Master's Thesis in Intelligent Multimedia at Aalborg University, DK

The Hitch Hiker's Guide to the Galaxy

DON'T PANIC!

Adam Cohen Søren Bach Christiansen

sponsored by The Digital Village

AALBORG UNIVERSITY
INSTITUTE OF ELECTRONIC SYSTEMS
INTELLIGENT MULTIMEDIA 10th. SEMESTER

The Hitch Hiker's Guide to the Galaxy

Project period: 1. February 1999 –1. July 1999

Project group: 1071 IMM

Authors:

Adam Cohen

Søren Bach Christiansen

Supervisors:

Dr. Paul Mc Kevitt

Number printed: 9

Number of pages: 125

Abstract:

The *Hitch Hiker's Guide to the Galaxy* is a science fiction story written by Douglas Adams that foresees an electronic guide to Life, the Universe and Everything. Together with *The Digital Village*, Douglas Adams's company, we define the capabilities that a modern version of the *Guide* should have: it should provide information in a contextual setting; use a third party interaction metaphor via software agents; and provide personality-based feedback to encourage the user to trust the agents.
Since a complete version of the *Guide* is not possible with current technology, we prototype h2g2, a system that can provide guidance over a more limited domain. We then discuss h2g2 in the context of other similar systems from Aalborg University and other research centres such as MIT's Media Lab. and Apple Computer, Inc.

Preface

This thesis is the report of the third and final project of the Master's Programme in Intelligent Multimedia (IMM) at Aalborg University, Denmark. This Master's programme is the educational face of IntelliMedia 2000+, a research and education programme at Aalborg University. Intelligent Multimedia (IntelliMedia) is different from traditional multimedia in that the computer has some understanding of the semantics and representation of the media involved. More information on the IntelliMedia 2000+ programme can be found on the web at: `http://www.cpk.auc.dk/imm/`.

Students of the Master's programme have the choice of combining their last two semesters into one "long thesis" project or of widening their experience with two separate six month projects. This project is a six month project that is loosely based on the previous semester's project (Christiansen, Cohen, Nielsen & Ortega 1999). As such, we have not had as much time to develop our project in depth as other students conducting one year Master's Theses. However, over the two semesters we have been able to build two different systems, sponsored by two different companies (Bosch Telecom and The Digital Village, respectively). We also had the chance to present our previous semester's project as a paper and a poster at the 7SEMCON98 conference at Aalborg University.

h2g2 makes use of knowledge gained over all three semesters of the Master's Programme, from courses on Spoken Dialogue Systems to readings in IntelliMedia research from around the world. We were also fortunate to be able to visit several research centres in Boston and New York during the 1999 IMM Study Tour to the USA, including MIT's Media, Speech and AI Labs, Bell Labs, Harvard and Rutgers University. Several concepts in h2g2 were inspired by research we saw on this trip.

We would like to thank Richard Harris of The Digital Village for allowing us to use their Universal Context Model and Principles of Guidance document, and Dr. Ken Haase of the Machine Understanding Group at the MIT Media Lab for his help in compiling FramerD and for giving us his unpublished parser software.

We would also like to thank our supervisor, Dr. Paul Mc Kevitt, for his invaluable advice, and Ruth Kushner (Adam's fiancée), for her support all the way through this Master's Programme.

Adam Cohen　　Søren Bach Christiansen

Contents

Introduction

Chapter 1

In many of the more relaxed civilizations on the Outer Eastern Rim of the Galaxy, the Hitch Hiker's Guide has already supplanted the great Encyclopedia Galactica as the standard repository of all knowledge and wisdom, for though it has many omissions and contains much that is apocryphal, or at least wildly inaccurate, it scores over the older, more pedestrian work in two important respects.

First, it is slightly cheaper; and secondly it has the words "Don't Panic" inscribed in large friendly letters on its cover.

Douglas Adams, The Hitch Hiker's Guide to the Galaxy

1.1 Providing Context

The Internet could be viewed as a huge source of information, similar to the fictional *Hitch Hiker's Guide to the Galaxy* (at least in the apocryphal and wildly inaccurate senses). Unfortunately, it is missing the all important, large friendly letters on the cover.

There are many sites on the Web that offer searching facilities. You type in a phrase and the site gives you a list of other sites that have something to do with that phrase. Unfortunately, each linked page might not have the exact answer that you are looking for, resulting in a process of flipping to and from the list of links while you get the disparate parts to the information that you are looking for. This process is now so well recognised that Microsoft has built a "page holder" feature into its Web browser so that users don't have to keep reloading the page of links.

There are three problems occurring here. Firstly, there is no way to tell from the page of links which page contains the exact information you are looking for. Some search engines have tried to remedy this by including an excerpt of text from the page, but automatic summarization of arbitrary text is a notoriously hard problem and the current results are far from satisfactory. Secondly, the information you are looking for might be spread across several pages. This happens because the people writing the material are writing it from their perspective, which can vary from biased to simply ignorant. Thirdly, the pages gathered by the search engine have no semantic links between them - taken together they are simply a collection of documents with similar words but that don't necessarily relate or refer to each other.

Some sites, such as *Yahoo*, try a different tack. Instead of tying together documents based on keyword searches, they create their own index of relevant documents carefully selected by a team of editors. However, not only are these sites almost always out of date (since they rely on people to catalogue new stuff all the time) but they can also only provide a few keywords per linked page – it's still hard to decide if the desired information is beyond the link. They are also constrained to a fixed, hierarchical structure which itself becomes hard to navigate through.

Donald Norman (Norman 1988b) presents a reason for this. He proposes that conscious thought involves using short term memory, which can only store five to seven items. We can only keep more things in mind if we combine them into a structure and store them as one item. If we are presented with lots of unconnected things, they rapidly pass out of our recollection. However, if the items are organised into a structure in some way, then we can store all of them as a single item.

For example, imagine someone is giving you directions in a city you have never visited before. He gives you a list of 15 different turns you must make to get to your destination. The chances are that you will have forgotten the first turn before he even reaches the end. The list of directions has no structure to it – it's boring. Unless, that is, you have a map in front of you and can relate his information to a structure.

Roger Schank presents the same idea in "Engines for Education" and "Tell Me A Story". He argues that lists of facts and keywords are not a good way to get information across to people:

> The important issue in learning a new fact is having someplace relevant to put it in memory. New facts are only absorbed in terms of old facts that are already present in memory. If we are told about pirates and can only understand them as an instance of muggers in the street, we will have missed something important.
>
> *Roger Shank, Engines for Education* (Schank & Cleary 1994)

In other words, it's much easier for people to understand and make use of information when it's provided in a context such as a story.

1.2 Agent Technology

For the information to be put into context, however, there needs to be a fundamental shift in the idea of the user interface.

Most current day software uses what is called the *direct manipulation* metaphor in their user interfaces. This implies not only that the user can see what choices are available at any time but that the user is entirely responsible for making those choices. This idea, while perfectly suitable for handling small amounts of local information, breaks down when the user must handle the large amounts of information currently available to search engines. As Pattie Maes, head of the Software Agents Group at MIT's Media Lab, says:

> The currently dominant interaction metaphor of direct manipulation requires the user to initiate all tasks explicitly and to monitor all events. This metaphor will have to change if untrained users are to make effective use of the computers and networks of tomorrow.
>
> *Pattie Maes, Agents that Reduce Work and Information Overload* (Maes 1994)

A solution to this problem is to enable the user to work in collaboration with other agents (both human and software based). While this strategy means that the user can let the system deal with more tasks, it also implies that the user must relinquish some control of the interface. For this to work, the user must trust the agents to complete their tasks satisfactorily. If a software agent is to be accepted by users, it must therefore have some way of proving its competence and gaining their trust.

One way of doing this is to allow the agent to gradually learn from the user's actions and reactions. In this way the user can see the agent developing and is "given time to gradually build up a model of how the agent makes decisions, which is one of the prerequisites for a trust relationship (Maes 1994)".

To do this, the agent must build up its own model of how it expects the *user* to behave. There are various problems associated with this. One of them is that the model can never fully capture the user's entire decision making process. There will always be some occasions when the user wishes to do something unusual and unexpected. For example, if a restaurant guidance agent is familiar with the restaurants that *you* like, what happens when you want a restaurant that's suitable for your parents?

Another problem is to do with the trust relationship. For the user to see how the agent is developing, the agent must make its model of the user apparent in some way. Since this model is usually quite abstract, conveying it to the user in an understandable way can be difficult.

1.3 Personality-based Expression

One kind of information that people can understand extremely well is facial expression. People have so many expectations of faces that they can receive information even from the vaguest of expressions.

Naoko Tosa's *Neurobaby* (Tosa 1993) used neural networks to model an artificial baby that reacted in emotional ways to the sounds made by a user looking into its crib. The neurobaby was a fairly simple system that wasn't able to do complicated tasks, but because of the users' knowledge and expectations of a *real* baby, it was judged very convincing.

Tomoko Koda (Koda 1996) even found that people assign more intelligence to an agent if it has a human face than if it is represented by a cartoon creature. This result agrees with Roger Schank's ideas about the intelligence of storytellers:

> If the skin of the teller of the story is fleshy and humanlike, we are likely to consider the algorithm that produced the story to be an intelligent one... But if the skin is plastic and we suspect that a computer is inside we are likely to claim that the algorithms being used to produce the same story were somehow just unintelligent retrieval methods.
>
> *Roger Schank, Tell Me A Story, p. 16* (Schank 1990)

Agents can harness people's expectations by using facial expressions to convey their internal state (how they "feel"). If done consistently, the user will quickly build a complex model of the agent's behaviour. However, the choice of graphical representation must be made carefully so that the users do not expect too much of the system.

1.4 Proposal

While there have been several thought experiments that combined the concepts in the previous three sections (for example, Apple's *Knowledge Navigator* (Sculley 1989), see description in Section 2.1), no one has yet come up with a satisfactory working product. Probably the closest attempt so far has been MIT Media Lab's *Rea* (Cassell, Bickmore, Billinghurst, Campbell, Chang, Vilhjalmsson & Yan 1999), a real estate agent that can converse about the accommodation in its database. However, the emphasis here is more on conversational actions such as gestures and turn-taking and less on putting complex information into context.

It is our belief that a successful presentation of information from varied sources must be both *relevant* and *easily understandable.* From our research we have discovered that these two criteria could be satisfied in a system that combines the three concepts of:

- providing information in a contextual setting;
- changing the interaction metaphor from direct to third party interaction;
- and providing personality-based feedback to encourage the user to trust that third party.

Since it is very difficult to judge these two criteria without extensive user testing and since this is only a six month project, our primary aim is to design and construct a system that successfully combines these three concepts.

1.5 Outline of the Thesis

The structure of this thesis approximately reflects the structure of the h2g2 project. We start with an idea of our goals for the project, as specified in this Introduction. We then investigate a large number of associated research projects and theories in order to see what is possible and to find systems that can help us achieve our goals. This search is detailed in the Background chapter.

Next we proceed to the Analysis, in which we specify the capabilities of h2g2. This culminates in a model of the information flow in the system which details how it interprets and processes the user's input in order to tell a relevant, personalised story.

This is followed by the Design and Implementation chapter which gives an overview of how we fulfilled the specifications of the previous chapter. This chapter does not give an in-depth technicial description of the h2g2 system – that can be found in the Javadoc source documentation and the edited worknotes in Appendix D. Instead we describe the development of the major processes involved in the system: storytelling, personalisation, the dialogue system and the graphical interface. This chapter ends with an example of the h2g2 system in use.

Once we have described the operation of h2g2, we move on to our Evaluation. Since this was a six month project we did not carry out extensive user tests but simply evaluated whether the separate modules worked according to their specifications from the Analysis. In the Evaluation chapter we give the results of these tests and also of our brief integration test of the system as a whole.

Finally, in the Discussion and Conclusion, we look at the topics raised by this project and see how h2g2 compares to the other projects we investigated in the Background chapter. We also look at what might be possible in the future in order to realise the *Hitch Hiker's Guide to the Galaxy*.

This report is intended to be read in conjunction with the associated appendices and Javadoc source documentation. These give further details of the workflow and implementation of this Master's Thesis. The Javadoc source documentation is available either on the CD together with this report or on the group web site at:
`http://www.kom.auc.dk/~adamc/group/`. The appendices include several edited worknotes that give more detail on development processes that are only mentioned briefly in the report.

Background

Chapter 2

'What is it?' asked Arthur.

'*The Hitch Hiker's Guide to the Galaxy.* It's a sort of electronic book. It tells you everything you need to know about anything. That's its job.'

Douglas Adams, The Hitch Hiker's Guide to the Galaxy

2.1 Science Fiction

The Hitch Hiker's Guide to the Galaxy (Adams 1979) started life in 1978 as a science fiction comedy on radio. Its story revolved around a book (the Guide) that could tell you about Life, the Universe and Everything, but that had so little room for Earth that its entry read "Mostly Harmless". The radio series became a book itself, then a trilogy, then a trilogy in five parts, and also metamorphosed into a TV series somewhere along the way. Such is its continuing popularity that even a Hollywood film is in the pipeline. But still the idea of the Guide remained: a source of information that was quite possibly strange and inaccurate but was most definitely hip and trendy!

More recently, its author, Douglas Adams, co-founded The Digital Village (TDV), a digital media and communications company. One of their long term projects is *h2g2*, which will consist of multiple information and entertainment services based on the attitude and humour of the original Guide. TDV's introduction to the *h2g2* project (TDV & AT&T 1997) contains the following vision of what it should eventually be capable of:

> A businessman arrives at work. He tells his Guide what he is working on for the day. He wants to sell a consignment of Peril-Sensitive Sunglasses, fashion eyewear which darkens in proportion to the danger surrounding the wearer. He also asks the Guide to book a restaurant for lunch near the New York Stock Exchange and find tickets for the opera tonight. No problem. The lunch booking is confirmed in 10 minutes. An hour later the Guide flags him on his desktop with a list of freshly located danger enthusiasts' phone numbers, the latest Travel Warnings from the State Department matched against this week's sunniest vacation spots, and some old-style World Wide Web URLs for ski-wear, scuba and hang-gliding merchandising. Over lunch, a pager message hits his cellphone: there are no tickets left for The Barber of Seville, but his four o'clock meeting is a known ice-hockey fan, and the Rangers are playing at home. No, he tells the Guide, he'll stay in tonight – please look out some good comedy shows and advise by TV.

There are many components of this vision which are still a long way off from the capabilities of today's technology. However, some ideas are applicable even today. Firstly, a Guide should be able to communicate with the user with whatever devices are available

– it's no good just being able to access information from an office computer when you're in a restaurant and only have your mobile phone with you. Secondly, it should adapt to the user's preferences, telling him about ice-hockey if he's into fast and dangerous sports, rather than cricket.

These ideas are not new. In 1987, Apple Computer produced a video of their vision of the future of computer interfaces (Sculley 1989). Apple's *Knowledge Navigator* was an onscreen butler that conversed with the user in real time and presented the information he wanted to see using appropriate media. Here is an example of the *Knowledge Navigator* in action (Lee 1993):

> **User:** Find me the paper I read on deforestation published by uh Flemson or something.
>
> **KN:** There is a paper by John Flemming, published in the Australian Journal of Earth Science, 2006.
>
> **User:** Yes, that's it. I'd like to recheck his Figures.
>
> (KN plots the deforestation predictions from the paper)
>
> **User:** And what happened?
>
> (KN plots actual rates of deforestation)

Neal Stephenson also put forward the idea of an electronic book that could adapt to its user in *The Diamond Age* (Stephenson 1995), another book about a book. Here the book in question was a primer – a teaching aid designed to teach a child how to behave. *The Diamond Age* is set in a future in which nanotechnology is commonplace: the Primer is a powerful computer in the shape of a book, with pages that can display colourful moving pictures. Not only can the Primer output sound, but it is also linked to the global network such that the stories in it are read by a human actor in some other location.

In the world of *The Diamond Age*, people no longer read newspapers consisting of several large sheets of printer paper gathered together. Instead, any sheet of nanotechnological paper is a newspaper – the user holds it, tells it to display the Times and reads his own personalised copy. The paper scans the user's fingerprints, retrieves the articles appropriate to their interests and displays them.

Surprisingly, this idea is not so far off as it may seem. Scientists in MIT's Media Lab are already working on paper-based displays (Jacobson, Comiskey, Turner, Albert & Tsao 1997) and another project in the same lab has developed a fully-working personalised newspaper.

2.2 Personalised Newspapers

At MIT, the News in the Future group has been researching new ways to present news. One of their projects has been *fishWrap* (Chesnais, Mucklo & Sheena 1995), named after the journalist's proverb: "Yesterday's news wraps today's fish". This is a web-based newspaper that allows users to receive news both about their own interests and about general affairs.

Initially, the system asks each user three questions: "Where are you from?", "What is your affiliation with MIT?" and "What majors interest you?". The answers are then used to create an individual user profile which is used to create a personalised newspaper on the fly each time the user logs into the system.

fishWrap takes input both from traditional news wires (Associated Press, Reuters, Knight-Ridder/Tribune and BPI Entertainment) and from local submissions. All articles are then converted to a common internal format called a *DType*. The conversion process also adds a signature to each article, containing such information as its source, time of publication, headline, keywords and sometimes even a summary. No complex information extraction is used – all this information is provided by the article itself (news wires use the ANPA (Bender 1995) format). The point of the DType is that all this information can be accessed easily and passed around the network with little fuss.

In further research, building on *fishWrap*, the *PLUM* system (Elo 1995) actually operates within the scope of individual articles. *PLUM* (Peace, Love and Understanding Machine) recognises disaster news and annotates the articles. It relates the facts of the disaster to information from the user's home town, in order to make them more meaningful to him.

The system scans through articles, looking for ones that match its general template of a disaster article. When it finds one, it picks out any statistics it recognises and compares them to equivalent statistics of the user's home town. For example: if 30,000 people have been killed in an earthquake in Papua New Guinea, *PLUM* will generate a meaningful local comparison such as "30,000 is 1 out of 19 people living in Boston". *PLUM* then creates a new DType consisting of an annotated article: the original article with links to the new information inserted at the appropriate points.

These two projects have a slightly different purpose to the *Hitch Hiker's Guide*, but they use some of the same techniques:

- storing information according to the concepts it contains
- taking large amounts of information and presenting the parts that are relevant to the user
- putting the new information into a context with which the user is already familiar

Underlying both of these systems are DTypes. This flexible way of storing and passing around information is based on a data structure well-known in Artificial Intelligence: Marvin Minsky's *frames*.

2.3 Frames

Minsky presented the idea of frames as a data structure well suited to storing knowledge used in Artificial Intelligence programs (Minsky 1974). A frame can be described as a list of slots each with an associated value (or set of values).

A simple frame is shown below, containing the recipe for the drink "White Russian".

Drink	
TYPE:	Drink
NAME:	White Russian
INGREDIENTS:	Milk or Cream, Coffee Liqueur, Vodka
ATTRIBUTES:	Alcohol, Mixed Drink

Frames can also contain references to other frames (The value ‘Alcohol’ could be a link to a frame describing the definition of ‘Alcohol’) meaning that a set of frames can be linked to create a network of information. In this way frames become a way of representing knowledge. The structure of each frame is usually decided by hand, though the contents and the links can be filled in automatically.

Besides describing information, frames can also be used as a means of communicating “intentions” (information needed to complete the current goal of the user or the system). Aalborg University has developed a system called *Chameleon* (Brœndsted, Dalsgaard, Larsen, Manthey, Mc Kevitt, Moeslund & Olesen 1998) which uses frames as the basis of communication between modules. One application of *Chameleon* is the “IntelliMedia Workbench”, a multimodal system in which the user is able to ask questions and use pointing gestures.

Chameleon uses a “blackboard” architecture in which each module communicates by sending and getting frames to and from a common notice board (see Figure 2.1). Examples of questions the IntelliMedia Workbench can cope with are: “Show me Hanne’s office,” “Point to Poul’s office,” “Whose office is this?” and “Show me a route from Hanne’s office to Poul’s office.” The Chameleon system will speak the answer and point with a laser on a plan of one of the University buldings.

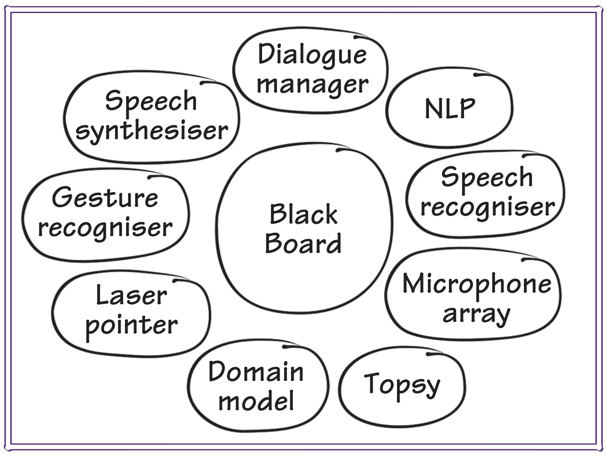


Figure 2.1: Modules in *Chameleon*

To demonstrate how the modules in *Chameleon* use frames to communicate, we can use an example in which the user wants to know where Hanne’s office is:

User says: “Point to Hanne’s office”

The **Natural Language Parser** indentifies the user’s intention (point instruction). It then sends a frame to the **Blackboard** so other modules can fill in the missing information.

NLP (int)	
INTENTION:	instruction! (pointing)
LOCATION:	office (tenant Hanne) (coordinates (X, Y))
TIME:	timestamp

The **Domain Model** has information on whose office belongs to whom and the location of each office. It is therefore able to find the coordinates for Hanne's office and fill in the the missing information in the frame.

DOMAIN-MODEL (int)	
INTENTION:	instruction! (pointing)
LOCATION:	office (owner Hanne) (**coordinates (5, 2)**)
TIME:	timestamp

Now that all the information gathering is completed, the next intention of the system is to indicate to the user where Hanne's office is. The **NLP** module chooses the appropriate sentence ("This is Hanne's office").

NLP (int)	
INTENTION:	description (pointing)
LOCATION:	office (owner Hanne) (coordinates (5, 2))
UTTERANCE:	**(This is Hanne's office)**
TIME:	timestamp

The **Laser** module sees the coordinates in the previous frame and puts a frame on the **Blackboard** to indicate that it wants to point at coordinate (5,2).

LASER (out)	
INTENTION:	description (pointing)
LOCATION:	coordinates (5, 2)
TIME:	timestamp

The **Speech Synthesiser** performs a similar action with the text to be spoken and the system then outputs its answer by saying "This is Hanne's office" while pointing to it at coordinate (5,2).

2.4 Statistical Methods

Since the basic structure of the frames for a particular system must designed by hand, frame-based systems are domain specific. For example, *PLUM*, a frame-based system, "knows" how to describe disasters but cannot deal with other types of events. Whilst this approach can be very successful for individual subject areas, the time and effort involved in describing new domains makes it ineffective for dealing with arbitrary text.

In the last few years, DARPA (the Defence Advanced Research Projects Agency of the USA) has been funding a major research project on information extraction. The aim of this project is to be able to automatically summarise the topics from a wide variety of information sources, from newspaper reports in different languages to broadcast news from CNN and other channels[1]. The DARPA research focusses on picking out *Named Entities* (people, locations, organisations, dates, times, percentages and currency values) from the information sources.

On our study tour to Boston and New York (IMM 1999), several research labs presented the current state of their information extraction projects to us. Most of these projects use statistical approaches rather than frames. These statistical approaches do not need any hand-designed knowledge structures to operate. Instead, they rely upon having large corpora of previously-annotated information available. For example, if the system is to discover the main topics in the current world news, it is first *trained* on old news in which the Named Entities have already been picked out (usually by hand). It builds up a statistical model of the language involved for each kind of Named Entity and uses this model to pick out other Named Entities from the current (unannotated) news sources.

One example of the statistical approach that is publicly available is *NewsMaps.com* (Cartia 1999). This is a web site run by Cartia, Inc. that uses a similar approach to the DARPA Named Entities projects to summarise different news areas as information *maps*. Each map shows topics as groups of keywords, with similar topics positioned close to each other and the more popular one shown as "peaks" using shading and contour lines. An example NewsMap from May the 19th 1999 is shown in Figure 2.2.

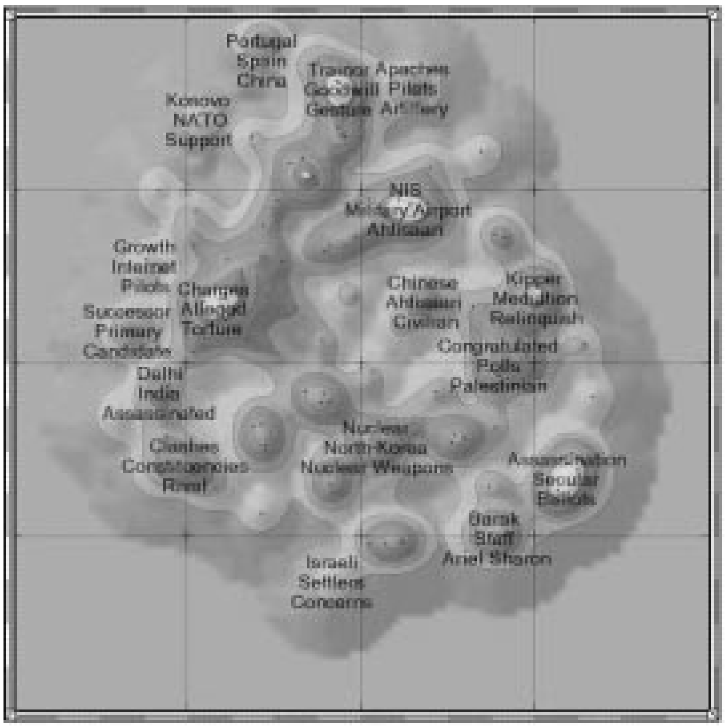


Figure 2.2: Topic landscape from ThemeScape.

2.4.1 Amalthaea

Each topic in a NewsMap could be represented as a vector of the associated entities. Alexandros Moukas, a research assistant in MIT's Media Lab, has developed an informa-

[1]Presumably the US Department of Defense is also interested in telephone conversations between suspected terrorists.

tion filtering system that uses such keyword vectors to summarise and select information that an individual user finds interesting. *Amalthaea* (Moukas 1996) generates its keywords using a simpler technique than the Named Entities projects: it simply stems all the words in a document, removes all common words such as "the, "it", "for" and "will") and assigns a weight to each resulting term proportional to its frequency in the document. As such, it cannot differentiate between different types of entity.

Nevertheless, Moukas's system does have some very interesting capabilities. It uses two populations of genetically-based agents to filter information from the web. One population is of information discovery agents, which look on a particular web page (possibly a search engine) for a given set of keywords. These keywords are provided in a contract with an agent from the other population of information filtering agents.

Each information filtering agent has a *genotype* consisting of a weighted keyword vector with which it assigns confidence values to the information it receives. A confidence value is calculated as the distance from the agent's weighted keyword vector to the keyword vector computed from the information under consideration. Each filtering agent assumes its keyword vector is a perfect representation of the user's interests, but then each is assigned a fitness level according to how much the user likes the information they select. Discovery agents try to make contracts with filtering agents that have a high level of fitness.

The agents then go through a process of selection by having to "pay rent" every generation. They can only get credit by providing useful information to the user. Filtering agents get credit (and debit) directly from the user, whereas discovery agents get credit passed down to them from the filtering agents they have contracts from. Only the information selected by the fittest 40% of the information filtering agents is presented to the user – the rest are there for the diversity of the population. Both populations are subject to evolution via mutation and crossover of their genotypes with only the top percentage of the populations allowed to reproduce. This evolutionary process means that both populations of agents gradually adapt to the user's preferences. Once adapted, *Amalthaea* is able not only to select but also to discover relevant information for the user.

