wordnet Bindings: This includes python bindings for the wordnet \(^2\) tool. Wordnet is used for finding bases or words as well as synonyms and hypernyms.

- pyCLIPS: This includes python bindings to the CLIPS language (discussed below).

The CLIPS (C Language Integrated Production System) language is an expert system created by NASA. It is mainly a forward chaining rule system. Forward chaining starts with the available data and uses inference rules and pattern matching to extract more data until an optimal goal is reached. CLIPS will be used for the decision-making in the environment. It is extendible, portable and handles information very well. It

\(^1\)http://nltk.sourceforge.net/
\(^2\)http://wordnet.princeton.edu/
forms its decisions based on information stored in RDF about the environment as well as pre-defined rules that are passed to it. Below is the format of a standard CLIPS rule:

(defrule Tired
  (question-is tired)
  (status-is just-woke-up)
  =>
  (assert (solution-is drink-coffee))
  (printout t "Have some coffee." crlf)
)

This rule can be de-constructed in the following manner: The first part:

defrule Tired

refers to the name of the module.

The elements on the left hand side of the module (i.e before the => sign) are the conditions or facts of a rule.

(question-is tired)
(status-is just-woke-up)

The elements on the right hand side of the rule (after the => sign) are the results of the rule.

(assert (solution-is drink-coffee))
(printout t 'Have some coffee.' crlf)

So, the rule above states that if a person is tired and has just woken up, make them have coffee.

6.2 Design and Implementation

Having established the requirements for the prototype system, we now turn to the details of its implementations (i.e. its design and construction).
6.2.1 System Architecture

![Diagram of System Architecture](image)

Figure 6.1: Diagrammatic Representation of the System

Figure 6.1 is an overall representation of the networked environment includes the interactive system and the collection of devices with which the later interacts. The elements in the networked environment intercommunicate by means of software tools which can be divided into the following classes:

- **Network Tools**: This includes publishing and browsing for services on a network as well as communication between users and services across a network.

- **Services**: The services are provided by different devices or shops (depending on the environment); they are displayed to the user using the network tools.

- **System Tools**: These include tools that enable user-network interaction as well as the network representation tool discussed in the previous chapter.

6.2.2 The Network Tools

As mentioned in chapter 3, the back-bone for most smart environments are wireless ad-hoc networks. Elements in smart environments should be able to communicate with each other without prior knowledge of one another. The tool used in this demonstration to simulate an ad-hoc wireless network is Avahi, introduced in 4.1.1. Avahi is
a multicast service discovery tool based on the Zeroconf standard. This tool enables its users to publish and browse the services that join a wireless network. Services that have the same service type will be part of the same multicast group. The following sections will discuss how the two main components of Avahi, the browser and publisher, are used, and how the servers used, for information exchange.

Publisher

The Avahi publisher registers services on the local network using the Avahi daemon. The main parameters required for registering a service included a service name, service type, domain and address, which is the address of the avahi-multi-cast group on the machine. Optional information about the service can be sent as a number of TXT record strings. So to demonstrate, we will use, as an example, a TV service of type ‘houseServices’ coming online registering with the Avahi network. The information it needs to provide is its service name (e.g. Display), service type (e.g. _LHhouseServices._tcp), domain (local) and its address, which is the address of the multi-cast group it is joining. The additional information it can provide is the port its server is listening to, and the location of its RDF file. This information can displayed as follows:

Service Type: _LHhouseServices._tcp
Service Name: Display
Domain Name: local
Interface: eth1 IPv4
Address: herbert.local/192.168.1.102:9999
TXT RDF Location = /.../RDFfile/Display.rdf
TXT Server Port = 2189

Browser

The Avahi browser shows a real-time browse list for network services running on the local network using the Avahi daemon. The browser tool available on the Linux machine is the avahi-discover tool which is a graphical browser tool.
Figure 6.2: Avahi Browser

Figure 6.2 is an example of the avahi-discover tool. As can be seen, the services are grouped in the Avahi browser by their service type. And information about the TV device discussed in the previous section is displayed.

It also prints a message out to the command line when a service joins or leaves the network. The message for joining a network is:

```
Found service ‘Display’ of type ‘_LHouseServices._tcp’
in domain ‘local’ on 3.0.
```

and the message for leaving a network is:

```
Service ‘Display’ of type ‘_LHouseServices._tcp’
in domain ‘local’ on 3.0 disappeared.
```

The browser was adapted to remove its graphical interface. It uses dictionary and list structures to pass network information between the different elements of the environment.

Other Network Tools

The browser and publisher simply provide and publish information about the environment. Other tools are needed to pass information between the different parts of the
network. These tools consist of a simple socket server and client that are created to transfer RDF information and streams between different elements in this environment. Each device in this environment has:

- A Client that can send requests to other elements.
- Servers that act on the requests invoked by the clients.

The requests that a client sends can either be ‘Services’ which requests an RDF file from its connected server, or ‘Streams’, which requests a sending stream to the client.

So, for example, if a client were to send a request ‘Services’ to the server, then the server would take the following action:

```
Listening to port 2189
Accepted Connection from: 127.0.0.1
Receiving data on Server side...
Received request ‘Services’
Sending data of size 61 from /../../RDFfile/Display.rdf
Data sent!!
```

6.2.3 Devices

As previously stated, since we do not have a real environment to work with, we will work with a simulated environment created by connecting various laptops to a network. We also have substitutes for real life devices. So, for example, in the house environment web browsers will be used to display information and simulate a game console, mPlayer and realplayer will be used to simulate DVD-Players and TVs respectively.

The devices sit on a server, each server has the following mandatory files:

- A service publisher that calls the Avahi publisher to register it on the network.
- An RDF file that has the information about all its services.
- A server that where listens for requests.
- A client that sends requests to other services.

The RDF file can have the following information:

- Name.
- Services.
- Location of Devices.

Index files can also be included. They behave like index files in html pages which store information about the other files. The index file helps provide more contextual information about the environment. For example in the house environment it can include information about the number of rooms. If the index file is missing, the system will try to pick up the room names from the RDF files and if it does not find the information there it will just assume that it is one room.

### 6.2.4 User Interface Design

The interface provides the users with a visual representation of the environment, as well as a method of manipulating and querying the environment. It connects information from the network to the parser and decision making tools discussed later in this chapter. A semiotic engineering approach is used here in the the design of the system. DeSouza[22] refers to an article by Philip Armour[4] where he states that software should be seen as a medium not a product. This notion is true in this case; the software here is a medium through which the system can communicate information about itself to the user and conversely the user can communicate intentions and commands to the system. Of the three groups that Peirce classed signs under (discussed in 2.4), the system to user communication can be seen as mainly iconic, as the elements in the system will be displayed as images on the screen and will simulate any action performed on the actual elements. On the other hand the user to system communication can be seen as mainly symbolic as the user will interact using the buttons on the screen and the parsers (discussed in 6.2.6) to convey their commands.
to the system. Also within Semiotic Engineering, design-user communication is important and users are involved early on in the design process. The way this idea was incorporated in the Interface Design can be seen in 6.2.4.

The interface is made of the following parts:

**Environment Menu**

This part of the GUI is similar to the wireless network browser tool on a computer. The Environment Menu is dependent on the avahi-based network tools discussed in the previous section. As mentioned in 6.2.2 the avahi browser was adapted to connect to the environment menu. The browser then continuously searches the environment for services that join or leave the network and sends the information to the environment menu. At the same time, it sends the RDF information from the services to the network parser discussed in chapter 5 to get information about the valid network connections available and stores them in an RDF file. If it can not find any environments then it will display a ‘No environments to display’ message in the menu.

In figure 6.3 the menu has discovered four environments: house, hotel, office and shop. These are the friendly names of the environment servicetypes. The user can then click on an item from the environment menu to get a visualisation of the

![Environment Menu](image)

Figure 6.3: Environment Menu
Environment which will be displayed in the environment display discussed in the next section.

Environment Display

Once a user selects an environment from the Environment Menu, it will be displayed in the Environment Display. Figure 6.4 describes the various parts of the Environment Display.

Figure 6.4: Breakdown of the Environment Display

The parts of the Display as shown in figure 6.4 are:

1. Drawing Area: This is the area where information from the environment will be displayed; it will be discussed further in this section.

2. User Tools: These are tools a user can use through a stylus to impose actions on an element in the environment; it will also be discussed in this section.

3. Teaching Button: This calls the teaching area tool, which will be discussed in 6.2.4.

4. Microphone Button: This calls a voice recorder which will send a wave file to the speech processing tool discussed in 6.2.5.
5. Natural Language Parser Entry and Button: When the user types in a command in the entry box and clicks on the return button, the command will be passed to the Parser which is discussed in 6.2.6.

6. Status Bar: This gives feedback on actions in the environment.

**Drawing Area:**
In the drawing area, a 2D visual representation of the environment will be shown. The display can be seen as an iconic representation of the surrounding environment. The displays get this information by retrieving all the information from the environment selected; for example the HouseServices from the Environment Menu in figure 6.3 is a multicast group on the avahi daemon for all services that have the house services servicetype. The system connects to that multicast group and retrieves the IP address of all the devices that fall under that servicetype. The system then asks each device for their services (in RDF format) and stores them under a local folder. The first RDF file the system will look for is the index file; this includes information about the environment, whether it is a house or office or shopping center and, where required, the number of rooms. If an index file does not exist it either tries to get the information from the RDF files of the devices, or just assumes there is one room/area and displays all the devices or shops there. Also for each device it finds the path to the icon that represents it, or if no image is found then the system will display as a noPhoto icon. Figures 6.5 and 6.6 are examples of how the Environment Display would show a house environment with an index file and a shop environment without an index file.

As can be seen, the drawing area changes but the other tools are the same.

**User Buttons:**
The system should be user-centric and allow users to choose how to interact with the system. Because we will be designing for a handheld device, we will have to take into account that the user will be interacting through a stylus, therefore some mouse actions (such as right click or the middle click) can not be used. So the system needs to compensate and provide the user with more methods of interaction. At the same time, we should avoid cluttering the interface with too much information and allow

---

3The icons used in this environment were taken from http://www.iconarchive.com/.

85
Figure 6.5: Display of a House Environment

Figure 6.6: Display of a Shopping Environment
the user to decide when the information is required. A simple solution to this issue is the provision of the Info and Action buttons to the right hand of the drawing area. With the info button, the user can get more information about a certain device. The user simply has to click on the info button and then click on the icon he wants more information about. The information is simply taken from the RDF files of the icon. There are are two formats of information that can be displayed, simple and detailed, as seen in figures 6.7 and 6.8. The basic information can be edited by the user while the other information cannot. The editing rights of the elements are set by the RDF files for these icons.

The user can also save information after editing it, and this in turn will save the information in the RDF files of the icons selected.

Another tool is the action button, which gives the user more choices of what to do with an icon than simply clicking and dragging. The combobox lists a set of actions that the user can do. These actions are created in the teaching area, which will be discussed below. Once an action is selected, then the Action Button’s label changes to the action selected in the combobox. The user then clicks on the action box and then on the icon he wishes to perform the action on.
Feedback and Error Messages

A useful component in all GUIs is appropriate feedback. Users need to know if a command went through and what actions to take if it did not.

For example, in the house environment, if the user tells the system that he wants to watch a TV show, then that means that the appropriate devices (such as the TV, Tuner, and Speakers) are to be connected. This information should also be displayed on the GUI. To do that we simply highlight the relevant icons and draw lines between them to refer to the connections. Also, a message is displayed in the statusbar notifying the user of a connection as can be seen in Figure 6.9. If however, a connection was not able to be made an error message will appear telling the user that it was not possible to make the connection and give some information on how the user can remedy this problem. The feedback here depends on accurate parsing and decision making from the system. The decision making tool, in some cases, provides helpful error messages so the user can see where he went wrong.

