Decisions To Go: An Intelligent Mobile Decision Support System

by

Hendrik Fischer

(Under the direction of Walter D. Potter)

Abstract

Decisions To Go is an intelligent mobile decision support system that facilitates the task of product research for consumers on the go. This paper describes the development of software that can be used on a PDA or a regular cell phone, and which enables the customer to access valuable information about a product when needed most – while being in a retail environment struggling with information overload. Providing the "right" information turns out to be a difficult problem. The expert system not only has to be able to identify appropriate recommendations and advice for each individual user, it is also used to reason with the huge amount of data found on the Internet, and in doing so must be able to retrieve specifically the information that is considered important to the customer in supporting the decision making process.

Index words: Decision Support Systems, Expert Systems, Knowledge Management, Mobile Applications, Intelligent Information System, Information Retrieval, JESS

Decisions To Go: An Intelligent Mobile Decision Support System

by

Hendrik Fischer

Vordiplom, Universität Koblenz, Germany, 2003

A Thesis Submitted to the Graduate Faculty

of The University of Georgia in Partial Fulfillment

of the

Requirements for the Degree

Master of Science

Athens, Georgia

2005

Decisions To Go: An Intelligent Mobile Decision Support System

by

Hendrik Fischer

Approved:

Major Professor: Walter D. Potter

Committee: Jay E. Aronson
Scott A. Shamp

Electronic Version Approved:

Maureen Grasso
Dean of the Graduate School
The University of Georgia
August 2005

Dedication

This thesis is dedicated to Heike for her everlasting love and support.

Acknowledgments

"A central lesson of science is that to understand complex issues, or even simple ones, we must try to free our minds of dogma and to guarantee the freedom to publish, to contradict, and to experiment."

— Carl Sagan

I always had a feeling that the Artificial Intelligence Center is adhering to this principle, probably because the field itself challenges the human mind in a way Computer Science in general doesn't. Instead of following the well-known paths, we were always encouraged to break out of the framework and think for ourselves. I want to express my deepest gratitude to the University of Georgia and the faculty at the AI Center who created this open-minded environment.

I would like to thank everyone who was part of my endeavor in this program, especially: My major professor Dr. Potter, for his support and encouragement for my research project and his advice and ideas during my course work and prototype development. I would also like to thank you for coming up with the perfect name for my application. Dr. Shamp from the New Media Institute, who taught me many valuable things and constantly challenged me in a way such that I had no choice but to improve myself as a professional. Dr. Aronson for serving on my committee and providing interesting ideas for my thesis project.

I would also like to thank my fellow students who made my time at the A.I. Center worthwhile and fun as well as contributed ideas and interesting thoughts during countless discussions. Special thanks to David for being so jewrific all these times.

Finally, I like to thank my family for supporting me throughout my graduate career and for always being there for me.

Table of Contents

List of Figures

Chapter 1

Introduction

1.1 Overview

Decisions To Go is a decision support system (DSS) for making informed decisions in retail environments. This is achieved by pushing important information about the product the customer is interested in to his or her mobile device (e.g. a PDA[1] or a cell phone). The software automates the product research by accessing the Internet remotely from an external server, gathering information about the desired product, which is then analyzed by an expert system. The latter takes into account the user's situation, product data, feature descriptions, other customers' evaluations and expert opinions. The result of the search, in the form of product information and recommendations, is then pushed to the customer's cell phone or PDA in condensed form, giving an overview of the most important facts that help make an informed decision on behalf of the customer.

An expert system is an application that combines domain-specific knowledge with an inference engine in order to infer new knowledge from the data stored in a knowledge-base and from additional input. It is an interesting subfield of Artificial Intelligence, because it is concerned with the design and implementation of programs capable of emulating human cognitive skills (Jackson, 1998). "An expert system is a computer program intended to embody the knowledge and ability of an expert in a certain domain" (McCarthy, 1984).

Hence, the main goal of such a system is to incorporate human expertise in a computer, so it can be used for all kinds of problem-solving tasks. One such task is reasoning about purchasing decisions, given certain information about the buyer, the situation and the product

[1]Personal Digital Assistant, a hand held computer that fits in the palm of one's hand

itself. The user is assumed to provide some information about his or her situation and intentions and the expert system will use this knowledge and the information retrieved from the Internet (i.e. general product information, customer reviews and evaluations, expert opinions and ratings, etc.) to provide an individual report on the product or to guide the shopper through the process, like some sort of virtual salesperson.

This task is considered to be very difficult, since it is hard to know the individual customer's belief system and to make accurate predictions on what is important to him or her and consequently, what the best product recommendation would be. The domain is very fuzzy and not clear cut as opposed to most traditional expert systems. Expert Systems that are realized as production systems are very peculiar in the way they are implemented and differ from traditional algorithmic approaches. Usually, there are certain preconditions to build an expert system, which Prerau explains in detail in an article about when it is appropriate to build such a system (Prerau, 1985). If these conditions are not met for a certain domain, then this approach is usually considered not suitable for such a domain, even though theoretically, "every formal system can be realized as a canonical system" (Jackson, 1998). Shopping is a very subjective matter, which makes it hard to find a qualified expert from whom one can extract the necessary knowledge on how to identify customer types and their needs. Nevertheless, most of us are taught how to spend money wisely and certain key concepts of successful shopping are common sense. For example, if 99% of all people who ever bought a certain product mention that they don't like it at all and feel sorry having spent money on it, common sense tells one that it probably wouldn't be wise to buy it. Of course this is a very simplistic example of "shopping expertise", but with the current number of rules identifying such key concepts in the current version of the Expert System, the resulting computerized educated guesses have led to promising results.

1.2 Consumer Behavior

The challenge for most people is to be well-informed when walking into a retail store. Intuitively, everyone wants to know all relevant facts about a certain product before buying it. But nowadays, people are overwhelmed by the huge amount of information that is made available to them: product descriptions, technical terms, acronyms – something that big retail chains like Best Buy internally refer to as "feature vomiting" (Sims, 2004). As a consequence, customers have to deal with an extremely large amount of information that they are not ready to "digest" in a short amount of time. In order to cope with these problems, people find other ways to access information that they feel plays a crucial role in their purchase decision – the Internet, an abundant source of information regarding all kinds of products. However, online product research often cannot replace the experience one has while interacting with the actual product. Most people will want to actually see the item before buying it. After all, who wants to buy a pig in a poke?

It turns out that going to a retail store is still an important step in the whole shopping cycle for a couple of reasons. For one, the service personnel can help to find out about one's more specific needs, especially if one considers well-trained salespeople that know how to get to this kind of information by asking precise questions. Secondly, even people that are comfortable with using the Internet as a major source of information still want to see the product in real life before buying it (Jupiter, 2002). This gave rise to a phenomenon that Forrester Research calls "cross-channel shopping" (Forrester, 2004). A customer falling into this category usually does extensive online product research, before going to a retail store to make the purchase (after having interacted with the product). The number of people who visit a regular "brick store" after doing online product research, and who are feeling well-informed and knowledgeable due to the acquired *online knowledge*, "remains steady at 75%" (Annenberg, 2004). Another variant of this goes the other way round: people who did some online research, went to a store to interact with the product and then made the purchase back home at an online store. "69% of U.S. online shoppers admit to browsing in

traditional stores before buying over the Internet" (Annenberg, 2004). Naturally, customers of the latter kind have recently been marked as "evil customers" by big retail chains like Best Buy, because they just *use* their facilities without ever *buying* anything in their stores. As a consequence, this group is considered to be part of the 20% of customers Best Buy wishes to ban from their stores (Freed, 2004) through various ways (e.g. by increasing restocking fees to prevent people from "borrowing" a product for a while to test it at home and then returning it, actually buying it from a cheaper online store).