2.5 Scripts

Whilst frames and Named Entities are good ways of representing knowledge in computers, they are not necessarily any good at conveying that information to people. In 1977, Roger Schank and Robert Abelson in 1977 (Schank & Abelson 1977) proposed that human memory is episodic, i.e. that we remember things by the stories, or *scripts*, in which they occur. The most quoted example of this idea is a visit to a restaurant, in which the customer (and the waiter) is simply following a well-known script that goes something like this (Schank 1986):

1. entering
2. seating
3. ordering
4. eating

5. paying
6. leaving

Schank claims that such a script forms by repeatedly having the same experiences – all the restaurants in which this sequence of events happens reinforce the script. More importantly, he claims that *all* events are stored in memory by being related to a script. For example, a visit to Legal Seafood (where you are expected to pay immediately after ordering) would be stored as an exception to the ordering or paying scene in the general restaurant script. This kind of knowledge representation fits in very well with our proposal, not only because everything in a script is in context by definition, but also because it is all about telling stories.

Schank went on to develop his theories in the field of education. He proposed that since people remember things in terms of scripts (now split into two levels: scriptlets and Memory Organisation Packets or MOPs), their intelligence depends on how easily they can relate new information to existing MOPs and how well they can relate different MOPs together. In *Engines for Education* (Schank & Cleary 1994), he says that asking questions is the way to index items in our memories and that "better indexing allows our memories to be more flexible."

2.5.1 Interesting Heuristics

At Georgia Institute for Technology, Ashwin Ram and his research group IGOR have been working along similar lines. They have developed a series of systems that use *goal-driven learning* to read and understand natural language stories. All of these systems are very complex and so we can only give a brief overview of parts of them, but they do introduce some important concepts.

The first system is called *AQUA* (Asking Questions and Understanding Answers) (Ram 1989). It is designed to learn about terrorism by reading newspaper stories about terrorist incidents in the Middle East, picking out the parts that it deems to be interesting. The system decides what is interesting based on knowledge goals that change as the system processes the stories. For example, if *AQUA* is reading an article about a terrorist bombing motivated by blackmail, it will note that its previously held idea that terrorist bombings are motivated by religious fanaticism does not hold. It will then pose several questions such as, "What did the bomber value more than his own life?" and look for other information to answer these.

AQUA highlights the idea of *interestingness*. Though its implementation is too complex for us to look at in detail, Ram does provide 14 heuristics (Ram 1990) that the system uses to decide whether a topic is interesting. These are relevant to our proposal in that they could be flipped on their head and used by a *storytelling* system to make its stories interesting to the user.

Another system under development at IGOR is *PIES* (Personal Interest Engine for Stories) (Ram 1991). This is like *fishWrap* at MIT in that it provides a personalised newspaper for the user. However, *PIES* works at a lower level, personalising the stories themselves rather than the choice of articles in the newspaper. It does this in a similar way to *AQUA* – looking for interesting parts of the stories. The difference from *AQUA* is that *PIES* uses a model of the user's interests to prune its knowledge structures before it applies them to the incoming story. Any story is then understood from the user's point of view and the summary it generates should be extremely pertinent.

The *PIES* system would seem to be extremely relevant to our proposal since it can output information in a context guaranteed to be understood by the user. However, in order to do this, it needs a hand-coded model of the user's interests. Not only is this difficult to create but it does not change, since the system gets no feedback from the user.

2.5.2 Bizarre Heuristics

A more relevant project at IGOR is called *ISAAC* (Integrated Story Analysis And Creativity) (Moorman & Ram 1994a), a system that reads and understands science fiction stories. *ISAAC* differs from the previous two systems in that its subject matter does not hold by the same rules. In science fiction anything can happen, and *ISAAC* must be creative when understanding the stories. For example, one story (Bloch 1963) tells of a future in which humans are extinct and humanoid robots have taken their place. Before reading the story, *ISAAC* only knows of robots as industrial machines and so it must create the concept of a humanoid robot. It does this by merging its concept of a robot with its concept of a man, guided by the title of the story, *Men Are Different*.

	Physical	**Mental**	**Social**	**Emotional**	**Temporal**
Agents	person	consciousness	boss	Ares	entropy
Actions	waiting	thinking	selling	loving	getting closer to March
Objects	rock	idea	teacher-student relationship	hatred	second
States	young	lack of knowledge	public dishonour	being angry	early

Figure 2.3: The *ISAAC* knowledge grid

To control the amount of creativity that *ISAAC* employs during its understanding process, the system uses *bizarreness heuristics* (Moorman & Ram 1996) which could be very appropriate for the Hitchhiker's Guide. All its knowledge is organised into a grid as shown in 2.3. The vertical axis represents the type of a concept (Schank & Abelson 1977), whereas the horizontal axis indicates its domain. When trying to understand a new concept, the system is guided by the bizarreness heuristics as to which kinds of movement are allowed in the grid. The example of the humanoid robot would take an industrial robot, a concept from the *physical object* cell, and combine it with a man, a concept from the *physical agent* cell. This kind of movement from objects to agents is considered quite reasonable by the heuristics.

One of the attractions of Douglas Adams' writing style is its very bizarreness – take for example, "The ships hung in the sky in much same way that bricks don't". Though we realise that there is much more than bizarreness to such statements, a system which could control how bizarre it made its stories could have some interesting properties.

2.6 Summary

In this chapter we have reviewed some of the current research relevant to our project. We have gone from the ideas of science fiction, through Minsky's frames to the latest DARPA competitions in information extraction. We also mentioned Schank's scripts and finished with Moorman and Ram's *ISAAC*, a system that could understand the ideas of science fiction but not carry them out.

Analysis

Chapter 3

The Hitch Hiker's Guide to the Galaxy is a very unevenly edited book and contains many passages that simply seemed to the editors like a good idea at the time.

Douglas Adams, The Hitch Hiker's Guide to the Galaxy

3.1 Preliminary Specification

As specified in Section 1.4, we intend to make a system that has the following capabilities:

- providing information in a contextual setting;
- using third party interaction via interface agents;
- providing personality-based feedback to encourage the user to trust the agents.

The original *Hitch Hiker's Guide to the Galaxy* only had the first of these capabilities. As portrayed in the books and other media, it was basically an encyclopedia: the user could search the index by typing in a keyword and would get back a fixed article that matched the keyword.

TDV's *h2g2* project, as described in Section 2.1, was our main inspiration. This goes much further than the original Guide. It uses an in-depth profile of the user to answer natural language queries; it works by itself while the user is doing something else; and it can even use several methods of communication to keep in contact with the user.

3.1.1 Capabilities of h2g2

We do not intend to keep up with all these aims: our h2g2 will lie between the two. We will keep the "omissions, apocrypha and inaccuracies" of the original version and add some amount of the personalisation of TDV's *h2g2*. We also hope to have some contextual knowledge but not as much as that envisioned by TDV's *h2g2*. For example, our system might be able to understand that "lunchtime" is around midday, but would not be able to link Peril-Sensitive Sunglasses to ski-wear, scuba and hang-gliding enthusiasts.

As a prototype, h2g2 will only run on a desktop machine. However, we intend to design it so that the interface could easily be transferred to a portable device such as a Personal Digital Assistant (PDA) or a mobile phone. To this end, we shall aim to include modalities available to such a mobile device: speech input and output, text and graphics display, pointing input (mouse-based in our case, but pen-based on a mobile device) and positional input (such as Global Positioning System signals). As such the system should be accessible by just voice, just pen input (or mouse and keyboard input) or any combination of these. However, since one of major criteria was the graphical display of the agent's current "feelings", graphical output will always be required.

h2g2 will take the user's queries and tell stories in response. We hope to include some storytelling capability in the system so that the stories told are not word-for-word the information that is provided to the system.

3.2 Storytelling

Telling stories is a complicated process. The process by which people tell stories (and understand them) is not fully understood, though as we discovered in Chapter 2, many people are researching this area.

The main avenue of research (initiated by Schank in (Schank & Abelson 1977)) involves representing stories as scripts. Each script is a series of causal events and actions: exactly the kind of information that frames were developed to represent. Thus it would seem that a script could be built up by gathering and ordering associated frames. This script could then be used to generate a story.

However, this process is not as simple as it might seem. Firstly, to build up a script of associated frames, we need some way of indexing a large number of frames and of finding related ones quickly. Secondly, the process of building a story from a script involves a lot of knowledge about language – the script is the bare bones of a story and it needs to be fleshed out to be readable.

3.2.1 Introducing FramerD

In the Machine Understanding Group at MIT's Media Lab, Professor Ken Haase is developing *FramerD* (Haase 1996). This is a cross-platform, object-oriented database designed to store objects (called DTypes) that can contain references to other objects. Its major purpose is to store and index frames for easy retrieval.

This is the perfect answer to the first part of our storytelling problem, but *FramerD* doesn't stop there. It also comes with a huge (almost 200 Mb) database of frames called the *Brico* Knowledge Base. This combines the *WordNet* lexical reference system (Chodorow, Fellbaum, Johnson-Laird, Landes, Miller, Tengi, Wakefield, Ziskind & Lipski 1999), a semantic network representing the 1911 version of Roget's thesaurus and the top level of the *CYC ontology*[1] (Cycorp 1999). This allows *FramerD* not only to relate different words together, but also to discover underlying topics. In short, exactly the language knowledge that we need for the second part of our storytelling problem.

As well as being a perfect fit for our project, *FramerD* is also available in source and binary form free of charge for educational and research purposes from the *FramerD* web site (`http://www.framerd.org`). Since it is a complicated system we will not describe it in detail here. An overview can be found in Appendix B and more details are available in (Haase 1996).

[1]Only a very small subset of the CYC ontology is freely available.

3.3 Personalisation

We want our system to personalise the information it displays to the individual user. To do this we need to create a model of the user's likes and dislikes. In this section we discuss what this model should consist of and how it should be represented to the user.

3.3.1 Previous User Models

Several of the systems mentioned in Chapter 2 have models of their users. *PIES* has a hand-coded model made of rules such as "user likes baseball" and "user is a friend of the baseball player called Bret Saberhagen[2]". Similarly, *fishWrap* keeps a fixed list of details about its users – where they live, what courses they are taking, etc. This style of user model can only contain fixed data and requires the user to spend time setting it up before using the system.

Amalthaea uses a different approach – its user model resides in the fitness levels of its populations of agents. As users interact with the system, the agents evolve into a better representation of their preferences. This machine-learning approach has two main advantages. Firstly, the user doesn't have to specify their preferences before they can make use of the system. Secondly, the system can adapt to changes in the users' preferences, allowing much more flexibility.

However, in order to evolve the agents in the system, the *Amalthaea* user has to provide "relevance feedback". This consists of evaluating how well each answer matched the query. Given that the *Amalthaea* agents take over fifty generations to adapt to an individual user, this could get quite tiresome.

A Probabilistic Approach

In our previous semester's project, *MPIA* (Christiansen et al. 1999), we prototyped a handheld city-guide that learned the user's preferences and recommended suitable places to visit. The preferences that the system attempted to learn were the user's criteria for choosing one suggestion over another.

We modelled these criteria by proposing that the user's decision was based on a set of independent preferences. The *MPIA* system the system modelled two preferences: *weariness* (dislike of travelling) and *stinginess* (dislike of spending money). Each suggestion was then treated as a set of attributes that related to these preferences. In the case of *MPIA*, each suggestion was reduced to its distance away from the user and the cost of visiting it.

The Decision Support System (DSS) predicted the user's choice by assigning penalty values to the places in the database. These penalty values reflected how much it believed the user cared about the attributes of each place (based on the associated preferences). The DSS then suggested the places with the least penalty values.

The DSS modelled each preference as a probability distribution over a single random variable: it did not give an exact value to the stinginess of the user but instead assigned probabilities to different levels of stinginess. Each time the user made a choice, the distributions were updated using Bayes theorem. This involved combining the distributions

[2] a baseball player for the Atlanta Braves

into a joint probability distribution. The joint distribution was then multiplied by the conditional probability of the actual choice, normalised and marginalised to get the new distributions of each preference. For a more detailed explanation of this application of Bayes theorem see (Pearl 1987).

Obviously, this model did not represent the user's complete decision-making process. However, it could quickly guess the user's feelings towards travelling and spending money (at least when in the context of restaurants – see Section 6.3). This meant that its predictions often agreed with the user's choices when the user was not taking other factors into account.

This probabilistic system allowed the city-guide to adapt to the user very quickly using only the choice of recommendation to update the user model. This avoided both the *fishWrap*-style question and answer session before the system could be used and the *Amalthaea*-style feedback after every query.

3.3.2 A User Model for h2g2

The user model in h2g2 has a similar purpose to that of *MPIA* – to predict the user's choice. However, there is one important difference: *MPIA* rates how much the user would like to go to a place, whereas h2g2 rates how much the user would like to be told a particular story.

The major effect of this change in focus is that stories differ from each other a great deal more than places. There are only so many attributes of places on which people base their decisions of where to go. Stories, on the other hand, can be liked or disliked for any number of reasons: from the myriad of subjects they can contain to the author's writing style.

The DSS in *MPIA* had specially coded penalty functions for each attribute with a preference. These functions took the attribute, converted it to a value in a standard measure and then multiplied the resulting value by a weighting calculated from the user's preference. The DSS calculated a penalty value for a specific place by summing the individual penalties of each of its attributes.

If we want to apply a similar model to h2g2, we have to make the DSS much more flexible. Not only will we have to compare items that have different numbers of attributes, but since we cannot predict what the attributes of stories will be, the DSS will have to be able to add new preferences on the fly.

Two Kinds of Attributes

Comparing items with differing numbers of attributes is a simple enough problem to solve. Instead of summing the penalties for each of the attributes we return the average penalty. This returns just as much information and does not penalise items on which we have more information.

Adding new preferences on the fly is a much harder problem. As noted above, each attribute in *MPIA* had its own hard-coded function to convert the attribute's value into a standard measure. Adding a preference for a new attribute on the fly would mean automatically generating a new conversion function.

Doing this for attributes such as distance and cost is virtually impossible. There is no way to write a general algorithm that can take a distance and compare it to a price.

However, not all attributes are so complicated. Distance and price are examples of *ordered attributes*: attributes that have scalar values. Some attributes, like the type of food served at a restaurant do not have any particular ordering.

These *unordered attributes* are much easier to deal with. The DSS can use an indicator function for each of the different states of the unordered attribute (returning one when the state is present and zero otherwise). Converting the output of the indicator functions to the standard measure is simply a matter of multiplying them by a weight. The DSS then assigns a preference for each state of the unordered attribute.

For example, a food type attribute could be automatically added to the DSS by creating indicator functions and a preference distribution for each type of food (one for Italian, one for French, etc.). For a restaurant that served Mexican food, only the preference for Mexican food would come into effect (the indicator functions for the other types of food would return zero).

We therefore decided to fix the ordered attributes that the DSS knew about to distance and price, but to allow the DSS to add unordered attributes as it came across them.

3.4 Personality Representation

This complicated user model is a perfect example of the need to have a personality-based presentation of the agent's internal state. It is hard enough to explain the structure of the Decision Support System in several paragraphs of (reasonably) technical language, let alone explain its current state in a quick glance to the screen of a handheld device.

We chose to represent the state of the user model with a cartoon humanoid face. Although this risked overestimates of h2g2's intelligence, we felt that users would feel more comfortable receiving guidance from a human-style character rather than the animal-style character as used in *MPIA*.

We also chose to use a graduation of emotions from happy, through confused, to hopeless as a representation of the state of the user model. The mood would be updated each time the user made a choice and would be linked to the conditional probability of the user's choice. If the choice agreed with the prediction of the DSS then the presentation agent would get happier; conversely, if the DSS got its prediction wrong then the agent would get more confused.

One reason for choosing such expressions is our research for the *MPIA* project. We investigated the appeal of Tamagotchis and discovered that one reason for their success was that people felt *responsible* for them. The *MPIA* system was designed to take advantage of this effect in order to encourage the user to reveal more information about her preferences. When the presentation agent wasn't happy, the user could select a *Pet Psychiatrist* feature. This would request that the user tell her pet why she had made certain decisions that the DSS had not predicted. Once the user had explained her decision strategy (and the DSS had updated itself), the pet became happier.

The Pet Psychiatrist is a feature that we would like to include in h2g2. However, since it is not necessary for the basic functionality of telling stories, we shall leave it out of the initial prototype. Instead of the mood only getting better when the user visits the Pet Psychiatrist, we shall implement a weighted history. The mood of the presentation agent will therefore depend on the success of the more recent predictions.

3.4.1 Multiple Personalities

One problem with personalisation that we noted in *MPIA* was that sometimes the user wants to do something different from their usual behaviour. For example, a student might be going out to eat with his parents and so wants to look at some more up-market restaurants than his usual fare.

In *Amalthaea*, each information filtering agent believed itself to have an exact model of the user. The information they provided was then ranked according to the "real" user model (in *Amalthaea* this corresponded to the fitness of the information filtering agents) before it was presented to the user. This increased the diversity of the presented information: the user received several different viewpoints of similar information.

This idea helps solve the problem noted above. Instead of receiving just one view of information, tailored to the system's model of the user, the user is presented with several variations. For example, one agent (the *HooterAgent*) could be a sports fanatic, wanting to involve sports with everything it recommends. Another could be based on an Actual Human Being (AHB) who makes reviews and puts them in the system for later use. When the user asks for a place to eat, the HooterAgent finds a cafe where you can watch basketball, whereas the AHB agent returns the place that the human reviewer currently likes most. These results (and others) are then rated according to the user model and displayed.

3.5 System Description

We are now at a stage where we can specify the flow of information in our system. Figure 3.1 summarises how the knowledge stored in the FramerD database is selected and ordered by the Storytelling Agents into *smarticles* (smart articles) which are then displayed to the user. The sections that follow provide more detail on each part of the diagram.

3.5.1 FramerD Database

The FramerD database stores h2g2's knowledge as frames linked together by their content. This knowledge comes in two forms: language knowledge, as represented by the Brico knowledge base; and knowledge of the world. This knowledge of the world represents the information about which h2g2 can tell stories. As it is stored in frames, we will have to define a structure for this information. However, information discovery agents could fill these frames by searching the web and other information sources (in a similar way to *Amalthaea*). Since we are concentrating on the ability of the system to make presentations, we will not spend time on information discovery and will instead construct a fixed world for the storytelling agents to present.

3.5.2 Storytelling Agents

As specified in Section 3.4.1, we want h2g2 to have a number of storytelling agents, each with its own personality, to provide the user with a diverse selection of stories. All the agents can access the FramerD database to create their stories. Three possible types of storytelling agent are shown in Figure 3.1:

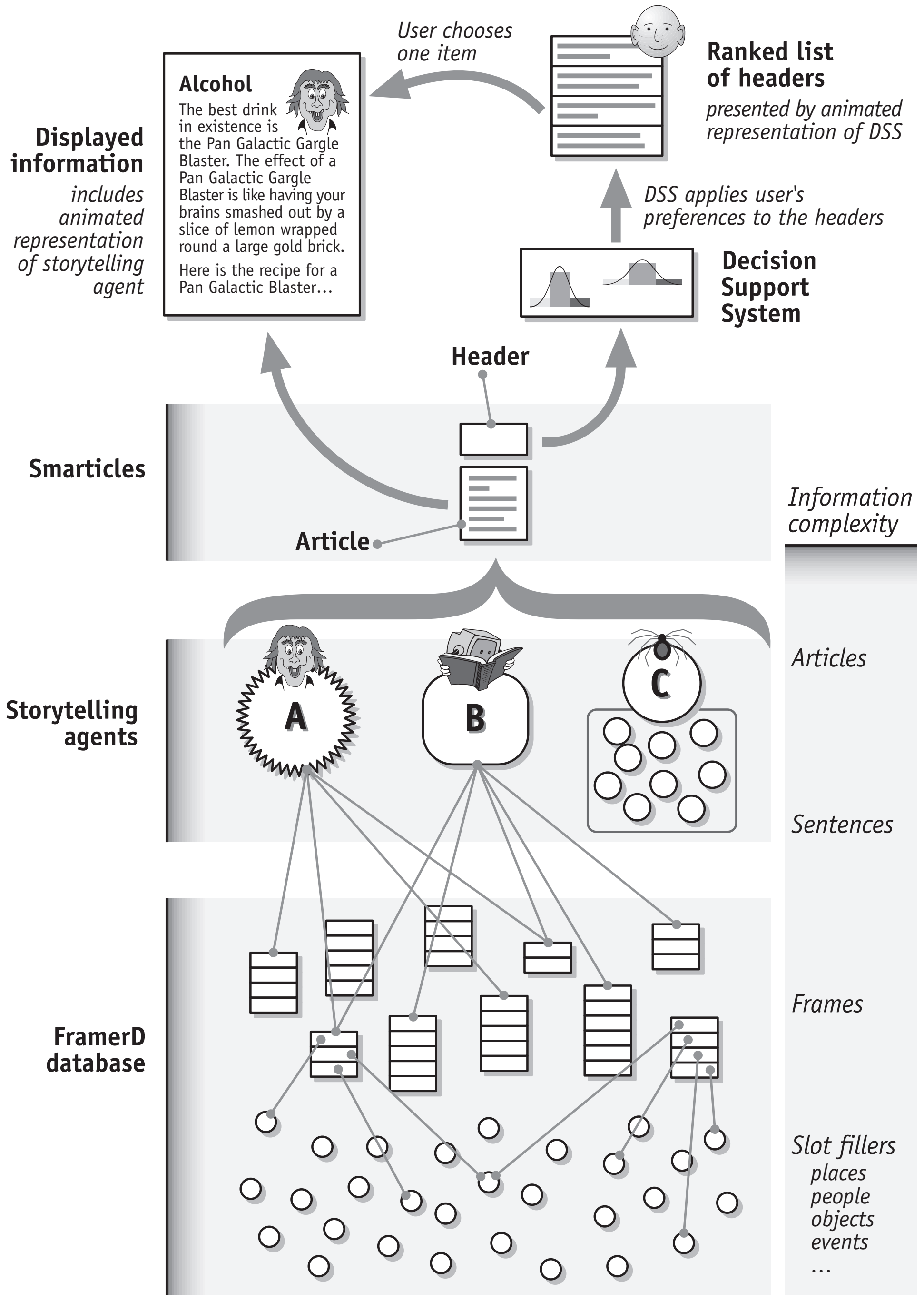

User chooses one item
Alcohol
The best drink in existence is the Pan Galactic Gargle Blaster. The effect of a Pan Galactic Gargle Blaster is like having your brains smashed out by a slice of lemon wrapped round a large gold brick.
Here is the recipe for a Pan Galactic Blaster...
Displayed information
includes animated representation of storytelling agent
Ranked list of headers
presented by animated representation of DSS
DSS applies user's preferences to the headers
Decision Support System
Header
Smarticles
Article
Information complexity
Articles
Storytelling agents
A
B
C
Sentences
FramerD database
Frames
Slot fillers
places
people
objects
events
...

Figure 3.1: Information flow in h2g2

Actual Human Being agent (A): The AHB agent could be a human operator writing smarticles in real-time to answer the user's queries or it could be a software agent that selects smarticles from a database of human-written ones.

Predefined agent (B): This kind of agent has some fixed code that is executed to generate a smarticle each time it gets a query from the user. One way that this might be done is for the agent to have a bias, for example the HooterAgent with its sports bias, or even an agent that only tells fictitious stories.

Genetically-based agent (C): This is a special type of agent because it is not just one agent, but rather a whole population of agents similar to those in *Amalthaea* (see Section 2.4.1). Each of the agents in the population would have some genotype that determines how it tells its stories. When a genetically-based agent receives a query, the fittest agents get to tell stories. These agents then receive credit or debit according to the scores assigned to their stories by the Decision Support System and whether or not the user chooses to read them. The agents then go through a selection process based on their level of credit and the survivors get to produce offspring by mating their genotypes or mutating. The genetically-based agent thus gradually adapts to the user, producing better and better stories.

3.5.3 Smarticles

"Smarticles" are defined in TDV's Principles of Guidance (TDV & AT&T 1997) as "multimedia constructs delivering short, easily digested and flexible items of information, plus any control functions unique to that content". In h2g2, smarticles are the stories that the storytelling agents tell.

Each smarticle will have a header containing the attributes it involves so that it can be rated by the Decision Support System. As specified in Section 3.3.2, these attributes can either be distance or price values or any unordered attribute such as food type.

h2g2 also uses the smarticle headers to make a list of the stories available for the user to choose. Therefore, the smarticle headers must also contain a title for the smarticle and an identification of the authoring agent.

3.5.4 Decision Support System

When the smarticles are listed for the user to choose amongst them, they are sorted according to the penalty values they receive from the Decision Support System. The smarticles with the least penalties (those that best fit the current user model) are placed at the top of the list. One of the purposes of h2g2 is to avoid long lists of unrelated information, so we will not have more than five items in this list. Smarticles that have too high a penalty will not be displayed.

The DSS calculates its penalties according to the probabilistic user model described in Section 3.3.2. It updates this user model each time the user chooses a smarticle to read from the list.

3.5.5 Display

h2g2 has two main displays for presenting information:

- the list of smarticles, presented by an animated representation of the DSS
- and the smarticle the user chooses, presented by the authoring agent.

For the purposes of this prototype, smarticles will consist of plain text so that they can easily be displayed on a PDA-style screen. However, future implementations could have smarticles that presented themselves differently according to the output device: a desktop computer user would get full-colour animation, sound and speech; whereas a mobile phone user would only see a brief textual summary and hear a longer spoken version.

3.6 User Input

The system we have described so far is designed to tell stories that satisfy our original criteria: they will put information in context and will be selected as relevant to the user (or at least to our model of the user). However, we have missed out a vital component – how the user specifies what information she wants.

Most current internet search engines ask the user to provide some combination of keywords to define their search. The problem with this approach is that the user is not stating their actual need, but is guessing some keywords linked to it. Searching for less well-defined information, such as "What's on TV tonight?", is difficult, often requiring several searches. This problem is exacerbated by the overwhelming bias towards American information – searching for European-specific information is always harder.

3.6.1 Natural Language

However, some search engines, such as *Ask Jeeves* (Ask Jeeves 1997), try a different approach. Instead of users having to think of keywords, all they have to do is type their question in plain English (unfortunately no other languages are supported at the moment). The users' questions are then matched to question templates, each of which has its own hand-picked answering pages. For example, asking "What's on TV tonight?" not only returns a page on which you can look up the current schedules of all the US TV stations, but also the answers to several more detailed questions such as, "What is on television tonight in <country>?" and "What is playing on the <network-name> network?" These more detailed answers are parameterised so that the user can choose the particular question to which they want the answer.

There is a similar change of focus between the original Hitchhiker's Guide, as described in the books, and TDV's *h2g2* as envisioned in (TDV & AT&T 1997). This was the original way of searching the Guide:

> 'You press this button here you see and the screen lights up giving you the index.'
>
> A screen, about three inches by four, lit up and characters began to flicker across the surface.

'You want to know about Vogons, so I enter that name so.' His fingers tapped some more keys. 'And there we are.'

The words *Vogon Constructor Fleets* flared in green across the screen.

Ford pressed a large red button at the bottom of the screen and words began to undulate across it. At the same time, the book began to speak the entry as well in a still quiet measured voice.