In the shopping environment, if a user asks a question such as ‘Who sells Milk?’ then the appropriate shops are highlighted and listed in a table for the user to investigate as can be seen in Figure 6.10. Again, if no answers are found then a message box is displayed telling the user that the system has not found a match.
Figure 6.9: Feedback on a Connection in the House Environment

Figure 6.10: Feedback on a Query in the Shopping Environment
Teaching area

The teaching area tool is called from the Teaching button in the Environment Display. This tool allows users to customise the way they interact with the system and allows them to add different semantics to words and gestures. Here is where users can use other symbolic or iconic signs to convey their intention. Figure 6.11 is the teaching area. To explain it clearly, this component will be broken down into the following sections:

- New Actions
- Previous Actions
- Modifying Semantics

New Actions

Figure 6.12 is the section where the user can add new actions to the system. It is made up of textbox at the top of the screen followed by a combo box where the user can add actions, underneath that is what we call the ‘Action Box’ area. The ‘Action Box’ has an action list where the user can add new actions, an icon list (which contains the icon names from the environment) and an area list (that contains the
names of the areas and rooms in the environment). Underneath that is a drawing area with two rectangles representing an icon and an area. Starting from the top, in the textbox a user can either type a shorthand command that can be attributed to one action (for example ‘movie’ to watch a movie) or use a sentence to group together a set of previous actions, for example, ‘Movie-mode’ which will dim the lights, turn on the DVD player and TV and, if the user asks for it, turn on the popcorn maker. In the ‘Action Box’ a user can click or double click on one of the rectangles in the drawing area, and the appropriate action will be displayed in the textbox at the top. To make it more interesting a user can add an action (eg: shake) from the action list then click on an item in drawing area, which makes the gesture: Shake icon. Also the user can specify which rooms and which icons he wants a certain gesture specific to. For example a user can select a shake action and then click on fridge icon to perform the ‘shake fridge’ action, which is intended to indicate to the system that the user in hungry.

**Previous Actions**

The previous actions box in figure 6.12 contains a list of actions the system has already performed. It is made up of a textbox at the top and listbox. This list is what
new actions or sentences can be attributed to. So for example if a user typed ‘Food’ in the new action box then selected ‘I am hungry’ from the previous actions list, then the system simply connects the word ‘Food’ to the RDF file (called Intentions.rdf) that contains the parser output from the phrase ‘I am hungry’. This way, every time a user says ‘Food’, the parser will not be called as the Intentions RDF file is already available. A user can also select one or more actions from that list to attribute a new action to. Even though this section is called the previous actions box, there is still room to add a new action that is not included in the list. For example, the sentence ‘I want to watch a movie’ is not included in the previous actions list but supposed that the user wants to create a shorthand notation for it called ‘Movie’. In this case the user types ‘Movie’ in the new actions text box and types ‘I want to watch a movie’ in the previous actions box and then clicks the Save Button. The system then parses the phrase ‘I want to watch a movie’ and if it is successful in doing that then it will attribute the output of that parse (in RDF again) to the ‘Movie’ phrase and include both of them in the previous actions list.

Modifying Semantics

Here a user can modify the semantics of a mouse gesture. This will be used in the gesture parser discussed in the semantic section of the gesture parser in 6.2.6. Figure
6.14 shows how the overlap action is edited to mean connect.

**Testing the user interface**

One of the main concepts in Semiotic Engineering is to encourage communication between users and designers [20][21]. This helps gives the designers a feel of what the user would want. This notion is similar to the co-design or participatory design approach in User Centered Design. This was useful for helping us choose the right programming language for designing our interface. The earlier version of the interface was done using pygame-extended(PyGext). It was selected because it provided a sleek sharp interface. Figures 6.15 6.16 and 6.17 are of the interface written in Pygame.
In line with Jacob Nielsen’s theory that even as little as 5 users are enough to find the main problems of an interface[53], a small selection of 9 participants was used to evaluate the system. They were between the ages of 24 and 49 split between 5 females and 4 males.

The test was done on the emulator system on the computer. The participants were given a brief description of the system, to familiarise themselves with it, and
instructions (Appendix C) for both the house and shopping environments. They were then asked to complete a questionnaire (Appendix D), regarding their experience with the system. The results were as follows:

- Application was Easy to Use: 77%
- Application was Easy to Learn: 88%
- Effectively Completed Tasks: 82%
- Clear representation of icons: 75%
- House Environment Easy to Use: 80%
- Shopping Environment Easy to Use: 80%
- Teacher Tool was Useful: 64%
- TextBox was Easy to Use: 55%
- Application Accurately Represented the User’s Intention: 77%

The main issues raised by the participants was regarding the GUI. It got slower each time a new user came, it did not show cursors for the toolbox, mouse clicks did not register with the GUI and one participant found the colours too bright. These all held the user back from freely interacting with the system and was enough reason to replace Pygext with Pyglade and Gtk when importing to handheld. Other comments included giving feedback on actions and the ability to allow the users to use shorthand for a set of previous actions. These were all noted for redesigning the GUI in Pyglade as can be seen in the teaching and feedback sections above.

6.2.5 Speech Processing

The speech processing tool is called from the microphone icon in the house display (Figure 6.5). This icon will call a voice recorder where users can pass their commands vocally. We did not implement a purpose-built speech recogniser as it is a research field on its own, and the time constraints did not allow for developing one. Instead, the software Dragon Naturally Speaking ⁴ was used. This software has a tool that transcribes wave files into text output. Because this software is too large to be installed on the handheld device, it was installed on a windows machine where it waits on a

⁴http://www.nuance.com/naturallyspeaking/
server, and the system sends wave files to it when required and it sends back text files.

### 6.2.6 Parsers

The parser is the tool that allows users to communicate with the system. It processes mainly symbolic communication as the user uses words and gestures to convey their intentions. The parsers are designed semiotically to conform with Stamper’s framework (discussed in section 2.6). The three levels of the framework that relate to the parser are the syntactics, semantics and pragmatics levels. The same parser is used for the network representation, discussed in chapter 5, and for the natural language and gesture recognition input.

**Parser Overview**

This parser will be used for different modes: natural language input and gesture input, and will have the same format. The parser has three parts:

**Syntax level**

The syntax level produces the grammatical structure of an input in tree form. It is specific to the form of input. At this level, the individual elements of the input are tokenized, and then tagged. In natural language the elements are words and the tags are parts of speech. The elements are processed and some correction is performed on faulty tags. Once the individual tags are processed, they are then put into a chunk parser. Chunk parsing as mentioned in 4.3.1 is a form of shallow parsing. This will be demonstrated for natural language input as well as gesture input later in this section.

**Semantic level**

The semantic level takes in the output produced by the syntax parser, to create a semantic tree. In short a semantic parser attributes meaning to the elements produced by the syntax parser.

A dictionary is available based on the chosen environment and chosen input method. A semantic parse usually has the following keywords:
PROCESS → A produced action.
AGENT/CARRIER → The process producer.
GOAL → The goal of the process.
TIME → The time of the process.
ATTRIBUTE → A attribute of the process.

For example

(2)
(Semantic Tree:
(Semantics:
...
(AGENT: ...)
(PROCESS: ...)
(ATRIBUTE: ...)
(GOAL: ...)
,LOCATION: ...)
)

Each element in this list has one or more of the following properties:

(3)
LABEL → The name of the element.
PROPERTY → The property of the element.
RESTRICTOR → The constraints of an certain element.
RESTRICTOR PROPERTY → The property of the constraints.

Example 4 is the semantic output tree for the phrase: ‘Who sells cheap grampian?’.

(4)
(Semantic Tree:
(Semantics:

(AGENT: (LABEL: ‘who’) (PROPERTY: ‘query’))
(PROCESS: (LABEL: ‘sell’) (PROPERTY: ‘present’))

97
Pragmatic level

The pragmatic level is where the semantic output is put into context. This will be split into two parts:

- The pragmatic parser: This will be discussed here, in the parser section, where deictics are replaced and keywords are reassigned.

- The decision making element: This will be discussed in 6.2.7 where the output from the parsers along with contextual information are passed through an expert system.

This output from the pragmatic parser is the same for all modes of input and different syntax and semantics models will be processed the same way. Therefore different modes of representing the same goal will give the same result.

The pragmatics level takes in the semantic tree, reassigns keywords where required (e.g. more than one GOAL → CO-GOAL), sets default values for missing items and checks for the deictics, such as location and time values. Usually the TIME Value will be written in a way that the system will understand, for example the TIME value ‘now’ is seen as 20077232123702041 in the system, but for readability purposes, we will use words such as ‘now’, ‘in 3 seconds’, ‘later in the afternoon’, when displaying output from the parser. If it faces vague values it will ask the user for clarification. The pragmatic tree is produced by putting the process as the Tree node and the other attributes as the leaves. This is then converted into RDF format and printed out:

(5)

------------------------------------------
Natural Language

Natural language is one form of input into the system. This parser takes in a sentence and processes it across the different levels, syntax, semantics and pragmatics. To demonstrate how this parser works, a simple walk-through will be given of the sentence: ‘Connect the tv and the tuner.’ asked in a house environment.

The natural language parser first takes an input:

(6)
Connect the tv and the tuner.

Syntax Level

As discussed above the syntax parser tokenizes each word and then tags it.

(7)
Tagged Input Tokens are:

\[
[('connect', 'vb'), ('the', 'at'), ('tv', None),
 ('and', 'cc'), ('the', 'at'), ('tuner', None),
 ('.', '.')]\]

The tags are above are based on the Brown corpus and stored in a storage form called a pickle file; more information about tags in the Brown corpus can be found in the appendix B. However, in some cases the tags may be faulty.
The pickle file generated was based on unigram tagging of the Brown corpus. An attempt was made to create a bigram corpus but it generated a large file that did not correct all of the errors. A larger file not only meant using more storage space (remembering that it will be put on a handheld) but it also required more processing time. Since the pay off was not that generous it was decided to add a simple correction tool described in the next section. There are two types of corrections that the correction tool does. The first type is to correct mis-tagged tokens. Examples of mis-tagged tokens are tokens that do not have a part of speech. By default all words that do not have tags are nouns and only change if they meet other conditions. Other examples are gerunds/verbs for example: The word ‘working’ can be seen as a present verb in the sentence ‘I am working on a program’ but is a gerund in the sentence ‘The inner workings of the program’. Also, some words can be nouns in one sentence and verbs in another, e.g.: ‘What is the cost of this item’ and ‘How much does this item cost’. The second form of corrections done by the tool is clarifying the meaning of certain words. This is mainly used to make the semantic parser’s job much easier. Some examples are changing verbs to catenative verbs for those that can be seen as intention verbs such as ‘want’, ‘like’, ‘intend’ or specifying if something is a place or a location. This is done by connecting it to the wordnet tool and finding hypernyms. Again, more information about the other tags can be found in the appendix B. So if we put our tagged sentence into the correction tool we will get the following result.

(8)

Corrected tokens are

\[
[(\text{'connect', 'vb'}, \text{'the', 'at'}, \text{'tv', 'nn'}),
(\text{'and', 'cc'}), (\text{'the', 'at'}, \text{'tuner', 'nn'}),
(\text{'.', '.'})]
\]

As it is a simple sentence, it hardly needs any corrections; the only thing changed in this sentence is the tags for ‘tv’ and ‘tuner’.

After the tags have been corrected the sentence is then put through the chunk parser. The chunk parser is fed with the following instructions:

- Chunk sequences of catenative verbs into Catenative Phrases (CP).
• Chunk sequences of determiners, adjectives and nouns into noun phrases (NP).
• Chunk the first NP into a beginning noun phrases (B-NP).
• Chunk sequences of adverbs followed by post-determiners(e.g: ‘much’, ‘many’) into Determinative phrases (DetP).
• Chunk one of more adverbs into adverb phrases (ADV).
• Chunk one or more verbs into verb phrases (VP).
• Chunk sequences of the ‘in’ and ‘to’ prepositions followed by NP’s into prepo-
sition phrases (PP).
• Chunk sequences of tightly bonded prepositional phrases (TPP).
• Chunk one or more adjectives into adjective phrases (ADJ).

This first step in parsing will give the following result for the sentence above:

(9)
(VP: (‘connect’, ‘vb’))
(NP: (‘the’, ‘at’) (‘tv’, ‘nn’))
(‘and’, ‘cc’)
(NP: (‘the’, ‘at’) (‘tuner’, ‘nn’))
(‘.’, ‘.’)

This initial parse gives us information about the types of phrases available in the sentence. Another parse will be performed to give information about the elements in the sentence such as Subject, Objects, Predicates,Adverbials and so on. So for the next parse this set of rules are put into the system:

• Chunk sequences of beginning nounphrases (B-NP) followed by 0 or more prepo-
sitional phrases PP’s into subjects (Subj).
• Chunk sequences of catenative phrases (CP) into a predicate (Pred1).
• Chunk sequences of verbphrases (VP) into predicates (Pred2).
• Chunk sequences of non-object adjective and nounphrases that follow an intensive verb into complements (Comp).

• Chunk sequences of nounphrases (NP) into objects (Obj).

• Chunk sequences of prepositional phrases (PP) or Adverbs (ADV) or prepositions followed by Determinative phrases (DetP) into Averbials (AdvL).

• Chunk all the elements into Sentences (Sentence).