1.3 Related Work

Information about a product is one of the most important factors in a customer's decision-making process. Apart from general sources of information, like advertisements, articles in magazines and newspapers and word-of-mouth, merchants are putting much effort into informing their customers in a better way. In fact, the need for such information is large enough that various third parties also address it.

Apart from intelligent store layout and cleverly placed advertisements and information booths, a few retail chains have started to experiment with electronic aids that guide the shopper through the store, providing useful facts about the products on sale. This is usually realized as computerized information kiosks that are placed in strategically good locations throughout the store. They are intended to give access to information about the products that are on sale or about certain brands in general. The idea is that these kiosks are used in order to get more detailed product descriptions and features, which a salesperson normally couldn't memorize or present accurately. Some approaches even grant access to the Internet (e.g. Comcast Internet Kiosks in some consumer electronics stores), which could be used by the customer to do some product research online. Other stores provide the customers with electronic handheld devices that can be used to retrieve information about a specific product, identifying it by scanning its bar code. While much effort has been spent on improving the overall shopping experience, not much has been done in terms of bringing knowledge from the

outside of the store to the customer while he or she is shopping *in* the store. Most attempts to inform the customer in new ways are misleading in some way and shouldn't be pursued on a long-term basis. The most popular approach was undertaken by Google. They recently added a feature that allows getting up to date price information via text messaging to one's cell phone. That is, a customer can send a text message containing the product name to Google and immediately get back the cheapest price for this product that was found online. The major flaw of this idea is the fact that it focuses only on price, which will lead to bad decisions much of the time. It seems to be very easy for an unreliable Web shop with bad service and no return policy to offer extremely low prices just to draw customers to its Web site. The Google service as it is right now offers no protection against such situations, like a trust-based ranking of the stores. Moreover, it fails to take into account the differences among the consumers perception of importance of information (Yuan, 2002). Information on different prices might be an important factor in choosing the right store to purchase the product, but is less likely to help with evaluating the actual utility for one's personal situation.

Another related field is dealing with giving people advice on the general product category, rather than providing information for a specific product. This type of service can be found on the Internet, where more and more companies and a couple of third party organizations are offering so-called "recommender bots". The idea is that the customer enters some information about individual needs and preferences and is presented with a small selection of suitable products. In general, this seems to be very useful. However, the user is forced to do this at home, neglecting the need for interacting with the actual product. The approach is thus not getting rid of the issue that the information is not available to the customer while being in the store. Moreover, the customer would still have to do additional product research himself once the "bot" found the best suited item for this particular customer.

Finally, there are first attempts to enhance the decision making process within a store using mobile devices as a platform. A reasonable approach to provide the customer with

more than just price information is to compute a "product attractiveness cue" based on the decision matrix for a given product category (van der Heijden, 2002). A decision matrix contains a list of products in rows with each of its attributes listed in columns. The cue is supposed to help the customer make a decision regarding a set of product alternatives. The main issue with this approach is that it makes no effort in bringing information from the outside of the store, e.g. the Internet, into the hands of the customers "on the go".

1.4 Conclusion

All in all, the current status quo is not very satisfying. On the one hand, we are confronted with an ever-growing source of information in the form of the Internet. But on the other hand, we are also overwhelmed by the sheer amount of available data and are looking for better ways to deal with it. Consequently, most people with online expertise who are shopping for something are cross-channeling. But this seems to be a hassle first and foremost for the customers, because they have to *commute* between two worlds – the "online world" and the retail environment – in order to be really informed and not as likely to make a wrong decision anymore. However, in the "online world", the customers are *out of context*, since they don't have access to the actual product. In the retail environment on the other hand, they are *out of context*, too, because they are disconnected from the Web and don't have access to this kind of knowledge anymore. The natural thing to do seems to merge these two environments in order to serve the customer's needs better. Hence the need for a decision support tool that can be accessed from a mobile device (van der Heijden, 2002). *Decisions To Go* is an attempt to do so and this paper describes the research and development that has been done to achieve these goals.

Developing such a tool confronts one with many issues that will be discussed in the next chapters. The results were very encouraging and the software was able to provide accurate reports and recommendations based on different scenarios and products that are further explained in chapter 5.

Chapter 2

System Architecture

2.1 Overview

Developing for mobile devices is a challenge that confronts one with several problems and constraints. Cell phones and PDAs or Smartphones have limited capabilities when it comes to memory requirements, computational power and user interaction.

This led to two conclusions. For one, running a full-blown intelligent decision support system on a mobile device is unlikely to succeed, mainly because of the computational requirements that would exceed the capabilities of a CPU on a regular cell phone. Secondly, since it is not easy to browse the Web on a cell phone, no one is willing to do any kind of manual product research on the go with such a device. Moreover, reading from a small screen in noisy environments with bad lighting is a big hassle. Using the Web features on a cell phone will probably never go beyond activities that require less than pushing 4-5 buttons at most.

Hence *Decisions To Go*'s infrastructure is based on an external intelligent server system that communicates with a mobile device through its wireless internet connection. For the PDA, the solution was a Web portal that can be accessed easily from the device's Web browser. The Web site itself was laid out in a way that reduced textual input, the biggest hassle of mobile Web browsing, to a minimum. The cell phone on the other hand poses a bigger problem, since for most hand sets, the Web sites have to be in a specific format based on the Wireless Markup Language (WML). Moreover, browsing the Web on these devices is not very popular, due to speed constraints and issues with the user interface on a cell phone. Many people are alienated by the complicated input method through the cell phones keypad

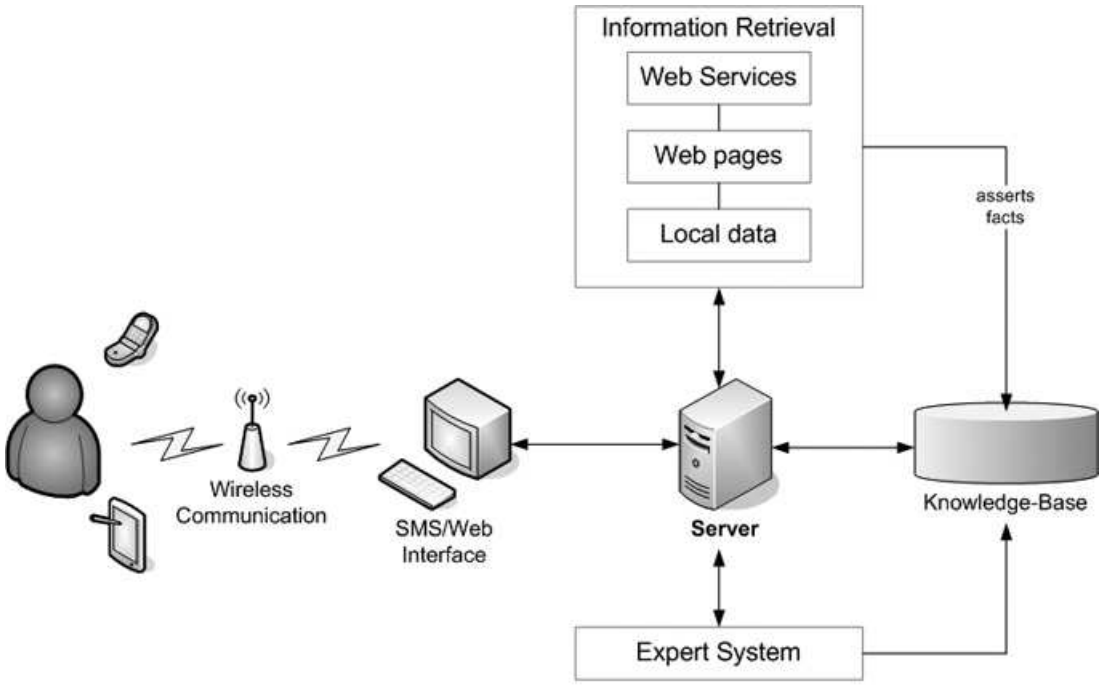

Figure 2.1: System architecture

and won't even try Web browsing to begin with. Therefore, the content had to be pushed to the customer's cell phone in a certain way, which will be further explained in section 2.5.