Douglas Adams, The Hitch Hiker's Guide to the Galaxy

TDV's *h2g2* (in Section 2.1) envisions natural language queries that extend way beyond those of *Ask Jeeves*: given that the user says he wants to sell Peril-Sensitive sunglasses, the Guide doesn't look for information about the glasses themselves but for people who would be interested in buying them. Instead of looking for keywords in the user's question, it looks for the user's *intention*.

3.6.2 Intentions

Intentions (sometimes called *speech acts* in spoken dialogue systems) are not so much the syntax or semantics of a sentence but *why* the sentence was said. They are the goals and beliefs behind it, part of the *pragmatics* of communication. Intentions cannot be observed directly, since we cannot see inside people's heads, but they can be inferred and recognised from the syntax, semantics and structure of the communication.

According to Schank (Schank 1990), people communicate by telling stories related to their intentions (or goals) and judge other people's intelligence by whether they recognise those goals and respond with stories that relate to them. In more closely related work, Mc Kevitt (Mc Kevitt 1991) argues that analysing intentions provides more effective human-computer communication. Applying these theories, a successful Guide should be able to detect the users' intentions and respond with relevant stories.

The problem with this approach is that determining the users' intention from what they say is a very difficult task. Firstly, the system must know about the subject in question in some detail to be able to determine, for example, that Peril-Sensitive sunglasses might be worn by people interested in dangerous sports such as hang-gliding. Secondly, it must also have a deep knowledge of language, since the same intention could be expressed in many different ways.

Fortunately, the Machine Understanding group at MIT has developed a parser to go with *FramerD*. It makes use of the extensive language knowledge represented by the *Brico* knowledge base (KB) and is designed to be able to draw analogies between sentences. For example, it can detect that the sentences, "Clinton named William Perry to be his secretary of defense" and, "Jocelyn Elders was chosen by President Clinton to be the new surgeon general" contain similar actions although the wording is different. This is exactly the language knowledge that we need to detect the users' intentions.

The Domain Knowledge Problem

Unfortunately, the other half of the problem, the knowledge of the subject (the *domain* knowledge) is not so easy to provide. A Guide that could answer questions on any subject would need to be able to reason about every possible subject in great detail. This kind of knowledge is not available and is not likely to be in the near future. Apart from the huge storage that such a knowledge base would require, all the links and connections between the different subjects would need to be determined and represented in some way. *Cycorp* (Cycorp 1999) is attempting to create a structure along these lines and their top-level structure is used in the *Brico* KB in *FramerD*. However, though the full *Cycorp* KB is much more detailed, it is still only successful with topics that it has been specifically taught about.

Nevertheless, we have to start somewhere. A Guide that knows about Life, the Universe and Everything is not feasible with today's technology, but a Guide that knows a large amount about a particular subject is. And anyway, when you want to know about TV schedules you don't go and ask a sunglasses salesman. The only question, then, is over which domain should h2g2, our Guide, provide guidance.

Limiting the Domain

On our study tour to Boston and New York, we met Victor Zue, head of MIT's Spoken Language Systems group. He has four criteria by which he selects new domains for spoken language systems (Gross, Judge, Port & Wildstrom 1998):

1. millions of people must care about the information, for example sports, stocks or weather;
2. the context must be clearly defined;
3. the information must change, so people come back for more;
4. and it must be something you can make money from

We chose the domain of places to eat and drink. This certainly fulfills the first two criteria and though the basic information does not change particularly rapidly, a Guide should be able to provide differing stories each time it is queried. Zue's final criterion doesn't really apply to this project as we do not intend to create a customer-ready product. However, magazines like *TimeOut* already make money offering similar advice so there is definitely a market!

3.6.3 Intention Analysis

Within a specific domain, people have specific intentions. Mc Kevitt (Mc Kevitt 1991) has defined (and categorised from Wizard of Oz[3] data) a list of intention types relevant for the operating systems consultancy domain. He argues that these types apply to many other consultancy domains. However, the domain of consultancy is slightly different from the domain of guidance that comes with the Hitch Hiker's Guide. For a start a consultant is assumed to be telling the truth with respect to achieving some specific task. Consequently, these intentions might not apply.

One way to discover what intentions people have is to derive them from the questions they ask. We conducted an informal survey of three people in which we asked them to

[3]A Wizard of Oz experiment is one in which potential system responses are simulated by a hidden human operator.

think of all the questions that they might ask a system that knew everything there was to know about restaurants and pubs. The subjects were familiar with spoken dialogue systems but we asked them to imagine that they were talking to a person. However, our survey did not include a conversational context: the subjects were asked to think of individual questions and were not presented with any answers that they could follow up.

The resulting list of questions can be seen in Appendix A. We then grouped the similar questions together and found there to be five main types of intention which we have labelled as follows:

1. Show a list of places with specific attributes (ShowListWithAttributes)
   - I want a free-range, organic meal in town. What can you suggest?
   - Where can I get a pint of *Guinness*?
   - Which Italian restaurants have a low corkage price?
2. Give details of a specific attribute at a specific place (DetailAttribute)
   - Does *Natalie's* accept visa?
   - Do I need to make a reservation at *Benny's* on a Saturday evening?
   - What is the corkage price at *La Provence*?
3. Give an opinion about a specific subject (GiveOpinion)
   - What do you know about *Carlsberg Classic*?
   - What would you recommend at *Natalie's*?
4. Show a list of places suitable for a specific situation (ShowListForSituation)
   - I want to take my Grandma to a restaurant she'd like. What can you suggest?
   - It's a first date – where should I take him?
5. Generalisations (Generalisation)
   - What percentage service do I have to pay in a Danish restaurant?
   - What types of restaurant are there in Aalborg?

Because our survey did not include a conversational context, we did not find any discourse-related intentions such as repetition (repeated requests) or elaboration (requesting more information). This was a limitation of our survey, but we do nevertheless intend to design a system that could interpret sequences of intentions as well as single, isolated ones.

Comparison to Intentions in the Consultancy Domain

Some of the intentions we found are similar to those in (Mc Kevitt 1991) but there are significant differences due to the difference in domain.

The two ShowList intentions (1 and 4 above) are the basic user requests. As such they could be compared to Mc Kevitt's basic information intention. However, Mc Kevitt's intentions were originally derived for *OSCON*, a *UNIX* help system in which the user was given direct answers in response to their questions. Since h2g2 provides *guidance* and not direct information there is a difference: Mc Kevitt's information intentions request a PLAN[4] to achieve a specified GOAL[5] (Schank & Abelson 1977); the ShowList intentions request some *guidance* as to how to achieve a GOAL. As the Principles of Guidance state (TDV & AT&T 1997): "A guide – human, print or electronic – provides a service. However, it isn't the service itself." For example, if the user asked for somewhere to drink a pint of Guinness, h2g2 might provide some reviews of nearby pubs. These would not necessarily be PLANS that could be executed to achieve the GOAL of drinking a Guinness, but information for the user to insert into her own plans for ordering a pint of Guinness in a pub. Some reviews could include a DETAILED-PLAN giving instructions on how to get to the pub and a layout of the bar showing where to stand to get served quickly, but this information is not necessarily needed for the user to make use of the guidance.

DetailAttribute (2 above) is very similar to three of Mc Kevitt's intentions, confirmation, description and explanation, in which the user is asking for confirmation of a belief or for a description or explanation of an object. The same discrimination can be made in our domain: some DetailAttribute questions expect h2g2 to confirm or deny a belief about a particular restaurant whereas others expect a description about that particular attribute. However, from a Guide that tells stories about places, a simple yes or no answer is not very interesting and so we decided that our system should interpret these two intentions in the same way.

Similarly, Generalisation (5 above) and GiveOpinion (3 above) could be other kinds of description intentions: Generalisation intentions ask for a description of the average attributes of a group of objects, whereas GiveOpinion intentions ask for some subjective input on behalf of h2g2. These again reflect the difference between the consultancy and guidance domains.

The more general Mc Kevitt intention types covered the user's basic goals for each type of question we detected, but our specific intention types captured the detailed nuances of the user's goals over our domain. In a similar way, Mc Kevitt has detailed sub-types of his general categories for the *UNIX* help domain.

3.6.4 Specification of User Input

We can now specify how the user determines what information she wants h2g2 to tell her. As can be seen in Figure 3.2, the user's spoken or typed text is parsed by the MIT parser and matched to an intention type. This intention is then passed to the storytelling agents who can tell exactly what kind of smarticle to generate from the *FramerD* database. The resulting smarticles are passed back to the user via the decision support system and the GUI as in Figure 3.1.

[4]A PLAN is defined as a set of actions to achieve some goal

[5]A GOAL is defined as an operation a speaker wishes to achieve

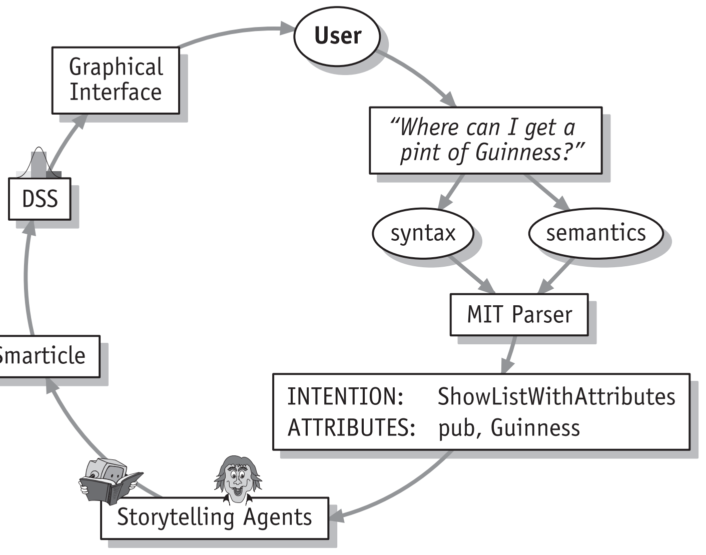

User
"Where can I get a pint of Guinness?"
syntax
semantics
MIT Parser
INTENTION: ShowListWithAttributes
ATTRIBUTES: pub, Guinness
Storytelling Agents
Smarticle
DSS
Graphical Interface

Figure 3.2: The user's input to system.

Since we did not have time to implement a complete system, we decided to restrict the intentions that h2g2 would detect to the simpler ones: ShowListWithAttributes, DetailAttribute and GiveOpinion. We then specified the information that each of these intentions should carry by designing the frames shown below:

ShowListWithAttributes	
INTENTION:	ShowListWithAttributes
ATTRIBUTES:	*list of topics that the user is interested in*
USERLOC:	*location of the user when they made the query*
USERTIME:	*the time the user made the query*

DetailAttribute	
INTENTION:	DetailAttribute
DETAIL-ATTRIBUTE:	*attribute about which the user wants information*
PLACENAME:	*name of the place of interest*
USERLOC:	*location of the user when they made the query*
USERTIME:	*the time the user made the query*

GiveOpinion	
INTENTION:	GiveOpinion
SUBJECT:	*topic on which the user wants an opinion*
USERLOC:	*location of the user when they made the query*
USERTIME:	*the time the user made the query*

3.7 Domain Knowledge

Domain knowledge is not only important for answering the user's questions. As we discovered in Section 3.6.2, the same knowledge is vital for understanding those questions in the first place. The knowledge structure we choose must relate to our expectations of the system. Despite the fact that we have limited our domain to places to eat and drink, we want h2g2 to have a knowledge structure that can handle information on any subject. This will allow us to add new domains at a later stage without having to undo all our existing work.

There are many different ideas on how to structure information. The first influence on the knowledge structure of h2g2 was the knowledge grid proposed by Ashwin Ram and Kenneth Moorman in the *ISAAC* system (see section 2.5.2). This seemed attractive as it was able to detect bizarre concepts – a clear fit with the Hitch Hiker's Guide. However, this grid is designed for reading and understanding information, not for searching through it. It is also too complicated for our purposes: he gives consciousness as an example of a *Mental Agent*, but not many other concepts would fit into this category.

Most of the systems we looked at in Chapter 2 were based on Schank's original *types* of *Events, Places, People* and *Things.* Even the DARPA funded research into information extraction (see Section 2.4) uses similar categories for its Named Entities: *Person, Organisation, Location, Date* and *Currency.* However, these are more tailored to corporate information.

3.7.1 The Universal Context Model

The Universal Context Model (UCM) from TDV (see Appendix C) goes further, detailing the attributes that each category should have. The UCM also introduces two special attributes common to all types of knowledge, *TRUST* and *VERACITY.* TRUST specifies the respect given to the source of the information: whether the information is verified or whether it is stated as coming directly from the source, or having been referred. VERACITY, on the other hand, indicates whether the information is fact (the capital of England), fiction (the capital of Narnia) or speculation (the capital of Barnard 4).

This way of classifying the information could be extremely useful for us since we are going to have different story agents each with their own biases. The agents could use these attributes to determine whether they wanted to use the information. For example, an agent could choose to tell stories based on purely factual information, using the VERACITY attribute to filter the information.

In a similar way, these attributes work very well with the probabilistic user model – the DSS can quickly detect whether the user likes to live in a dreamworld by having preferences for each of the states of the VERACITY and TRUST attributes. These fit straight in with our concept of unordered attributes.

3.7.2 h2g2's Knowledge Structure

We therefore chose to use a simplified form of the UCM for our basic knowledge structure. Each domain knowledge frame has at least UCM TYPE, VERACITY and TRUST slots. The five UCM types have the following attributes (aside from VERACITY and TRUST):

- *events* have a NAME, a TIME and a LOCATION;
- *places* have a NAME and a LOCATION;
- *entities* have a NAME;

- *concepts* have a NAME; and
- *relationships* have a NAME and a list of RELATED items

By combining this structure with the language knowledge from the *Brico* KB, we came up with the following structure for the frames of our domain knowledge:

PLACE	
UCM TYPE:	Place
UCM VERACITY:	*(fact/fiction/speculation)*
UCM TRUST:	*(verified/source/referred)*
UCM NAME:	*Name of the place*
UCM LOCATION:	*GPS location*
OWNER:	*refers to a PERSON frame*
MENU:	*List of MENUITEM frames*
ATTRIBUTES:	*a list of CONCEPTs that this place is related to*

PERSON	
UCM TYPE:	Entity
UCM VERACITY:	*(fact/fiction/speculation)*
UCM TRUST:	*(verified/source/referred)*
UCM NAME:	*Name of the person*

MENUITEM	
UCM TYPE:	Relationship
UCM VERACITY:	*(fact/fiction/speculation)*
UCM TRUST:	*(verified/source/referred)*
UCM NAME:	Menu item
UCM RELATED:	*refers to the restaurant and the item*
PLACE:	*refers to the PLACE to whose menu this belongs*
ITEM:	*refers to a FOOD or DRINK frame*
PRICE:	*the price of the item at the place*

FOOD	
UCM TYPE:	Concept
UCM VERACITY:	*(fact/fiction/speculation)*
UCM TRUST:	*(verified/source/referred)*
UCM NAME:	*Food name*
INGREDIENTS:	*a list of ingredients (FOOD and DRINK frames)*
ATTRIBUTES:	*a list of CONCEPTs that this food is related to (including the CONCEPT of food itself)*

DRINK	
UCM TYPE:	Concept
UCM VERACITY:	*(fact/fiction/speculation)*
UCM TRUST:	*(verified/source/referred)*
UCM NAME:	*Drink name*
INGREDIENTS:	*a list of ingredients (FOOD and DRINK frames)*
ATTRIBUTES:	*a list of CONCEPTs that this drink is related to (including the CONCEPT of drink itself)*

REVIEW	
UCM TYPE:	Relationship
UCM VERACITY:	*(fact/fiction/speculation)*
UCM TRUST:	*(verified/source/referred)*
UCM NAME:	Review
UCM RELATED:	*lists the place and the author*
PLACE:	*refers to the PLACE reviewed*
AUTHOR:	*refers to the ENTITY who wrote the review*
TITLE:	*title of the review*
TEXT:	*text of the review*

Figure 3.3 shows how the different frames are linked together. Arrows point from frames to the frames that contain a slot in which they are referenced.

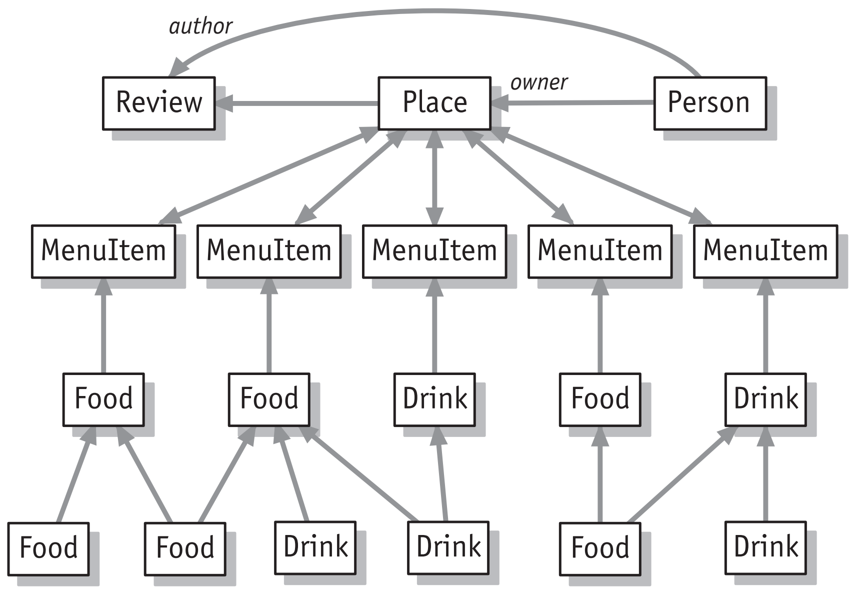


Figure 3.3: The domain frames linked together

3.8 Graphical Interface

Donald Norman's book, "The Design Of Everyday Things", provides a useful starting point for our design process. His fundamental messages in the design of all everyday things is that the designer must "make sure that 1) the user can figure out what to do 2) the user can tell what is going on." (Norman 1988a). He goes on to elaborate that a good design should:

- Make it easy to determine what actions are possible at any moment (make use of constraints).
- Make things visible, including the conceptual model of the system, the alternative actions, and the result of actions.
- Make it easy to evaluate the current state of the system.

- Follow natural mappings between intentions and the required action; between action and the resulting effect; and between the information that is visible and the interpretation of the system state.

Good design should allow the user to be intuitive, using minimal instructions and labelling. It should make use of the natural properties of people and the world. This includes how people ordinarily relate, learn and respond. Any remaining instruction or training should be needed only once; with each explanation the person should be able to say, “Of course,” or “Yes, I see,” as opposed to, “How am I going to remember that?”

3.8.1 Focusing on computer interface design

Jakob Nielsen, in his lecture “Skills For The Future” (Nielsen 1999a), examined effective interface design through critical analysis of web pages that provided user services. He emphasised the dangers of crowding a user interface so that the actual information the user needs becomes less obvious. To illustrate his point, he analysed a service called *MapQuest*[6].

This system has a interface in which users can type in an address and be shown a map of that location. The map is displayed in a field of its own. Figure 3.4 shows a screenshot from Netscape Navigator displaying the *MapQuest* site which has been colour coded according to the purpose of each area. Table 3.1 gives a description of each coloured region.

MapQuest has its own interface to enable the user to navigate the map (marked in red), it takes up almost 30% of the total space. The reason why it takes up so much space is that all of its functions are directly accessible from the page. A large proportion of users have no interest in the sophisticated functions: they simply want to view a map and perform simple actions such as zooming in on an area or moving to other section of the map. However the information they want is only displayed in 13% of the available space.

Colour	**Region type**	**Percent size**
Green	Program overhead	17.9%
Blue	Sponsor	18.64%
Red	Navigation	27.8%
Yellow	**Map area**	**13%**
White	Unused area	22.7%

Table 3.1: *Key to the colour codes in Figure 3.4.*

The *MapQuest* example does not fulfil Norman’s principles of being easy to understand and use without instruction, and is overly complicated.

[6]http://www.mapquest.com

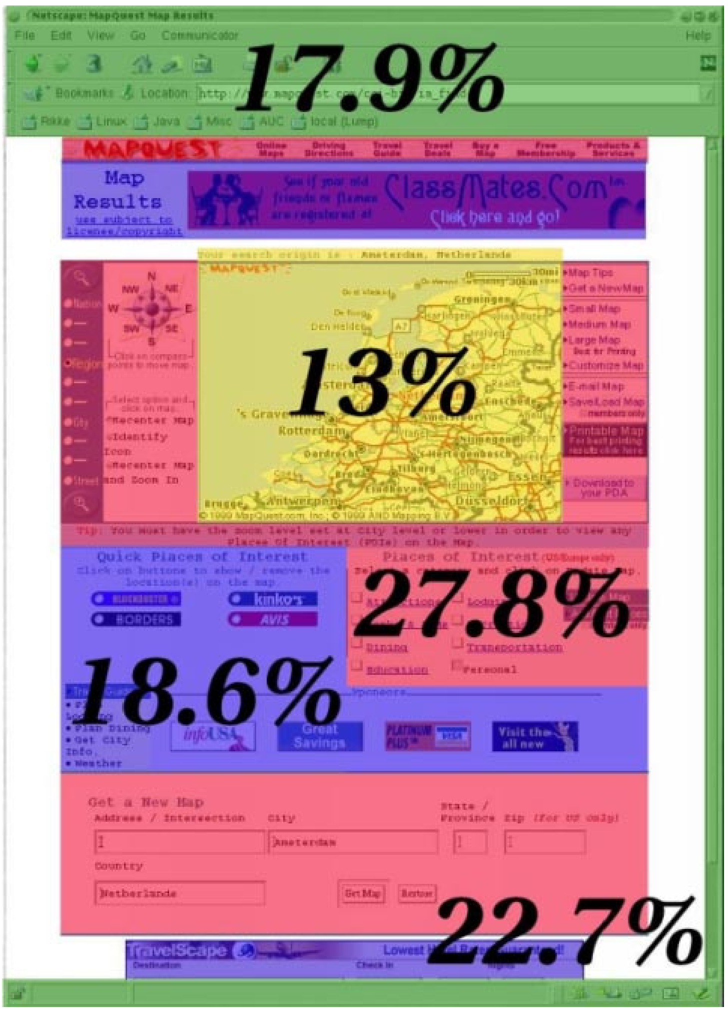


Figure 3.4: The *MapQuest* site as viewed by Netscape Navigator and colour coded according to its use of space.

3.8.2 Designing a Guide

TDV's *Principles of Guidance* (TDV & AT&T 1997) suggest that a Guide should not be limited to one method of access to the user but rather be able to scale its interface from the small display on a cellphone to a big screen TV. By being present at all times, the Guide becomes an integral part of the user's life.

Our Specification

We wish to apply Norman's principles by creating an interface in which the model of the system and possible actions are clearly visible to the user.

We also wish to make use of our experience from last semester's project (Christiansen et al. 1999) in which we designed a city guide that was limited to a mobile phone sized display. One of the mistakes we made in that project was not to inform the user when the system was busy. User tests showed that users became quickly confused if the system didn't respond, even if it was only for a short while. They often repeated the last command or started to click on buttons to get the system to react. We will therefore include a busy indicator in this project.

Following the *Principles of Guidance*, we shall make our interface as scaleable as possible. We choose to make the prototype the size of a PDA, an intermediate stage between a mobile phone or pager and a desktop machine. However, this format should not enforce any limitations on the design.

A further contribution to our design goals is the value of consistency in creating a good interface. Jenny Preece, in the book "Human-Computer Interaction" (Preece et al. 1994), explains that consistent interfaces help users learn quickly how to use the system. Consistency means that information is grouped into logical categories, and that the same information is *always* presented in the same way.

The stipulation that good design uses "natural properties of people and the world" is already incorporated into our goals for the system. Our choice of a natural language interface (English), combined with the system's capacity for learning what the user "likes", frees the user from the task of finding keywords to describe what she wants. The system decodes the request of the user, rather than the user having to decode the system. This is a basic principle of third party interaction.

We feel that *MapQuest*'s weaknesses were based around the fact that its complex nature was reflected in a complex interface. We hope to create a successful interface to a similarly complex system in a way that stands up to Nielsen's analysis of intelligent design.

3.9 Summary

At the beginning of this chapter, we started with our preliminary specifications of a Guide to Life, the Universe and Everything that:

- could provide information in a contextual setting;
- involved third party interaction through the use of software agents;
- and could provide personality-based feedback to encourage the user to trust the agents.

By examining the implications of these specifications, we ended with h2g2, a Guide to places to eat and drink that can:

1. take a user's query and interpret her *intention* using the syntax and semantics of the query and a knowledge of the possible intentions in its domain;

2. pass the intention as a frame to several storytelling agents. Each of these agents then makes a *smarticle* using a FramerD database containing the Brico language knowledge base linked to domain frames defining a fixed world of places to eat and drink;

3. rate the resulting smarticles according to the attributes in their headers using a *probabilistic user model*;

4. display no more than five of the top-rated smarticles in a *simple, scaleable graphical interface*;

5. also display the current accuracy of the user model as a *humanoid face displaying a mood* between happy and hopeless;

6. *update the user model* using Bayes' Theorem when the user selects a smarticle to read;

7. and show the user the text of the smarticle together with a representation of the authoring agent.

In the next chapter we explain how we designed and implemented this system.

Design and Implementation

Chapter 4

Deep Thought pondered for a moment.

'Tricky,' he said.

'But can you do it?' cried Loonquawl.

Deep Thought pondered this for another long moment.

Finally: 'No,' he said firmly.

Both men collapsed on to their chairs in despair.

'But I'll tell you who can,' said Deep Thought.

Douglas Adams, The Hitch Hiker's Guide to the Galaxy

4.1 Introduction

The analysis provided us with an informal specification of what the system should do. The next stage was to go into more detail and to design how the system should work.

Section 4.2 explains our design processes and gives a general overview of our system. The sections that follow provide more detail on some of the subject areas involved. These sections do not cover the whole design and implementation of the system – that would be too long to include in the report. More detail on design decisions can be found in the edited worknotes supplied in the Appendix. Our whole implementation can also be found on the group web site or on CD.

This chapter ends with an example of h2g2 in use, illustrating how the different modules communicate (Section 4.7).

4.2 System Design

The design stage borrowed heavily from the design of our previous two projects in this Master's degree, the *Intelligent Multimedia Based Pool Trainer* (Buck, Christiansen, Cohen, Ortega & Muhammed 1998) and the *Mobile Intelligent Agent* (Christiansen et al. 1999). In these projects we developed an agent-based design based on Java Remote Method Invocation (RMI). Each agent provides services as specified in its interface and can run anywhere on the network as long as one agent, the **Manager**, knows where it is. The **Manager** is responsible for starting up the system and connecting the different agents as necessary.

An added advantage of this design is that each agent only knows of the others through their interfaces and via the **Manager**. Any service can then be simulated by implementing an agent that simulated that service (perhaps involving an operator in a Wizard of Oz setup) and informing the **Manager**.

The agents in h2g2 are shown in Figure 4.1. They fall into four categories:

- **Sensor** agents take input from the user and translate it into events that other agents can understand;
- **Actuator** agents do the opposite – providing output to the user;
- **Device** agents provide basic services that other agents can make use of;
- and **Control** agents combine the services of other agents to provide more complex services.