Running this set of rules over the previously parsed sentence will give us the following:

(10)

(Sentence:
  (Pred2: (VP: ('connect', 'vb')))
  (Obj: (NP: ('the', 'at') ('tv', 'nn')))
  ('and', 'cc')
  (Obj: (NP: ('the', 'at') ('tuner', 'nn')))
  (',', ',')
)

The next step here is to find a description of this parse tree; we will call this description a clause. We are interested in the following types of clauses:

• Active: A sentence where the subject(s) performs the action(s) expressed in the predicate (eg: ‘I connect the TV and the Tuner’).

• Passive: A sentence where the subject(s) receives the action(s) expressed in the predicate (eg: ‘The TV and Tuner is connected by me’).

• Query: An interrogative sentence (eg: ‘Who connected the TV and the Tuner’).

• Intensive: A sentence that has an intensive verb (eg: ‘I am hungry’).

• Catenative: A sentence where a lexical main verb is followed by another main verb (eg: ‘I want to connect the TV and the Tuner’).
• Transitive: A sentence where non-catenative main verbs take an object (eg: ‘I want this book’).

• Subjectless: A sentence that does not have a subject (eg: ‘Connect the TV and the Tuner’).

In the current example the clause is an active subjectless clause. This clause description is added to the syntax tree giving us this final output:

(11)
(Syntax Tree:
  (Sentence:
    (Clause: ‘active’ ‘Subjectless’)
    (Pred2: (VP: (‘connect’, ‘vb’)))
    (Obj: (NP: (‘the’, ‘at’) (‘tv’, ‘nn’))
      (‘and’, ‘cc’)
      (Obj: (NP: (‘the’, ‘at’) (‘tuner’, ‘nn’))
        (‘.’, ‘.’))))

Semantic Level

Now that we have our syntactical tree we then move to our semantic parser. This parser takes the syntactic input and uses its elements to create a semantic parse tree. In a regular active sentence there is a one-to-one mapping between the syntactic and semantic terms for the most part.

(12)
Pred1  →  PROCESS
Pred2  →  PROCESS
Subj   →  AGENT
Obj    →  GOAL
AdvL   →  GOAL
Comp   →  ATTRIBUTE

If the sentence is intensive then it is the same format but the Subj becomes a Carrier rather than an Agent.
If the sentence is passive then the only change is the reversal of the Subject and Object.

\[(14)\]

\[
\begin{align*}
\text{Obj} & \rightarrow \text{AGENT/CARRIER} \\
\text{Subj} & \rightarrow \text{GOAL/ATTRIBUTE}
\end{align*}
\]

So using this rule, two similar meaning sentences that are different syntactically will have the same semantics. For example, the sentences ‘The TV was watched by me’ and ‘I watched TV’ are different syntactically as the position of the Subject changes. Because the clause description states that it is a passive sentence, the semantic parser knows that the roles of the two nounphrases will have to be reversed as both sentences have the same meaning.

If however the sentence is subjectless, as the example is, then the mapping of the syntax to semantic nodes will be as follows:

\[(15)\]

\[
\begin{align*}
\text{Pred1} & \rightarrow \text{PROCESS} \\
\text{Pred2} & \rightarrow \text{PROCESS} \\
\text{Subj} & \rightarrow \text{SOURCE} \\
\text{Obj} & \rightarrow \text{GOAL} \\
\text{AdvL} & \rightarrow \text{DESTINATION}
\end{align*}
\]

The next step is to get all the words in their singular base forms. The equivalent to a word and tag in the semantic level is Label and Property. The semantic value of the tags are stored in a dictionary, and it is the meaning of the tag that is used as a property. Verbs are converted to their base forms by using wordnet to get the base form of each word. The tense is analysed as a property. Personal pronouns are all converted to ‘me’ to give a uniform structure and all nouns are put into singular form (with their grammatical number stored in the property value). Extra tags like nn-place and nn-time are converted into their rightful semantic tags. So for example the syntactic leaf is converted to its semantic counterpart as follows:
The clause description is carried over from the syntax parser. The output of the parser looks like this:

\[(16)\]
\[(\text{Pred2: (VP: ('connect', 'vb')))}\]
\[(\text{Obj: (NP: ('the', 'at') ('tv', 'nn')))}\]
\[('and', 'cc')\]
\[(\text{Obj: (NP: ('the', 'at') ('tuner', 'nn')))}\]
\[\rightarrow\]
\[(\text{PROCESS: (LABEL: 'connect') (PROPERTY: 'present')))}\]
\[(\text{GOAL: (LABEL: 'tv') (PROPERTY: 'singular')))}\]
\[(\text{GOAL: (LABEL: 'tuner') (PROPERTY: 'singular')))}\]

The semantic output tree has two GOALS, this will be interpreted as CO-GOALS by the pragmatics parser.

**Pragmatics Level**

The pragmatics parser takes in the semantic tree. It makes the process into the tree node and uses the other nodes as descriptions of that node. It also checks for deictics (like the time, location and agent values). If no values are given it sets the time to the current time by default, and location to current location. If no agent is given it sets the agent to the system by default. If the agent is ‘me’ then it sets the agent to ‘user’. The pragmatic tree will look like this:

\[(17)\]
\[(\text{Semantic Tree:)}\]
\[(\text{Semantics:)}\]
\[\text{(Clause: ‘active’ ‘Subjectless’)}\]
\[(\text{PROCESS: (LABEL: ‘connect’) (PROPERTY: ‘present’))}\]
\[(\text{GOAL: (LABEL: ‘tv’) (PROPERTY: ‘singular’))}\]
\[(\text{GOAL: (LABEL: ‘tuner’) (PROPERTY: ‘singular’))}\]
(Pragmatics Tree:
  (connect:
    (AGENT: ‘system’)
    (CO-GOAL: ‘tv’)
    (CO-GOAL: ‘tuner’)
    (LOCATION: ‘any’)
    (TIME: ‘now’)
  )
)

The next step is to store this information in RDF format. It does that by using the RDF module that converts trees to RDF files. The RDF is printed out as follows:

(19)
Printing RDF:
----------------------------------------
| Resource | Property | Value |
----------------------------------------
| Connect  | AGENT    | system |
| Connect  | CO-GOAL  | TV     |
| Connect  | CO-GOAL  | Tuner  |
| Connect  | TIME     | now    |
| Connect  | LOCATION | any    |

Graphical Input Parser

Another form of the input is gestures performed by the user on the user interface. It monitors the motion of the mouse and passes it to a parser. Associated tags are stored (based on user preference) and those tags are used. Alternatively the user can add different meanings to their gestures by using the teaching area tool 6.2.4 of the interface. So for example, a user could drag a Tuner icon on top of a TV icon with the
intention of connecting them. If overlapping icons mean connect in the environment then the interface will send this action as input for the parser:

(20)

OVERLAP TV Tuner

Syntax Level

The first step is to tokenize the input and give it the relevant tags.

(21)

Gesture tokens are [('OVERLAP', 'overlap'), ('TV', 'icon'), ('Tuner', 'icon')]

Overlap is classed as a motion while the TV and Tuner are considered icons. Because the input is fairly standard and provided by the user there is no need for any corrections.

The next step is to pass the input to the chunk parser. The chunk parser is given the following rules:

- Doubleclicks, Dragging and overlapping motions and right Clicks are chunked as Motions.

- Icons are chunked as Targets.

- Sequences of Motions and Targets are chunked as gestures.

These rules are fed into the parser and produce the following tree:

(22)

(Syntactical Parsing

(Gesture:

(Motion: ('OVERLAP', 'overlap'))
(Target: ('TV', 'icon'))
(Target: ('Tuner', 'icon'))
)
)
Semantic Level

In the semantic level again meaning is added to the grammar structures. Here there is a one-to-one mapping between the syntactic and semantic elements:

(23)

\[
\begin{align*}
\text{Motion} & \rightarrow \text{PROCESS} \\
\text{Target} & \rightarrow \text{GOAL}
\end{align*}
\]

A dictionary is provided with the meaning for the different elements.

- Right Click means display more information about a device.
- Double Click means turn on.
- Overlap means connect.

The rules give the following output.

(24)

(Semantic Tree:

(\text{Action:}

 (\text{PROCESS: (LABEL: ‘Connect’) )}

 (\text{GOAL: (LABEL: ‘Tuner’) )}

 (\text{GOAL: (LABEL: ‘TV’) )}

 )

 )

Pragmatics Level

This uses the exact same pragmatic parser as the natural language parser as the output provided by the semantic level is similar to the output provided by a natural language parser. So here the pragmatic tree is produced using the process from the semantic tree.

(25)

(Pragmatics Tree:
(Connect:
    (AGENT: ‘system’)
    (CO-GOAL: ‘Tuner’)
    (CO-GOAL: ‘TV’)
    (LOCATION: ‘any’)
    (TIME: ‘now’)
)
)

And the output is then converted to RDF format.

(26)

Printing RDF:

<table>
<thead>
<tr>
<th>Resource</th>
<th>Property</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>Connect</td>
<td>AGENT</td>
<td>system</td>
</tr>
<tr>
<td>Connect</td>
<td>CO-GOAL</td>
<td>TV</td>
</tr>
<tr>
<td>Connect</td>
<td>CO-GOAL</td>
<td>Tuner</td>
</tr>
<tr>
<td>Connect</td>
<td>TIME</td>
<td>now</td>
</tr>
<tr>
<td>Connect</td>
<td>LOCATION</td>
<td>any</td>
</tr>
</tbody>
</table>

As can be seen from these examples, different forms of input are processed the same way. Gesture and natural language input can be different on syntactical and even semantic levels, but they are the same on the pragmatics level.

### 6.2.7 Decision Making

As stated before the decision making is written in CLIPS. Therefore the decision making will start with available data and use inference rules from a rule base and pattern matching to extract more data until an optimal goal is reached. The first step involves processing all the RDF information available. This includes:
• Information about the network representation (discussed in chapter 5).

• Information about the user’s goals from the parsers.

• Information from the devices in the user’s environment.

This information is in RPV triples in RDF. It is fairly easy to convert them into CLIPS format. CLIPS will simply take the name of the RDF file followed by the RPV values to create a CLIPS fact. So for example information from the network parser in chapter 5 is stored in an RDF file Network.rdf, and this will be passed to CLIPS as follows:

(27)
(Network Connection Tuner TV)
(Network Connection Tuner Speakers)
(Network Connection Tuner Transcoder)

Information from the intention files produced by the parser is also passed across:

(28)
(Intentions Connect AGENT system)
(Intentions Connect CO-GOAL TV )
(Intentions Connect CO-GOAL Tuner )
(Intentions Connect TIME now )
(Intentions Connect LOCATION any )

All other RDF files (such as device information and information the system picks up) are stored under Context.

(29)
(Context Tuner location lounge)
(Context Tuner for channel)
(Context BBC isA channel)
(Context TV isA interface)

...
All this information is passed into the CLIPS system as facts about the environment. These will be matches against the rule base of the system.

The system starts with the following simple rules:

1. If a device is used for a GOAL then it is a beginDevice.

2. If a device interfaces with the user then it is a terminalDevice.

This is generally the base rule for most cases. The only exception is direct commands such as our example. In most cases the beginDevice finds connections until it reaches a terminalDevice. To do that, the beginDevice is set as matchDevice then:

1. If matchDevice has a network connection with another device set second device to matchDevice and append to List.

2. If matchDevice is terminalDevice then End.

It is possible to have many different routes for achieving a particular service. The system here will simply select the shortest route/list.

For a direct command example like ‘Connect the TV to the Tuner’, the system has to check network connections to see if there is a valid connection between the two devices. If there is then it proceeds to produce an Action RDF with instructions of what the system will do. So for our example the actions required by the system are as follows:

(30)
Turn on TV.
Turn on Tuner.
Send a stream from the Tuner to the TV.

6.2.8 Porting to Handheld device

As mentioned earlier, the hand held device for this prototype was a Nokia 770. This runs a debian linux machine. It has a development platform called MAEMO. Moving the system to a handheld device was the final step in the development. However,
There were some issues with porting the software, mainly that the libraries required were not available for the MAEMO. Because there was not enough time to using ssh to log into the emulator machine. SSH stands for Secure SHell; it is a method of remote logging into another machine and running programs from there. The version of ssh used for the MAEMO was called openSSH\(^5\). The aim was to simulate how a machine could run once the required libraries were available on the MAEMO. A keyboard GUI was also created so that the users could use it to type commands to the system (as the original MAEMO soft keyboard would not appear when logging on from another machine). Since both machines were on the same network, there were not any delays in running the GUI and the actions were real time as if it were running on the handheld itself. As for simulating devices, a tool called Netcat\(^6\) was used as a server to accept commands that run scripts on a computer. This way users would interact with the handheld, which in turn will send their commands to the computer (which includes the simulated devices) to run the movie players or websites that the user would require.