The general idea behind *Decisions To Go* is that the user queries the system, requesting either information about a specific product or help with finding the right product from a general product category. The former will put the system into Automated Product Research (ARP) mode, whereas the latter will let the user access the Virtual Salesperson (VSP). The server is waiting for the initial contact for either type of query and then responds accordingly. After all information is gathered and analyzed, the system will dynamically create a summary report that is sent back to the user. This can be either an information page for a specific

product (ARP) or a list of recommended products in case the user is looking for general guidance on a product category (VSP).

2.1.1 Automated Product Research

For ARP type requests, the server parses the product name that is included in the query, which will allow it to identify the product in the data base. It can then start the information retrieval module in order to access all sources of information known to the system about this specific product. These sources are predefined entry points to the online world. This includes Web services that give access to product databases, Web sites that can be automatically parsed or retrieving data from a local database. The inner workings of this module are further explained in section 2.2. The result of the information retrieval phase is stored in a temporary container for the query. Then, the expert system is invoked in order to identify the user's needs and current situation. The goal is to come up with a "concept" of the user, which has a great impact on how the report is generated, focusing on what information the expert system thinks matters most to this particular type of user and the situation he or she is in.

2.1.2 Virtual Salesperson

The VSP mode follows a different strategy. The server will parse the incoming request in order to identify the product category and then invoke the expert system to find out more about the user's needs. In other words, the expert system acts like a virtual salesperson by asking typical questions that go from general to specific until the search space is narrowed down sufficiently to make an accurate prediction on what product or products will suit this particular user's needs the best way. The questions are supposed to reveal the user's background and the general circumstances of the intended purchase. For each category, the server maintains basic information about the items in that category (e.g. features, price points, etc.). This serves as the foundation for the reasoning process of the virtual salesperson.

In the end, it will come up with a list of recommendations that the system believes will match the user's needs best. Upon selecting any of these, the Automated Product Research module is activated in order to gather more detailed information about the chosen product and redirect the user to the summary report generated for it.

2.2 Information Retrieval

The information retrieval module, once activated, gathers information following a predefined pattern. There are several sources of information that are known to the system and from where it retrieves all the data that is used in the process. The system is actively looking for specific data. For example, "price" is an important factor in a purchasing decision on behalf of the customer. Hence, the information retrieval is storing all the different prices for a product that are found during the search in order to come up with a price range that can be included in the report and the reasoning process (e.g. to exclude certain products from the report if it is thought to be too expensive for this type of customer or the specific need he or she has, as determined by the expert system). All the results are stored in a temporary container, which can be later accessed by the other modules of the software. The expert system module will include this information later in the reasoning process in order to determine what pieces of information are used by the report generator module.

As mentioned before, the Internet is an abundant source of information. The real challenge is to sift through the data in a meaningful way. It is especially difficult to analyze human language or simple HTML Web pages that have no semantic structure to them. Therefore, the emphasis for this project was on Web content that contained at least some semantic markup. Web services play a crucial role in distributing information in this way. Big companies like Amazon and Ebay are opening the doors to their product and customer databases in order to facilitate the creation of third party tools to use their Web shops. The Web service approach makes it easy to identify different pieces of information, such that all the data can be processed automatically. For each product, we can easily get the average customer rating,

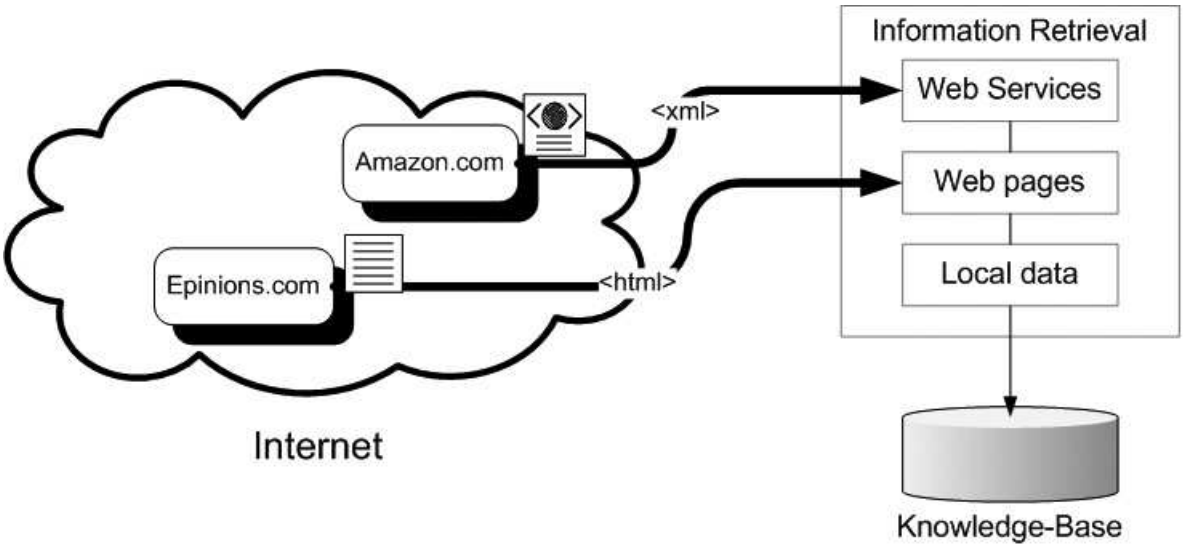

Figure 2.2: Information Retrieval module

the price for a new product, the average price for a used product, and so forth. All information that can be found manually on Amazon's Web page can be retrieved automatically by our information retrieval module and then further processed within the intelligent decision support system.

Another method to retrieve certain types of information is parsing public product review Web sites like *Epinions.com* or *BizRate.com*. The product information sheets for each product are laid out following a generic pattern, making it easy to retrieve key elements that are considered useful. In contrast to Amazon's Web service, Web sites like Epinions.com offer price comparisons and customer ratings and reviews even for different Web shops. This allows a more bias-free overview of how other customers have evaluated a specific product and gives a general idea about what the price range looks like. The challenge with this approach is that not all product review Web sites provide access to their data, protecting their reports through various mechanisms that will be discussed in chapter 3.

The last resort for searching for information is manually looking for it and storing it in a local product database. This can make sense for certain types of data (e.g. feature list), but would require manual updates of the database and is therefore not desired.

After all the information has been acquired, it is stored in our knowledge-base, which is in turn accessed by the expert system and report generator. Chapter 3 describes the information retrieval process in more detail.

2.3 Expert System

The other main pillar of the prototype is the Expert System module. Its purpose is to make sense of all the information that is related to the particular product or product category that the user is inquiring about. This not only entails product specific information like prices, features and average ratings, but also general information about the customer, the product domain (e.g. "television sets" or "portable audio players") and the act of shopping in general. The latter includes knowledge that an experienced shopper would have about making good decisions.