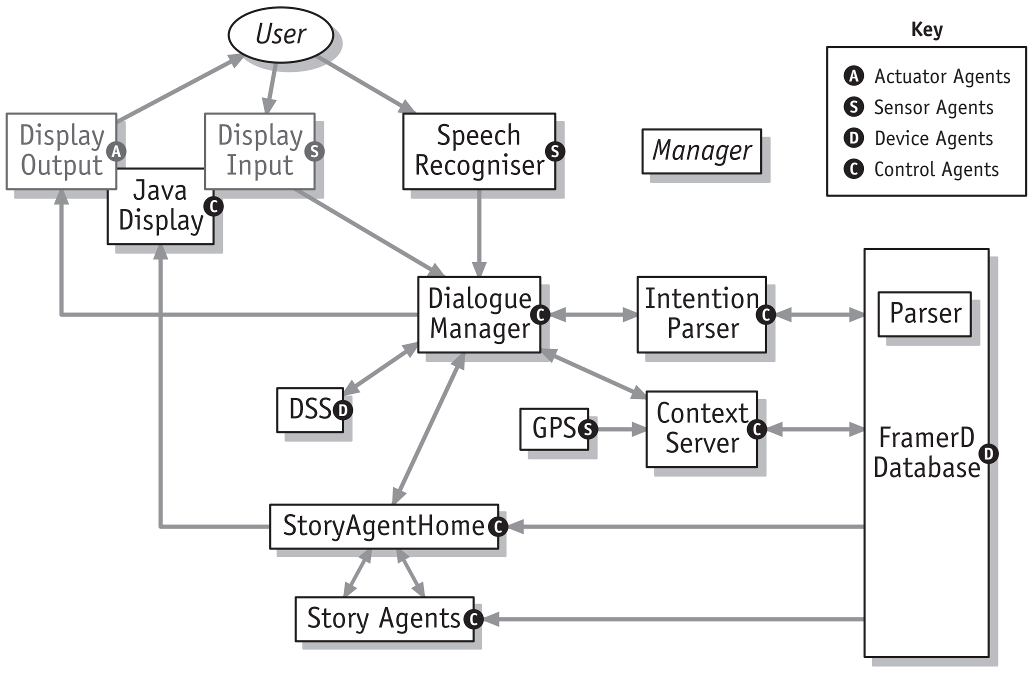


Figure 4.1: Agents in h2g2

4.2.1 Sensor and Actuator Agents

These define the appearance and communicative abilities of h2g2 to the user. As specified in Section 3.1, we use five modalities: spoken and written language, pointing with the mouse, facial expressions and the user's geographical position. The users speak or type input to the system and receive written and graphical output on the display. They can also use the mouse to choose items on the display. The system can also take input from a Global Positioning System (GPS) device to give localisation data. Due to time constraints, we did not implement any sound or speech output, though this could easily added to the system in future.

These modalities are represented in the system by the **Speech Recogniser**, **Display Input**, **Display Output** and **GPS** agents. Most of these agents are very simple, merely packaging up events from lower-level devices.

The **Speech Recogniser** agent is not based on a real generalized speech recognition system (though such systems are becoming available) but is instead simulated by an operator typing in the user's commands.

4.2.2 Device Agents

The Device Agents provide the basic capabilities of the system. The **Decision Support System** (**DSS**) agent builds up the probabilistic user model from the choices that the user makes and ranks the list of smarticles produced by the **Storytelling Agents**. See Section 4.4 for more details of the user model and how the **DSS** agent uses it.

The **FramerD Database** agent stores (and could manipulate) all four knowledge bases of the system: language knowledge (*Brico*), domain knowledge, knowledge of the user's intentions and knowledge of existing restaurant reviews. The details of how FramerD stores information can be found in Appendix B and examples of how the information is used can be found in Section 4.3.

4.2.3 Control Agents

These agents provide the more complex services of dialogue management, the parsing process and the storytelling process.

The system is controlled by a central, dictatorial **Dialogue Manager**. This is a control agent that coordinates the actions of the other agents, keeping track of the state of the system and tranferring information to and from each agent.

The **Intention Parser** is responsible for tranforming the user's utterances and spoken queries into preliminary intention frames. These are then passed to the **Context Server**, which fills in references to contextual information such as the user's position.

For details of all three of these agents, see Section 4.5.

The **Java Display** is merely a centralised place for all the graphical interface code – it implements both the **DisplayInput** and **DisplayOutput** interfaces. For more details of the design of the graphical interface, see Section 4.6.

The other control agents are the **StoryAgentHome** and the **Story Agents** themselves. The **StoryAgentHome** agent is just a collating mechanism to isolate the rest of the system from the **Story Agents**. This design allows **Story Agents** to start up (and shut down) independently of the rest of the system. The **Story Agents** are responsible for creating the smarticles in answer to the query intention frames passed to them by the **StoryAgentHome**. See Section 4.3 for details of this process and of their interaction with the **FramerD** agent.

4.3 Storytelling

4.3.1 Filling in World Knowledge

For the **Story Agents** to be able to tell stories we need to fill in the domain knowledge. Since our system is only a prototype, we chose to make a test "world" in which we place some fictive restaurants.

Three types of information are needed:

Restaurants: We need their locations, owners, some attributes (Mexican, Fastfood, etc.) and, of course, their names. We also need to know the menu from each restaurant to make the MENUITEM frames. The "Restaurant Row" web site is probably one of the biggest restaurant search sites in the world. It has over 100,000 restaurants in 47 countries (RestaurantRow 1999) so it has more information then we could ever need. It is just a question of picking out some restaurants that have both reviews and a picture of the menu.

Reviews: Since we're not using a real world scenario, we could take any restaurant review and assign it to one of our fictive restaurants. However, we tried to keep the information as real as we could, so all the reviews in the domain knowledge are "real" reviews written by different people.

Food/Drink: There are many different drink and recipe databases freely available on the web. We chose to extract information from two of these: "The Acats Internet Bar" (Saloon 1999) and "Morten's Recipe Collection" (Nielsen 1999b), both of which have more than enough information to test our system.

We added the frames partly by hand and partly using specially written Java programs that could parse pages from a specific site into specific frames. These Java programs are almost equivalent to *Amalthaea*'s information discovery agents. In a more complete implementation of h2g2 they could be developed into Sensor agents that continuously added new knowledge into the system.

4.3.2 Definition of a Smarticle

The *Story Agents* had to return their stories in a form that the **DSS** could rate and the **DisplayOutput** could display. Accordingly we had to define the structure of a smarticle and its header.

Each smarticle is a **Story Agent**'s attempt to inform the user of something. As such it is effectively an intention of that **Story Agent** in the same way as the user's query is an intention of the user. Fortunately, it is much easier to determine the system's intentions than the user's.

Firstly, we decided that every smarticle in h2g2 should mention exactly one place. This makes the **DSS**'s comparison more meaningful: instead of comparing the attributes of different stories, it compares the attributes of reviews of specific places.

We then defined three intentions for the **Story Agents**, one for each of the user's intentions that we are recognising. These three agent intentions were identical in structure, with the name of the intention providing the only difference. Our three agent intention types are:

1. Give an opinion of a suitable place for the specified attributes given the user's current location and time of query (**OpinionPlace**)
2. Give an opinion of the specified attribute at the specified place given the user's current location and time of query (**OpinionAttribute**)
3. Give an opinion of the subject at a suitable place given the user's current location and time of query (**OpinionSubject**)

The resulting frame for passing the information back to the **Dialogue Manager** is:

Agent Intention Frame	
UCM TYPE:	Relationship (between agent and restaurant)
UCM VERACITY:	*Based on that of the agent*
UCM TRUST:	*Based on the agent's source material*
UCM NAME:	Smarticle
UCM RELATED:	*AUTHOR, PLACE-NAME*
INTENTION:	*The agent's intention*
AUTHOR:	*Storytelling agent*
TITLE:	*Title of smarticle*
TEXT:	*Text of smarticle*
USERLOC:	*User location at query*
USERTIME:	*Time of query*
PLACE-NAME:	*Name of the place*
PLACE-LOC:	*Location*
ATTRIBUTES:	*The unordered attributes involved*
AVERAGEPRICE:	*The average price of any mentioned food or drink*

4.3.3 Telling a Story

The **Story Agents** do not receive the user's query intention directly from the **Dialogue Manager**. Instead, the **StoryAgentHome** agent acts as an intermediary, dispatching the user's query to all the registered **Story Agents** and collating the resulting smarticles before sending them back to the **Dialogue Manager**. The **StoryAgentHome** agent also operates a timeout procedure to make sure that the user does not have to wait too long for her answer. In this way the **Dialogue Manager** does not need to know anything about how many **Story Agents** there are nor how long they take to write a smarticle.

Each **Story Agent** is derived from the **Story Agent** abstract base class. This defines all the methods for sending and receiving messages. Each **Story Agent** subclass then has to define just one method that takes a query intention frame and returns a list of smarticles. This method is called from inside a thread that is started when the **Story Agent** receives a query from the **Story Agent Home**.

The **Story Agents** can choose exactly how they want to use the domain knowledge. Indeed, the **Dada Agent** chooses not to use it at all.

The Dada Agent

The **Dada Agent** is a very strange agent. It does tell stories, but they have nothing to do with the user's intentions. It merely uses the *Dada engine* (Bulhak 1996a) to create some feasible-sounding gibberish. The *Dada engine* is a natural language generation engine that uses recursive transition networks to build up sentences and paragraphs. The most impressive application of the *Dada engine* (and its original purpose) is the *Postmodernism generator* (Bulhak 1996b), which can generate an entire paper, complete with references, on "postmodernism, literary criticism, cultural theory and similar issues" (Bulhak 1996a).

To fill in the rest of the smarticle frame, the **Dada Agent** needs the name and location of a place to eat or drink and some attributes of that place. Since the text of the smarticle does not refer to any specific restaurant this is somewhat difficult. We chose to allow the **Dada Agent** to select a place from a fixed database of fictional places. The rest of the information in the frame would then be as fictional as the text itself. Consequently, the frame is given UCM Veracity value of *fiction* and a UCM Trust value of *source* (since the **Dada Agent** has created this information).

By building up an appropriate grammar, the **Dada Agent** could talk nonsense in the style of a restaurant review, even involving the concepts in the user's intention. The resulting smarticle would not be very useful in the practical sense but, as Bulhak noted, the results are often "quite amusing".

The HooterAgent

The **HooterAgent** is only interested in going to sports bars and does not have much intelligence of its own. It takes the user's intention and adds its own bias towards sports by including new, sports-oriented attributes. It then uses these new criteria to find a match amongst the database of previously written reviews. Any matching reviews are returned whole, with no processing applied to them at all.

The matching process involves creating a *PLACE* frame that contains all the information specified in the user's query intention. The **HooterAgent** then uses the FramerD `find-similar` function to find the *PLACE* frames in the database which have similar slots to the manufactured frame (for more details on FramerD's searching algorithms see Appendix B.1).

Once the **HooterAgent** has found a matching restaurant, it simply picks an existing review, puts its name on it, adjusts the UCM Trust attribute to be *referred* and returns it as its own work.

The Shuffling Agent

This agent works in a similar way to the **HooterAgent**. However, it does not have any bias of its own and consequently searches using the original criteria as specified in the user's query. The major difference between the two is that the **Shuffling Agent** creates a new review of a restaurant rather than returning one that has already been written.

It does this in a very simple way: by combining two reviews of similar restaurants. Instead of using just one matching restaurant, the **Shuffling Agent** finds the two best matches and uses one review of each. It then creates its own review of one of the restaurants by shuffling the sentences of two reviews together. This does not involve much intelligence and sometimes results in a review which is only marginally better than one created by the **Dada Agent**. However, since the original reviews are talking about very similar subjects, the resulting combination is usually fairly believable.

Reviews created by the **Shuffling Agent** are given a UCM Trust value of *referred* and a UCM Veracity value of *speculation* to reflect their origin.

4.4 Personalisation

The **DSS** agent is basically a parameter estimation machine. We tell it which parameters we want to estimate and the **DSS** makes better and better estimates each time the user makes a choice. As specified in Section 3.3, the parameters we want to estimate are the user's preferences for particular attributes. These are split between preferences for specified *ordered* attributes and an unlimited number of preferences for *unordered* attributes. The unordered attributes are either present or not present, whereas the ordered attributes can have scalar values.

4.4.1 The MPIA DSS

In last semester's DSS (Christiansen et al. 1999), we had fixed attributes of *distance* and *price*. All penalties were calculated in Kroner.

The parameters we evaluated were the *stinginess* and *weariness* of the user, i.e. how much weight they put on paying out money and how much weight they put on travelling. Thus we had two scales of preference, one for aversion to distance and one for aversion to cost. The DSS then estimated where along these scales the user lay.

Since both of these attributes were ordered, each of them had a conversion function. The conversion function used for distance was:

$$penalty = 10^{\frac{distance}{d10}} \tag{4.1}$$

where $d10$ was the distance that made a difference of 10Kr. This was set to 400m. This function was intended to represent the preferences of a person on foot.

The conversion function for price was simple: since all penalties were calculated in Kroner and since we only had Danish prices in the system, we just took the price.

A Problem With Distance

One problem with the conversion function for distance (Equation 4.1) is that it can easily generate numbers that are too big for the computer to handle. To solve this problem we capped the penalty: for penalties greater than a specific amount, the conversion function switches to a linear function. This also ties in with people's perception of distance: at a certain point, a place is "far away" and if another place is a a bit further it doesn't really make that much difference.

4.4.2 Applying the DSS to h2g2

Since we are using the same ordered attributes as last semester, it makes sense to use the same parameters and conversion functions. This means that all our penalties are in the local currency. All we had to do was to ensure that we could tie this in with the user model for the unordered attributes.

The unordered attributes are simpler than the ordered ones in that they don't need a conversion function. All they have is a weight. However, we had to be careful in choosing these weights since the penalties for price and distance, having no limits, could easily swamp the penalties for the unordered attributes.

Weights for new unordered attributes are estimated by taking an average of the existing ones. In the final version of h2g2, they could be adjusted later by a visit to the Pet Psychiatrist (see Section 3.4). However, we needed an initial weight to start things going.

Choosing an Initial Weight

The preferences for the unordered attributes are limited by the value of their weights – the weight is directly proportional to the maximum penalty a preference can carry. Therefore we should relate the initial weight to some well-defined point in the penalty scale.

If we consider that the purpose of h2g2 is to recommend places for the user to go (at least in the limited domain that we are considering), a maximum penalty would be high enough that the user decided to stay where she was. This would, of course, vary depending on how much she earned. However, a limit could be found by considering the most expensive place around. For example, in London this could be having dinner at the Ritz Hotel – people are just about willing to pay for the priviledge (otherwise the Ritz would go out of business).

We decided to set the initial weight to 1000 Kr, the price of a very expensive meal. For a complete system, this weight could be set to be relative to the most expensive place that h2g2 initially knew about.

4.4.3 Discretization

The **DSS** represents all its probability distributions by discretizing them into separate states (see Figure 4.2). We therefore had to consider the range of the parameters and how they were discretized. Last semester, each preference parameter could have one of three possible values: 1, 3 or 10 (a simplified logarithmic scale).

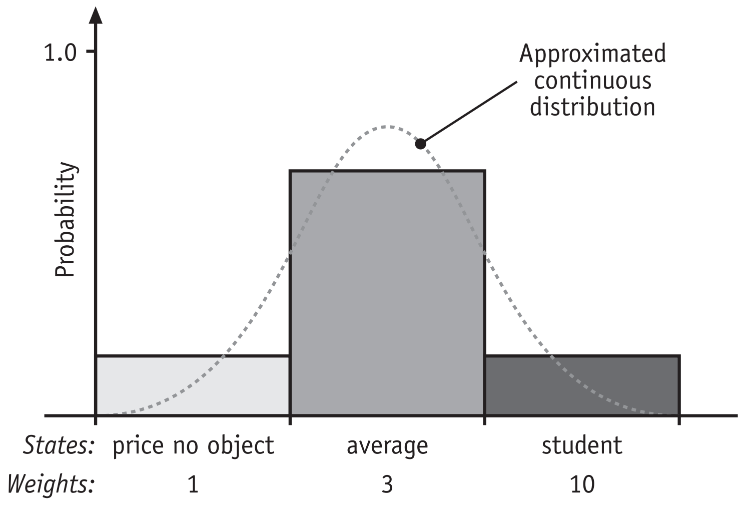


Figure 4.2: A sample distribution for the *stinginess* parameter

Especially considering the unordered attributes, it made sense to add a 0 state – a special case in which the user doesn't care at all about that specific attribute.

To simplify the calculations, we can also set each preference range to be from 0 to 1. Thus the resulting possible values are 0, 0.1, 0.3 and 1.0.

4.4.4 Combinatorial Explosion

While the structure of the DSS in Section 3.3.2 works very well in theory, it contains a major problem. With four states for each attribute, the computer grinds to a halt calculating more than a few attributes worth of preferences. This is due to the combinatorial explosion of the joint distribution.

Even with only two states per unordered attribute (UA), each new UA will double the number of states in the joint distribution. With 10 different attributes (not that many) we therefore have 1024 states to deal with. And this number must be multiplied by the number of combinations of states for the ordered attributes.

Our solution is to combine all the unordered attributes into one distribution. This does not by itself solve our problem – if this distribution has two states per UA we still get the same explosion when we calculate the joint distribution. However, if we can make do with one state per UA, then the growth slows from exponential to linear.

What would a distribution with one state per UA mean?

The DSS is estimating parameters and each state should represent a parameter value. Thus this distribution represents the UAs that the user dislikes. The value we are estimating is the user's least favourite UA.

How does this new distribution calculate penalties?

In the previous design for the DSS, each UA had its own distribution with several possible values. The penalty was then the expected value multiplied by a weight. If all the UAs are combined into one distribution we have to calculate the penalty differently.

If we consider a smarticle with UAs A, B and C, then one way we could calculate its penalty would be to sum the probabilities of each of the specified UAs being the user's least favourite attribute. This would get us the total probability of the smarticle containing the user's least favourite attribute. This probability could then be converted to a penalty by multiplying it by a weight as before.

This penalty measure makes sense in two ways. Firstly, if a smarticle contains all the attributes we know about, then the probability that it contains the user's least favourite attribute will be 1. The penalty will then be the maximum weight. Secondly, if we add states by assigning them an average probability, the distribution will continue to be uniform until we start making choices. With this penalty calculation, a uniform distribution will give each attribute the same penalty.

One disadvantage is that smarticles with more attributes will be penalized until the distribution becomes less uniform. To solve this problem, we can look at the attributes one at a time and calculate their penalties separately. The penalties are then combined together by averaging them as in the original DSS design.

As an example, a smarticle with four UAs A, B, C and D has its penalty calculated as follows:

The penalty for each UA is the probability that it is the user's least favourite UA multiplied by the weight assigned to that attribute. The combined penalty is then the average of each of these.

4.5 The Dialogue System

This section describes the agents responsible for the natural language dialogue with the user. As mentioned in Section 4.2, the **Speech Recogniser** merely transcribes the user's spoken words and passes them onto the **Dialogue Manager** for processing. This agent is the central control of our system. It uses the **Intention Parser** and **Context Server** to determine the user's intention and then acts accordingly by sending events to the **StoryAgentHome** and **DisplayOutput** agents.

4.5.1 The Dialogue Manager

According to our specification (Section 3.1), h2g2 should be accessible by just voice. Time did not permit us to implement any voice output but the user still held a dialogue with the system by receiving graphical responses to her utterances.

Last semester's project (Christiansen et al. 1999) used a very structured dialogue in which the initiative remained with the system at most points. While this made the dialogue easier to implement, we wanted h2g2 to be more flexible. From our intention analysis in Section 3.6.3 we had discovered the kinds of questions that people could ask h2g2 and this allowed us to use natural language for the actual queries (see Section 4.5.2 below). However, we also use natural language to control the interface.

Apart from the query itself, there are two other commands involved in controlling the graphical interface: the choice of one of the offered smarticles and a cancel (or undo) command to allow the user to change her mind. These commands are easy to interpret when the user controlled the system using a pointer: a different event is sent to the **Dialogue Manager** depending on which area the user pointed to. Spoken commands are harder to tell apart, especially as we allow natural language. For example, the command "Show me the first restaurant in the list!" could be misinterpreted as a query rather than a choice.

To avoid this kind of misinterpretation, we split the dialogue into four separate states related to what was displayed on the screen. These states are shown in Figure 4.3:

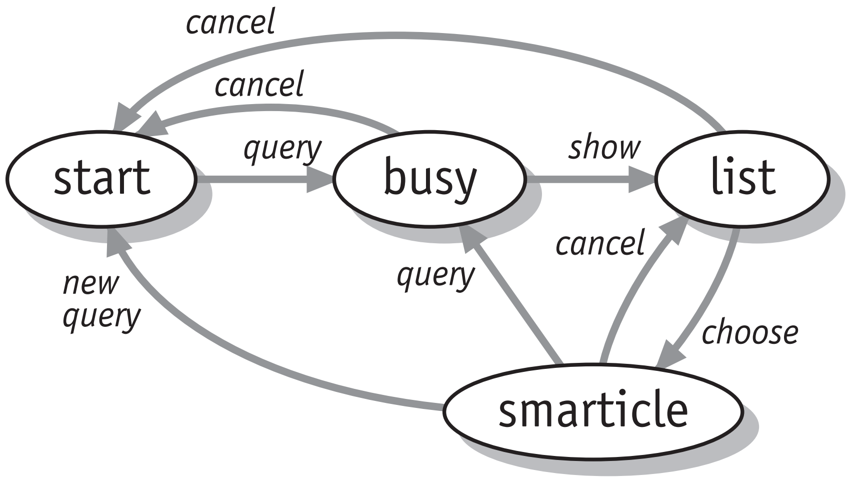


Figure 4.3: Dialogue Manager states

- a *start* state in which the system is waiting for a query from the user;
- a *list* state in which it is displaying a list of different smarticles for the user to choose;
- a *smarticle* state in which it is presenting an individual smarticle to the user;
- and a *busy* state used to indicate that a query is being processed and that the user should wait for an answer. The **Dialogue Manager** exits this state when it is ready to show its answer to the user.

This last state ensures that the user always has feedback as to what the system is doing, even when it does not have any new information to display.

This separation of the dialogue into separate states ensures that the **Dialogue Manager** never has to differentiate between a choice command and a query. However, it still has to perform some disambiguation – in the *smarticle* state the user can ask another query, cancel the choice she has just made or say something that h2g2 was not equipped to understand (such as “Make me a cup of coffee”).

The interpretation of the user’s intention is performed by the **Intention Parser**.

4.5.2 The Intention Parser

The intention processing is split into three main tasks:

1. Detecting if the user has told h2g2 to stop its current processing and go back to the previous state. This is especially important if other commands are possible in the current state.
2. Determining which smarticle the user wishes to select from a spoken choice command.
3. Discovering the query intention type from a spoken or typed query and extracting the information needed for the query intention frame.

Detecting Cancel Commands

The **Intention Parser** detects spoken cancel commands using the simple technique of word-spotting. It looks for specific words or phrases such as “cancel”, “stop” or “go back” in the user’s speech and returns a Cancel intention to the **Dialogue Manager** when one is detected.

However, in cases where other commands with similar wording are possible, the **Intention Parser** checks for the other commands *first*. This is done to avoid wrongly categorising such sentences as “Can I cancel a reservation at Natalie’s?”

Parsing Choice Commands

Each time the **Intention Parser** is given a choice command to interpret, it is also provided with the list of smarticles displayed on the screen at the time. This makes it possible for h2g2 to understand commands such as “Show me the review by Ford Prefect” by examining each smarticle in turn and looking for a match with the user’s utterance.

However, for our prototype implementation, we limited the capability of this function to only detecting utterances that mention the number of the smarticle in the list. This is done in much the same way as interpreting Cancel commands: a Choice command is recognised if the user’s speech includes specific phrases and the number of one the items in the list. This detects commands such as “Show me review three” or simply “Number two, please.”

Recognising Query Intentions

The **Intention Parser** detects query intentions by comparing the user's query with templates for the selected intention types. We designed these templates by examinining the example questions that we had originally used to identify the possible user intentions (see Section 3.6.3).

The first part of the templates is specific syntax that is used for specific intentions. For example, DetailAttribute queries always include phrases such as "Do I need", "Can I", or "What is". However, this is not enough by itself to uniquely identify the intention types of all queries since some intentions share phrases: while "What is the menu at *Natalie's*?" is a DetailAttribute intention, "What is a Calzone?" is a GiveOpinion intention.

To disambiguate such queries we have to take into account their semantics. In the context of h2g2, this means examining the queries for subjects that are contained in its domain knowledge. For example, the two queries above can be disambiguated by the fact that DetailAttribute intentions always mention a specific place name (*Natalie's*) and an attribute (the menu). h2g2 has to know about both of these items to successfully detect the intention.

The resulting template for the DetailAttribute intention is as follows:

- Can I <attribute> <place-name>?
- Can you <attribute> <place-name>?
- Does <place-name> <attribute>?
- Do I need <attribute> <place-name>?
- Do you have any suggestions as to <attribute> <place-name>?
- Tell me about <attribute> <place-name>.
- What is/are <attribute> <place-name>?

For the templates of the other two intentions we detect, see Worknote D.10.

Filling in the Intention Frames

Once the intention type had been detected, the **Intention Parser** has to fill in an intention frame to pass back to the **Dialogue Manager**. In some cases, this is directly linked to the detection of the intention. For instance, detecting a GiveOpinion intention sometimes requires looking up the subject of the query as a food type in the domain knowledge. If a matching food type is found then that is the value that should be passed back in the GiveOpinion intention frame.

In other cases, the information for the frame has to be extracted from the query. Information extraction from arbitrary text is a very difficult problem, but our task is made much easier by the fact that we have already identified the user's intention. This not only constrains the ways that the user might provide their information, but also the kind of information they would provide.

The MIT Parser

We also simplified our task by using the MIT parser. This is able to tag every word in the query with its grammatical class and root. Thus for ShowListWithAttributes intentions, we can determine the attributes the user is interested in by picking out the adjectives, nouns and important verbs from the query and returning the roots of each word.

For example, the query "Where is a good children's restaurant?" is split up as follows:

```
#(
   ( "Where " ("where")
       #("Where" PRONOUN "where") )

   ( "is a good children's restaurant?" ("restaurant")
       #("is" BE-VERB) #("a" DETERMINER) #("good" ADJECTIVE)
       #("children's" POSSESSIVE "child") #("restaurant" NOUN) )
)
```

The **Intention Parser** then returns a ShowListWithAttributes frame that looks like this:

	Drink
INTENTION:	ShowListWithAttributes
ATTRIBUTES:	child, good, restaurant
USERLOC:	<empty>
USERTIME:	time of query

The USERLOC slot is filled in later by the **Context Server**.

Communication with the Dialogue Manager

The **Dialogue Manager** does not accept any input from the user while the **Intention Parser** is processing. This is a design decision made to avoid confusion in the dialogue and to simplify the **Dialogue Manager**. Since the time taken to interpret the user's intention was about one second (even when the MIT parser is involved) this does not have a noticeable effect on the dialogue.

In any dialogue system it is very important that the system can tell if the user has said something out of context and can respond appropriately. The **Intention Parser** does not attempt to understand all the user's utterances. When it cannot make an intention frame from the user's utterance, it returns a null response to the **Dialogue Manager**. In this case h2g2 displays the presentation agent with a question mark thought bubble, indicating that it cannot understand. Since graphical input is always available, we did not implement a complex dialogue-based error handling process for the interface commands. However, more complex dialogue behaviour could be added at a later date through the use of the **Context Server**.

4.5.3 The Context Server

The **Context Server** contains a complete history of the dialogue, kept up-to-date by the **Dialogue Manager**, and uses it to resolve any references that the user makes in her utterances.

In the current version of h2g2, the **Context Server** is only aware of the user's geographical location, using the **GPS** agent[1] and completely ignores the dialogue history. However, our design allows future implementations to understand complex contextual references and to interpret intentions in *sequence* as well as in isolation.