### 6.3 Final Remarks

This chapter discussed the design of the prototype system using the technologies that were discussed separately in the previous chapters. HCI principles were used with semiotic engineering to design the interface. Avahi was used to simulate ad-hoc networks. Principles from smart home designs were mimicked to create an intelligent environment. The NICHE project discussed in 3.1.5 was a good illustration on how natural language could influence smart environments, the Obje project (discussed in 3.2.2) and the Microsoft easy living project (discussed in 3.1.2) had design principles, such as mobile code and abstraction across devices, that influenced the design of the prototype system. By tapping into these various research areas, we aimed to design a system in accordance with Stamper’s six level framework. The prototype adapted to the six-level framework as follows:

\(^5\)http://www.openssh.com  
\(^6\)http://netcat.sourceforge.net
• Physical: This level depended on the hardware that was used, such as laptops and handheld devices, as well as the tool of interacting with them such as the keyboards, microphones, mice and touch screen styluses.

• Empirics: This level was concerned with the interaction with the physical level such as: the touching of the keypads, tapping of the styluses and speaking to the microphone (depending on the mode of input).

• Syntax: This level dealt with the rules that governed its structures, in natural language it was concerned with the grammatical structure of the phrases and in the gesture mode it dealt with the ‘grammar’ of the mouse clicks.

• Semantics: This level involved taking the structured elements and extracting ‘meaning’ from them.

• Pragmatics: This level is where all the information about the user intention was put into context. First, it took information from the syntax and semantic levels, clarified any vagueness in them and converted them into RDF. The RDF output of the user intention information along with information about services and valid connections were passed to the decision making system to trigger the necessary actions.

• Social World: This level conveyed the effect of the communication. To do this a GUI was designed that would communicate information from system to user as well as from user to system. The GUI used iconic (such as highlighting the icon representation of services) and symbolic (such as feedback) signs to notify the user about the state of the services in the networked environment. The GUI also took gesture and natural language input from the user and converted them to machine understandable commands for the system.

Ad-hoc networks are useful to both the pragmatics and the social world levels in the semiotic framework. In the pragmatics level, it helped clarify communication. The same command could mean different things based on the environment the user was in. For example, ‘I am hungry’ would search for recipes from the fridge content in the
house environment while it would search for restaurant in the shopping environment. As for the social world, it is a useful tool in letting the interface know the state of the services in the environment (and the interface in turn could represent this information iconic-ally for the user) or by gathering information on why a certain set of actions failed.
Chapter 7

Testing and User Evaluation

A series of tests were run to check if the application was usable. These tests can be divided between system tests, mostly blackbox, and user evaluation performed by participants.

7.1 System Testing

Individual modules were tested separately, but because the application was made up of many different parts, blackbox testing was seen as the best way to test the system. In this section we will only list the main tests that were performed: the interface and the parser tests.

7.1.1 Interface

The goal was to create an interface that adapted to the surrounding environment. The first test was to check whether the menu would notice if servicetypes go online and offline in real time. To do this a number of different services were setup to go online and offline in order to test whether the menu would keep up to date with the changes in the environment. This was successful. The next step was to test how the prototype would draw different types of environments on demand. Different environments with different numbers of rooms were created. The largest number of visible rooms possible on the device was found to be 10 as can be seen in figure 7.1. Any number more than
10 would make the rooms really small and not leave room to place the devices. Since 10 rooms were more than enough for a demo application, it was not deemed necessary to find a way of allowing for more rooms as this instance. Another test was to create a different new environment that the system has not seen before and then checking with how the interface would cope. So an office environment was created that was made up of 2 rooms, a meeting room and an open plan office. Devices like laptops, coffee machines, projectors and screens were included. The system had no problems in displaying it as can be seen in figure 7.2.

### 7.1.2 Parsers

One of the major tests for the system was regarding the parsers. In the previous chapter, some simple examples were given. Here is a sample of four slightly more complicated sentences that have similar meaning but are worded differently:

- **Who sells grampian and for how much?**
- **How much is grampian?**
- **Which grampian is the cheapest grampian?**
- **What is the cheapest grampian?**

![Figure 7.1: The Preferred Maximum Number of Rooms](image-url)
Figure 7.2: The New Office Environment

(Note: Grampian is the name of a fictitious brand of whisky). This gave the following output:

```
Sentence is: Who sells grampian and for how much
```

Tagged Input Tokens are:

```
[('who', 'wps'), ('sells', 'vbz'), ('grampian', None),
 ('and', 'cc'), ('for', 'in'), ('how', 'wrb'),
 ('much', 'ap'), ('.', '.')]`

Corrected tokens are:

```
[('who', 'wps'), ('sells', 'vbz'), ('grampian', 'nn'),
 ('and', 'cc'), ('for', 'in'), ('how', 'wrb'),
 ('much', 'ap'), ('.', '.')]`

Syntactical Parsing:

```
(Syntax Tree:

117
```
Sentence is: How much is grampian

Tagged Input Tokens are:
[['how', 'wrb'], ['much', 'ap'], ['is', 'bez'], ['grampian', None], ['.', '.']]

Corrected tokens are:
[['how', 'wrb'], ['much', 'ap'], ['is', 'bez'], ['grampian', 'nn'], ['.', '.']]

Syntactical Parsing:

(Syntax Tree:
(Sentence:
 (Clause: 'active' 'intensive' 'query')
 (Comp: (DetP: ('how', 'wrb') ('much', 'ap')))  
 (Pred2: (VP: ('is', 'bez'))) 
 (Subj: (B-NP: ('grampian', 'nn')))  
 ('.', '.')))

(Semantic Tree:
(Semantics: 
 (Clause: 'active' 'intensive' 'query')
 (ATTRIBUTE: (LABEL: 'much') (PROPERTY: 'query'))
 (PROCESS: (LABEL: 'be') (PROPERTY: 'present'))
 (CARRIER: (LABEL: 'grampian') (PROPERTY: 'singular'))))
<table>
<thead>
<tr>
<th>Resource</th>
<th>Property</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>be</td>
<td>CARRIER</td>
<td>grampian</td>
</tr>
<tr>
<td>be</td>
<td>ATTRIBUTE</td>
<td>price?</td>
</tr>
<tr>
<td>be</td>
<td>TIME</td>
<td>now</td>
</tr>
<tr>
<td>be</td>
<td>LOCATION</td>
<td>any</td>
</tr>
</tbody>
</table>

Sentence is: Which grampian is the cheapest grampian

Tagged Input Tokens are:

```python
[('which', 'wdt'), ('grampian', None), ('is', 'bez'), ('the', 'at'), ('cheapest', 'jjt'), ('grampian', None), ('.', '.')]
```