Therefore, the main focus of *Decisions To Go* lies on analyzing the currently available information about these categories in order to figure out what part of the data can be considered important and useful for further analysis and reporting. The aim is to inform the customer the best possible way. This is especially important for the automated product research, for which the expert system has to incorporate rules that reflect general shopping knowledge, knowledge that is specific to the product category and knowledge about consumer behavior that lead to a good prediction of which target group the user most likely belongs to. This is important, since the preferences of each individual user might differ noticeably and, on a similar note, their interest in different kinds of information will vary. The shopping guide, or virtual salesperson, on the other hand is going a step further by incorporating additional knowledge about the market, as well as the target customers for certain product categories. In order to find out more about the customer seeking advice regarding a general product domain,

the expert system is asking several questions about personal preferences. These questions go from general to specific, as the search space for appropriate solutions is narrowed down. Eventually, the expert system will be able to come up with a product recommendation. This is currently done by providing a selection of the three products considered most suitable, given the information volunteered by the customer as well as information acquired from the Web and local databases.

In both cases, the overall goal is to perform an intelligent online product research *for* the customer, both reducing the time to do this type of research to a minimum and at the same time providing all necessary information on the go. This calls for a comprehensive summary report that contains all key elements of traditional, manual product research done at home. Hence the software agent doing the search has to identify all the important information and facts, and present it to the user in a condensed form that can be understood at a glance. In case of general shopping advice, we had to make sure that the virtual salesperson is recommending the right products, predicting the user's preferences accurately and matching them to the current situation on the respective market.

The expert system is realized as a rule-based production system in Java. Information about the current state of the environment (in the form of facts about the product, customer and market) is induced into the system's knowledge-base where it causes rules to fire and put forward the reasoning process. The inference engine is based on a stacked forward chaining approach. Instead of sub-goaling from each possible outcome to the fundamental facts (backward-chaining) or deriving the global goals from initial knowledge (forward-chaining), *Decision To Go*'s expert system is laid out to work in three major tiers of reasoning. Each tier represents a degree of specificity regarding the possible solutions for the initial query. The first tier represents the attempt to lead the search in the right way from a general perspective. The second tier is accessed once the general direction becomes clearer in the reasoning process: here, the system is narrowing down the search space until the confidence is high enough to access the third and final tier leading to specific product recom-

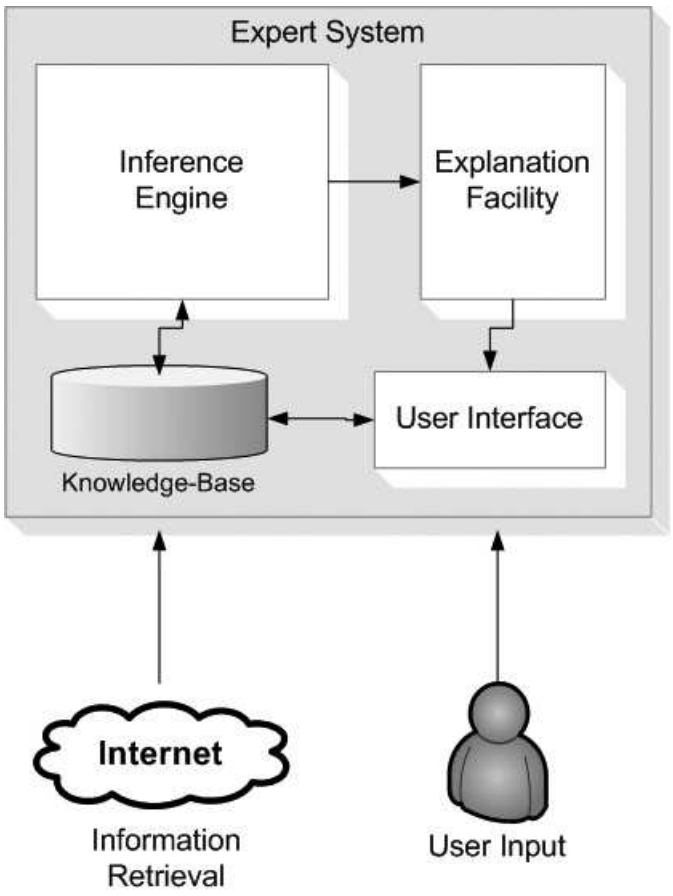

Figure 2.3: Expert System module

mendations as outcomes to the initial query. This approach might be called *stacked* forward chaining, since there are two levels of abstraction, both working forward in the reasoning process.

The way certain information is gathered depends on which mode the user chose for the query. In the case of an automated product research query for a specific item, the user profile is retrieved from the local database in order to perform basic reasoning about the user's individual situation. After that, further analysis of the known facts is done, which

eventually yields certain guidelines in laying out the summary report for the individual user. A more comprehensive phase of acquiring important facts about the user's preferences is necessary for the virtual salesperson mode. Here, the expert system is asking the user a number of questions for each tier. What questions are asked in each step heavily depends on the current accumulated knowledge based on the previous answers of the customer and his or her profile. Chapter 4 will describe this in great detail as well as go over the issues with reasoning under uncertainty and missing information.

2.4 User Interface

The design for a portable device like a cell phone or a personal digital assistant (PDA) has to take into account the limitations that come with these platforms. The main issues that had to be addressed during the development of *Decisions To Go* were mostly restrictions regarding computational power and the communication channel bottleneck: the wireless internet connection. Furthermore, interacting with a small screen device is drastically different from the experience one has with a regular personal computer, which called for a whole different design philosophy altogether.

These issues were solved in slightly different ways for PDAs and cell phones. Being both equipped with a rather small screen, the graphical user interface (GUI) was designed to be intuitive and easy to use in mobile environments. Textual input and user interaction were reduced to a minimum, focusing on delivering the required information about products as quick as possible. Regarding slow internet connections, the solution was to push only small pieces of information at a time, instead of providing the full report at once. Finally, the problem with limited computational resources on a mobile device was dodged altogether, by moving the artificial intelligence component completely to a remote intelligent server system.

2.4.1 Smartphones and PDAs

A Smartphone is usually seen as a handheld device that merges cellular phone capabilities and personal information management features into one device. This can be either adding voice communication features to a PDA or combining PDA functionality with an already existing cell phone. There are several operating systems, each of which supports a Web browser able to interpret standard HTML, which is needed for accessing *Decisions To Go*'s user interface for PDAs and Smartphones. This is important, because the interface for these devices is based on a Web page, that has been specifically designed for the PDA screen size. The remote intelligent server system was completely written in Java. Thus, the natural thing to do was to implement the user interface as Java Server Pages (JSP) that allowed us to easily connect the user interface to the Java-based expert system and information retrieval modules. For each user, a separate Java Bean is created to track a single query that carries all information about the user and the results from the automated product research or the virtual shopping guide, respectively. The report is then provided as a dynamically created Web page based on the results of the search and the expert systems recommendations.

The summary report page is divided into a number of sections (see figure 2.4). Each section represents a space that can be filled with content. Due to the way the human mind acquires information visually, the order in which the results are presented inherently reflects the hierarchy of importance. For example, the top left box represents a very important piece of information, whereas the bottom right corner is considered least important. The screen is further divided into three major sections: the title (holding the product name), the text column (for textual information like customer reviews or expert opinions) and the facts column (for simple facts like price, average rating etc.). Which box is filled with what content is decided after gathering and analyzing the available information.