Cancel and Choice commands can be completely understood by the **Intention Parser** since they involved very little external context. However, the user's queries can depend heavily on contextual information. For instance, a user might want more information on a restaurant that h2g2 mentioned the day before and refer to it using this time reference instead of its name ("Show me the menu of that Italian restaurant you told me about yesterday"). In this case, the **Intention Parser** would only be able to identify the attribute of the user's **DetailAttribute** intention and the **Context Server** would be responsible for filling the place name slot of the intention frame.

Detecting sequences of intentions can be instrumental in understanding the user's goals. Intentions such as **repetition** and **elaboration** just cannot be detected without some concept of dialogue history. We did not look for such discourse related intentions in our brief survey of possible questions (Appendix A) in order to save time. However, a fully implemented **Context Server** should be capable of reinterpreting the output of the **Intention Parser**, potentially returning a new intention type to the **Dialogue Manager** representing the user's goal in the context of the dialogue history.

4.6 Graphical Interface Design

In designing the graphical user interface (GUI), we chose to create one general interface structure to be used throughout the system (see Figure 4.4. The purpose of this is to keep the interface simple and consistent. Important features such as the input fields not only remain in place but keep their semantics similar. We have also positioned our fields in the logical order in which they will be used, following the natural movement of the eyes when reading: i.e. from top left to bottom right.

4.6.1 The Prompt

The minimal instruction needed in each state is provided at the top left corner of each page in the form of a prompt. The text explains to the user what she needs to do with the further information in the display.

The five states of the display and their respective prompts (see Figure 4.7) are closely related to the four states of the **Dialogue Manager** (see Figure 4.3). Table 4.1 relates these two figures and details the prompt in each state.

Screen 4 in Figure 4.7 has no direct comparison with a state in the **Dialogue Manager**. It occurs when the **Dialogue Manager** has been unable to interpret the user's intention and remains in the *start* state instead of beginning to process the user's query in the *busy* state.

[1]Since we are using a fictional world, we simulate the GPS information.

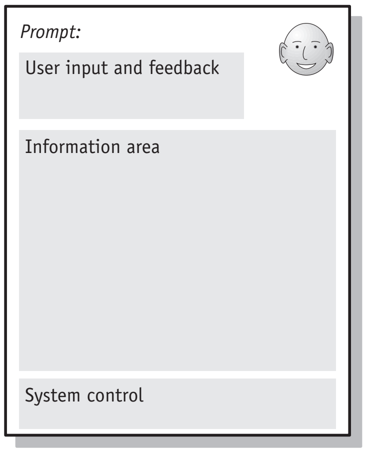


Figure 4.4: Basic structure of the graphical interface.

GUI Screen	GUI Prompt	Dialogue Manager state
1	Please enter your question:	*Start*
2	You asked:	*Busy*
3	Please choose an answer:	*List*
5	<Agent> recommends:	*Smarticle*

Table 4.1: *The relation between the states of the GUI and the* Dialogue Manager.

4.6.2 Representation of the Personality

As specified in Section 3.4, the presentation agent should reflect the "mood" of the system: from happy, through confused, to hopeless depending on the state of the user model. In order to provide a realistic representation of these emotions we went to the source of emotive animation, Walt Disney. "The Illusion of Life" (Thomas & Johnston 1995) describes many ways to breathe life into an animated character. An example from the book is shown in Figure 4.5: three ways of animating a look the right.

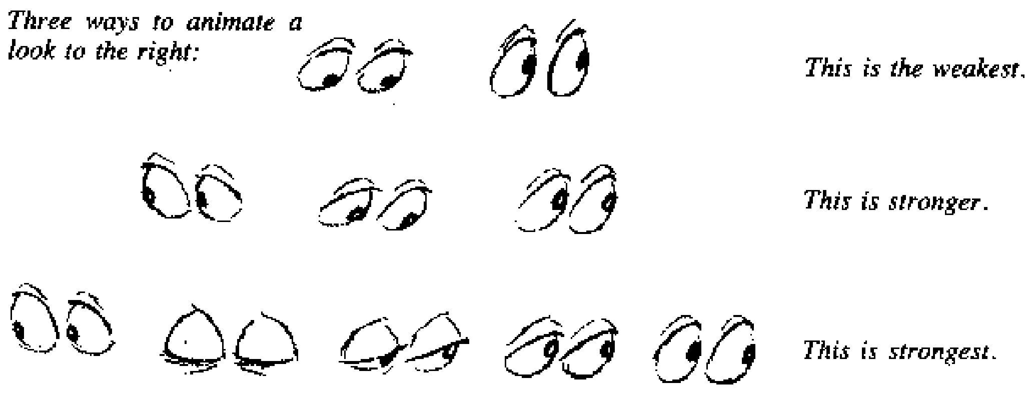


Figure 4.5: How to be the next Bugs Bunny.

However, this book was in many ways too complicated and so we settled for less sophisticated animation. "1000+ Pictures for Teachers to Copy" (Wright 1996) describes how to convey different emotions with simple line drawings. It details how to represent sequences of emotions, for example from happy to sad, and which facial features to move in order to achieve different effects. Using this book as our guide, the resulting expressions are shown in Figure 4.6.

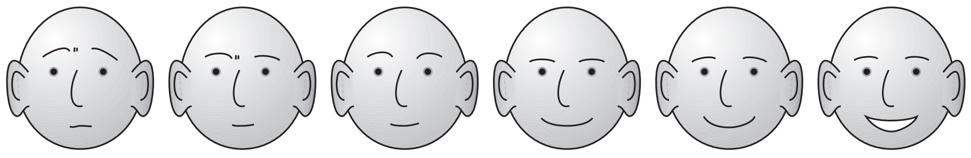


Figure 4.6: Shows the different moods of the presentation agent, from "Hopeless" through "Confused" to "Happy".

The *Presentation Agent*, the personality-based representation of the system, is the "voice" of the prompts and the "ears" of the system. It therefore sits right next to the prompt and the *text field*.

4.6.3 User Input and Feedback

The text field has three purposes, all of which are extremely similar. Firstly, it provides feedback from the *Speech Recogniser*, giving the user direct feedback as to exactly what the system recognised. Secondly, if the user is unable to use speech input, she can write her question directly into the text field using keyboard or pen input. The third purpose is to display the title of the smarticle that the user has chosen. Since the user selects which smarticle she wants to read by choosing a title, this purpose is also providing feedback of the user's previous action.

4.6.4 Information Area

The central part of the interface, the *result field*, is used by the system to answer the user's questions. In the two cases (Screen 1 and Screen 4) in which the user asks h2g2 a question, it has the words "Don't Panic" inscribed in large friendly letters so that the user feels relaxed, ready and in control. In the other cases the result field is either used by the Presentation Agent to show a list of the smarticle titles (Screen 3) or it is used by a **Story Agent** to present its own smarticle (Screen 5).

4.6.5 System Control

The Presentation Agent never leaves the display. When a **Story Agent** takes its place while it tells its story, the Presentation Agent moves to the bottom of the screen to indicate that it is still listening to the user even though another agent is presenting.

This area at the bottom of the screen is the last area that the user's eye falls upon. As such it is consistently used for system controls that the user can operate once they have seen the rest of the display.

The *Cancel* button is available whenever the user may wish to undo her last command. This allows h2g2 to be *explorable* – the user can play with the system knowing that very little they do will have a permanent effect.

The *New Query* button is only available when the user is reading a smarticle that she has requested. It lies next to the Presentation Agent at the bottom of the screen to indicate that this is the way back to asking a new query.

4.7 The System in Action

This section gives a brief description of how the agents work together to answer the user's question.

4.7.1 Welcome

When the system is started, the user is presented with a welcome screen (Screen 1 in 4.7) controlled by the **Display Output** agent. She can then either speak or type her question to the guide.

Spoken language is transcribed by the **Speech Recogniser** or the **Display Input** agent takes the user's typewritten text directly. This text is then sent to the **Dialogue Manager**.

4.7.2 Processing

The **Dialogue Manager** first tells the **Display Output** to inform the user that it is busy processing (Screen 2 in 4.7). It then tries to interpret the intention of her query using the **Intention Parser**. This uses a combination of word-spotting and the syntactic and semantic processing of the MIT parser to fill in an *intention frame*.

If the **Intention Parser** detects the user's intention, it returns the *intention frame* to the **Dialogue Manager** which passes the frame onto the **Context Server**. This is responsible for filling in any empty slots in the frame, such as the user's location in a query for a "nearby" place. Finally, the *contextualized intention frame* is sent to the **StoryAgentHome** for distribution to the **Story Agents**.

Each **Story Agent** is free to operate in its own way, using the information from all four knowledge bases in the **FramerD** database. Once they have generated their *smarticles*, they pass them back to the **Dialogue Manager**, which then uses the **DSS** to assign them grades according to the user model.

4.7.3 Choice

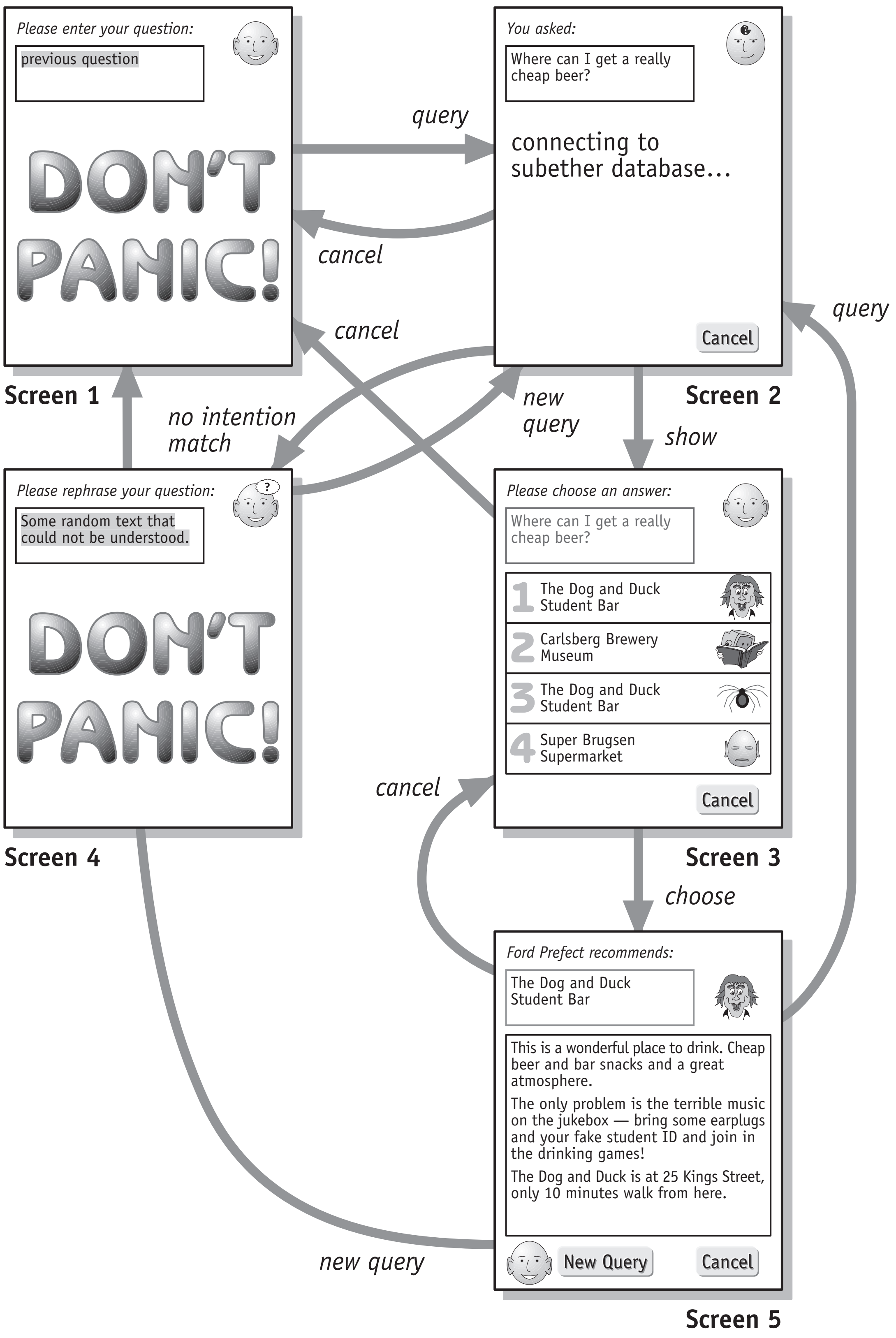
The **Dialogue Manager** then puts all the *smarticles* together in a sorted list and sends them off to the **Display Output** agent (Screen 3 in 4.7). The user can then either choose one of the *smarticles*, or cancel her question.

A spoken choice is processed in a similar way to the original question – the **Dialogue Manager** passes it to the **Intention Parser** and waits for a choice *intention frame*. However, if the user uses the pointer to indicate her choice the **Display Input** sends a choice *intention frame* directly to the **Dialogue Manager**, avoiding any processing by the **Intention Parser**.

4.7.4 Smarticle

Once the user's choice has been evaluated, the **Dialogue Manager** sends the relevant *smarticle* to the **Display Output**. This uses the reference to the authoring agent in the *smarticle* to display the graphics for the agent's face at the top of the smarticle (Screen 5 in 4.7).

At this point, the user can cancel and go back to the list of *smarticles*, can ask another question straight away or can use the pointer to indicate that she wants to ask another question.



Please enter your question:

previous question

You asked:

Where can I get a really cheap beer?

connecting to subether database...

Please rephrase your question:

Some random text that could not be understood.

Please choose an answer:

Where can I get a really cheap beer?

The Dog and Duck Student Bar

Carlsberg Brewery Museum

The Dog and Duck Student Bar

Super Brugsen Supermarket

Ford Prefect recommends:

The Dog and Duck Student Bar

This is a wonderful place to drink. Cheap beer and bar snacks and a great atmosphere.

The only problem is the terrible music on the jukebox — bring some earplugs and your fake student ID and join in the drinking games!

The Dog and Duck is at 25 Kings Street, only 10 minutes walk from here.

Figure 4.7: h2g2: what the user sees

Evaluation

Chapter 5

Many speak of the legendary and gigantic Starship *Titanic*, a majestic and luxurious cruise-liner launched from the great ship-building asteroid complexes of Artrifactovol some hundreds of years ago, and with good reason.

...

The designers and engineers decided, in their innocence, to build a prototype Improbability Field into it, which was meant, supposedly, to ensure that it was Infinitely Improbable that anything would ever go wrong with any part of the ship.

...

The Starship *Titanic* was a monstrously pretty sight as it lay beached like a silver Arcturan Megavoidwhale amongst the laser-lit tracery of its construction gantries, a brilliant crowd of pins and needles of light against the deep interstellar blackness; but when launched, it did not even manage to complete its very first radio message – an SOS – before undergoing a sudden and gratuitous total existence failure.

Douglas Adams, Life, The Universe And Everything

5.1 Individual Module Tests

The agents in our system were tested individually to ensure that they fulfilled the requirements we specified in the Analysis (Chapter 3). The Java language made this process easier by allowing each agent class to define its own testing code.

5.2 Storytelling

We tested the separate **Story Agents** using the user intention frame shown below. Working backwards through the processes of the **Intention Parser**, this frame could have been generated from a query like, "Show me a restaurant that serves steaks."

ShowListWithAttributes	
INTENTION:	"ShowListWithAttributes"
ATTRIBUTES:	restaurant,steaks
USERLOC:	(0, 0)
USERTIME:	1999/05/14 13:00:00.000

The following sections show the smarticles that the three implemented **Story Agents** wrote as a result.

5.2.1 The Dada Agent

This agent ignores the information from the user's intention and simply makes up its own story. We did not develop our own "restaurant review" Dada grammar but instead used *spout*, one of the grammars that comes with the *Dada engine*. The resulting smarticle is shown below.

Dada Smarticle	
UCM TYPE:	*Relationship*
UCM VERACITY:	*Speculation*
UCM TRUST:	*Source*
UCM NAME:	"Smarticle"
UCM RELATED:	AUTHOR, PLACE-NAME
INTENTION:	"OpinionPlace"
AUTHOR:	"Dada Agent"
TITLE:	"The entity called "Bob"."
TEXT:	"The entity called "Bob" of Jehovah-1 can be "sold" to humanity if it is of utmost concern to the fate of the planet. The cosmic joke known to Emergentiles and Rewardians alike fucks that which can be, figuratively, 'smoked in "Bob's" Pipe' until it is of utmost concern to the fate of the planet. The absolute stupidity seen once a Millennium was implanted in our minds only to be part of the Conspiracy."
USERLOC:	(0, 0)
USERTIME:	1999/05/14 13:00:00.000
PLACE-NAME:	"Milliways: The Restaurant at the End of the Universe"
PLACE-LOC:	(1000, 1000)
ATTRIBUTES:	future, impossible
AVERAGEPRICE:	1

5.2.2 The HooterAgent

The **HooterAgent** likes sport. Consequently, it takes the user's intention and adds "sport" to the ATTRIBUTES slot. It then uses the `find-frames` FramerD function (see Appendix B.1) to find PLACE frames that have similar values in their ATTRIBUTES slot. When this process is applied to the user intention frame defined above, it currently results in the two PLACE frames summarised below:

Summarised Place Frame	
UCM NAME:	"Hooters Of Manchester"
ATTRIBUTES:	party, bar, girl, steak, hamburger, sport, television

Summarised Place Frame	
UCM NAME:	"Bandanas Cafe"
ATTRIBUTES:	american, hamburger, mexican, seafood, steak music, party, bar, wheelchair access, sport live music, television

Once it has a list of places that match the query, the **HooterAgent** looks for a review of one of them and repackages it as its own. Thus the resulting smarticle frame is:

HooterAgent Smarticle	
UCM TYPE:	*Relationship*
UCM VERACITY:	*Fact*
UCM TRUST:	*Referred*
UCM NAME:	"Smarticle"
UCM RELATED:	AUTHOR, PLACE-NAME
INTENTION:	"OpinionPlace"
AUTHOR:	"HooterAgent"
TITLE:	"Hooters of Manchester"
TEXT:	"This is one of my favorite Hooters. The people were nice and I had a lot of fun when I visited there. A bartender noticed that I was from Denver, and made sure that I had a great time while I was there with my cousin. I bought a shirt and all the girls signed them. From what they said on my shirt, it was noticeable that they took time out to find out where I was from and a little about me. They did not put normal notes but took time to put something unique on my shirt. In St. Louis, this is the best one I have visited!"
USERLOC:	(0, 0)
USERTIME:	1999/05/14 13:00:00.000
PLACE-NAME:	"Hooters of Manchester"
PLACE-LOC:	(250, 120)
ATTRIBUTES:	party, bar, girl, steak, hamburger, sport, television
AVERAGEPRICE:	70

5.2.3 The Shuffling Agent

This agent does not have any bias, it simply uses the attributes specified in the user intention frame to look for matching PLACE frames. As with the **HooterAgent**, once it finds a match it looks for matching reviews. However, instead of just returning a previously written review it combines the text from two of the reviews to make its own.

This process is shown in the three frames below. The first two are summaries of the reviews found using the `find-frames` function:

Summarised Review Frame	
AUTHOR:	"Ian Hill"
TITLE:	"Hooters of Manchester"
TEXT:	"This is one of my favorite Hooters. The people were nice and I had a lot of fun when I visited there. A bartender noticed that I was from Denver, and made sure that I had a great time while I was there with my cousin. I bought a shirt and all the girls signed it. From what they said on my shirt, it was noticeable that they took time out to find out where I was from and a little about me. They did not put normal notes but took time to put something unique on my shirt. In St. Louis, this is the best one I have visited!"

Summarised Review Frame	
AUTHOR:	"Lynn Hubbard"
TITLE:	"Bandanas Cafe at the Bullpen"
TEXT:	"My wife and I went to Bandanas when it was first opened and had a great meal. The waitress was very nice and the food was plenty and very good. I had a pepper steak that was as good as any that I have had. I highly recommend this restaurant, it's small and quaint but very good."

The third frame is the resulting smarticle. As mentioned in Section 4.3.3, this is not a very effective method of creating reviews but it often produces results which are almost believable:

Shuffling Agent Smarticle	
UCM TYPE:	*Relationship*
UCM VERACITY:	*Speculation*
UCM TRUST:	*Referred*
UCM NAME:	"Smarticle"
UCM RELATED:	AUTHOR, PLACE-NAME
INTENTION:	"OpinionPlace"
AUTHOR:	"Shuffling Agent"
TITLE:	"Bandanas Cafe at the Bullpen"
TEXT:	"This is one of my favorite Hooters. The waitress was very nice and the food was plenty and very good. I bought a shirt and all the girls signed it. I had a pepper steak that was as good as any that I have had. They did not put normal notes but took time to put something unique on my shirt."
USERLOC:	(0, 0)
USERTIME:	1999/05/14 13:00:00.000
PLACE-NAME:	"Bandanas Cafe"
PLACE-LOC:	(140, 302)
ATTRIBUTES:	american, hamburger, mexican, seafood, steak music, party, bar, wheelchair access, sport live music, television
AVERAGEPRICE:	90

5.3 Decision Support System

The **DSS** was tested to see if it could successfully adapt the user model to capture the criteria used to make a series of choices. This test was more complicated than the previous ones since we had to be able to provide the **DSS** with a list of smarticles each time a choice was to be made.

To make this test easier, we implemented a new Java class called the **SmarticleDB**. This was a simple database that could read smarticle definitions from text files, store them and subsequently retrieve them by keywords or by name. The **SmarticleDB** meant that we could temporarily cut the **Intention Parser** and the **Story Agents** out of the system and reduce h2g2 to a simple keyword search engine.

The **DSS** test thus involved making a list of keyword vectors representing the user's queries and combining it with a list of choices. By tailoring the choices to the smarticles selected by the keyword vectors we could emulate a user making choices using any criteria we wanted. For example, we could emulate a user who always chose smarticles written by a particular agent, or who always chose the smarticle describing the nearest place.

We used several such manufactured user profiles to evaluate the **DSS**. The results from two such tests are shown below. In the first test the user profile is biased towards closer places, whereas in the second profile the bias is towards factual information. These tests used the same query each time, but varied the choices.

The initial penalties awarded to the results of the query are shown in the table below. The smarticles referred to places that were increasingly further away, but decreasingly expensive. Smarticles 0 and 10 had UCM Veracity values of *fact*, whereas the rest all were *fiction*. These smarticles were chosen because they had quite similar penalties and thus provide a very clear example of the effect of changes in the user model.

Initial Penalties	
Smarticle 10:	137.65538
Smarticle 9:	141.68110
Smarticle 8:	145.53967
Smarticle 7:	149.07353
Smarticle 6:	152.08879
Smarticle 1:	153.06388
Smarticle 0:	153.21547
Smarticle 2:	154.33213
Smarticle 5:	154.33869
Smarticle 3:	155.48844
Smarticle 4:	155.57370

5.3.1 Test 1: Bias towards closer places

After eight choices of the smarticles referring to closer places (smarticles 0, 1 and 2), the penalties of the test smarticles were as follows:

Penalties after choosing the nearest places	
Smarticle 0:	140.83460
Smarticle 1:	154.97811
Smarticle 2:	168.72309
Smarticle 3:	182.15166
Smarticle 4:	192.55286
Smarticle 10:	198.45535
Smarticle 5:	199.22226
Smarticle 6:	203.02302
Smarticle 9:	204.08582
Smarticle 7:	204.66046
Smarticle 8:	204.83205

This change was due to the distribution of the distance preference tightening around the "lazy" end. The **DSS** succeeded in adapting to a user who did not like travelling.

5.3.2 Test 2: Bias towards factual information

After eight choices of the smarticles referring to factual information (smarticles 0 and 10), the penalties of the test smarticles were as follows:

Penalties after choosing the factual smarticles	
Smarticle 10:	29.56418
Smarticle 0:	38.15752
Smarticle 9:	157.77787
Smarticle 8:	160.86571
Smarticle 1:	163.62151
Smarticle 7:	163.63928
Smarticle 2:	165.49560
Smarticle 6:	165.91093
Smarticle 3:	167.26134
Smarticle 5:	167.44183
Smarticle 4:	167.99049

This change was entirely due to a change in the distribution of the unordered attributes preference. After the eight choices had been made, *fact* was clearly the user's favourite unordered attribute. However, the change that had the most effect on the penalties awarded to the smarticles was that *fiction* was by far the least favourite unordered attribute.

This "negative preference" occurred due to the mechanisms of the **DSS**. The user who chooses factual information is also *avoiding* fictional information. When a similar test was run in which the rest of the smarticles were equally divided between UCM Veracity values of *fiction* and *speculation* the effect was not nearly as pronounced. Instead of the penalties for the non-factual smarticles rising to around 160, they were evenly spread between 80 and 140.

5.4 Intention Parsing

We tested the **Intention Parser** by asking it to parse the list of questions from our survey in Section 3.6.3 as query or cancel intentions. Since this list contained intentions which we had chosen not to detect, we expected the **Intention Parser** to misrecognise some of the intentions in the questions. Some of the results from this test are shown below:

Example 1:
User says: "I want a vegetarian lasagne. Who delivers?"
This is should match a **ShowListWithAttributes** intention.

By using the templates specified in Appendix D.10, the **Intention Parser** finds that the sentence appears to match a **ShowListWithAttributes** intention. It therefore looks for attributes from the rest of the sentence using the MIT Parser:

```
((()
  #(("I " ("i" SUBJECT "want") #("I" PRONOUN "i"))
    ("want " ("want") #("want" VERB))
    ("a vegetarian lasagne. " ("lasagne" OBJECT "want") #("a" DETERMINER)
     #("vegetarian" ADJECTIVE) #("lasagne" NOUN))))
 (()
  #(("Who " ("who" SUBJECT "deliver") #("Who" PRONOUN "who"))
    ("delivers?" ("deliver") #("delivers" VERB "deliver")))))
```

The resulting intention frame is therefore:

ShowListWithAttributes	
INTENTION:	ShowListWithAttributes
ATTRIBUTES:	vegetarian, lasange, deliver
USERLOC:	*empty – filled in by the Context Server*
USERTIME:	*the time the user made the query*

Example 2:
User says: "I want to take my Grandma to a restaurant she'd like. What can you suggest?"
This is a **ShowListForSituation** intention, but the **Intention Parser** is not set up to detect these. Consequently, it matches it with the template for the **ShowListWithAttributes** intention with the result shown below:

ShowListWithAttributes	
INTENTION:	ShowListWithAttributes
ATTRIBUTES:	grandma, restaurant
USERLOC:	*empty*
USERTIME:	*the time the user made the query*

This is clearly *not* what the user wanted. The system fails to interpret the user's question and consequently produces smarticles about grandmothers rather than genteel restaurants.

Example 3:
User says:"What do you know about *Carlsberg Classic*?"
This should match a **GiveOpinion** intention.

The **Intention Parser** matches this with the template for the **GiveOpinion** intention and looks in the domain knowledge for foods or drinks that match "Carlsberg Classic". When it finds a match it returns the following frame:

GiveOpinion	
INTENTION:	GiveOpinion
SUBJECT:	Carlsberg Classic
USERLOC:	*empty*
USERTIME:	*the time the user made the query*

This test did not evaluate the **Intention Parser**'s ability to correctly identify and interpret Choice or Cancel commands, though it did evaluate its ability to differentiate Queries from Cancel commands.