Corrected tokens are:

```python
[('which', 'wdt'), ('grampian', 'nn'), ('is', 'bez'), ('the', 'at'), ('cheapest', 'jjt'), ('grampian', 'nn'), ('.', '.')]```
(Pred2: (VP: ('is', 'bez')))  
(Comp:  
  (NP: ('the', 'at') ('cheapest', 'jkt') ('grampian', 'nn'))  
  ('.', '.'))

(Semantic Tree:  
  (Semantics:  
    (Clause: 'active' 'intensive' 'query')  
    (CARRIER:  
      (LABEL: 'grampian')  
      (PROPERTY: 'singular')  
      (LABEL: 'which')  
      (PROPERTY: 'query'))  
    (PROCESS: (LABEL: 'be') (PROPERTY: 'present'))  
    (ATTRIBUTE:  
      (LABEL: 'grampian')  
      (PROPERTY: 'singular')  
      (RESTRICTOR: 'cheap')  
      (RESTRICTOR PROPERTY: 'superlative')))

Printing RDF:  
----------------------------------------  
Resource | Property | Value  
----------------------------------------  
be | CARRIER | grampian  
be | ATTRIBUTE | price?  
be | EXCHANGE | minimum_money  
be | TIME | now  
be | LOCATION | any
Sentence is: What is the cheapest grampian

Tagged Input Tokens are: [('what', 'wdt'), ('is', 'bez'), ('the', 'at'),
('cheapest', 'jjt'), ('grampian', None), ('.', '.')]  
Corrected tokens are [(‘what’, ‘wdt’), (‘is’, ‘bez’), (‘the’, ‘at’),
(‘cheapest’, ‘jjt’), (‘grampian’, ‘nn’), (‘.’, ‘.’)]

Syntactical Parsing:

(Syntax Tree:
(Sentence:
(Clause: ‘active’ ‘intensive’ ‘query’)
(Subj: (B-NP: (‘what’, ‘wdt’)))
(Pred2: (VP: (‘is’, ‘bez’)))
(Comp:
  (NP: (‘the’, ‘at’) (‘cheapest’, ‘jjt’) (‘grampian’, ‘nn’))
  (‘.’, ‘.’)))

(Semantic Tree:
(Semantics:
(Clause: ‘active’ ‘intensive’ ‘query’)
(CARRIER: (LABEL: ‘what’) (PROPERTY: ‘query’))
(PROCESS: (LABEL: ‘be’) (PROPERTY: ‘present’))
(ATTRIBUTE:
  (LABEL: ‘grampian’)
  (PROPERTY: ‘singular’)
  (RESTRICTOR: ‘cheap’)
  (RESTRICTOR PROPERTY: ‘superlative’)))

122
As can be seen the last two sentences - ‘Which grampian is the cheapest grampian?’ and ‘What is the cheapest grampian?’ - produce the same pragmatic output even though they are syntactically different. More parser output can be found in Appendix G. Obviously the parser does not work for all sentences, for example it can not deal with most one word commands or phrases that do not include verbs. In those cases it simply returns a null value and does not proceed any further.

7.2 User Evaluation

7.2.1 Scenario

As mentioned in the previous chapter, the services were simulated using servers on the computer. A netcat server (briefly discussed in section 6.2.8) was setup to accept commands to run scripts that call movie players, videos and websites when required by the user.

In both cases the scenario was the same: the participants were given a brief description of the system, to familiarise themselves with it, and instructions (Appendix E) for both the house and shopping environments.
The speech aspect of the system was not tested as the Dragon Speak software needed training for each new participant which would have taken a lot of time.

House Environment

Once the program runs, the ‘Environment Menu’ appears, which lists all the ad-hoc environments it can find via Avahi (discussed earlier in sections 4.1.1 and 6.2.2). Once the menu loaded, the participants were first asked to select the house environment. This environment was made up of two rooms with device icons distributed between them.

The participants were expected to input commands to:

1. Watch a movie: The devices highlighted should be DVD-Player, TV and Speakers and a movie will play.

2. State that you are hungry: Fridge should be highlighted and a browser will open with recipe search.

3. Play a Game: The devices highlighted should be Game Console, TV and Speakers and a browser with game emulators will appear.

The participants were then asked to use the teaching tool and add new shorthand commands that would trigger the above actions. They were told to

1. Create a shorthand name for previous actions (e.g: ‘Movie-mode’ for ‘Play a movie’).

2. Create an action in the combo box and use it on an icon (e.g: Perform a shake action on the Fridge for the ‘I am hungry’ action).

Shopping Environment

The participants were then asked to return to the ‘Environment Menu’ and select the shopping environment. This environment consisted of a large room, representing the town, with different shop icons in it. In the shopping environment, the users were asked to do ask the system the following: 

124
1. Find out who sells what.

2. Find out what the store ‘jam king’ sells.

3. Find out who sells the cheapest jam.

And again the participants were required to use the teaching tool and add new shorthand commands that would trigger the above actions.

1. Create a shorthand name for one of the previous actions (e.g: Typing ‘Coffee’ for ‘What is the price of coffee’).

2. Create a shorthand name for a set of previous actions (e.g: Typing ‘Price’ to get the price of a list of items).

Also the participants who were part of the interface test in 6.2.4 were asked if the new system was better and where they saw the improvements.

7.3 Results and Final Remarks

The participants were asked to fill a questionnaire (see Appendix F) after completing the test. The questionnaire guidelines were taken from ACM’s Computer System Usability Questionnaire ¹ and the questionnaire itself was made up of nine close format questions, two open-format questions regarding usability and an empty space for how the system has improved (this question was for users who used the interface described in 6.2.4) and an empty space for further comments. The evaluation was carried out on 15 participants. The age range was 23 to 40 and was split between nine female participants and six male participants. Six of the total participants already tested the system in section 6.2.4. Their results were not that different from ‘new’ users of the system (i.e. users who have used the system for the first time), therefore the results were grouped together and are as follows:

¹http://oldwww.acm.org/perlman/question.cgi
Application was Easy to Use: 78%
Application was Easy to Learn: 88%
Effectively Completed Tasks: 82%
Clear representation of icons: 78%
House Environment Easy to Use: 89%
Shopping Environment Easy to Use: 84%
Teacher Tool was Useful: 84%
TextBox was Easy to Use: 78%
Application Accurately Represented the User’s Intention: 85%

As can be seen, the new interface received more positive feedback than the interface tested in section 6.2.4. Major improvements were seen in the use of the teaching tool (which was very basic in the original interface) and the text box (which was difficult to access in the original interface).

As mentioned earlier, six of the participants had taken the test before. They provided the following responses when asked about the improvements they saw in the new system:

1. System was now faster.
2. Navigation and feedback was clearer.
3. Remote control feel made the new test more interesting.
4. Teacher tool improved.
5. Better interface design.

Obviously there is always room for further improvement, and participants gave the following opinions on how the prototype could be further improved:

1. Faster response times from the server.
2. Expand the vocabulary further.
3. Find a better substitute for the soft keyboard
The test only covered the general aspects of the system. The results gave ideas about how to improve the prototype further. For example, vocabulary can be expanded by including more synonyms via wordnet, we can also add more information to the context RDF file about the environment the user is in. As for substituting the keyboard, one option is to use the microphone or shorthand terms and actions created by the teaching tool.
Chapter 8

Conclusion

Intelligent environments should enable users to express their intentions and not restrict them to a pre-ordained set of choices. By applying a semiotically based approach to the design of the interaction between the user and a network, we have demonstrated how it is possible to overcome such a limitation. Although we have presented a relatively simple example here for ease of exposition, the approach is in principle extendible to other more complicated examples. The semiotic framework readily accommodates situations where different inputs are assigned the same meaning and it also supports a variety of possible input methods. We therefore consider that our work represents a promising approach to the design and implementation of home as well as m-commerce environments.

8.1 What has been Achieved

The purpose of this work was to demonstrate a practical application of semiotics. This was achieved by using the theories behind semiotics to develop a system that operated in intelligent ad-hoc networked environments. This novel application of semiotics provided another approach to the area of HCI in pervasive computing. Communication with the interface (through the use of natural language or gestures) and communication within the system were parsed the same way. The semiotic framework also accommodated different modes of communication such as verbal or written
commands. These qualities of the semiotic framework are beneficial in promoting user centered design in the area of intelligent environments. This gives users more freedom in choosing how they want to interact with their environment. Semiotics was also used as an inspiration behind the representation and parsing of elements in the ad-hoc network. This research is also useful for the area of semiotics as it encourages more practical application of its theories to current computer science issues. We also were able to exploit semantic web tools such as RDF. It was beneficial in the following areas:

1. Store and represent information about the network: This also helped in providing a simple 2D visualisation for the user.

2. Store information about the devices: This helped infer information about valid connections between devices which again were stored in RDF.

3. Store output from the natural language parsing of users intentions.

4. Store output from actions: This would tell the system what actions to do and when.

RDF was the ideal data storage tool for this environment as it was portable, easy to format and provided a very good representation of information about the environment and elements within in.

8.2 Potential Extensions of the Research

This research demonstrated the novel idea of applying semiotics to networks. M-commerce and smart home environments were used as a testbed for this concept, as they are both modern problem-areas in the field of human-computer interaction and are open to interdisciplinary research. However, because this work was mainly a proof of concept, there are many areas that are still open to investigation. They include:

1. Machine Learning: This was an area this research hoped to cover, but due to the time limitation as well as the depth of the machine learning field, we had
to compromise by just making the system ‘memorise’ new intentions by users and use the teaching tool as a manual way of ‘understanding’ shorthands that a user would user. Having a strong machine learning system would also be useful to the next two points.

2. Deeper Gesture Recognition: The research covered simple gesture recognition, such as mouse movements and mouse clicks. Interesting examples would be through the use of web-cams or video cameras in the environment and machine learning techniques, such as Hidden Markov models, to associate a user’s gestures with the actions they want from the system. Also, since handheld devices are used, it would be interesting to make more use of the stylus and allow input methods such as drawing (e.g.: draw a light switch and click on it to turn the lights on). These can be seen as another mode of input and fit well within the semiotic framework.

3. Security Issues: This research did not cover any security issues, as it is a field on its own. But the semiotic application here could include recognising user patterns to know whether an intruder is using the system and perform actions like freezing the system or denying permission if that happens.

4. Adaptability and Accessibility: Modern systems are designed to be more inclusive and accessible to people with disabilities as well as those without. Current applications can be extended to be accessible, such as websites and games [5]. The prototype device was designed with the ability to be extended by other developers to cater for a wider audience.