The virtual salesperson works differently. During the process, the user is asked a number of questions similar to the ones shown in figure 2.5. Each question has been selected by the expert system based on what it knows about the user's preferences already. As described

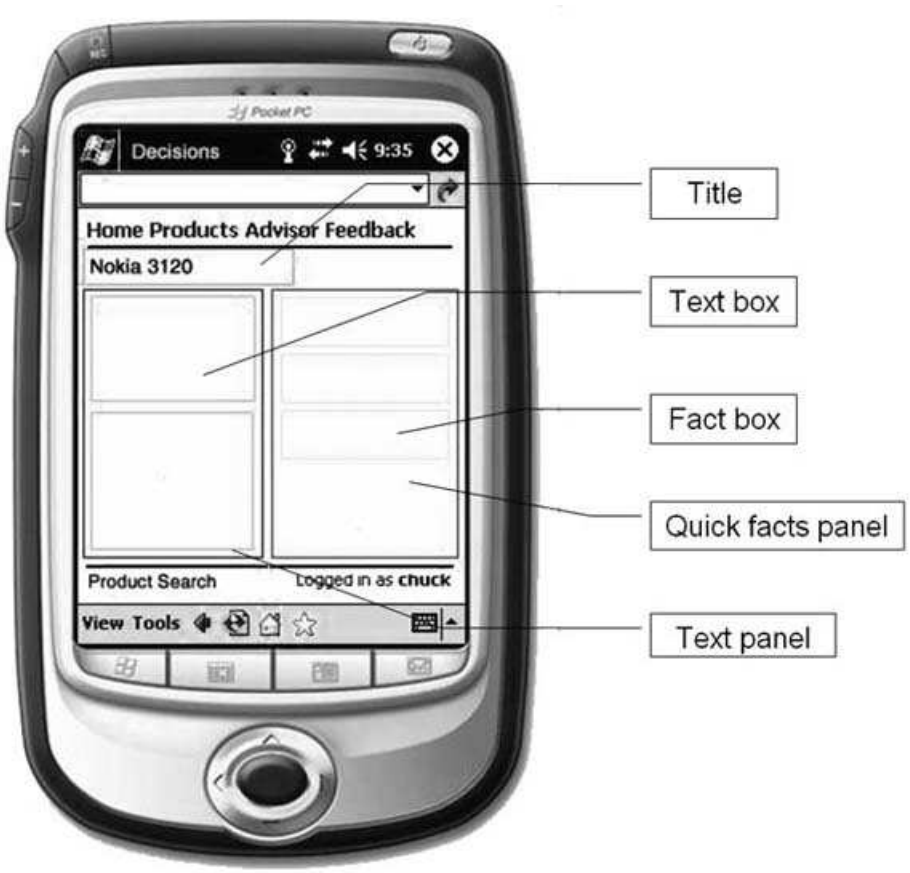

Figure 2.4: User interface on a PDA with summary report outline

briefly in the previous section, the expert system is working its way from general to specific, narrowing down the search space until it is confident enough to recommend a selection of products most likely to suit the user's needs best. Therefore, the questions are carefully selected in each step in order to avoid asking unnecessary ones. In addition to that, each question comes with a set of possible answers that are ordered in a particular way to reflect the system's current best guess about the user's preferences. In addition to that, as figure 2.5 shows, the answers to the question about preferred coverage are pre-ordered: a local plan is most likely to be the best choice for that user based on the current evidence. The purpose

Figure 2.5: User interface on a PDA with virtual salesperson showing

of this is to induce a small bias towards answering the question that way in order to help the user in situations where he or she doesn't know exactly what to answer. Research has shown that people are most likely to select the first possible answer in those cases.

Answering all the questions the VSP asks will eventually lead to a list of recommended products in this category. Upon selecting one of the recommendations, the automated product research module is activated for this particular product and the user is redirected to the resulting summary report page.

2.4.2 Cell Phones

Compared to the user interface developed for PDAs and Smartphones, cell phones pose a much bigger problem regarding the communication between the device itself and the remote server as well as in terms of user interaction. Most modern devices allow a mobile Internet connection over the cellular network, but apart from being very slow, most included Web browsers don't support simple HTML pages. The most common way to display Internet content on cell phones is to write Web sites in a special format following the Wireless Application Protocol (WAP). Even though WAP supports a form of Java script (i.e. Wireless Script Language, WSL), it lacks the possibility to create more complex Web applications like the one developed for accessing of *Decisions To Go* from a PDA. The question therefore is how to push content to cell phones without putting too much burden on the user.

The main issue with mobile Web browsing on cell phones has always been the inconvenience with which it takes place. Entering URLs on a keypad is a hassle very few are willing to go through. The alternative to that are Web portals that most carriers offer. However, these portals limit the user in experiencing the Web, because Web sites not available through the portal still have to be accessed manually. Therefore, users generally have to rely on the initial selection of bookmarks and are most likely to be unwilling to add new ones.

In the current version of the prototype, the user can initiate the automated product research via text messaging, also known as Short Message Service (SMS). SMS is a set of protocols deployed by most carriers that allows the transmission of short text messages containing plain text of up to 160 Bytes. This enables the user to send the product name to our server, where it is further processed. The system recognizes it as a text message (as opposed to a Web request as it would come from a PDA or Smartphone) and treats it in a slightly different way. After retrieving the product name from the message, the automated product research can be initiated as usual. The only difference is that the resulting summary report is split into multiple parts and sent back to the user's cell phone as a series of text

Figure 2.6: Communicating with the user via text messaging

messages containing the most important facts about the product he or she intends to buy (see figure 2.6).

Due to the limitations of this communication channel the virtual salesperson cannot be accessed from a cell phone. In future versions of this prototype, interaction with a cell phone might be realized using a transmission technology called "WAP Push", which basically is a simple SMS containing a binary encoded message that any WAP-enabled phone can decode to a WAP Web page. This Web page could be the starting point of the virtual salesperson asking questions and sending another WAP Push message for each new question until coming to a conclusion, effectively realizing the virtual salesperson mode for cell phones.

2.5 Wireless Communication

The easiest way to communicate with a mobile handheld device is over WiFi, a set of wireless communication protocols, most notably used in modern wireless networks (WLAN). This can be used to set up an Internet connection on a WiFi-enabled PDA or Smartphone. Hence, connecting to our server is very straightforward. If no wireless networks are available, Smartphones can usually connect to the Internet over a cellular connection. It is important to note that the main advantage of PDAs and Smartphones is the HTML Web browser that is pre-installed on those devices. In the end, this enables the user to access our Web application in a direct manner, by browsing to the server's URL (i.e. Web address).

Cell phones are usually restricted to a cellular Internet connection. However, as mentioned in the previous section, the browser is expecting the Web sites to be in a special format that prevents one from developing a complex and highly dynamic Web application. In addition to that, most cell phone subscribers choose not to include a "data plan" in their contract, which allows mobile internet access, and are most likely limited to text messaging only.

In the current version of the prototype, text messages from the server are sent as emails to the user's cell phone. All major carriers have deployed email gateways in their networks, over which emails can be sent directly to a cell phone destination for free, although the latency is significantly higher than directly sending a text message (SMS) to the cell phone. Since this would be a costly operation, the prototype was realized using email gateways rather than direct SMS.

Chapter 3

Information Retrieval

3.1 Overview

The main purpose of the prototype is to provide product information on a mobile platform for users on the go. This requires first and foremost accessing the Internet, a vast source of information, in a way that automates the product research for the user. Moreover, we want to produce meaningful reports about the information that was gathered online in order to enable the user to be well-informed before making a purchase decision. These summary reports are created by the report generator of the expert system incorporated in the intelligent server, which will be further discussed in chapter 4.

In the traditional sense, information retrieval often means nothing more than the retrieval of objects holding information by the means of search (Lancaster, 1968). For this prototype, we focus on a very limited search on product information in three sources: Amazon.com, Epinions.com and a local database containing specific product data on a number of products. The first Web site is widely known for its numerous product reviews and evaluations, and for being a platform for exchanging opinions on almost any given product in the form of customer reviews and product ratings. The second Web site also serves as a customer review forum, with a more extensive overview on prices and ratings from a greater number of different stores, thus not only focusing on Amazon's customer base. In addition to that, certain information was gathered manually for the products that were included in the tests of the prototype's functionality to cross-check the data retrieved from online sources. For this purpose, the name and the general product category for each individual product were stored in a local database.