The test also failed to evaluate whether the **Intention Parser** could interpret the intentions of new queries that were not in our original list. However, until the system was integrated there was no easy way to do this.

5.4.1 GUI evaluation

In the section on Graphical interface design (see page 60) we evaluated the *MapQuest* user interface according to Jacob Nielsen's criteria. This involved analysing how well it made use of the available space (see Figure 3.4 on page 43) to make a service avalible to the user.

We performed the same analysis on our own user interface to see how we fared. The result can be seen in Figure 5.1. This shows that the main area takes up 51.7% of the space while feedback and control uses 32.5% (the two other areas combined). It is hard to compare our system with *MapQuest* because many of our displays (Screens 1-4 in Figure 4.7) are used to find a "story" that answers the user's question. This corresponds to the form users have to fill to find the right map in *MapQuest*). It's only when the h2g2 is displaying a story or a list of stories (Screens 3 and 5) that we use 51.7% of the display to show what the user wants.

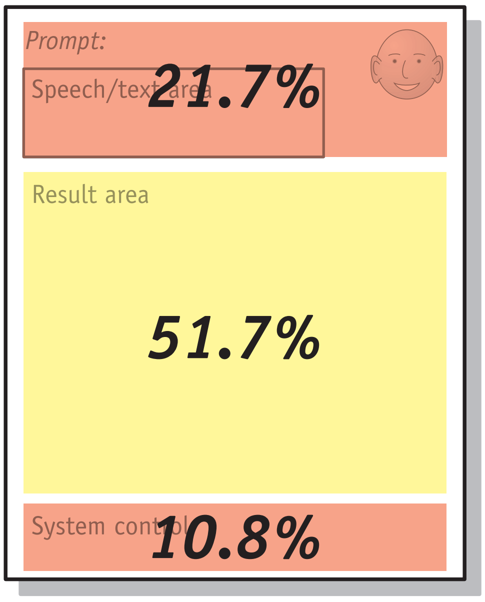


Figure 5.1: The /hhgg interface colour coded to indicate its use of space.

5.5 System Evaluation

Having successfully tested each module, we conducted an integration test and ensured that the system worked when put together. This was made simple by the use of the **Manager** agent, which can control which of the other agents to start up by use of a text-based configuration file. Once the system was completely running, we asked it some questions from our survey. Each of the modules communicated with each other as expected and h2g2 returned smarticles in a few seconds. The first question took much longer since the MIT parser initialises its storage space the first time it is called.

5.5.1 Multiple Users

An unexpected advantage of our design is that h2g2 is immediately ready to be deployed as a multi-user system. The **FramerD** and **Intention Parser** agents and all the **Story Agents** are entirely user-independent and can be run separately, catering for as many users as the machines they are running on can cope with.

The other agents contain user-specific information and so must be duplicated for each user, but each user's **Manager** agent only needs to know the location of the shared agents to be able to run a complete system. The flexibility of the design means that even the user-specific agents need not run on the same machine. If h2g2 is to be used on a handheld device, only the **JavaDisplay** (and the **SpeechRecogniser** if it is used) has to run on the device itself – all the other agents can be run on more powerful, not so mobile machines and be accessed over a wireless network.

5.6 Summary

Due to our short timescale we did not test the system further by conducting a user test, or even by asking it queries that were not in our original list. However, simply by putting the system together we have achieved our goals as set out in the Introduction.

Providing information in a contextual setting: h2g2 tells stories (albeit simple ones) in response to the user's queries rather than giving a list of unrelated information.

Third party interaction: the user interacts with the system via the presentation agent, delegating the search to the system rather than specifying keywords herself.

Providing personality-based feedback: the state of the system's probabilistic model of the user is conveyed back to the user by an animated humanoid face.

Discussion

CHAPTER 6

They gazed at God's Final message in wonderment, and were slowly and ineffably filled with a great sense of peace, and of final and complete understanding.

Fenchurch sighed. 'Yes,' she said, 'that was it.'

Douglas Adams, So Long And Thanks For All The Fish

While we were finishing this project, The Digital Village (TDV) launched the first part of its *h2g2* project – a web site that aims to be the Earth Edition of the Hitch Hiker's Guide to the Galaxy (TDV 1999). The main purpose of this version of the Guide is to create an online community in which users (or *researchers*, as they are termed) submit their own articles. A team of editors at TDV then accept or reject the new entries.

The *h2g2* web site provides opinions galore and, if successful, will soon have information about a wide range of subjects. However, it is still based around keyword searching and cannot combine information from varied sources. As such it is less sophisticated than *Ask Jeeves* (Ask Jeeves 1997), the search engine we mentioned in Section 3.6.1. *Ask Jeeves* has a similar collection of pages containing information (culled from the rest of the web rather than on its own site), but allows the user to say what they want using natural language queries.

In terms of the criteria we established in our Introduction (Section 1.4) (presenting information from varied sources in a *relevant* and *easily understandable* way), neither site has succeeded. *Ask Jeeves* makes a good attempt at relevancy by allowing users to *delegate* their search tasks to the system: they can specify "what" they want rather than detailing "how" to find it. The *h2g2* web site tries to make subjects easier to comprehend by allowing users to comment on anything on the site, providing more and more contextual information as the site develops.

More importantly, neither site has any concept of personalisation or of any kind of feedback from the system. Although the *h2g2* site allows the user to create her own pages, these have no effect on subsequent queries. Both sites are still a long way from the kind of system envisaged by Apple's *Knowledge Navigator*.

6.1 The Missing Sciences

In 1993, Kai Fu Lee made a presentation of Apple's *Knowledge Navigator* (Lee 1993) in which he detailed its central concepts. These concepts were remarkably similar to our own, though more centred around spoken dialogue interfaces[1]: multimodal interaction, collaboration, multi-media, intelligent assistance and speech-centric interface. He also listed the "missing sciences" – research areas that he saw as central to the development of such systems:

[1]He was, after all, making a keynote at Eurospeech '93.

1. *Knowledge representation:* "a rich, deep knowledge base that includes task knowledge, domain knowledge, user preferences and common-sense knowledge."

2. *User adaptibility:* Lee looked at this area from the perspective of speech systems: adapting the dialogue to the style and manner of the user. However, this overlaps with the knowledge of the user's preferences – the whole system should adapt itself to the user's style and manner.

3. *Improved speech recognition:* better accuracy and robustness; recognising "real" speech as opposed to read or segmented speech; rejection of sounds not meant for the system and assigning confidence values to those that are recognised; and integrating natural language processing with speech recognition.

4. *Improved speech synthesis:* achieving voices with "naturalness (voices that one is willing to listen to for hours), prosody (based on front-end natural language processing) and flexibility (multi-voice, multi-lingual, multi-style)."

5. *Task independence:* "technologies that perform adequately irrespective of the domain." Though Lee was talking about speech recognition and synthesis, he raises an important point about intelligent systems in general that we mentioned in Section 3.6.2.

6. *User interface:* combining multiple modalities in a transparent user interface that allows delegation and provides assistance. Lee defines a transparent user interface as one that "empowers users to reach directly into the task domain and accomplish their goals as effectively and naturally as possible."

Whilst Lee approached the area from a strongly speech-oriented angle, it is interesting to see how well our system and those of others have fared in each of these "missing sciences" and also what might be possible in the future.

6.2 Knowledge Representation

Thanks to the use of *FramerD* and its *Brico KB*, the knowledge representation in h2g2 rests on firm foundations. Our task (knowledge of what the user is looking for and how the system should answer) is encoded in the intention frames of both the **Intention Parser** and the **Story Agents** and also stored in the language knowledge of the *Brico KB*. We have a structure for the domain knowledge, though at present it only contains information about a few restaurants. The **DSS** provides information about user preferences, though not to the extent of TDV's *h2g2* or even of the *Knowledge Navigator*[2]. And due to *Brico*'s inclusion of the *CYC ontology*, all of this knowledge is given some kind of basis in common-sense.

[2]Lee mentions that the *Knowledge Navigator* should know whether the user would want to wake someone to ask them a question!

6.2.1 Storytelling

Unfortunately, we have not had time to take full advantage of this richness and depth of knowledge. h2g2 is only a very basic storyteller at the moment, capable of repeating stories it has already been told, spouting complete nonsense or making an unintelligent mishmash of different stories. Our system could be compared to Schank's librarian or grandfather models (Schank 1990):

The librarian listens to what you are looking for and returns you a book which suits your needs (much like h2g2 repeating stories it has already been told). This model can seem intelligent if your query is for an idea and not a specific book *and* if the librarian has a large selection of books to choose from. h2g2 can interpret complex queries but at present it only has a small library to look through.

This model also suffers from the limitations of its library in another way – if what you are looking for is only covered by a combination of several books, the librarian can only give you all the relevant books to look through yourself. He cannot make up a new story that summarises just the information you want to know. This is the limitation we noticed in the Introduction that applies to all the existing Internet search engines.

The grandfather is based on the kind of grandfather who seems to tell the same story over and over again. In an extension of Schank's metaphor, an even older grandfather might start to babble: talking nonsense or switching from story to story as he lost his memory. This would be similar to the second and third ways that h2g2 can currently tell stories – you can tell that some knowledge has been involved in the answer, but the story has got garbled somewhere along the way.

6.2.2 Stories of the Future

h2g2's storytelling process could be improved in several ways. Some simple ideas could be use Ken Haase's analogy matching (Haase 1995) or to combine different points of view.

The MIT parser that we use to process both the restaurant reviews and the user's queries was designed to be able to draw analogies between sentences. As such, it could be used to swap sentences from one review with similar-meaning sentences in other reviews. This would enable h2g2 to make up "new" stories while still retaining the flow of meaning (as long as the analogies were not stretched too far).

Combining different points of view would necessitate having several reviews of the same place. A **Story Agent** could then quote several reviews in one smarticle. However, as TDV's Principles of Guidance say, "guidance is subjective", and so such a **Story Agent** should make sure to give its own opinion as well, even if this is only to the extent of saying which review it "believes in" most.

6.2.3 Stories from Scripts

In Section 3.2 we put forward the suggestion that the stories told by h2g2 could be generated from scripts consisting of frames put together in a causal manner. Though this would seem to be the ideal solution it has turned out to be beyond our capabilities in this project. However, research into natural language generation is at a stage where this is nearly possible:

- *SPUD* (Stone, Webber & Doran 1997), the natural language generation engine behind MIT's *Rea*, takes a set of communicative goals and outputs a sentence that attempts to achieve them.

- The *MAGIC* system at Columbia University (McKeown, Shaw, Pan, Jordan & Allen 1997) uses clause aggregation to build up complex sentences from information stored in a database.

Unfortunately, the types of stories generated so far are a long way even from the unedited user-contributed pages on TDV's *h2g2* web site. As with speech synthesis, the best results so far are achieved by chopping and changing pre-recorded material.

6.2.4 Back to Statistics

A completely different approach would be to take the user's intention and use a statistical methods-based system to find an answer. Such hybrid systems already exist: *Ask Jeeves* not only uses its own database of previously answered questions to answer a query, but also conducts a meta-search amongst several leading keyword-based search engines. An example of a similar application in h2g2 could be for a **Story Agent** to use a Cartia *ThemeScape* map (see Section 2.4) as an alternative knowledge representation structure on which to base its story.

This flexibility of storytelling methods is made possible by the design of our system – the **Story Agents** can operate completely independently from the rest of the system. All that is required of them is that they know the user intention types and can output equivalent smarticles (or agent intentions).

6.3 User Adaptibility

The Decision Support System in h2g2 captures a limited sense of the user's interests and adapts the system accordingly. As can be seen in the Evaluation (Section 5.3), h2g2 builds up profiles of users based on both the questions they ask and the answers they choose. It does this reasonably quickly without the user having to fill in a questionnaire before she starts and without her having to provide relevance feedback for every story that the system creates.

The limitations of the user model that we noted in last semester's report (Christiansen et al. 1999) still apply. Firstly, it assumes that each of the user's preferences are independent of each other. This rules out the possibility of detecting such links as "only goes to pubs that serve Guinness". Secondly, it assumes that the preferences are independent of the subject matter, ruling out inferences such as "prefers noisy pubs but quiet restaurants". Thirdly, it has no concept of time – people behave differently in the morning than in the evening.

This last limitation can be partially removed by having different preferences for different times of day, but the first two are inherent to the workings of the **DSS**.

6.3.1 Comparison with Other Systems

It is hard to make a direct comparison of the capabilities of the h2g2 user model with those of other systems. However, the limitations noted above mean that the **DSS** could never aspire to such in-depth knowledge of the user as required by TDV's vision of the Guide (see Section 2.1). Nevertheless, it achieves a greater level of personalisation than *PLUM* or *fishWrap* (in both of which the user profile is fixed after an initial questionnaire).

Moukas's *Amalthaea* is probably more capable – it can discover links between interests – but it is certainly slower. Moukas reports (Moukas 1996) that his system took 60 generations (that's 60 queries) to reach an 80% fitness level against a manufactured user profile.

6.3.2 Artificial Life

The extra capabilities of *Amalthaea* could be added to h2g2 by implementing Story Agents based on genetic algorithms and artificial life (ALife). This was briefly mentioned in our Analysis (Section 3.5) and could be achieved in several ways. One way would be to have a single Story Agent that contained within it a whole population of ALife-based agents. The Story Agent would return the stories of the fittest ALife agents, just like in *Amalthaea*, and would give them credit based on how well the stories scored according to the **DSS** and whether the user chose them. Such an agent could customise itself to an individual user, providing better and better smarticles as its internal population evolved.

Using the inbuilt multi-user capabilities of our system (see Section 5.5.1), similar agents could customise themselves to groups of users. This would not only allow them to evolve faster but would give them a wider scope, since they would be answering more queries. Yet another way of involving ALife would be to have groups of Story Agents working together to collectively explore and summarise the information in the knowledge base.

6.3.3 User Psychology

The Psychiatrist feature introduced in Section 3.4 could be capable of more than adjusting the weights of the different preferences. A process called *repertory grid analysis*, invented by George Kelly (Kelly 1955), is an automated way of discovering exactly which criteria people use for their decisions. The process involves building up a table of items to choose between and some initial criteria to distinguish between them. By following through the steps, users can discover which criteria are more or less important to them and which criteria are similar or equivalent. Since most of this process is automatic, it can be (and has been (Boxer 1985)) incorporated into software. This, surely, is the ultimate in user adaptability – discovering the way that the user thinks, even though the user may be unaware of it herself!

6.4 Speech Recognition

Our project did not involve any speech recognition – the **Speech Recogniser** agent was simulated by an operator typing in the user's utterances. Nevertheless, the six years since Lee gave his keynote have seen great advances in speech recognition technology, to the extent that off-the-shelf systems like IBM's *ViaVoice* are accurate and robust enough to replace the operator (after a training period with the individual user).

Even with one of the products, the h2g2 system would not be able to cope with "real" speech. Firstly, the available products require a high-quality sound source – they usually come with a head-mounted, noise-cancelling microphone. This would not be feasible for a handheld device, though it would be fine for a desktop version. Current speech recognition products also find it very difficult to accurately transcribe telephone speech, and even more difficult to recognise words spoken on a mobile telephone. Secondly, the **Intention Parser** is set up to interpret text without interjections such as "er" or "um" and could easily get confused.

Other problems are related to the intentions themselves. In order to complete the system in the time available, we decided to ignore two of the five intentions that we abstracted from our sample questions. This means that in most cases h2g2 successfully detects and interprets ShowListWithAttributes, DetailAttribute and GiveOpinion intentions. However, it misrecognises ShowListForSituation and Generalise intentions, classing them as ShowList-WithAttributes and DetailAttribute intentions respectively. This is not such a problem as it may seem – false interpretations still produce an answer which takes into account the ideas expressed in the question. It is as if h2g2 misheard or misunderstood the user's question.

Mc Kevitt (Mc Kevitt 1991) has shown that recognising *sequences* of intentions is extremely important for dialogue-based systems in communicating with the user. Though we designed h2g2 in such a way that intention sequences could have been detected and acted on (see Section 4.5.3), we did not fully implement the **Context Server**, leaving it only capable of filling in the user's geographical location. Our survey of possible questions was also limited in that we did not look for sequences of intentions or discourse related intentions. As a result, intention types such as elaboration go completely unnoticed by our system.

Other systems such as *OSCON* (Mc Kevitt 1991) and *GALAXY* (Zue, Glass, Hazen, Hetherington, Lau & Seneff 1999) are already capable of dealing with sequences of intentions and with contextual references. However, Lee's idea that natural language should become an integral part of speech recognition itself is still in the early stages of research.

6.5 Speech Synthesis

h2g2 uses no speech synthesis at present. This is not a limitation of the design, however – it would be easy to implement a **SpeechOutput** actuator agent that the **Dialogue Manager** could use in tandem with the **DisplayOutput** agent.

Adding voice output would make h2g2 accessible from telephones without displays, though the dialogue system would have to be made much more robust to compensate for the lack of visual feedback. At present h2g2 does not provide more information when the user is making errors, relying on the simplicity of its graphical interface.

The WebGALAXY system (Zue et al. 1999) offers an interesting combination. It provides an extension to the solely telephone-based GALAXY system: a web-based interface to give users extra feedback (as long as they can still talk on a phone line when connected to the Internet).

Speech synthesis would also bring h2g2 closer to the original Guide of the Hitch Hiker's Guide to the Galaxy radio series. This spoke with the "still quiet measured voice" of Peter

Jones. Such a well-chosen voice adds character and respectability to even the weirdest articles, but this kind of speech synthesis is still a long way off from today's technology.

Looking further into the future, multi-voice and multi-style capabilities would allow h2g2 to distinguish between storytelling agents by using different voices as well as different graphical representations.

6.6 Task Independence

We can add new knowledge to h2g2 very easily, as long as it is within the domain we have defined. Adding new restaurants, foods, reviews and anything else for which we have defined frames is simply a matter of storing the frames in the database. Similarly, the system is quite happy answering questions within the defined domain.

However, adding a new area of knowledge is a completely different matter. Not only would we have to define new frames for the new subjects, but we would also have to analyse the possible user intentions over this new area. The resulting intentions would (hopefully) be similar to our specialised eating and drinking guidance intentions, but we would almost definitely have to alter the detection methods used in the **Intention Parser**. As mentioned in Section 3.6.2, a Guide that knows about Life, the Universe and Everything is simply not possible with today's technology.

Even the most successful applications of artificial intelligence today operate on limited domains. For example, Ram's powerful story understanding systems – *AQUA*, *PIES* and *ISAAC* – can grasp complex concepts such as irony and the suspension of disbelief, but they are severely limited to their domains. ISAAC, for instance, only reads three science fiction stories and synopses of *Star Trek: The Original Series* episodes, and we know what a warped sense of the world Trekkies have!

On the other hand, the most successful applications of artificial intelligence today operate on limited domains. The spoken dialogue systems developed by Victor Zue and his team at MIT's Spoken Language Systems Group work extremely well and are in constant use by the general public (and are presumably gathering income for MIT as they go!). Maybe it's just a question of time before all the necessary subject areas are covered by separate systems.

6.7 User Interface

The h2g2 system successfully integrates pen, speech and positional input with graphical output, resulting in a scaleable interface suitable for desktop and mobile applications. The intelligence in the system allows the interface to be very simple and the user to accomplish her goals effectively.

Two areas in which other systems have an advantage over h2g2 are synchronisation between the different modalities and facial expression.

6.7.1 Synchronising Modalities

h2g2 does not synchronise information between its different input modalities. A user cannot point to an item in a list and say, "Show me me *that* one." However, some systems already have this capability. Aalborg University's *IntelliMedia Workbench* (Brœndsted et al. 1998) can understand such deictic references and MIT's *Rea* (Cassell et al. 1999) can interpret conversational gestures such as raising a hand to interrupt. Both systems can also make deictic references of their own, pointing to objects at the same time as referring to them in speech.

Not only is this synchronisation essential to any system that attempts to hold a conversation with the user, but such capabilities make it easier for the system to comprehend the user in noisy environments. Human conversation is full of redundant information and systems should be able to take advantage of this. For example, someone might point to the item at the top of a list *and* say, "Show me the one at the top."

6.7.2 Facial Expressions

This is a relatively new modality (at least for computer systems!) that is in rapid development. Apple's *Knowledge Navigator* used a primitive cartoon butler with little animation and h2g2 takes a similar approach. A simple, static face is adequate to convey the necessary information (the state of the user model), but much more can be achieved.

Recent developments take these ideas to a new dimension, literally. The MPEG-4 standard (MPEG 1998) contains specifications for parameterised facial expressions. This allows any system with a model of the human head to display synchronised lip motions, eye movements and even emotional reactions. One such model has been developed by a Master's student at Aalborg and Edinburgh Universities: Buck's *SimpleFace* for Java (Buck 1999) is a 3D human face directly accessible from Java. Now any applet can express itself in fully-animated three dimensions.

A system that combined the synchronisation of modalities and a three-dimensional animated face would allow the user to accomplish her goals in a much more natural way.

Conclusion

Chapter 7

‘All right,’ said Deep Thought. ‘The Answer to the Great Question...’

‘Yes...!’

‘Of Life, the Universe and Everything...’ said Deep Thought.

‘Yes...!’

‘Is...’ said Deep Thought, and paused.

‘Yes...!’

‘Is...’

‘Yes...!!!...?’

‘Forty-two,’ said Deep Thought, with infinite majesty and calm.

Douglas Adams, The Hitch Hiker’s Guide to the Galaxy

We succeeded in our aim of designing and constructing a system that combined the three concepts of:

- providing information in a contextual setting;
- changing the interaction metaphor from direct to third party interaction;
- and providing personality-based feedback to encourage the user to trust that third party.

The resulting system, h2g2, is a prototype of the *Hitch Hiker’s Guide to the Galaxy* limited to the domain of places to eat and drink. It interprets the user’s intentions in order to give her guidance in the form of a story. h2g2 uses frames as a flexible way of representing and accessing its knowledge, and builds up a probabilistic model of the user’s preferences. The state of the user model is conveyed to the user via the facial expressions of an animated presentation agent. The information is displayed in an interface that is scaleable from a desktop computer to a mobile phone.

We successfully tested each module of h2g2 individually and then completed an integration test of the entire system.

Having made a comprehensive survey of similar research projects, we looked to the future through the eyes of Kai Fu Lee. Lee was a member of Apple’s Advanced Technology Group who in 1993 listed the research areas that he believed were needed to achieve Apple’s vision of the Knowledge Navigator. This is a video presentation of a fictional conversational computer very much like the Hitch Hiker’s Guide.

Lee’s “missing sciences” were:

1. Knowledge representation
2. User adaptibility
3. Improved speech recognition
4. Improved speech synthesis
5. Task independence
6. User interface

We discovered that there has been a great deal of development in these fields since 1993, but that there is still a lot more to be accomplished. Compared to many of the big research projects in these fields, h2g2 is a simple prototype. However, we noticed that the most effective projects were the ones that integrated all of Lee's "missing sciences" and h2g2 is firmly oriented in this direction.

We also found that, though there are many successful applications that operate over limited domains, a Guide to Life, the Universe and Everything is still in the realm of science fiction. Fortunately, h2g2 has the words "Don't Panic" inscribed in large friendly letters on its cover.

Bibliography

Adams, D. (1979). *The Hitch Hiker's Guide to the Galaxy*, Barker, London.

Ask Jeeves (1997). Ask jeeves, http://www.ask.com.

Bender, W. (1995). News in the future, *Proceedings of the IS&T 48th Annual Conference*, Washington, DC.

Bloch, A. (1963). *Men Are Different*, MacMillan Publishing Co., New York.

Boxer, P. (1985). Judging the quality of development: the subject of knowing, *in* D. Boud, R. Keogh & D. Walker (eds), *Reflection: Turning Experience into Learning*, Kogan Page.

Brœndsted, T., Dalsgaard, P., Larsen, L. B., Manthey, M., Mc Kevitt, P., Moeslund, T. B. & Olesen, K. G. (1998). A platform for developing intelligent multimedia applications, *Technical Report R-98-1004*, Center for PersonKommunikation (CPK), Aalborg University, Aalborg, DK.

Buck, J. (1999). Buck's SimpleFace for Java 1.0, http://www.kom.auc.dk/~jbuc93/thesis/simpleface/.

Buck, J., Christiansen, S. B., Cohen, A., Ortega, S. & Muhammed, S. (1998). Intelligent multimedia based pool trainer, *Technical report*, IMM Group 870, CPK, Aalborg University, Denmark.

Bulhak, A. C. (1996a). On the simulation of postmodernism and mental debility using recursive transition networks, *Technical Report 96/264*, Dept Computer Science, Monash Univ., Melbourne, Australia.

Bulhak, A. C. (1996b). The postmodernism generator, http://www.csse.monash.edu.au/cgi-bin/postmodern.

Cartia (1999). Newsmaps.com, http://www.newsmaps.com.

Cassell, J., Bickmore, T., Billinghurst, M., Campbell, L., Chang, K., Vilhjalmsson, H. & Yan, H. (1999). Embodiment in conversational interfaces: Rea, *CHI '99 Proceedings*, ACM Press, Pittsburgh, PA.

Chesnais, P. R., Mucklo, M. J. & Sheena, J. A. (1995). The fishwrap personalized news system, *Second International Workshop on Community Networking Integrating Multimedia Services to the Home*, IEEE.

Chodorow, M., Fellbaum, C., Johnson-Laird, P., Landes, S., Miller, G. A., Tengi, R., Wakefield, P., Ziskind, L. & Lipski, A. (1999). Wordnet – a lexical database for English, http://www.cogsci.princeton.edu/~wn/.

Christiansen, S. B., Cohen, A., Nielsen, T. D. & Ortega, S. (1999). Mobile intelligent agent with a multimodal interface, *Technical report*, IMM Group 971, CPK, Aalborg University, Denmark.

Cycorp (1999). Cyc ontology, http://www.cyc.com.

Elo, S. (1995). *PLUM: Contextualizing news for communities through augmentation*, Master's thesis, MIT, Cambridge, MA.

Gross, N., Judge, P. C., Port, O. & Wildstrom, S. H. (1998). Let's talk!, *Business Week* p. 60.

Haase, K. (1995). Analogy in the large, *Proceedings IJCAI-95*.

Haase, K. (1996). FramerD: Representing knowledge in the large, *IBM System Journal* **35**: 381–397.

IMM (1999). Study trip to Boston and New York, http://www.kom.auc.dk/Boston99.

Jacobson, J., Comiskey, B., Turner, C., Albert, J. & Tsao, P. (1997). The last book, *IBM Systems Journal* **36**(3).

Kelly, G. (1955). *The Psychology of Personal Constructs*, W.W. Norton & Co.

Koda, T. (1996). Agents with faces: The effect of personification, *5th IEEE International Workshop on Robot and Human Communication*.

Lee, K.-F. (1993). The conversational computer: An Apple perspective, *Proceedings of the 3rd European Conference on Speech Communication and Technology*, Vol. 2, ESCA, Berlin, pp. 1377–1384.

Maes, P. (1994). Agents that reduce work and information overload, *Communications of the ACM* **37**(7).

Mc Kevitt, P. (1991). *Analysing coherence of intention in natural language dialogue*, PhD thesis, Department of Computer Science, University of Exeter, England.

McKeown, K., Shaw, J., Pan, S., Jordan, D. & Allen, B. (1997). Language generation for multimedia healthcare briefings, *Proceedings of Fifth Conference on Applied Natural Language Processing*, pp. 277–282.

Minsky, M. (1974). A framework for representing knowledge, *MIT-AI Laboratory Memo 306* .

Moorman, K. & Ram, A. (1994a). A functional theory of creative reading, *Technical Report GIT-CC-94/01*, College of Computing, Georgia Institute of Technology, Atlanta, Georgia.