This work addressed the problem of enabling users to communicate with, and receive helpful responses from, their IT platforms. We have attempted to show that by applying a semiotic analysis to the problem, we have been able to find an avenue towards a promising solution, while acknowledging that more research needs to be carried out before an operational implementation can be made available. Crucial to this approach was the realisation that the interpretation of users’ intentions is a pragmatic process which is to be understood in relation to the other levels within the semiotic
framework. Choosing appropriate methods of representing and reasoning about intentions within the networked system has also been a vital semiotic consideration in our research. Thus semiotics can be seen as playing a central role in the engineering of network-based computer systems, present and future.
Publications


References


140
Appendix A

Roger Bacon’s Division of signs

1. NATURAL SIGNS

1.1 signifying by inference, concomitance, consequence

1.1.1 signifying necessarily

1.1.1.1 signifying something present
(large extremeties -> strength)

1.1.1.2 signifying something past
(lactation -> birth of a child)

1.1.1.3 signifying something future
(dawn -> imminent sunrise)

1.1.2 signifying with probability

1.1.2.1 signifying sth. present
(to be a mother -> love)

1.1.2.2 signifying sth. past
(wet ground -> previous rain)

1.1.2.3 signifying sth. future
(red sky in the morning -> rain)

1.2 signifying by configuration and likeness
(images, pictures, species of colour)

1.3 signifying by causality
(tracks -> animal)
2. SIGNS GIVEN AND DIRECTED BY A SOUL

2.1 signifying instinctively without deliberation
   (sigh $\rightarrow$ pain; laughter $\rightarrow$ joy)

2.2 signifying with deliberation
   (words)

2.3 interjections
Appendix B

Brown Corpus

* . sentence closer (. ; ? *)
* ( left paren
* ) right paren
* * not, n’t
* -- dash
* , comma
* : colon
* ABL pre-qualifier (quite, rather)
* ABN pre-quantifier (half, all)
* ABX pre-quantifier (both)
* AP post-determiner (many, several, next )
* AT article (a, the, no)
* BE be
* BED were
* BEDZ was
* BEG being
* BEM am
* BEN been
* BER are, art
* BEZ is
* CC coordinating conjunction (and, or)
* CD cardinal numberal (one, two, 2, etc.)
* CS subordinating conjunction (if, although)
* DO do
* DOD did
* DOZ does
* DT singular determiner/quantifier (this, that)
* DTI singular or plural determiner/quantifier (some, any)
* DTS plural determiner (these, those)
* DTX determiner/double conjunction (either)
* EX existential there
* FW foreign word (hyphenated before regular tag)
* HV have
* HVD had (past tense)
* HVG having
* HVN had (past participle)
* IN preposition
* JJ adjective
* JJR comparative adjective
* JJS semantically superlative adjective (chief, top)
* JJT morphologically superlative adjective (biggest)
* MD modal auxiliary (can, should, will)
* NC cited word (hyphenated after regular tag)
* NN singular or mass noun
* NN$ possessive singular noun
* NNS plural noun
* NNS$ possessive plural noun
* NP proper noun or part of name phrase
* NP$ possessive proper noun
* NPS$ possessive plural proper noun
* NR adverbial noun (home, today, west)
* OD ordinal numeral (first, 2nd)
* PN nominal pronoun (everybody, nothing)
* PN$ possessive nominal pronoun
Appendix C

First Set of Instructions

C.1 About the Application

This program is a demonstration of how to manipulate the surrounding environment using a simple interface. The environment here is simulated using an Avahi daemon. The two environments dealt with here are:

- House Environment: Where the services are provided by domestic appliances (such as TVs, speakers, fridges, coffee machines etc).

- Shopping Environment: Where the services are provided by stores.

This the program is split into 3 parts:

- An environment menu: This menu displays the names of the different environments it finds.

- An environment display: This will display the environment selected from the menu.

- A teacher screen: This is a simple tool that allows the user to attribute different names to actions (will be explained in example).

In this example you will deal with both the house and shopping environments.
C.2 House Environment

When the application starts, a menu of the available environments will appear. Using the up and down keys, select the ‘House Services’ option and press enter. A screen will appear depicting a house with four rooms. The rooms include a list of icons. These represent devices you will use in this environment. Right clicking on the devices gives you more information about them. You can edit the fields by clicking on them and typing. Under the rooms is a white text box where you can enter commands. To enter a command you need to click the box, then type the command and press enter. In this example you need to type in commands that allow you to do the actions below, you are successful if the items required are highlighted by a pink square and action is committed:

1. Watch a movie: (The devices highlighted should be DVD-Player, TV and Speakers and a movie will play).

2. State that you are hungry (Fridge should be highlighted and a browser will open with recipe search).

3. Play a Game (The devices highlighted should be Game Console, TV and Speakers and browser with game emulators will appear).

4. Watch BBC in 3 seconds (The devices highlighted should be Tuner, TV and Speakers and a BBC show will play in 3 seconds).

Above the return key there is an icon with the image of a hat on it. This is the teacher icon, click on it. Here you will be able to add shorthand commands to existing actions.

On the left hand side you have a textbox, here you can add new actions. On the right hand side you have a list of previously written commands. Write the short hand in the box in the left and click on an item in the box on the right. Save by clicking the save button. Now go back to the house environment and try out the new command.
C.3 Shop Environment

Now, go back to the main menu by clicking the "Back to Main Menu" button at the bottom of the screen. We are back in the main menu. Now click the "Shop Services". Here you have one large area, that represents a town center and shop icons. Again more information about the icons can be attained by right clicking on them. Try the following commands in the textbox:

1. Find out who sells milk.

2. Find out who sells coffee.

3. Find out who sells the cheapest coffee.

Again you can use teacher icon to add new actions, same as for the house environment.
Appendix D

First Questionnaire

Gender:

Age:

1. The application is easy to use:
   

2. The application is easy to learn:
   

3. I effectively completed my tasks using this application:
   

4. It was clear what the icons represented:
   

5. The House Environment was easy to use:
   

6. The Shopping Environment was easy to use:

7. The Teacher tool was useful:


8. It was easy to use the textbox:


9. The application accurately represented my intentions:


10. What was the most useful feature:

11. What was the least useful feature:

Please include further comments here:
Appendix E

Second Set of Instructions

E.1 About the Application

This program is a demonstration of how to manipulate the surrounding environment using a simple interface. The environment here is simulated using an Avahi daemon. The two environments dealt with here are:

- House Environment: Where the services are provided by domestic appliances (such as TVs, speakers, fridges, DVD-players etc).
- Shopping Environment: Where the services are provided by stores.

The program is split into 3 parts:

- An environment menu: This menu displays the names of the different environments it finds.
- An environment display: This will display the environment selected from the menu.
- A teacher screen: This is a simple tool that allows the user to attribute different names to actions (will be explained in example).

In this example you will deal with both the house and shopping environments.
E.2 Using the Virtual Keyboard

To use the keyboard, you need to click on a text area that you want to edit. A virtual keyboard will appear. Use the keys to type in the information required and click on the save button.

E.3 House Environment

When the application starts, a menu of the available environments will appear. Select the ‘HouseServices’ option. A screen will appear depicting a house with 2 rooms. The rooms include a list of icons. These represent devices you will use in this environment. To get more information about a certain icon, first click the ‘Info’ Button. Once the mouse pointer changes into a question mark, you can click on the icon you want. The information box comes in two forms simple and detailed. ‘Simple’ information is editable while detailed is not. Under the drawing area is a text box where you can enter commands. In this example you need to type in commands that allow you to do the actions below and then click on the return key next to the textbox, you are successful if the items required are highlighted by a red square and action is committed:

1. Watch a movie: (The devices highlighted should be DVD-Player, TV and Speakers and a movie will play).
2. State that you are hungry (Fridge should be highlighted and a browser will open with recipe search).
3. Play a Game (The devices highlighted should be Game Console, TV and Speakers and browser with game emulators will appear).

Next to the return key there is an icon with the image of a hat on it. This is the teacher icon, click on it. Here you will be able to add shorthand commands to existing actions.
The left hand side of the teacher screen is where you can add new actions. On the right hand side you have a list of previously written commands. To add new commands, you can either:

1. Use the textbox at the top left of the page: Select the text radiobutton on the left hand side of the screen and type in a command, then select the ‘previous action’ radio button and select an already performed action on the right hand screen and press Save. (For example type movie time and select ”I want to watch a movie” from the right hand side)

2. Use the action box: Select the Action radiobutton on the left hand side of the screen. Here you can perform a gesture action and attribute it to an action on the right hand side of the screen. For example in the action area select the ‘shake’ action from the ‘action list’ and then select ‘Fridge’ from the ‘all icons list’. You will notice that the label of the icon in the drawing area has changed to ‘fridge’. Now click once on the fridge icon and you will notice that the text area above says ‘Shake Fridge’. Now, select the sentence ‘I am hungry’ from the right hand side and click save.

You can go back to the house environment and try out the new commands. For the fridge example click on the shake button then click on the fridge.

E.4 Shop Environment

Now, close the ‘House environment’ and go back to the main menu. Click on the ‘ShopServices’

Here you have one large area (this represents a town center) and shop icons. Again more information about the icons can be attained by using the info button. Try writing commands that will help you find out:


2. What does jamking sell.
3. Who sells the cheapest jam.

Again you can use teacher icon to add new actions, same as for the house environment. So for example you can attribute the new phrase ‘I am hungry’ to the new phrase ‘Who sells buffet’.
Appendix F

Second Questionnaire

Gender:

Age:

1. The application is easy to use:


2. The application is easy to learn:


3. I effectively completed my tasks using this application:


4. It was clear what the icons represented:


5. The House Environment was easy to use:


6. The Shopping Environment was easy to use:


156
7. The Teacher tool was useful:


8. It was easy to use the textbox:


9. The application accurately represented my intentions:


10. What was the most useful feature:

11. What was the least useful feature:

If you have done the test before, how has it improved?

Please include further comments here:
Appendix G

Parser Output

Sentence is: I watch TV in the garden

Tagged Input Tokens are: [('i', 'nn'), ('watch', 'vb'), ('TV', 'nn'), ('in', 'in'), ('the', 'at'), ('garden', 'nn'), ('.', '.')]  

Syntactical Parsing:

(Syntax Tree:
(Sentence:
(Clause: ‘active’)
(Subj: (B-NP: (‘i’, ‘ppss’)))
(Pred2: (VP: (‘watch’, ‘vb’)))
(Obj: (NP: (‘TV’, ‘nn’)))
(AdvL:
(‘.’, ‘.’)))

158
(Semantic Tree:
(Semantics:
(Clauses: ‘active’)

(AGENT: (LABEL: ‘me’) (PROPERTY: ‘singular’))
(PROCESS: (LABEL: ‘watch’) (PROPERTY: ‘present’))
(GOAL: (LABEL: ‘TV’) (PROPERTY: ‘singular’))
(LOCATION: (LABEL: ‘garden’) (PROPERTY: ‘location’))))

Printing RDF:

----------------------------------------
Resource | Property | Value
----------------------------------------
watch    | AGENT   | User
watch    | GOAL    | TV
watch    | TIME    | 2007232123602041
watch    | LOCATION | garden
----------------------------------------

Sentence is: connect the tv and the tuner

---

Tagged Input Tokens are: [(‘connect’, ‘vb’), (‘the’, ‘at’), (‘tv’, None),
(‘and’, ‘cc’), (‘the’, ‘at’), (‘tuner’, None), (‘.’, ‘.’)]
Corrected tokens are [(‘connect’, ‘vb’), (‘the’, ‘at’), (‘tv’, ‘nn’),
(‘and’, ‘cc’), (‘the’, ‘at’), (‘tuner’, ‘nn’), (‘.’, ‘.’)]
Syntactical Parsing:
Sentence is: connect the tv to the tuner

Tagged Input Tokens are: [('connect', 'vb'), ('the', 'at'), ('tv', None), ('to', 'to'), ('the', 'at'), ('tuner', None), ('.', '.')]  
Corrected tokens are [(‘connect’, ‘vb’), (‘the’, ‘at’), (‘tv’, ‘nn’), (‘to’, ‘to’), (‘the’, ‘at’), (‘tuner’, ‘nn’), (‘.’, ‘.’)]

Syntactical Parsing:

(Syntax Tree:

(Sentence:

(Clause: ‘active’ ‘Subjectless’)

(Pred2: (VP: (‘connect’, ‘vb’)))

(Obj: (NP: (‘the’, ‘at’) (‘tv’, ‘nn’)))

(AdvL: (PP: (‘to’, ‘to’) (NP: (‘the’, ‘at’) (‘tuner’, ‘nn’))))

(‘.’, ‘.’))

(Semantic Tree:

(Semantics:

(Clause: ‘active’ ‘Subjectless’)

(PROCESS: (LABEL: ‘connect’) (PROPERTY: ‘present’))

(GOAL: (LABEL: ‘tv’) (PROPERTY: ‘singular’))

(DESTINATION: (LABEL: ‘tuner’) (PROPERTY: ‘singular’))

Printing RDF:

<table>
<thead>
<tr>
<th>Resource</th>
<th>Property</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>connect</td>
<td>AGENT</td>
<td>system</td>
</tr>
</tbody>
</table>

161
Sentence is: Who sells the very cheapest grampian
(Semantic Tree:
  (Semantics:
    (Clause: ‘active’ ’query’)
    (AGENT: (LABEL: ‘who’) (PROPERTY: ’query’))
    (PROCESS: (LABEL: ‘sell’) (PROPERTY: ’present’))
    (GOAL:
      (LABEL: ‘grampian’)
      (PROPERTY: ‘singular’)
      (RESTRICTOR: ‘cheap’)
      (RESTRICTOR PROPERTY: ‘superlative’))))

Printing RDF:
----------------------------------------
<table>
<thead>
<tr>
<th>Resource</th>
<th>Property</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>sell</td>
<td>AGENT</td>
<td>shop?</td>
</tr>
<tr>
<td>sell</td>
<td>GOAL</td>
<td>grampian</td>
</tr>
<tr>
<td>sell</td>
<td>EXCHANGE</td>
<td>minimum_money</td>
</tr>
<tr>
<td>sell</td>
<td>TIME</td>
<td>20077232123702041</td>
</tr>
<tr>
<td>sell</td>
<td>LOCATION</td>
<td>any</td>
</tr>
</tbody>
</table>

----------------------------------------

Sentence is: I want the tv to be connected to the tuner
Tagged Input Tokens are: [('i', 'nn'), ('want', 'vb'), ('the', 'at'), ('tv', None),
('to', 'to'), ('be', 'be'), ('connected', 'vbn'), ('to', 'to'),
('the', 'at'), ('tuner', None), (',', ',')]

Corrected tokens are [(‘i’, ‘ppss’), (‘want’, ‘catVerb’), (‘the’, ‘at’), (‘tv’, ‘nn’),
(‘to’, ‘to’), (‘be’, ‘be’), (‘connected’, ‘vbn’), (‘to’, ‘to’),
(‘the’, ‘at’), (‘tuner’, ‘nn’), (‘.’, ‘.’)]

Syntactical Parsing:

(Syntax Tree:
(Clause: ‘active’ ‘transitive’)
(Subj: (BNP: (‘i’, ‘ppss’)))
(Pred1: (CP: (‘want’, ‘catVerb’)))
(Obj: (NP: (‘the’, ‘at’) (‘tv’, ‘nn’)))
(Pred2: (VP: (‘to’, ‘to’) (‘be’, ‘be’) (‘connected’, ‘vbn’)))
(AdvL: (PP: (‘to’, ‘to’) (NP: (‘the’, ‘at’) (‘tuner’, ‘nn’)))
(‘.’, ‘.’))

(Semantic Tree:
(Semantics: ‘active’ ‘transitive’)
(AGENT: (LABEL: ‘me’) (PROPERTY: ‘singular’))
(PROCESS: (LABEL: ‘want’) (PROPERTY: ‘present’))
(GOAL:
(GOAL: (LABEL: ‘tv’) (PROPERTY: ‘singular’))
(PROCESS: (LABEL: ‘be’) (PROPERTY: ‘future’))
(GOAL: (LABEL: ‘tuner’) (PROPERTY: ‘singular’)))

Printing RDF:
----------------------------------------
<table>
<thead>
<tr>
<th>Resource</th>
<th>Property</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>be</td>
<td>AGENT</td>
<td>system</td>
</tr>
<tr>
<td>be</td>
<td>CO-GOAL</td>
<td>tuner</td>
</tr>
<tr>
<td>be</td>
<td>CO-GOAL</td>
<td>tv</td>
</tr>
<tr>
<td>be</td>
<td>TIME</td>
<td>20077232123702041</td>
</tr>
<tr>
<td>be</td>
<td>LOCATION</td>
<td>any</td>
</tr>
</tbody>
</table>

Sentence is: Who sells grampian and for how much.

Tagged Input Tokens are: [(‘who’, ‘wps’), (‘sells’, ‘vbz’), (‘grampian’, None),
(‘and’, ‘cc’), (‘for’, ‘in’), (‘how’, ‘wrb’),
(‘much’, ‘ap’), (‘.’, ‘.’)]

Corrected tokens are [(‘who’, ‘wps’), (‘sells’, ‘vbz’), (‘grampian’, ‘nn’),
(‘and’, ‘cc’), (‘for’, ‘in’), (‘how’, ‘wrb’),
(‘much’, ‘ap’), (‘.’, ‘.’)]

Syntactical Parsing:

(Syntax Tree:
  (Sentence: (Clause: ‘active’ ‘query’)
    (Subj: (B-NP: (‘who’, ‘wps’)))
    (Pred2: (VP: (‘sells’, ‘vbz’)))
    (Obj: (NP: (‘grampian’, ‘nn’)))
    (‘and’, ‘cc’))
('for', 'in')
(Comp: (DetP: ('how', 'wrb') ('much', 'ap')))
(‘.’, ‘.’))

(Semantic Tree:
(Semantics:
(Clause: ‘active’ ’query’)
(AGENT: (LABEL: ‘who’) (PROPERTY: ’query’))
(PROCESS: (LABEL: ‘sell’) (PROPERTY: ’present’))
(GOAL: (LABEL: ‘grampian’) (PROPERTY: ’singular’))
(ATTRIBUTE: (LABEL: ‘much’) (PROPERTY: ’query’))))

Printing RDF:
----------------------------------------
<table>
<thead>
<tr>
<th>Resource</th>
<th>Property</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>sell</td>
<td>AGENT</td>
<td>shop?</td>
</tr>
<tr>
<td>sell</td>
<td>ATTRIBUTE</td>
<td>price?</td>
</tr>
<tr>
<td>sell</td>
<td>GOAL</td>
<td>grampian</td>
</tr>
<tr>
<td>sell</td>
<td>TIME</td>
<td>20077232123702041</td>
</tr>
<tr>
<td>sell</td>
<td>LOCATION</td>
<td>any</td>
</tr>
</tbody>
</table>

Sentence is: How much is grampian.
Tagged Input Tokens are: [('how', 'wrb'), ('much', 'ap'), ('is', 'bez'),
('grampian', None), ('.', '.')]
Corrected tokens are [(\'how\', \'wrb\'), (\'much\', \'ap\'), (\'is\', \'bez\'),
(\'grampian\', \'nn\'), (\'.\', \'.\')]

Syntactical Parsing:
---------------------
(Syntax Tree:
Sentence:
(Clause: \'active\' \'intensive\' \'query\'
(Comp: (DetP: (\'how\', \'wrb\') (\'much\', \'ap\'))
(Pred2: (VP: (\'is\', \'bez\'))
(Subj: (B-NP: (\'grampian\', \'nn\'))
(\'.\', \'.\')))<<<--------------------------------------------------------------------------------------------------

(Semantic Tree:
(Semantics:
(Clause: \'active\' \'intensive\' \'query\'
(ATTRIBUTE: (LABEL: \'much\') (PROPERTY: \'query\'))
(PROCESS: (LABEL: \'be\') (PROPERTY: \'present\'))
(CARRIER: (LABEL: \'grampian\') (PROPERTY: \'singular\')))))<--------------------------------------------------------------------------------------------------

Printing RDF:
----------------------------------------
<table>
<thead>
<tr>
<th>Resource</th>
<th>Property</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>be</td>
<td>CARRIER</td>
<td>grampian</td>
</tr>
</tbody>
</table>
Which grampian is the cheapest grampian
tagged input tokens are: [('which', 'wdt'), ('grampian', None), ('is', 'bez'),
('the', 'at'), ('cheapest', 'jtt'),
('grampian', None), ('.', '.')]corrected tokens are [('which', 'wdt'), ('grampian', 'nn'), ('is', 'bez'),
('the', 'at'), ('cheapest', 'jtt'),
('grampian', 'nn'), ('.', '.')]syntactical parsing:

(Syntax Tree:
(Sentence:
 (Clause: 'active' 'intensive' 'query')
 (Subj: (B-NP: ('which', 'wdt') ('grampian', 'nn')))
 (Pred2: (VP: ('is', 'bez')))
 (Comp:
  (NP: ('the', 'at') ('cheapest', 'jtt') ('grampian', 'nn'))
  ('.', '.')))

(Semantic Tree:
(Semantics:
 (Clause: 'active' 'intensive' 'query')
 (CARRIER: 168))
Sentence is: What is the cheapest grampian

Tagged Input Tokens are: [('what', 'wdt'), ('is', 'bez'), ('the', 'at'), ('cheapest', 'jjt'), ('grampian', None), ('.', '.')]
Corrected tokens are [('what', 'wdt'), ('is', 'bez'), ('the', 'at'), ('cheapest', 'jjt'), ('grampian', 'nn'), ('.', '.')]  

Syntactical Parsing:  
-----------------------------

(Syntax Tree:  
  (Sentence:  
    (Clause: 'active' 'intensive' 'query')  
    (Subj: (B-NP: ('what', 'wdt')))  
    (Pred2: (VP: ('is', 'bez')))  
    (Comp:  
      (NP: ('the', 'at') ('cheapest', 'jjt') ('grampian', 'nn'))  
      ('.', '.')))

(Semantic Tree:  
  (Semantics:  
    (Clause: 'active' 'intensive' 'query')  
    (CARRIER: (LABEL: 'what') (PROPERTY: 'query'))  
    (PROCESS: (LABEL: 'be') (PROPERTY: 'present'))  
    (ATTRIBUTE:  
      (LABEL: 'grampian')  
      (PROPERTY: 'singular')  
      (RESTRICCTOR: 'cheap')  
      (RESTRICTOR PROPERTY: 'superlative')))  

Printing RDF:  
----------------------------------------

<table>
<thead>
<tr>
<th>Resource</th>
<th>Property</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>be</td>
<td>CARRIER</td>
<td>grampian</td>
</tr>
</tbody>
</table>
Sentence is: what is the price of grampian in the shop

Tagged Input Tokens are: [('what', 'wdt'), ('is', 'bez'), ('the', 'at'), ('price', 'nn'), ('of', 'in'), ('grampian', None), ('in', 'in'), ('the', 'at'), ('shop', 'nn'), ('.', '.')]  
Corrected tokens are [('what', 'wdt'), ('is', 'bez'), ('the', 'at'), ('price', 'nn'), ('of', 'of'), ('grampian', 'nn'), ('in', 'in'), ('the', 'at'), ('shop', 'nn-place'), ('.', '.')]  

Syntactical Parsing:

(Syntax Tree:
  (Sentence: ('active' 'intensive' 'query')
   (Subj: (B-NP: ('what', 'wdt')))  
   (Pred2: (VP: ('is', 'bez')))  
   (Comp:
     (NP: ('the', 'at') ('price', 'nn'))
     (TPP: ('of', 'of') (NP: ('grampian', 'nn')))  
   (AdvL:  
   )))
(PP: (‘in’, ‘in’) (NP: (‘the’, ‘at’) (‘shop’, ‘nn-place’))))
(‘.’, ‘.’))

(Semantic Tree:

(Semantics:

(Clause: ‘active’ 'intensive' 'query')

(CARRIER: (LABEL: 'what') (PROPERTY: 'query'))

(PROCESS: (LABEL: 'be') (PROPERTY: 'present'))

(ATRIBUTE:

(LABEL: 'grampian')

(PROPERTY: 'singular')

(RESTRCTOR: 'price')

(RESTRCTOR PROPERTY: 'singular'))

(LOCATION: (LABEL: 'shop') (PROPERTY: 'location'))))

Printing RDF:

----------------------------------------
Resource | Property | Value
----------------------------------------
be | CARRIER | grampian
be | ATTRIBUTE | price?
be | TIME | 20077232123802041
be | LOCATION | shop

Sentence is: What is the price of grampian everywhere
Tagged Input Tokens are: [(‘what’, ‘wdt’), (‘is’, ‘bez’), (‘the’, ‘at’),
(‘price’, ‘nn’), (‘of’, ‘in’), (‘grampian’, None),
(‘everywhere’, ‘rb’), (‘.’, ‘.’)]

Corrected tokens are [(‘what’, ‘wdt’), (‘is’, ‘bez’), (‘the’, ‘at’),
(‘everywhere’, ‘rb-place’), (‘.’, ‘.’)]

Syntactical Parsing:
---------------------

(Syntax Tree: 
(Sentence: 
(Clause: ‘active’ ‘intensive’ ‘query’)
(Subj: (B-NP: (‘what’, ‘wdt’)))
(Pred2: (VP: (‘is’, ‘bez’)))
(Comp:
  (NP: (‘the’, ‘at’) (‘price’, ‘nn’))
  (TPP: (‘of’, ‘of’) (NP: (‘grampian’, ‘nn’))))
(AdvL: (ADV: (‘everywhere’, ‘rb-place’)))
(‘.’, ‘.’))

(Semantic Tree: 
(Semantics: 
(Clause: ‘active’ ‘intensive’ ‘query’)
(CARRIER: (LABEL: ‘what’) (PROPERTY: ‘query’))
(PROCESS: (LABEL: ‘be’) (PROPERTY: ‘present’))

(ATTRIBUTE:
  (LABEL: ‘grampian’)
  (PROPERTY: ‘singular’)
  (RESTRICITOR: ‘price’)
  (RESTRCTOR PROPERTY: ‘singular’))

173
<table>
<thead>
<tr>
<th>Resource</th>
<th>Property</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>be</td>
<td>CARRIER</td>
<td>grampian</td>
</tr>
<tr>
<td>be</td>
<td>ATTRIBUTE</td>
<td>price?</td>
</tr>
<tr>
<td>be</td>
<td>TIME</td>
<td>20077232123802041</td>
</tr>
<tr>
<td>be</td>
<td>LOCATION</td>
<td>everywhere</td>
</tr>
</tbody>
</table>

Sentence is: how much does grampian cost

Tagged Input Tokens are: [('how', 'wrb'), ('much', 'ap'), ('does', 'doz'), ('grampian', None), ('cost', 'nn'), ('.', '.')]  
Corrected tokens are [('how', 'wrb'), ('much', 'ap'), ('does', 'doz'), ('grampian', 'nn'), ('cost', 'vb'), ('.', '.')]  

Syntactical Parsing:

(Syntax Tree:  
(Sentence:  
(Clause: 'active' 'intensive' 'query')  
(Comp: (DetP: ('how', 'wrb') ('much', 'ap'))  
(Pred2: (VP: ('does', 'doz')))
(Subj: (B-NP: ('grampian', 'nn')))  
(Pred2: (VP: ('cost', 'vb')))  
(‘.’, ‘.’))

(Semantic Tree:
(Semantics:
  (Clause: ‘active’ ‘intensive’ ‘query’)
  (ATTRIBUTE: (LABEL: ‘much’) (PROPERTY: ’query’))
  (CARRIER: (LABEL: ‘grampian’) (PROPERTY: ’singular’))
  (PROCESS: (LABEL: ‘cost’) (PROPERTY: ’present’))))

Printing RDF:

----------------------------------------
Resource | Property     | Value            
----------------------------------------
cost      | CARRIER      | grampian         
cost      | ATTRIBUTE    | price?           
cost      | TIME         | 20077232123802041

cost      | LOCATION     | any              

Sentence is: I went to the shopping center

Tagged Input Tokens are: [('i', 'nn'), ('went', 'vbd'), ('to', 'to'), ('the', 'at'), ('shopping', 'vbg'),
Corrected tokens are [('i', 'ppss'), ('went', 'vbd'), ('to', 'to'),
('the', 'at'), ('shopping', 'jj'),