Each of these sources of information presents the available data and information in a different format. Amazon offers access to its product and customer review data bases through a Web service. A Web service is a set of self-contained functions that can be executed from anywhere in the Web using an XML-based protocol. The Amazon Web Services (AWS) offer methods for directly accessing their databases on product data, including, but not limited to, information about products, features, prices, average ratings and even complete customer reviews. Epinions on the other hand doesn't provide a Web service to access their data. The Web site is completely displayed in plain HTML, making it harder to extract information from it (Agosti and Smeaton, 1996). The information retrieval module loads the Web sites automatically upon receiving a query and analyzes the report page line by line, searching for certain keywords that have been previously determined to guide the search towards some important facts usually contained in the summary. This allows us to retrieve and store certain facts automatically, e.g. prices from different online stores, average customer rating, number of people who rated this product and so forth. Finally, the local database is accessed for retrieving the product specific data (i.e. the name and product category) in order to check if the query is valid.

It is important to note that each of these tasks can be handled with Java. It allows easy Web service integration through a number of class libraries dealing with the relevant protocols, including a powerful XML parser. Moreover, Java can be enabled to issue HTTP commands necessary for accessing simple Web sites as if a human user was browsing them manually. This is important because many public product review Web sites now are trying to prevent software agents from accessing their Web sites to protect their intellectual property. In addition to that, Java provides the necessary facilities to parse large text files in order to look for keywords. Finally, it includes a simple interface for a standard SQL database like the one that was used for this prototype to store other product information locally. Consequently, the module has been written completely in Java and incorporated into the intelligent server system.

The main goal of the information retrieval module is to access a given source of information to extract all the available data from it. The information is then adjusted to match a general format that is accepted by the other modules of the system (i.e. the server, the expert system and the report generator). Hence, the system is expecting the data to be in a certain format, although with varying degree of strictness. For example, price or rating information is always considered to be a crisp number. However, an opinion or customer review can just be any kind of text. All the information that has been retrieved for a user query on a specific product is then stored in a temporary container for later use. Temporary, because the information available online is highly volatile and subject to daily changes. This forces us to make sure to always retrieve the most recent data. In addition to that, the information retrieval module also asserts certain facts to the expert system's knowledge-base that have been extracted from the raw material.

3.2 Web Services

In general, Web services are used for exchanging data between applications on remote systems. Therefore they are widely used by companies who want to distribute information over the Web in a standardized form. The protocols used for handling the data transfer are XML-based, making it particularly easy to develop applications that integrate a Web service into their program flow. The idea is that the interface of a Web service is described in a publicly accessible file written in the Web Service Description Language (WSDL), which lists all usable function calls the client can issue in order to get certain information back. The client-side application can create queries using any of these methods.

These queries are sent to Amazon's Web service as a Representational State Transfer (REST) request, which basically encodes the requested method call into an HTTP post command. In other words, the system loads an URL on the Web service server of Amazon and passes a number of parameters in a query string. A query string is a set of parameters, delimited by an ampersand (see below). For example, to get the attributes of a specific

product, the system has to request the unique product ID from Amazon using the product name as the identifier. The response is searched for the unique product ID number. Once it is retrieved, our server can issue the actual request for looking up the item itself, which yields, among other things, the price, average rating and so forth, depending on the commands included in the query to the Web service. The main attributes of this particular product are retrieved by issuing the following request:

```
http://webservices.amazon.com/onca/xml?Service=AWSECommerceService
    &SubscriptionId=1234
    &AssociateTag=0000
    &Operation=ItemLookup
    &ItemId=B00008OE6I
    &ResponseGroup=ItemAttributes
```

This query performs the command "ItemLookup", as defined by the WSDL, for a specific product and limits the response to the item's attributes. The result is sent back to our server as an XML file that can be easily parsed due to the semantic mark-up that comes with it:

```
<Response>
...
<ItemAttributes>
   <Feature>4.0 megapixel sensor</Feature>
   <Feature>3x optical zoom</Feature>
   <Feature>Stainless steel shell</Feature>
   <Feature>32 MB card included</Feature>
   <Feature>lithium-ion battery (NB-1LH )</Feature>
   <ListPrice>
     <FormattedPrice>$449.99</FormattedPrice>
   </ListPrice>
   <Manufacturer>Canon Cameras US</Manufacturer>
</ItemAttributes>
...
</Response>
```

Since all the data are explicitly labeled, it is quite easy to store it in the appropriate fields of the temporary container. Furthermore, some of the data are also transformed into facts that can be asserted to the knowledge-base, so that it can be used during the reasoning

process. The program maintains a list of facts that are considered useful for the reasoning process, e.g. “price” or “customer rating”. If the information retrieval yields such a piece of information, it is entered into the knowledge-base. Adding these kinds of facts activates the reasoning process, as more and more rules can fire with more data becoming available. The way this influences the summary report is described in chapter 4. It is important to note that all the retrieved data are stored in a temporary container, specifically created for each individual request and discarded after the session expires.

3.3 Parsing Web Sites

Most Web sites providing information on products are not as easily accessible as Amazon’s product database, due to the lack of an appropriate interface, like a Web service, to automatically gather data. In those cases, the Web site is retrieved in simple HTML format, which makes it harder to analyze. However, Web pages are usually well-structured and can be described by a formal grammar. Knowing the structure of the summary report pages for such a Web site enables the information retrieval module to extract certain information by searching the HTML documents for keywords and certain constructs (e.g. a certain table cell holding price information), without having to rely on natural language processing (Knoblock, 1998).

However, more and more third party product review Web sites are closing their doors for automatic information retrieval agents. One way this is done is randomly generating a code that is added to the URL of the summary report page and linked to a manual search query. If this code is omitted, the Web server will display a blank page, making it impossible for a software agent to retrieve the otherwise available information. Therefore, the Web site parser realized for this prototype simulates human interaction with the target Web site to get to the report page for a particular product. Using Apache’s Jakarta Commons HTTP Client for Java, the system operates the search input mask as if a human would enter a product name

or keyword manually. For example, to get information about a popular PDA, the system has to access the Web page the following way:

```
HttpMethod m = new PostMethod("...search/?search_string=ipaq+hx4705);
Byte[] b = m.getResponseBody();
```

On Epinions.com, the Web site targeted in the current version of the prototype, this leads to an overview page where possible matches are listed. The information retrieval module then parses the titles contained in the `ResponseBody` Byte array and access the correct result page by matching the product name from the user query to one of the search results. It can then go about loading the HTML document and parsing it line by line, searching for keywords that have been previously determined to lead to important facts that are usually contained in the summary. The keywords and constructs that the parser is looking for have been determined during an extensive analysis of the general structure of Epinions.com's report pages. Examples are:

- "Product Rating:" + number

  Average product rating

- "Reviewed by" + number

  Number of people this rating is based on

- "Lowest base price " + number

  Lowest price found

- "prc=" + number

  Individual store's price and name of the store

After retrieving these key elements from the initial page, the information retrieval module can go on to access the report's customer review page, which contains further information on how

the product was evaluated by other users. The parser is looking for additional information on that page. In this case, it is searching for a specific construct in the Web site: a table titled "product rating" that contains a list of four key features that are considered most important for the product category. For example, in the domain of cell phones, Epinions.com lists individual ratings on durability, clarity of sound, portability and battery life. Each of these ratings can be retrieved from this table by searching for the table cells that contain the respective keyword.