Moorman, K. & Ram, A. (1994b). Integrating creativity and reading: A functional approach, *Proceedings of the 16th Annual Cognitive Science Conference*, Atlanta, GA.

Moorman, K. & Ram, A. (1996). The role of ontology in creative understanding, *Proceedings of the 18th Annual Cognitive Science Conference*, San Diego, CA.

Moukas, A. (1996). Amalthaea: Information discovery and filtering using a multiagent evolving ecosystem, *Proceedings of the Conference on Practical Application of Intelligent Agents and Multi-Agent Technology*, London.

MPEG (1998). MPEG-4, ISO/IEC 14496, http://drogo.cselt.stet.it/mpeg/.

Nielsen, J. (1999a). Skills for the future, Lecture.

Nielsen, M. K. (1999b). Morten's recipe collection, http://sunsite.auc.dk/recipes/english.

Norman, D. A. (1988a). *The design of everyday things*, Doubleday.

Norman, D. A. (1988b). *The Psychology of Everyday Things*, Basic Books.

Pearl, J. (1987). Bayesian decision methods, *Encyclopedia of AI*, John Wiley & Sons, Inc., pp. 48–56.

Postel, J. (1981). Rfc 973: Tcp, *Technical report*, DARPA. available from http://www.rfc-editor.org/.

Preece, J. et al. (1994). *Human-Computer Interaction*, Addison Wesley.

Ram, A. (1989). *Question-driven understanding: An integrated theory of story understanding, memory and learning*, PhD thesis, Yale University, New Haven, CT.

Ram, A. (1990). Knowledge goals: A theory of interestingness, *Proceedings of the 12th Annual Conference of the Cognitive Science Society*.

Ram, A. (1991). Interest-based information filtering and extraction in natural language understanding systems, *Bellcore Workshop on High-Performance Information Filtering*.

RestaurantRow (1999). Restaurant row, http://www.restaurantrow.com/.

Saloon, J. (1999). Acats internet bar page, http://www.epact.se/acats.

Schank, R. (1986). Language and memory, *in* B. Grosz, K. Sparck Jones & B. Lynn Webber (eds), *Readings in Natural Language Processing*, Morgan Kaufmann Publishers, Inc., Los Altos, CA, pp. 171–191.

Schank, R. (1990). *Tell Me A Story: a new look at real and artificial memory*, Charles Scribner's Sons.

Schank, R. & Abelson, R. (1977). *Scripts, Plans, Goals and Understanding*, Lawrence Erlbaum Associates, Inc., Hillsdale, NJ.

Schank, R. & Cleary, C. (1994). *Engines for Education*, Lawrence Erlbaum Associates. http://www.ils.nwu.edu/~e_for_e/.

Sculley, J. (1989). The relationship between business and higher education: A perspective on the 21st century, *Communications of the ACM* **32**(9): 1056–1061.

Stephenson, N. (1995). *The Diamond Age: A Young Lady's Illustrated Primer*, Bantam, New York.

Stone, M., Webber, B. & Doran, C. (1997). Spud, http://www.cs.rutgers.edu/~mdstone/nlg.html.

TDV (1999). h2g2.com, http://www.h2g2.com.

TDV & AT&T (1997). Principles of guidance. Commercial-in-confidence.

Thomas, F. & Johnston, O. (1995). *The Illusion of Life*, Hyperion, New York.

Tosa, N. (1993). Neurobaby, *Tomorrow's Realities*, ACM SIGGRAPH, pp. pp. 212–213.

Wright, A. (1996). *1000+ Pictures for Teachers to Copy*, Longman.

Zue, V., Glass, J., Hazen, T., Hetherington, L., Lau, R. & Seneff, S. (1999). The MIT GALAXY system, http://www.sls.lcs.mit.edu/GALAXY.html.

Question Survey

Appendix A

We conducted an informal survey of three people to investigate the intentions that people have when talking to a Guide to place to eat and drink.

Each subject was asked to think of as many questions as they could that they might ask a Guide that knew everything about places to eat and drink. They were asked not to think of the Guide as a computer but more as a human to whom they could talk normally.

The following list is not intended to be representative of a particular population – the questions are intended only as examples.

A.1 Intention Types

We grouped the questions into the *intention types* shown below. Each group contains questions which were asking for similar information (the subject's intention was the same). The questions shown here are only representative of the questions we received: we have removed the questions which were identical apart from the place name or attribute specified. After the questions we list the attributes that were included in the other questions.

A.1.1 ShowListWithAttributes

Here, the subjects' intention is to ask the Guide to show a list of places that have the particular attributes in which they are interested.

- Are there any restaurants in Aalborg that have live music?
- Does Aalborg have a kosher restaurant?
- Find me a restaurant that has a parking place nearby.
- I like *Pizza Hut* is there one near here?
- I need a restaurant with room for 100 people, can you suggest one?
- I want a free-range, organic meal in town. What can you suggest?
- I want a quiet, French meal and I don't mind driving. Where should I go?
- I want a vegetarian lasagne. Who delivers?
- I want some spicy food today. Find me a restaurant.
- I want to go to a pub with a nice atmosphere, is there one around here?
- I'm looking for an unusual place to eat.

- I'm looking for typical French food, which isn't too expensive. Where is the best place near here?
- Is there a non-smoking bar near here?
- Is there a restaurant near the park?
- Show me a cheap restaurant, not too far away.
- Where can I find a restaurant for a party?
- Where can I find a restaurant that allows smoking?
- Where can I find an out-of-the-way, interesting restaurant?
- Where can I get a ham and onion pizza?
- Where can I get a pint of *Guinness*?
- Where can I get a really cheap beer?
- Where can I get a special discount for a group?
- Where can I get friendly waiter service?
- Where can I get good champagne with my meal?
- Where can I get some Chinese food?
- Where can I get some Italian food?
- Where can I get vegetarian food?
- Where is a good children's restaurant?
- Where is the cheapest *MacDonald's*?
- Where is there a restaurant with disabled access?
- Where is there a restaurant with speedy service?
- Where's the nearest pub?
- Which Italian restaurants have a low corkage price?
- Which kebab house doesn't put seagulls in their food?

A.1.2 DetailAttribute

The subjects want details of a specific attribute at a specific place.

- Can I park nearby?
- Can you get *Guinness* at the *John Bull Pub*?
- Do I need to make a reservation at *Benny's* on a Saturday evening?
- Does *Natalie's* accept visa?
- Does *La Provence* have a children's menu?
- Does *La Provence* use free-range products?
- Do you have any suggestions as to what to order here?
- Tell me about the cook in this restaurant.
- What are the baby-changing facilities here?
- What is the corkage price at *La Provence*?

A.1.3 GiveOpinion

The subjects want an opinion about a particular subject.

- How does a *Bloody Mary* taste?
- Is this a good place to eat?
- Is this restaurant good?
- What do you know about *Carlsberg Classic*?
- What do you think of this restaurant?
- What does *Calzone* mean?
- What is in a *Long Island Ice Tea*?
- What would you recommend at *Natalie's*?

A.1.4 ShowListForSituation

Similar to the **ShowListWithAttributes** intention. Here the subjects want a list of places suitable for a particular situation.

- I want to take my Grandma to a restaurant she'd like. What can you suggest?
- I want to take my girlfriend for a romantic dinner. Where would you suggest?
- It's a first date – where should I take him?
- I want to go to eat after a movie at *Biffen.* Where can I go that's open, cheap and close?
- The weather is very good so I would like to eat outside. Show me a restaurant where this is possible.

A.1.5 Generalise

Similar to the **DetailAttribute** intention, the subject want a description of the attributes of a group of places.

- What percentage service do I have to pay in a Danish restaurant?
- What types of restaurant are there in Aalborg?

A.2 Possible attributes

Here we list some of the attributes that people mentioned in the questions we received.

- cheap
- exciting
- extravagant
- elegant
- quirky
- unusual
- ethnic
- vegetarian
- kosher
- low calorie
- healthy
- organic
- *country-based* (e.g. Italian food)
- romantic
- non-smoking (or smoking)
- diabetic
- fat free
- friendly
- fast food
- filling
- sandwiches
- snack
- buffet

A.3 Considerations

These are just example questions that could be asked. They represent likely ways of expressing potential users' needs.

The intentions we have identified could be subcategorised in various ways. For example, each type could be divided between those questions asking for places to eat and those that want places to drink. However, this can start to cause problems, since some questions are not clear: the question, "What good pubs are there nearby?" could be asking after a place to drink or a place to get good pub food.

This problem can also occur between the main intention types. For example, the question "I want to go to eat after a movie at *Biffen*. Where can I go that's open, cheap and close?" could be interpreted as a ShowListWithAttributes or a ShowListForSituation intention type.

Another problem is that making sense of the unrestricted situations in ShowListForSituation type questions is extremely difficult. We would have to have a system that knew about life in general, and we have already restricted its knowledge to pubs and restaurants.

Understanding a Generalise intention is also difficult. Firstly, it is difficult to differentiate the intention from a possibly ambiguous ShowListWithAttributes or ShowListForSituation intention. And secondly, generating a story about a generalised case is much more difficult than about a specific one that already has some stories dedicated to it.

To eliminate most problems of ambiguity and parsing, we only attempt to handle the first three intention types: ShowListWithAttributes, DetailAttribute and GiveOpinion. These intentions are fairly distinct and can be picked out using simple syntactic and semantic parsing (see D.10).

Questions that should be interpreted as other types of intention are either falsely interpreted or rejected. This is not such a problem as it may seem – false interpretations still produce an answer which takes into account the ideas expressed in the question. It is as if the Guide misheard or misunderstood the user's question.

A rejection in some ways is a success – the system simply does not know how to answer that kind of question and should not try. The user will then be requested to ask a slightly different question.

The other two intention types, ShowListForSituation and Generalise would need a much wider domain knowledge in the system. Given a finite database, the answer to the Generalise intention could be found simply by summarising across all the relevant entries. However, to tell a story that put this information in context we would also have to generalise the stories about each entry – a much more complex task which goes beyond our expectations of this project.

FramerD

Appendix B

This appendix gives a brief overview of the FramerD database. More details can be found in (Haase 1996).

FramerD is an object-oriented database that runs on a wide variety of platforms. It has a layered achitecture (see figure B.1) where the lowest level is a protocol called the *DType*. DTypes are an external data representation mainly used for sharing information between architectures and networks. This Data Layer also provides for object references defined by 64-bit unique object identifiers. On top of the Data Layer are the Object and Index Layers.

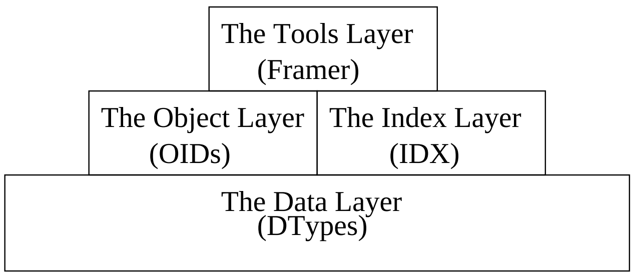


Figure B.1: Layered achitecture of FramerD

The Object Layer provides for the mapping of references into Data Layer values. These Object Indentifers (OIDs) are 64-bit (8 byte) integers. This large range is divided into sub-ranges called *pools*: each pool being a range of OIDs specially assigned to a particular project or program. When a pool is allocated only the "owner" of that range is permitted to modify the mapping within that pool.

An extension to OIDs is called a Frame. A frame is an OID which refers to a set of attributes and relations. Each attribute consists of a slot ID and a value. Values can either be symbols or references to other frames, in which case the attribute is treated as a relation between the frames involved. An example of a frame is shown below:

```
Object @44/B2D0
    Type: "Restaurant"
    Street: "P. Lundsvej 2"
    City: "Aaalborg"
    Owner: @44/35D2"Peter Olsen"
    Menu:  @44/25E4 @44/25E2 @44/25B4 @44/25A3 .....
```

All the values in the form `@44/32D2` are OIDs describing a relation to another frame (`Object @44/B2D0` is the OID of the frame). The Owner slot is a bit different because it illustrates the possibility of referring to an object by both its numeric address and its "human name": many interfaces to FramerD hide the numeric identifers from the user, using the human name whenever possible.

However, an OID gives no cues as to the structure associated with it. If we need to identify objects based on their structure something else is needed. The Index Layer provides this functionality: it associates keys (arbitrary DType structures) to sets of other DType structures. The resulting indices can be used, for example, to describe the meaning of a word by making a connection from the DType containing the word to frames representing its possible meanings.

The Tools Layer uses the Object and Index Layers to provide a basic frames-based "representation language". The basic operations the Tool Layer can perform on objects are:

1. Getting the values for a slot.
2. Testing whether or not the slot contains a specific value.
3. Adding an object to the values for a slot.
4. Removing an object from the values for a slot.

However, the Tools Layer is capable of much more than simple object manipulation – FramerD provides a language called *FDScript*, based on Scheme (which in turn is a simplified dialect of LISP). As such it is possible to define complex functions in FDScript and execute them from any system that can understand DTypes.

Our Interface to FramerD

This is the approach we used to interface FramerD to the h2g2 agents, written in Java. The **FramerD** agent could take text, tell FramerD to evaluate it as FDScript and return the result as a Java object. The **FramerD** agent also defined specialised functions for creating and fetching frames. See the Worknotes Appendix and our source code for more details on how we interfaced with FramerD.

For future implementations, the Machine Understanding Group at MIT's Media Lab is working on a native implementation of FramerD in Java. Unfortunately, this was still not complete by the time our project ended.

B.1 Framer Search

The following is an extract from the documentation provided with the FramerD database from the Machine Understanding Group at MIT's Media Lab. The rest of the documentation, source code and demonstrations of FramerD can be found on the web at:// `http://www.framerd.org/`.

B.1.1 Frame Indexing

FramerD builds an object indexing facility on top of the general association facility described above. Indexing a frame stores inverse pointers from its properties to the frame. This allows the frame to be found later based on a set of properties or to find objects similar (i.e. with common properties) to one particular frame.

There are three main ways of indexing frames:

`(index-frame index frame)` Indexes all the properties actually stored on frame in index.

`(index-frame index frame slot1 slot2... slotn)` Indexes all the specified slots (slot1, slot2, through slotn) on frame. If some of the slots are frames, they may be computed even if they aren't actually stored on frame.

`(index-slot-value index frame slot value)` Indexes frame as having SLOT with VALUE (whether it does or not).

Once a frame has been indexed, one can search for it either strictly based on certain combinations of slots and values or "fuzzily" based on similarity to an instance or set of instances.

B.1.2 Strict Searching

I'll have the french onion soup, without the cheese on top, the green salad with honey-mustard dressing on the side, and a decaffeinated coffee with one sugar.

Sally, in When Harry Met Sally

There are two ways of using the indices constructed by these functions. Strict searching searches for frames with exactly a particular set of attributes. Fuzzy searching searches for the "best match" with a particular set of attribues or with a particular frame.

Strict searching uses the FIND-FRAMES procedure:

```
(find-frames index
   slot1 value1
   ... slotn valuen)
```

which finds all objects that have all the specified slots and values. If any of the valuei are non-deterministic sets, the slot need only have one of the specified values, allowing some variations. For example, in the WordNet database, the expression:

```
Eval:  (find-frames "brico@framerd.org" 'words {"hack" "chop"})
    Value: {@1/31813"VERB.COMPETITION synset for hack and kick_on_the_arm"
           @1/151a5"NOUN.ARTIFACT synset for cab, hack, taxi, and taxicab"
           @1/316ff"VERB.COMPETITION synset for chop and hit_sharply"
           @1/23e1b"NOUN.PERSON synset for machine_politician, ward-heeler, politi
           @1/321a4"VERB.CONTACT synset for chop and strike_sharply"
```

```
          @1/303a5"VERB.CHANGE synset for hack and hack_on"
          @1/1299a"NOUN.ANIMAL synset for hack, jade, nag, and plug"
          @1/cdc6"NOUN.ACT synset for chop and chop_shot"
          @1/31c45"VERB.CONTACT synset for hack and clear"
          @1/20578"NOUN.PERSON synset for hack, hack_writer, and literary_hack"
          @1/31c3d"VERB.CONTACT synset for chop and hack"
          @1/2ee62"VERB.BODY synset for hack and whoop"
          @1/32cfe"VERB.MOTION synset for chop and move_suddenly"
          @1/31812"VERB.COMPETITION synset for hack and kick_on_the_shins"
          @1/129cd"NOUN.ANIMAL synset for hack"
          @1/1ed62"NOUN.FOOD synset for chop"
          @1/2262a"NOUN.PERSON synset for hack, drudge, and hacker"
          @1/31c3f"VERB.CONTACT synset for chop, chop_up, and cut_into_pieces"
          @1/2f7e3"VERB.CHANGE synset for hack and cut_up"
          @1/12999"NOUN.ANIMAL synset for hack"
          @1/bc17"NOUN.ACT synset for chop and chopper"}
```

finds all the synsets containing either the words "hack" or "chop", while

```
Eval:  (find-frames "brico@framerd.org" 'words "hack" 'words "chop")
Value: @1/31c3d"VERB.CONTACT synset for chop and hack"
```

finds only the synsets containing both the words "hack" and "chop". One way to think about this visually, is the search performed by find-frames is conjunctive horizontally (along the list of arguments) and disjunctive vertically (within each argument).

Non-deterministic indices. When the index term provided to find-frames is non-deterministic, the indices are combined for the purposes of searching, so that if one index records entries for one slot and another index records entries for another, that find-frames will act as though the two indices were one.

B.1.3 Fuzzy Searching

I'll have a burger with fries and a chocolate shake.

Harry in When Harry met Sally

A fuzzy search does not require an exact match, but returns the best possible match measured by the number of overlapping properties. There are a variety of fuzzy search functions as well as a set of tools for writing your own fuzzy search routines. The chief function find-best, looks just like find-frames:

```
(find-frames index
   slot1 value1
   ... slotn valuen)
```

but returns those objects with the largest number of matching properties.

B.1.4 Similarity Searching

I'll have what she's having.

from When Harry Met Sally

One of the most powerful search mechanisms in FramerD and FDscript is "similarity searching" which begins with an object or set of objects and finds objects which have the same properties as those objects, weighting as higher those which are more in common among the set of initial descriptions.

(find-similar indices frame) Returns the frames indexed in indices which have the most number of slot values in common with frame.

(find-similar indices frameslots) limits the search for similarity to slots. Returns the frames indexed in indices which have the most number of slot values in common with frame.

For example, the following search finds words with similar meanings to the common sense of "hack" and "chop":

```
Eval:  (find-similar "brico@framerd.org"
            @1/31c3d"VERB.CONTACT synset for chop and hack")
    Value: {@1/31be9"VERB.CONTACT synset for shave, trim, and cut_closely"
            @1/31c3f"VERB.CONTACT synset for chop, chop_up, and cut_into_pieces"
            @1/31e95"VERB.CONTACT synset for mow and cut_down"
            @1/325d4"VERB.CONTACT synset for rebate and cut_a_rebate_in"
            @1/325d7"VERB.CONTACT synset for saw and cut_with_a_saw"}
```

Fuzzy searching and non-determinism. When the frame argument to find-similar is non-deterministic, the search mechanism does a single search but uses features from each of the frames in combination. As a consequence, the search weighs properties common between the frames more heavily. For example, in this retrieval,

```
Eval:  (find-similar "brico@framerd"
        (amb @1/31c3f"VERB.CONTACT synset for chop, chop_up, and cut_into_pieces"
             @1/31e95"VERB.CONTACT synset for mow and cut_down"))
Value: @1/31c3d"VERB.CONTACT synset for chop and hack"
```

the properties common to the two synsets selects the search pattern which put them together in the first place.

Universal Context Model

Appendix C

The Universal Context Model (UCM) is part of the work Richard Harris and The Digital Village are doing on creating an information ecosystem. The following text is an email that he sent us describing its current state of development.

The term Universal Context Model and the abbrevation UCM are Copyright Richard Harris 1998.

If we consider the value path from raw data, through abstracted information to applied knowledge and thence (hopefully) to predictive wisdom, we find that, in the open environment of the Net, we need to be able to collect data from heterogeneous sources which of themselves may have a greater or lesser degree of explicit structure. As we collect the data, we also need to collect or infer as much information as we can about their local context – explicit XML tags, keyword definitions, link models, nearest neighbour/affinity analyses etc. Some of that is done at the point of collection, other at the host service. Once we've abstracted as much of that information as possible, we need a generic information model into which to map it. Without doing that, we simply can't then go on to create a consistently relevant basis for queries and added value services within the local domain (in our case h2g2). In an ideal world, everyone would build their web sites and corporate databases to a common information model and would publish that model so that search sites would simply collect the appropriate content with no need for inference or assumption. That however is a scenario whose probability can only be calculated by reference to the Infinite Improbability Drive... What we can do however (I hope) is create an abstracted metamodel for information context, into which we can map any content we wish to collect. Of course, given a sufficient level of abstraction we can model anything :) – the objective is to use the lowest level of abstraction that will effectively create a universal metamodel. We can then use that metamodel two ways:

1. to instantiate an information model for our own service, into which we explicitly place all content we create within the service; and
2. as the target model for the content we're bringing in from the outside by our bots and search agents

In the latter case, creating the context is by two paths:

1. by inference, analysis and categorisation by agents (software and human); and
2. by explicit mapping by the creators of the information source

The second of these obviously depends on the UCM (a) working and (b) being openly published and accepted by the Net at large, so that it's worth people's while to create the mappings from their own data models into the UCM. Any search bot could then pick up the mapping along with the source (XML is of course an ideal vehicle) and be able to map the content into its site's own database via its instantiation of the UCM.

Now let's look a little at the structure and model for the UCM. Firstly, there's a few basic requirements:

- fully scalable – it must be capable of describing the context of anything from a haiku to a universe
- abstractable – it be as applicable to a completely conceptual piece of information as it is to a real world description
- function independent – it must make no assumptions about how the information will be assembled , delivered and presented
- universal – it must be capable of positioning any information item in space, time, entity scope and subject.
- recursive – capable of description at any and all levels of granularity.

That was the easy bit. The tricky part is working out what the appropriate co-ordinate system for each context is.

Information is created at the atomic level (for the purposes of the Guide, we assume you can't split atoms on the Internet). It has associated with it an initial set of attributes which locate, describe and classify the information. Alongside these attributes is a relationship structure which identifies the initial set of uses for the information that we've defined. Over time the attributes expand to include a timeline of the uses of the information and the relevance information to particular queries and individuals, allowing us to build ever more effective profiling, affinity and referral models.

The most basic premise is that we can classify any item of information as being any or all of:

- an event – time co-ordinates
- a place or object – spatial co-ordinates
- a person or other entity
- a concept
- a relationship (to other relationships or any of the above

For h2g2, we'll start with the most basic of attributes that are needed to locate the information by space, time and subject and use the same basic structure to record assemblages of information at multiple levels. Over time, we'll add the more abstract classifications once they're approximately consistent. For example of how the thinking is going, here's a rather cryptic example cribbed (as much of this rather disjoint note is) from my rather ad hoc notes and notes of discussions between myself and Douglas:

- Context Type (1, mandatory)
  - Space
  - Time
  - Context
  - Individual
- Veracity (1, mandatory)
  - Fact (the capital of England)
  - Fiction (the capital of Narnia)
  - Speculation (the capital of Barnard 4)
- Trust (1, mandatory)
  - Verified
  - Source
  - referred
  - chain length
- Location (1, mandatory) * scale for each
  - co-ordinate system
  - dimensions (value 1..n) – move type stuff into here
  - Co-ordinates
  - Type (1, mandatory)
    - point
    - co-ordinates
    - area
    - area type
      - circle
      - rectangle
      - ellipse
      - polygon
      - irregular
      - polygon
    - co-ordinates (1..n)
    - area unit
    - area
    - volume (recursive)
    - spheroid
    - volume
    - construct (n-dimensional)
    - cuboid

        - path
        - co-ordinate type
            - absolute
            - relative
            - reference co-ordinates
        - measurement unit
        - nodes (1..n)
            - co-ordinates
            - angle
            - displacement
        - algorithm (e.g. orbital path)
        - number of parameters
        - parameter values (1..n)
        - algorithm type (lookup)
    - resolution
        - units
        - value

- Subject (1:n, mandatory)
- Time (n, optional)
    - Nature
    - absolute
    - relative
    - point | duration
    - Relevance (apply similar to overall object)
    - reference event
    - offset
    - finish | duration
    - Units
    - Reference (for relative)
    - Time system
        - Calendar (e.g. Christian Julian/Gregorian)
            - Date only – convert entries automatically – will require location for accuracy
            - Egyptian
            - Muslim
            - Mayan
            - etc.

Worknotes

Appendix D

D.1 DSS Attributes

Problems faced

How should the DSS grade items? What attributes should the DSS know about? After all, the ranking of the articles in the interface depends entirely on what the DSS knows about (i.e. how good a model of the user it can build up).

We would like the DSS to be independent of the information in the database. Otherwise new kinds of information would either require the DSS to be rebuilt or would have no effect on the ranking of the items. Therefore we would like the DSS to be able to add new preferences on the fly and to be able to adjust them as it goes along.

Various problems here:

1. We might have items with different numbers of attributes. This is currently not supported since the penalties are simply added up – more attributes means a greater penalty...

2. We need a conversion function for each attribute value before the preferences are applied to it. If we want to be able to add new preferences on the fly this could be a big problem.

3. Some attributes have scalar values (Price, Distance), whereas others are unordered (Gender, FoodServed) or are simply true or false (hasDisabledAccess). We need some way of coping with all of these.

Possible solutions

Differing numbers of attributes

To sort out the problem with differing numbers of attributes we need to change the method of combining different penalties. An arithmetic average would be an easy way to do it. Given that the penalties are already given correct weights in relation to each other this would work perfectly.

The other problems are not so easy to solve.

High-level preferences

Some preferences might relate to the style of the information presented rather than the information itself. For example, the user could have a preference as to what level of reality they like their reviews to have (pure fact or way out conjecture). These preferences would not have to be picked up on the fly since they would apply to all information in the database.

Perhaps the user should have preferences as to which part of the knowledge organisation grid (Moorman & Ram 1994b) they prefer to see information. Some users might prefer pure physical information (straight facts about concrete things), whereas others might want to see other people's opinions (mental, social and emotional types of information).

Conversion functions and unordered attributes

Maybe it's possible to get the DSS to learn about new attributes based on the information that's fed to it. The main problem with this is that we need a conversion function for each attribute to enable a meaningful comparison to be made with the other attributes.

One way to make a conversion function for unordered attributes is to have an indicator function for each of the different states i.e. a function that has the value 1 when the state is present and 0 when it is not. These indicator functions are then weighted to convert them into the standard measure and are then fed into different preferences.

So, for example, the FoodType attribute could be dealt with by having an indicator function for each type of food (one for Italian, one for French, etc.). All the indicator functions for FoodType would be given the same weight, since they are all about the same thing. There would then be one preference for each different type of food. Thus when calculating the penalty for a restaurant that served Mexican food, only the preference for Mexican food would come into effect (the others would have indicator functions that returned zero.

By using these indicator functions, we have reduced the problem from finding a conversion function to finding one weight per attribute. This weight could actually vary from user to user in that there is no authoritative conversion from FoodType to monetary value. Thus a default weight could be given (maybe an average of the weights of the other preferences) which could later be changed by visiting the Pet Psychiatrist.

Conversion functions for ordered attributes

Ordered attributes are not so easy to deal with. The conversion from distance to money is not necessarily linear (in fact, last semester we used a power function) and there is no obvious link between the conversions.