('center', 'nn-place'), ('.', '.')]
Resource | Property | Value
----------------------------------------
go | AGENT | User
go | TIME | past
go | LOCATION | center

Sentence is: I went to the garden at 4 in the morning

Tagged Input Tokens are: [('i', 'nn'), ('went', 'vbd'), ('to', 'to'), ('the', 'at'), ('garden', 'nn'), ('at', 'in'), ('4', 'cd'), ('in', 'in'), ('the', 'at'), ('morning', 'nn'), ('.', '.')]

Corrected tokens are [('i', 'ppss'), ('went', 'vbd'), ('to', 'to'), ('the', 'at'), ('garden', 'nn-place'), ('at', 'in'), ('4', 'nn-time'), ('in', 'in'), ('the', 'at'), ('morning', 'nn-time'), ('.', '.')]

Syntactical Parsing:
---------------------

(Syntax Tree:
  (Sentence:
    (Clause: 'active')
    (Subj: (B-NP: ('i', 'ppss')))
    (Pred2: (VP: ('went', 'vbd')))
    (AdvL:
      (PP: ('to', 'to') (NP: ('the', 'at') ('garden', 'nn-place')))
      (AdvL: (PP: ('at', 'in') (NP: ('4', 'nn-time')))))

177
(AdvL:
(PP: ('in', 'in') (NP: ('the', 'at') ('morning', 'nn-time')))
('.', '.')))

(Semantic Tree:
(Semantics:
 (Clause: 'active')
 (AGENT: (LABEL: 'me') (PROPERTY: 'singular'))
 (PROCESS: (LABEL: 'go') (PROPERTY: 'past'))
 (LOCATION: (LABEL: 'garden') (PROPERTY: 'location'))
 (TIME: (LABEL: '4') (PROPERTY: 'time'))
 (TIME: (LABEL: 'morning') (PROPERTY: 'time'))))

Printing RDF:
----------------------------------------
<table>
<thead>
<tr>
<th>Resource</th>
<th>Property</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>go</td>
<td>AGENT</td>
<td>User</td>
</tr>
<tr>
<td>go</td>
<td>TIME</td>
<td>past</td>
</tr>
<tr>
<td>go</td>
<td>LOCATION</td>
<td>garden</td>
</tr>
</tbody>
</table>

Sentence is: How much was grampian

Tagged Input Tokens are: [('how', 'wrb'), ('much', 'ap'), ('was', 'bedz'), ('grampian', None), ('.', '.')]
Corrected tokens are [('how', 'wrb'), ('much', 'ap'), ('was', 'bedz'), ('grampian', 'nn'), ('.', '.')]
Sentence is: who sells grampian

Tagged Input Tokens are: [('who', 'wps'), ('sells', 'vbz'), ('grampian', None), ('.', '.')]  
Corrected tokens are [('who', 'wps'), ('sells', 'vbz'), ('grampian', 'nn'), ('.', '.')]  

Syntactical Parsing:

(Syntax Tree:

(Sentence:  
(Clause: 'active' 'query')  
(Subj: (B-NP: ('who', 'wps')))  
(Pred2: (VP: ('sells', 'vbz')))  
(Obj: (NP: ('grampian', 'nn')))  
('.', '.')))

(Semantic Tree:

(Semantics:  
(Clause: 'active' 'query')  
(AGENT: (LABEL: 'who') (PROPERTY: 'query'))  
(PROCESS: (LABEL: 'sell') (PROPERTY: 'present'))  
(GOAL: (LABEL: 'grampian') (PROPERTY: 'singular'))))

Printing RDF:

----------------------------------------

180
Resource | Property | Value
----------------------------------------
sell    | AGENT     | shop?
sell    | GOAL      | grampian
sell    | TIME      | 20077232123902041
sell    | LOCATION  | any

Sentence is: I want to watch BBC in the garden

Tagged Input Tokens are: [('i', 'nn'), ('want', 'vb'), ('to', 'to'), ('watch', 'vb'), ('BBC', None), ('in', 'in'), ('the', 'at'), ('garden', 'nn'), ('.', '.')]  
Corrected tokens are [('i', 'ppss'), ('want', 'catVerb'), ('to', 'to'), ('watch', 'vb'), ('BBC', 'nn'), ('in', 'in'), ('the', 'at'), ('garden', 'nn-place'), ('.', '.')]  
Syntactical Parsing:

(Syntax Tree: (Sentence: (Clause: 'active' 'catenative') (Subj: (B-NP: ('i', 'ppss'))) (Pred1: (CP: ('want', 'catVerb'))) (Pred2: (VP: ('to', 'to') ('watch', 'vb'))) (Obj: (NP: ('BBC', 'nn'))) (AdvL: (PP: ('in', 'in') (NP: ('the', 'at') ('garden', 'nn-place')))))
(‘.’, ‘.’))

(Semantic Tree:
(Semantics:
(Clauses: ‘active’ ‘catenative’)
(AGENT: (LABEL: ‘me’) (PROPERTY: ‘singular’))
(PROCESS: (LABEL: ‘want’) (PROPERTY: ‘present’))
(GOAL:
(AGENT: (LABEL: ‘me’) (PROPERTY: ‘singular’))
(PROCESS: (LABEL: ‘watch’) (PROPERTY: ‘present’))
(GOAL: (LABEL: ‘BBC’) (PROPERTY: ‘singular’))
,LOCATION: (LABEL: ‘garden’) (PROPERTY: ‘location’)))

Printing RDF:
----------------------------------------
<table>
<thead>
<tr>
<th>Resource</th>
<th>Property</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>watch</td>
<td>AGENT</td>
<td>User</td>
</tr>
<tr>
<td>watch</td>
<td>GOAL</td>
<td>BBC</td>
</tr>
<tr>
<td>watch</td>
<td>TIME</td>
<td>20077232123902041</td>
</tr>
<tr>
<td>watch</td>
<td>LOCATION</td>
<td>garden</td>
</tr>
</tbody>
</table>

Sentence is: I want to watch tv
Tagged Input Tokens are: 
[('i', 'nn'), ('want', 'vb'), ('to', 'to'),
('watch', 'vb'), ('tv', None), ('.', '.')] 
Corrected tokens are 
[('i', 'ppss'), ('want', 'catVerb'), ('to', 'to'),
('watch', 'vb'), ('tv', 'nn'), ('.', '.')] 

Syntactical Parsing: 

(Syntax Tree: 
(Sentence: 
(Clause: 'active' 'catenative') 
(Subj: (B-NP: ('i', 'ppss'))) 
(Pred1: (CP: ('want', 'catVerb'))) 
(Pred2: (VP: ('to', 'to') ('watch', 'vb'))) 
(Obj: (NP: ('tv', 'nn'))) 
('.', '.'))) 

(Semantic Tree: 
(Semantics: 
(Clause: 'active' 'catenative') 
(AGENT: (LABEL: 'me') (PROPERTY: 'singular')) 
(PROCESS: (LABEL: 'want') (PROPERTY: 'present')) 
(GOAL: 
(AGENT: (LABEL: 'me') (PROPERTY: 'singular')) 
(PROCESS: (LABEL: 'watch') (PROPERTY: 'present')) 
(GOAL: (LABEL: 'tv') (PROPERTY: 'singular'))))) 

Printing RDF: 

<table>
<thead>
<tr>
<th>Resource</th>
<th>Property</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>watch</td>
<td>AGENT</td>
<td>User</td>
</tr>
</tbody>
</table>

183
Sentence is: I am hungry

Tagged Input Tokens are: [['i', 'nn'], ['am', 'bem'],
('hungry', 'jj'), ('.', '.')]  
Corrected tokens are [['i', 'ppss'], ['am', 'bem'],
('hungry', 'jj'), ('.', '.')]  

Syntactical Parsing:

(Syntax Tree:
  (Sentence:
    (Clause: 'active' 'intensive')
    (Subj: (B-NP: ('i', 'ppss')))  
    (Pred2: (VP: ('am', 'bem')))  
    (Comp: (ADJ: ('hungry', 'jj')))
    ('.', '.')))

(Semantic Tree:
  (Semantics:
    (Clause: 'active' 'intensive')
    (CARRIER: (LABEL: 'me') (PROPERTY: 'singular'))
    (PROCESS: (LABEL: 'be') (PROPERTY: 'present intensive')))

184
hungry is a discomfort word
This can be translated to the user not wanting to be hungry

Printing RDF:
----------------------------------------
Resource | Property | Value
----------------------------------------
be | CARRIER | User
be | ATTRIBUTE | non-hungry
be | TIME | 200772321231002041
be | LOCATION | any

Sentence is: I watched TV

Tagged Input Tokens are: [('i', 'nn'), ('watched', 'vbd'), ('TV', 'nn'), ('.', '.')]  
Corrected tokens are [('i', 'ppss'), ('watched', 'vbd'), ('TV', 'nn'), ('.', '.')]  
Syntactical Parsing:
---------------------
(Syntax Tree:
(Sentence:
(Clause: ‘active’)  
(Subj: (B-NP: (‘i’, ‘ppss’)))  
(Pred2: (VP: (‘watched’, ‘vbd’)))  
(Obj: (NP: (‘TV’, ‘nn’)))  
(‘.’, ‘.’))

(Semantic Tree)

(Semantics:

(Clause: ‘active’)

(AGENT: (LABEL: ‘me’) (PROPERTY: ‘singular’))

(PROCESS: (LABEL: ‘watch’) (PROPERTY: ‘past’))

(GOAL: (LABEL: ‘TV’) (PROPERTY: ‘singular’)))

past

Printing RDF:

----------------------------------------
<table>
<thead>
<tr>
<th>Resource</th>
<th>Property</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>watch</td>
<td>AGENT</td>
<td>User</td>
</tr>
<tr>
<td>watch</td>
<td>GOAL</td>
<td>TV</td>
</tr>
<tr>
<td>watch</td>
<td>TIME</td>
<td>past</td>
</tr>
<tr>
<td>watch</td>
<td>LOCATION</td>
<td>any</td>
</tr>
</tbody>
</table>

Sentence is: TV was watched the me.
Tagged Input Tokens are: [(‘TV’, None), (‘was’, ‘bedz’), (‘watched’, ‘vbd’), (‘the’, ‘at’), (‘me’, ‘ppo’), (‘.’, ‘.’)]
Corrected tokens are [(‘TV’, ‘nn’), (‘was’, ‘bedz’), (‘watched’, ‘vbd’), (‘the’, ‘at’), (‘me’, ‘ppo’), (‘.’, ‘.’)]

Syntactical Parsing:
---------------------
(Syntax Tree:

(Sentence:
 (Clause: ‘passive’)
 (Subj: (B-NP: (‘TV’, ‘nn’)))
 (Pred2: (VP: (‘was’, ‘bedz’) (‘watched’, ‘vbd’)))
 (Obj: (NP: (‘the’, ‘at’) (‘me’, ‘ppo’)))
 (‘.’, ‘.’)))

(Semantic Tree:

(Semantics:
 (Clause: ‘passive’)
 (GOAL: (LABEL: ‘TV’) (PROPERTY: ‘singular’))
 (PROCESS:
  (LABEL: ‘watch’)
  (PROPERTY: ‘past’)
  (LABEL: ‘be’)
  (PROPERTY: ‘past’)
  (AGENT: (LABEL: ‘me’) (PROPERTY: ‘singular’))))

Printing RDF:
----------------------------------------
<table>
<thead>
<tr>
<th>Resource</th>
<th>Property</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>watch</td>
<td>AGENT</td>
<td>User</td>
</tr>
</tbody>
</table>
----------------------------------------
Sentence is: TV was watched by me
I am hungry
hungry is a discomfort word
This can be translated to the user not wanting to be hungry

Printing RDF:

<table>
<thead>
<tr>
<th>Resource</th>
<th>Property</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>be</td>
<td>CARRIER</td>
<td>User</td>
</tr>
<tr>
<td>be</td>
<td>ATTRIBUTE</td>
<td>non-hungry</td>
</tr>
<tr>
<td>be</td>
<td>TIME</td>
<td>200772321231102041</td>
</tr>
<tr>
<td>be</td>
<td>LOCATION</td>
<td>any</td>
</tr>
</tbody>
</table>
Sentence is: I want to sleep in 3 hours

Tagged Input Tokens are: [('i', 'nn'), ('want', 'vb'), ('to', 'to'), ('sleep', 'nn'), ('in', 'in'), ('3', 'cd'), ('hours', 'nns'), ('.', '.')]

Corrected tokens are [('i', 'ppss'), ('want', 'catVerb'), ('to', 'to'), ('sleep', 'vb'), ('in', 'in'), ('3', 'nn-time'), ('hours', 'nn-time'), ('.', '.')]

Syntactical Parsing:

(Syntax Tree:
(Sentence: (Clause: 'active' 'catenative')
(Subj: (B-NP: ('i', 'ppss'))) (Pred1: (CP: ('want', 'catVerb'))) (Pred2: (VP: ('to', 'to') ('sleep', 'vb'))) (AdvL: (PP: ('in', 'in') (NP: ('3', 'nn-time') ('hours', 'nn-time'))))
('.', '.')))

(Semantic Tree:
(Semantics:
(Clause: 'active' 'catenative')
(AGENT: (LABEL: 'me') (PROPERTY: 'singular'))
(PROCESS: (LABEL: 'want') (PROPERTY: 'present'))
(GOAL:
(AGENT: (LABEL: 'me') (PROPERTY: 'singular'))
(PROCESS: (LABEL: 'sleep') (PROPERTY: 'present'))
(TIME: 191)
(LABEL: ‘hours’) 
(PROPERTY: ‘time’) 
(LABEL: ‘3’) 
(PROPERTY: ‘time’)))

Printing RDF:

<table>
<thead>
<tr>
<th>Resource</th>
<th>Property</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>sleep</td>
<td>AGENT</td>
<td>User</td>
</tr>
<tr>
<td>sleep</td>
<td>TIME</td>
<td>20072324231102041</td>
</tr>
<tr>
<td>sleep</td>
<td>LOCATION</td>
<td>any</td>
</tr>
</tbody>
</table>

Sentence is: universities are very hungry

Tagged Input Tokens are: [('universities', 'nns'), ('are', 'ber'),
('very', 'ql'), ('hungry', 'jj'), ('.', '.')] 
Corrected tokens are: [('universities', 'nns'), ('are', 'ber'),
('very', 'ql'), ('hungry', 'jj'), ('.', '.')] 
Syntactical Parsing:

(Syntax Tree:
  (Sentence:
    (Clause: ‘active’ ‘intensive’)
    (Subj: (B-NP: (‘universities’, ‘nns’)))
    (Pred2: (VP: (‘are’, ‘ber’))))

192
(Comp: (ADJ: ('very', 'ql') ('hungry', 'jj'))
('.', '.')))  

(Semantic Tree:  
  (Semantics:  
    (Clause: 'active' 'intensive')  
    (CARRIER: (LABEL: 'university') (PROPERTY: 'plural'))  
    (PROCESS: (LABEL: 'be') (PROPERTY: 'present'))  
    (ATTRIBUTE:  
      (RESTRICTOR: 'hungry')  
      (RESTRICTOR PROPERTY: 'positive')))))  

hungry is a discomfort word  
This can be translated to the user not wanting to be hungry  

Printing RDF:  
----------------------------------------  
<table>
<thead>
<tr>
<th>Resource</th>
<th>Property</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>be</td>
<td>CARRIER</td>
<td>university</td>
</tr>
<tr>
<td>be</td>
<td>ATTRIBUTE</td>
<td>non-hungry</td>
</tr>
<tr>
<td>be</td>
<td>TIME</td>
<td>20072321231102041</td>
</tr>
<tr>
<td>be</td>
<td>LOCATION</td>
<td>any</td>
</tr>
</tbody>
</table>

----------------------------------------  

Sentence is: what is happening  
----------------------------------------
Tagged Input Tokens are: [('what', 'wdt'), ('is', 'bez'),
('happening', 'vbg'), ('.', '.')] 
Corrected tokens are [('what', 'wdt'), ('is', 'bez'),
('happening', 'vbg'), ('.', '.')] 

Syntactical Parsing: 
--------------------

(Syntax Tree: 
Sentence:  
Clause: 'active' 'intensive' 'query'  
Subj: (B-NP: ('what', 'wdt'))  
Pred2: (VP: ('is', 'bez') ('happening', 'vbg'))  
('.', '.'))

(Semantic Tree: 
Semantics: 
Clause: 'active' 'intensive' 'query'  
CARRIER: (LABEL: 'what') (PROPERTY: 'query')  
PROCESS:  
(LABEL: 'happen')  
PROPERTY: 'present')  
(LABEL: 'be')  
PROPERTY: 'present'))

Printing RDF: 
----------------------------------------
 Resource | Property | Value 
----------------------------------------
happen | CARRIER | what

194
Sentence is: he was believed

Tagged Input Tokens are: [('he', 'pps'), ('was', 'bedz'), ('believed', 'vbd'), ('.', '.')]

Corrected tokens are [('he', 'pps'), ('was', 'bedz'), ('believed', 'vbd'), ('.', '.')]
(PROPERTY: 'past'))))

Printing RDF:

----------------------------------------
Resource | Property | Value
----------------------------------------
believe   | AGENT    | system
believe   | TIME     | past
believe   | LOCATION | any

----------------------------------------