Since the parser is looking explicitly for predefined content, it is quite easy to break down the information and put it into the right fields of the temporary container. In addition to that, certain data are transformed into facts that can be asserted to the knowledge-base, namely the domain-specific feature ratings, the lowest price and the ratio of product rating and number of people who rated the product. The latter results in a more meaningful measure of how important the rating really is. If only 2 people rated the product, the resulting number is not very useful due to the lack of supporting evidence (i.e. we cannot conclude that the product really deserves that rating based on the opinions of two people). If, however, 100 people rated the product so far, the number will be of greater significance.

Finally, it is worth mentioning that the barriers the operators of these kinds of Web sites put into their system are getting more sophisticated over time. Since they get their revenue by redirecting people to online stores, they don't want other parties to access their valuable information. This may lead to the conclusion that this kind of research will only be possible with the Web site's owner's explicit consent, in form of a public interface like a Web service.

3.4 Collecting Information Manually

Some of the product data can be easily retrieved manually for new products that are coming to market. The reason for doing this is to make sure the information retrieval module can supervise the search process and to make sure all queries are valid. Therefore, for the sake of making the prototype more robust to testing, we manually gathered a list of products

that can be queried. The information on these products includes their names and categories, which are stored in a local database. When the user sends a query to the system, the product is looked up and the actual information retrieval module is started. This is currently in use to prevent searches on non-existent or misspelled products, although in future versions of the prototype, Amazon's Web service could be used to search for products.

3.5 Integrating Sources of Information

In order to simplify the process of storing information, the data retrieved from Web resources are forced into a general pattern. The system uses *information templates* to store each element, which makes the search for information very explicit and dependent on the expert knowledge of what information contributes to the decision making process. The information retrieval module looks specifically for this type of data and stores it in the respective fields. Some of the more explicit data like price and average rating are transformed into facts that are asserted to the knowledge-base to bring forward the reasoning process (see figure 3.1). In fact, the expert system and the information retrieval module work hand in hand, both incorporating knowledge about the process of doing product research. The expert system incorporates knowledge that deals with linking pieces of information together in order to come to accurate conclusions about what parts of the retrieved information are important. The information retrieval module on the other hand includes some implicit rules about what data to gather to begin with. Which data are considered important has been evaluated in the planning phase preceding the development of the actual prototype.

The information retrieval module is not using any natural language processing and therefore does not have a real understanding of the content of any of the sources of information used during the process. The data that is retrieved is analyzed and categorized based on the systematic knowledge previously incorporated into the system. The sources of information vary in the way they are accessible. A Web service is a perfect way of exchanging information between external providers like Amazon and the prototype, due to the way the data is

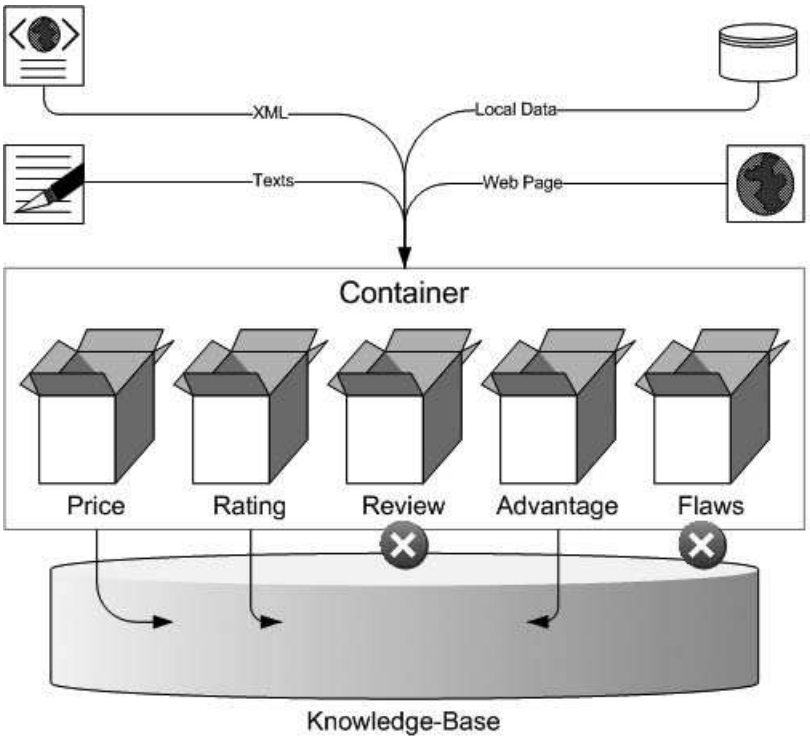

Figure 3.1: Storage of retrieved information

marked-up semantically. Parsing Web sites is much more complex, although a basic keyword search yields at least some important facts that add to the knowledge.

In order to make this module more powerful, another information retrieval agent with natural language processing capabilities would have to be added to evaluate customer reviews in a more meaningful way. Such a module could be used to extract the precise reasons why a particular customer liked or didn't like the product in question. Knowing these reasons would add another layer to the reasoning process in identifying opinions that are repeated throughout a large group of customers. If that was the case, the expert system could come to the conclusion that a given product has certain benefits and flaws that go beyond numeric ratings or feature descriptions.

Chapter 4

Expert System - Design and Implementation

4.1 Overview

The intelligent server program was entirely written in Java. Hence, the powerful Java Expert System Shell (JESS) was used to incorporate the expert system module. It was realized as a Java Bean, a program component that can be deployed on an application server - in our case the communication server interacting with the user's mobile device. Java Beans have the advantage of being reusable in multiple applications, thus acting as some sort of session object for each individual user query, each with its own expert system loaded into memory.

Once the user logs on to the system, a "Expert Bean" (i.e., a Java Bean containing an instance of the Expert System) is created and can be used either for automated product research or by the virtual salesperson. The former relies on the facts that were asserted to the knowledge-base by the information retrieval module uses the results in order to prepare the summary report (see figure 4.1). The latter takes the user's profile as a starting point and tries to come up with a set of reasonably good recommendations by asking the user a series of questions.

Each module of the intelligent server has the ability to access the current user's "Expert Bean" at any given time. This makes it easy to add new data whenever deemed necessary, by executing the appropriate JESS method, e.g.:

```
r.executeCommand("(assert (fact ...))");
```

Moreover, this means that any newly derived knowledge can be easily retrieved from anywhere in the program for the current query. In order to create a report for a specific product,

the system will first retrieve all information about it and assert the facts to the knowledge-base. Carefully designed rules in the expert system will fire and yield certain hypotheses about the product and the importance of individual facts. These results can then be used to assemble a meaningful report. The virtual salesperson on the other hand will interact with the user through a questionnaire, trying to gather more information about his or her preferences and the general circumstances of the intended purchase. These facts are also stored in the knowledge-base, eventually leading to a set of recommendations that the user can choose from. Upon selecting one of those, the automated product research is started, using all the information that was collected in the questionnaire phase. This is facilitated by the infrastructure design. The "Expert Bean", which contains the current state of working memory, is simply passed on to another module, where it is further processed. In this case, the information retrieval process will continue to add data to the knowledge-base, thus improving the report generation. The next sections will describe in more detail how this general concept was realized programmatically.

4.2 A Rule-Based System in JESS

The expert system was implemented as a rule-based system, which consists of a set of rules, an inference engine and a working memory. The rule-base holds the knowledge about the domain in the form of conditional rules (e.g. IF <condition> THEN <action>). The inference engine tests known facts that are currently stored in the working memory against the rules. If the conditions of a rule are matched by the known facts, the rule is said to fire and will perform an action. This can be to add new knowledge to the working memory or retract old knowledge. This process is continued until no further rules can be fired and the inference process comes to a halt or until new facts are added to the system, setting the process in motion again.