This problem could be dealt with in a similar way to the unordered attributes, only the indicator functions would be a set of fuzzy functions. For example, distance could be divided into three states: walkable, driveable and long trip. Each state would be specified by a fuzzy set defined over distance and there would be a preference and a weight for each state. 5km could then be 5% walkable, 100% drivable and 0% long trip. Each of these scores would be fed into a different preference and weighted according to the monetary value of each state.

Considerations

High-level preferences

In order for the user to have preferences as to which part of the knowledge organisation grid they prefer to see information, all slot fillers must be located somewhere in the knowledge organisation grid. This means that the parser (or fixed frames) that grab the information into the database must know about the ontology of the words and the ontology must be linked to the knowledge grid. The FramerD parser is already linked to an ontology (the upper CYC ontology) but unfortunately this ontology is not linked to Ram's knowlege grid.

Since we intend to use TDV's UCM (see section 3.7) rather than Ram's knowledge grid, the DSS can have some preferences that apply to all kinds of knowledge (e.g. what ratio of fiction to fact the user likes to see). However, we still need preferences for the actual information that is passing through or else we will end up making decisions entirely on the reviews, ignoring the information they contain.

Conversion functions for ordered attributes

Unfortunately, the fuzzy set solution still leaves us having to decide not only the weights of each state, but also where the boundaries between the states should be. This decision is not any easier than inventing a conversion function and cannot be done automatically.

Fortunately, there are not so many ordered attributes in our domain. Distance and price are the only objective ones we have found. FoodQuality could be considered ordered but it would be extremely difficult to judge objectively. Thus it should be possible to hard code the conversion functions for all of the ordered attributes involved.

Solutions chosen and why

We will implement indicator functions for unordered attributes so that the DSS can make up preferences about any attribute in the database on the fly.

We will precalculate the conversion functions for all the ordered attributes in our database.

The DSS will also have preferences about the type of information the user likes to see. These will be based on the general attributes of the UCM, such as *veracity* and *trust*.

D.2 Pet Psychiatrist

Problems faced

Should our system have a Pet Psychiatrist like last semester? What advantages would this give as there are no longer extra features to be added when the user gets more experienced...

Possible solutions

Maybe extra features could be translated to extra information sources, or extra storytelling agents. For example, after the user has kept their pet happy for long enough, they might get access to a storytelling agent who knows about something else.

If the system was a commercial one, then revealing more information about yourself to the Pet Psychiatrist could be rewarded with a reduction in the cost of the service. This information could then be used by the service provider to target advertising or to tailor other information sources.

The information gathered by the shrink is still useful. Primarily, it could be used to adjust the weights of preferences added on the fly (see D.1).

Another use of the shrink is to actually discover the mental constructs that the user uses to make decisions. This can be done using *Repertory Grid Analysis*, a method of psychoanalysis that involves the user making their decision making constructs apparent, and perhaps even learning what they are. This method was invented by George Kelly (Kelly 1955) and has been applied to Decision Support Systems by Ron Atkin as Q-Analysis. For more details, see `http://www.ship.edu/~cgboeree/kelly.html`.

Considerations

If we don't have the pet psychiatrist, there is not much point in giving the pet emotions, since the user won't be able to do anything about the pet getting confused.

The main point of the Pet Psychiatrist is to form a closer bond with the pet and to elicit information from the user.

We need to work out what will be in the DSS before this point can be decided.

As with last semester, we will not have any time to actually implement the Pet Psychiatrist.

Solutions chosen and why

There should be a Pet Psychiatrist in the completed system. However, we are not building a completed system, more a proof of concept. Therefore, we shall leave this part of the system sketched out but incomplete.

D.3 Sentence Forming

Problems faced

What is the sentence forming process? How are smarticles made?

Possible solutions

One simple way to do this would be to have an ARTICLE frame, consisting of a complete human written article and a short description so that the storytelling agents could tell what it was about.

While this would be suitable for the human-based agents it brings the whole system down to the level of a standard search engine.

The ideal solution is to model a human memory, based on Schank's concept of frames and scripts. To have the database know about everything would be an impossible task. Instead we will take a particular subject area and try to get a reasonable knowledge about that into the database.

We would like to be able to mix and match sentences and to conglomerate sentences as in MAGIC (McKeown et al. 1997). This system deals with information gathered from a relational database and consequently with a lot of lists of similar information. While the information in our system will not necessarily be so homogeneous, it should be possible to use similar principles to involve pronouns and conjunctions in the output of our system.

Considerations

It is very difficult to get a computer to understand free text. This is called *information extraction* and is what all the research centres we visited in the US (IMM 1999) were spending huge sums trying to achieve.

Solutions chosen and why

Instead of having to do full information extraction, we will put some domain information into the system to begin with (on the specified, narrow domain of restaurants and pubs). Other text can then be read into the system to give it a database of sentences. This text will be converted into case-frames by Ken Haase's parser, but not into any higher level of meaning.

D.4 Agent Personification

Problems faced

We want both the DSS and the agents to have some personality. The DSS has emotions that reflect its state of confusion (as per last semester). The storytelling agents just need to have character. How should the different personalities be represented graphically?

Possible solutions

One way would be to have one character that presented everything. It would have the emotions given to it by the DSS and the personality of the storytelling agents. However, this combination might be confusing to the user, since the personality will appear to be slightly schizoid – shifting its manner of speaking when a different storytelling agent presents the information.

It might also get confusing when the user visits the Pet Psychiatrist: who will he be visiting? The storytelling agents (with their associated characters) or the plain DSS agent?

Maybe it would be safer to separate the personalities. The DSS representation would be the introductory guide, displaying the list of possible information sources. When the user chose an item in the list, it would be presented by the appropriate storytelling agent (each of which would have some animation).

Considerations

Will the system still have a Pet Psychiatrist? See D.2 for more discussion on this point.

Solutions chosen and why

We chose to have separate animations – each storytelling agent will have its own, separate from the DSS agent.

D.5 Information Passing

Problems faced

How do we pass information *down* the structure, from the user to the storytelling agents? This is necessary since there could be an arbitrary amount of info in the database and the storytelling agents need to do some selection to avoid finding stuff which is totally irrelevant.

Possible solutions

Idea from Amalthea: each storytelling agent considers itself as a perfect representation of the user. However, it does need some context to find information: for example, the position of the user and the current time. Maybe we could also pass down the current user preferences.

We could treat the DSS as a query agent (some kind of limited avatar of the user). It knows about the current search topics and also what kind of things the user enjoyed before (using its probabilistic prefs). However, we would need some clever way of passing down the probabilistic preferences, and anyway this would limit the effectiveness of the storytelling agents in that they would not be as free to select information based on their biases.

Using intentions:

Analyze the possible query types beforehand so that we have several different intention types for the user. Since the domain is quite small, there should be a limited number of different intention types. Then parse each query into case-frames using the FramerD parser and match it to the nearest precalculated intention frame.

Each intention frame would need specific information. For example, the user's intention to find a nearby restaurant would need the user's current position. Gaps in the intention frames would be filled in by the system, either with information it knows (such as the position of the user) or with a default value. Default values would be defined together with the intention frames.

The resulting fully defined intention frame is then passed down to the storytelling agents. They add their own intentions to the frame (such as a bias towards sports bars with pretty waitresses) and use the resulting frame to search the database.

Considerations

In the last method, the DSS does not need to be involved in passing information around – it merely rates smarticles and updates its preferences.

Using intentions does involve some more work, however, since we need to work out all the possible questions that could be asked of our system in order to abstract out the different intention types. This is normally done by involving lots of different people and getting them to interact with a Wizard-of-Oz-like system. Unfortunately, we don't have enough time to conduct large user tests. Instead we will ask a few people to write down all the questions they can think of that they would like to ask a system that knew everything about Restaurants and Pubs.

Solutions chosen and why

We will use intention modelling, deriving the intention types from a long list of possible questions. This list will be built by asking a few people to think of as many questions to ask as possible.

D.6 Dialogue Manager

Problems faced

What controls the system? Is there a dialogue manager? Is it a dictator? Can the user interrupt while the system is processing?

Possible solutions

Since most queries are self contained, our system will not be built to handle complex conversations. This means that the Dialogue Manager (DM) can be quite simple.

A simple way to control the system is to have a central, dictatorial DM that keeps track of the state of the system and passes the relevant information on to each of the subsystems.

In order to keep track of some context in the dialogue, we could have a separate Context Server. This would be responsible not only for maintaining the dialogue history (a record of who said what), but also of relating any of the user's queries to this history. For example, if the user chose to read a smarticle on a particular restaurant and then asked, "Can I park nearby?", the Context Server would be responsible for adding the restaurant previously selected to the user's intention.

The Context Server would also be responsible for keeping track of other contextual information such as the user's position and the current time.

The DM would have three main states: a *start* state in which it is waiting for a query from the user; a *list* state in which it was displaying a list of different smarticles for the user to choose; and a *smarticle* state in which it is presenting an individual smarticle to the user.

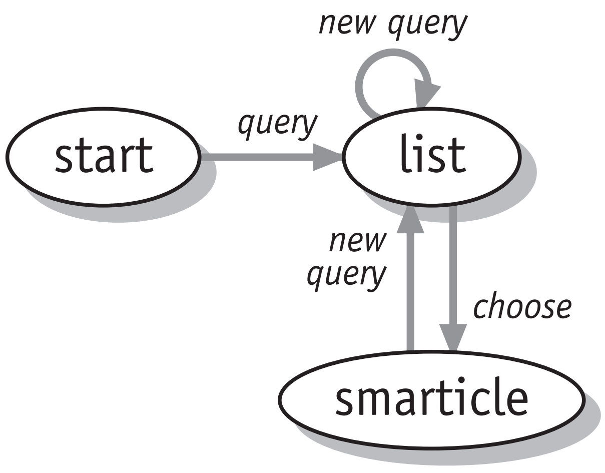


Figure D.1: State diagram for the Dialogue Manager

Considerations

One of the results we got from last semester's project was that the user must be kept informed if the system is busy processing their commands. This can be done by having a special state in the DM set aside for being busy. If the user interrupts while the system is busy, then the previous request should be ignored and replaced with the new one.

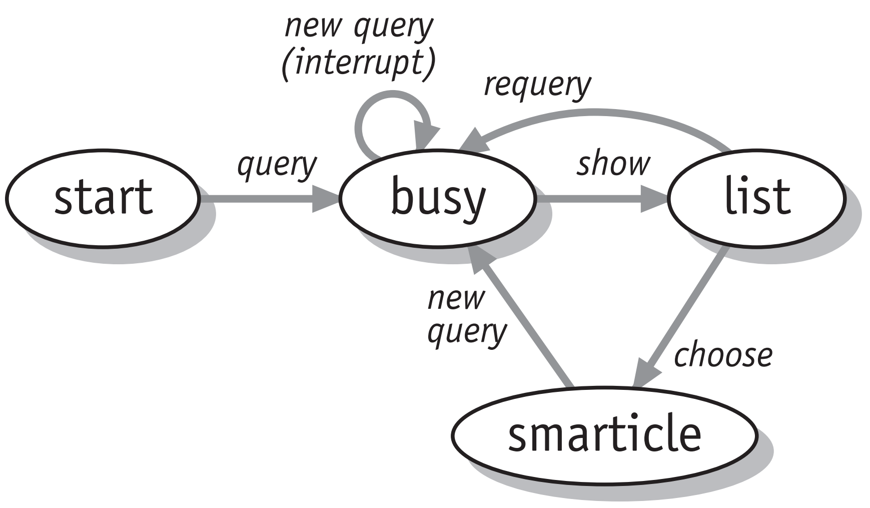


Figure D.2: DM with a busy state

This state would be placed between the listing state and the article state as in the diagram D.3.

In the *list* state, the DM is waiting for the user either to choose a smarticle or to ask a new query. If both of these choices are to be made by speech recognition, it would be very easy for the system to misunderstand. For example, if the user said, "I want to see the restaurant with a non-smoking section," the system could interpret this as a new query or as choosing one of the smarticles in the list. This kind of disambiguation is very hard to solve without in depth contextual analysis.

If the system is not sure which choice to make it must either ask the user for further information or offer the user the chance to go back when it makes the wrong choice. Either of these options add extra complexity to the system which offset the simplicity of always being able to make a new query.

To avoid this problem, we redesigned the states DM so that this problem could not occur (see D.3). In doing so, we made the dialogue a little less user-directed and more system-directed. Instead of the user being able to make a new query at any point, we opted to add a *cancel* command to each state. A new query could then only be made from the *start* and the *smarticle* states.

Although we have lost some of the flexibility of the original dialogue structure, the user is never more than one *cancel* command away from being able to make a new query. The *cancel* command also makes the structure of the dialogue more transparent to the user – instead of always having to do something new, they have the option to go back to what they were doing before.

Solutions chosen and why

We choose to have a dictatorial DM since the system is quite simple. We will use a busy state and a cancel command to make the dialogue process more transparent. We will implement a simple Context Server but will not spend much time on it.

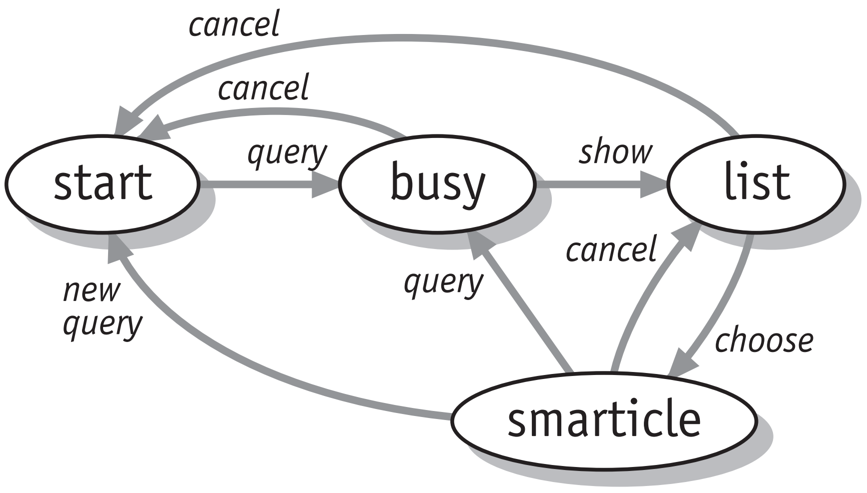


Figure D.3: DM with a cancel command

D.7 Agent Intentions

Problems faced

We need to identify the intentions of the storytelling agents. These should be directly related to the intentions of the users' queries (see Appendix A).

Possible solutions

The user intentions that we have identified and plan to answer are:

1. List places with *specified attributes* given the user's *current location* and *time of query*
2. Give details of *specified attribute* at *specified place* given the user's *current location* and *time of query*
3. Give an opinion of a *subject* given the user's *current location* and *time of query*

The emphasized words indicate information that must be present in the slots of the intention frame.

We also identified two other intentions, but do not plan to answer them in our system.

Each storytelling agent must produce at least one answer, each of which must contain a review of exactly one place. Thus their respective answering intentions could be:

1. Give an opinion of a *suitable place* for the *specified attributes* given the user's *current location* and *time of query* (**OpinionPlace**)
2. Give an opinion of the *specified attribute* at the *specified place* given the user's *current location* and *time of query* (**OpinionAttribute**)
3. Give an opinion of the *subject* at a *suitable place* given the user's *current location* and *time of query* (**OpinionSubject**)

As before, the emphasized words indicate information that must be present in the slots of the intention frame.

Considerations

The intention that the storytelling agents receive will already have all known contextual information (such as the user's position) included. The storytelling agent will then add its own biases, for example, a bias towards sports bars. *All* these attributes will then form the basis of the query and will be included in the answering intention.

It's important to note that the agents should not just intend to describe the subject matter, but should also intend to *give their opinion* of it. Some agents, like a purely factual one or one that always regurgitates complete human-written articles, have no opinions *of their own.* The reviews they return will thus solely contain the opinions of the original material provided to them.

Solutions chosen and why

We chose to implement the three answering intentions listed above. They are general enough to allow the agents free expression, but closed enough to allow the DSS to pick out the information it needs to rate the smarticles.

D.8 Smarticle Headers

Problems faced

What should be in the smarticle headers?

Possible solutions

Since each smarticle will refer to exactly one restaurant or pub, there will be information about that place. However, there should also be information about the agent that produced the article.

If we think about the flow of information through the system as a sequence of frames, the user's spoken request is translated into a case-frame representing its syntactic and semantic structure. This is then matched to an intention and passed to the storytelling agents together with all the necessary context (such as the user's location).

The storytelling agents then add their own intentions to the frame and look for matching information from the world knowledge database and the reviews database. The answer they generate is the smarticle. This itself could be viewed as the answer frame, with one of the slots being filled with the text of the smarticle. The other slots form its header.

If we are modelling the user's input using intentions, it makes sense to use the same technique to model the system's output. We therefore need to identify the intentions that the agents should have when they are answering the users' queries.

See D.7 for the agents' intentions.

Considerations

These intentions should then be incorporated into a structure that also gives a reference to the agent creating the smarticle.

An answering intention frame must understandable to the **DSS**. It should not only be able to pick out all the attributes in the frame, but be able to tell which ones are distance- or price-based (these are the only two ordered attributes we are considering). Each attribute needs to have a unique type so that the **DSS** can identify new types of attributes and create preferences for them.

The intention frames produced by the storytelling agents should be stored in the **Context Server** so that the system knows what it has said. A more complex **Dialogue Manager** would then be able to understand references like, "Show me the one on Kings Street," or even, "Tell me more about the pizza restaurant you mentioned yesterday."

Each frame should also be self-contained (for example, it should have no references to other frames in the World Knowledge database). Not only does this mean that the **DSS** does not need a connection to any of the FramerD databases, it also ensures that the history in the **Context Server** does not change when the World Knowledge changes.

Solutions chosen and why

Each smarticle will be an intention frame, using one of the intentions designed in D.7. Each frame will have at least the following slots:

- UCM Type (relation between agent and restaurant)
- UCM Veracity (based on that of the agent)
- UCM Trust (based on agent's source material)
- intention
- authoring storytelling agent
- title of smarticle
- text of smarticle
- place type (pub or restaurant)
- place name
- place location
- user location at query
- time of query

There will also be slots for the attributes of the place that are relevant to the smarticle. Each of these attributes must be typed to allow the **DSS** to find (or create) the relevant preference.

In addition to all these, the **DSS** will add a slot for the rating given to this smarticle according to the user model.

We will hopefully use the Java interface to FramerD, which allows us to access and create DTypes and frames directly from Java.

Although we will store all the system's answers in the **Context Server**, we will not do any complex dereferencing in this project. The user will not be able to refer to previous queries and will only be able to select a list item by its number. Although these actions would be possible using our system, we do not have the time to implement them.

D.9 FramerD Problems

Problems faced

The first problem encountered when running the Brico system was an error in the `brico.fdz` file. It tried to use a file called `word.index` but none was available. This file was supposedly the index to the WordNet database. By consulting Ken Haase, we found out that the reference could just be changed to `brico.index` because the Brico database also included the WordNet database

The next problem was the *parse-arg* function which is used to convert arguments from the command line into lisp objects that can be sent to a framerd server. Ken Haase didn't have any solutions to this and since it was not vital we to the project we didn't spent a lot of time trying to fix it.

The next big problem was that we were unable to create new pools with the command `make-file-pool`. We really needed this function to create two pools: one pool to store the world knowledge and another pool to store the history/context information. No matter what we did `make-file-pool` always returned:

```
;; !fdscript! Can't check lock file
             (/usr/local/lib/framerd/super-pool.LCK) #?
```

In the file `odb.html` which is part of the documentation on FramerD, there is a note that says:

> If your FDScript session exits abnormally, the file may still be locked by the existence of a file filename.LCK. The leftover lock can be cleared by simply removing this file.

but the file `super-pool.LCK` was not there and if it wanted to write one the directory protections were fine. It turned out that the solution was trivial. FramerD was installed as an rpm packages which means that all installation and "configuration" is done for you. But when the config file `/etc/framerd.cfg` says that FramerD is installed in `/usr/local/lib/framerd` and the rpm packages installed it in `/usr/lib/framerd` it is bound to give some problems. After this was corrected everything worked fine.

D.10 Recognising the User's Intention

Problems faced

How do we identify the intention of the user from their question?

From Section A we have a list of possible question formats and the intentions that they represent. The system needs to be able to identify the intention from the text of the user's questions.

Possible solutions

The parser from MIT will break down the question into syntactic and semantic case-frames. If we can identify common syntax and semantics for each identity type, we can match a question to its corresponding intention.

Basic structure

Most questions fit into one of the general templates listed below. In each of these templates, an item in angled brackets will be replaced with a phrase (or several phrases) of the relevant type. For example, the `conj` slot contains conjunctions such as "and", "that" or "which". Slots enclosed in square brackets can be missed out altogether.

1. *Intention:* Show a list of places that have these `attributes` (`ShowListWithAttributes`)

   - Are there any [<attributes>] <place> [<conj> <attributes>]?
   - Find me [<attributes>] <place> [<conj> <attributes>].
   - I like [<attributes>] <place> [<conj> <attributes>].
   - I need [<attributes>] <place> [<conj> <attributes>].
   - I want [<attributes>] <place> [<conj> <attributes>].
   - I'm looking for [<attributes>] <place> [<conj> <attributes>].
   - Is there [<attributes>] <place> [<conj> <attributes>]?
   - Show me [<attributes>] <place> [<conj> <attributes>].
   - Where can I find [<attributes>] <place> [<conj> <attributes>]?
   - Where can I get [<attributes>] <place> [<conj> <attributes>]?
   - Where is [<attributes>] <place> [<conj> <attributes>]?
   - Which [<attributes>] <place> <verb> [<conj> <attributes>]?

2. *Intention:* Give details of `attribute` at a `specific place` (`DetailAttributeAtPlace`)

   - Can I <attribute> <place-name>?
   - Can you <attribute> <place-name>?
   - Does <place-name> <attribute>?
   - Do I need <attribute> <place-name>?
   - Do you have any suggestions as to <attribute> <place-name>?

- Tell me about <attribute> <place-name>.
- What is/are <attribute> <place-name>?

3. *Intention:* Give an opinion about `subject` (`GiveOpinion`)

    - How does <food-type> taste?
    - Tell me about <subject>.
    - What do you know about <subject>?
    - What do you think of <subject>?
    - What does <subject> mean?
    - What does <food-type> taste like?
    - What is <subject>?
    - What is in <food-type>?
    - What are the ingredients of <food-type>?
    - What would you recommend <place>?

Some of the parts of these templates, such as `food-type` or `place-name`, will be references to fields in a world knowledge database. The `conj` slot will be detected directly from the parser's case-frames.

Attribute slots

`Attribute` slots contain adjectives, adverbs and adjectival phrases that provide information about the subject at hand. For example, if the user asks, "Is there an Italian restaurant that serves ham and onion pizza?" the intention parser should pick out *Italian* and *serves ham and onion pizza* as two attributes that describe the restaurant.

Place slots

The `place` slot contains a reference to a place. This reference can be in the form of the name of a specific place, as in "in The John Bull Pub". It can also be a spatial reference such as "nearby" or "here", or even a reference to a type of place, like "a restaurant". We would also like the system to be able to recognise indirect phrases: for example, "a meal" or "a drink" should refer to a restaurant and a pub respectively.

`Place-name` slots, on the other hand, are limited to the names of places that are in the world knowledge database.

Subject slots

The third type of intention introduces the idea of the `subject` slot. This can contain either a `food-type` or a `place` since these are the only subjects that the system can talk about.

Considerations

These templates cope with most of the questions that could be asked in each intention type. However, there will always be ways to phrase questions that will not match any of these. We do not intend the system to be complete and so questions which do not match the templates will be rejected. The user will then be asked to rephrase their question.

Though most of the templates have been chosen to be distinct, there is an overlap with "What is..." and "Tell me about..." type questions: they can either be `DetailAttributeAtPlace` or `GiveOpinion` intentions. The distinction made here is that `DetailAttributeAtPlace` intentions *must* mention a specific place.

Solutions chosen and why

We choose to implement the above question templates in our Intention Parser. The templates have been chosen in such a way as to eliminate ambiguity. This is at the expense of recognising certain questions – the Intention Parser will fail to parse some questions and the user will be forced to rephrase them.

D.11 Story Events

Problems faced

Since the StoryAgents might be quite slow to build smarticles, how should smarticles be collected and sent off to the DM? What happens when the user cancels?

Possible solutions

The StoryAgentHome (SAH) could wait until all the StoryAgents it asked to make smarticles have returned one. This could take a long time.

Alternatively, it could send smarticles off to the DM as it received them. In this case, the DM would need to know when to start displaying the list of smarticles and stop telling the user that it was busy thinking.

Neither of these two solutions sounds promising. The DM needs to have all its information ready at a specific point, whereas the SAH has to wait for information to come in slowly.

A compromise would be for the SAH to wait until a certain number of smarticles have been returned and then to send all of those off at once to the DM. Any StoryAgents who didn't send their smarticles in time would then be told to stop processing.

Considerations

How many smarticles should the SAH wait for? The interface can only display four, but these should be chosen by their DSS ratings not by the fastest agents. Maybe the SAH should wait for a specific amount of time first and *only then* start worrying about getting back to the DM. The SAH would then return a full set of smarticles if all StoryAgents returned within the time limit, and a list with a minimum length otherwise.

If the SAH is to wait for a specified length of time, how long should that be? This time should probably be dependent on the amount of time it takes for StoryAgents to respond and the number of times the user stop waiting for information and cancels their query.

Perhaps we could use an algorithm similar to that used by the TCP protocol to estimate how long a sender should wait until it receives an acknowledgement from the receiver. This algorithm ((Postel 1981)) creates a smoothed estimate of the time taken for a reply and then sets the timeout for a little bit longer to allow for some variance. If a reply is timed out, the protocol uses an *exponential backoff*: waiting for twice as long each time until a message gets through.

To use this algorithm in our case, we would have to add an extra case to deal with reducing the timeout when the user cancels. This should not be done by halving the timeout, since the user might cancel for other reasons than having to wait too long (for example, they might want to change their question). Instead, we should probably treat a cancelled query similarly to a successful query – using the time taken to reduce the estimated time for a reply.

What should be the minimum number of smarticles to return? The SAH should always return at least four smarticles (if there are four StoryAgents) since this is as many as the interface can display.

Solutions chosen and why

The StoryAgentHome server will estimate the amount of time it takes for all the StoryAgents to return their smarticles and set a timeout for a little bit longer. If all the smarticles are returned within this time, it will send them all off at once to the DM. If some StoryAgents do not return smarticles within the time limit, the SAH will wait indefinitely until it has at least four to send off and then cancel the others.

D.12 Cancelling Stories

Problems faced

Running threads over a network, we can never be sure when events will arrive at their destination. The DM could send a CancelEvent to stop the StoryAgents processing and then send a new QueryIntnEvent to get them looking for new stories. How are we to ensure that the CancelEvent (which could arrive later than the QueryIntnEvent) doesn't stop the new stories being processed?

Possible solutions

We could put the event object that the CancelEvent is cancelling into the CancelEvent object. Thus agents can ignore CancelEvents that are not related to the event that they are currently processing.

Considerations

Two considerations here: firstly, an agent could receive events from several sources – it needs to know *all* the events it is currently processing, not just the last one it received. Secondly, we do not need to worry about events getting lost (and thus a process missing out on its cancellation) since the Java RMI runs over TCP, ensuring that data gets through exactly once.

Solutions chosen and why

We add the event that is being cancelled to the CancelEvent definition.