The Java Expert System Shell (JESS), the rule engine that was used for this project was developed by Sandia Corporation in an effort to provide an expert system entirely written

in Java to facilitate complex Java-based applications that incorporate a knowledge-based system. For this project, JESS is used as the reasoning part of the intelligent information retrieval module for evaluating the available information on products. This makes it easier to maintain a knowledge-base that is accessed by the main program whenever necessary. The rule base is in a format that is widely known as CLIPS (C Language Integrated Production System), which essentially incorporates the knowledge in a way that resembles human logic and is somewhat based on the popular programming language LISP. For example, we might want to express the knowledge that if a cell phone has a camera built in, it should provide a number of ways to transfer the pictures to the owner's home computer. Hence whether or not it has certain connectivity features becomes important regarding the purchase decision:

```
(defrule
    "If CELL PHONE has a camera, then connectivity features are important"
    (cell-phone-has-camera)
    =>
    (assert (connectivity-features-are-important))
)
```

The above rule contains a single fact as its condition (i.e. "cell-phone-has-camera") and a single action, which asserts a new fact if triggered. Even though this rule seems to be easy to understand, the facts are not well-structured and will lead to problems in more complex rule-bases. A better way to store facts is through the use of templates that enforce a uniform format for the factual knowledge that is asserted to the knowledge-base. This makes it easier to maintain a large set of rules, since all of them will contain facts that follow the same guidelines. In the expert system used by *Decisions To Go*, there are two types of fact templates. The first one holds a simple statement that has a description and a certainty factor (which are further explained in the next section):

```
(deftemplate stmt
    "Holds a statement as evidence"
    (slot desc)
    (slot cf (type FLOAT))
)
```

The second type is an object-value pair that can be used to store more complex information like pricing and average ratings:

```
(deftemplate fact
    "Holds a object-value pair as evidence"
    (slot obj)
    (slot val)
    (slot cf (type FLOAT))
)
```

When it comes to using these templates, each fact or statement that is added to the knowledge-base has to obey the predefined format. Each "slot" can be filled with data, representing the slot's designator (in the above example, a fact's first slot is filled with the object itself, the second slot holds the value of that object and the certainty factor for this fact is stored in the third slot):

```
(assert (obj price) (val 100) (cf 0.4))
```

Finally, a typical rule used in the underlying expert system has a number of statements or facts as its conditions and lists one or more actions in case it is fired. Most of the time, an action will add new knowledge to the working memory, causing other rules to fire until we derive a conclusion. For example, to express what features might be important for a user if he or she is of a young age is expressed in the following form:

```
(defrule tier-2-features
    "If user is young, then features are ranked..."
    (fact (obj user-age) (val young) (cf ?x))
    =>
    (assert (fact (obj features) (val voice) (cf (* ?x 0.5))))
    (assert (fact (obj features) (val message) (cf (* ?x 0.9))))
    (assert (fact (obj features) (val camera) (cf (* ?x 0.6))))
    (assert (fact (obj features) (val data) (cf (* ?x -0.8))))
)
```

This rule encodes our knowledge that a user who is considered young, is most likely to have great interest in the messaging capabilities of the cell phone he or she intends to

buy. In addition to that, there seems to be at least some interest in voice and camera features, but there is no evidence at all for pure data features like calendar, organizer and Web browser. Note that JESS rules can include function calls within a condition (e.g. the expression `(val ?x:(< ?x 100))` only matches if the fact's second slot contains a number less than 100.

Finally, the rule-base is integrated into the Java server application that provides the infrastructure to acquire all necessary data, either from various Web sites through the information retrieval module or directly from the user's input. The intelligent server therefore implements a control program for the expert system that asserts important facts (once available), and retrieves the newly derived knowledge from the knowledge-base later on in order to further process it (e.g. for generating a report). The control program connects all the different modules, such that the information retrieval can contribute the gathered information to the reasoning process in an easy way. JESS provides a comprehensive set of methods that can be used to do this. After having created an instance of the expert system (i.e. the "Expert Bean"), we can assert new knowledge by executing the appropriate command through the Java API. For example, in order to acknowledge the fact that the user is young with complete certainty, the following command would be executed to assert it to the knowledge-base:

```
r.executeCommand("(assert (fact (obj user-age) (val young) (cf 1.0)))");
```

Facts stored in the working memory can be accessed by retrieving the specific fact and grabbing each of its slot values for further processing. For example, the following routine displays a list of all known facts in Java:

```
for (Iterator i = this.r.listFacts(); i.hasNext();) {
    Fact f = (Fact) i.next();
    Value v1 = f.getSlotValue("stmt");
    Value v2 = f.getSlotValue("cf");
    System.out.print("Statement: " v1);
    System.out.println(" CF: " + v2);
}
```

This gives the Java side total control over how to deal with this data and allows the integration of various sources of information through a separate module. Information retrieved from accessing a Web service, a Web site or local database is broken down into a predefined format that is stored in the knowledge-base. This guarantees that the expert system is always operating on the same basis and doesn't have to deal with exceptions or erroneous data.

4.3 Dealing with Uncertainty

It is the very nature of the problem addressed in this work that the information the system gathers is most likely to be imperfect or incomplete at best. In addition to that, we cannot be absolutely confident that our rule-base is absolutely correct in every aspect either. In fact, the domain knowledge incorporated into the Expert System might not be complete or even inaccurate. This is especially the case when dealing with subjective theories concerning consumer behavior and product evaluation, which are seldom provable. It is considered subjective, because many people would probably rate a given product differently, simply because each and everyone has a different opinion based on individual past experience and knowledge. The expert system therefore mainly incorporates knowledge that is considered common sense or would come from an expert of the domain, but still has to find a way to express the varying degree of certainty for each individual rule.

The expert system developed for *Decisions To Go* uses MYCIN[1]-style certainty factors to address this problem. This type of certainty factor ranges from -1 to 1, expressing the confidence in a hypothesis or piece of evidence, where -1 indicates strong disbelief and 1 strong belief. The approach explored by MYCIN was to incorporate this kind of uncertainty measure into the rule itself. Therefore, each rule has a certainty factor attached to it, reflecting to what extent this rule has an impact on the reasoning process. Very uncertain domain knowledge will receive a low certainty factor, whereas a rule that is very likely to be correct is assigned a high value. For example, a simple rule like "If 9 out of 10 people

[1]MYCIN is a diagnostic expert system for blood diseases

hate the product, then don't buy it" should get a high certainty factor, because it is very likely that the hypothesis is correct. The certainty factors for the system input can easily be calculated by combining the *measure of belief* and *measure of disbelief* for a piece or a set of evidence. The former depicts the confidence that the hypothesis is true, based on a set of evidence; the latter depicts the confidence that the hypothesis is not true, based on some evidence.

A rule in MYCIN can consist of conjunctions of pieces of evidence, for which it had to determine the combined certainty factor, by applying an *antecedent rule*, using the smallest certainty factor in case of a conjunction of evidence, the largest one if the antecedents where connected as a disjunction or the inverse of the certainty factor, if the premise was negated. The certainty factor of the input and the certainty factor of the rule itself were then combined by a *serial combination rule*. If multiple support for a single hypothesis was available, the certainty factors for each set of supporting evidence had to be combined using a *parallel combination rule*. In the end, the system could tell the user, for the given input, the hypothesis to be most likely true and how confident the system was of this decision, by returning the certainty factor that was computed during the process.

In its simplest form, such a mechanism accumulates all evidence and calculates the certainty factors for the resulting hypotheses, allowing us to see which one has the strongest support. If there is some evidence to back a given hypothesis, it will increase the belief in it to some extent. This is a very linear and straightforward approach that can sometimes be mislead by insufficient evidence, because the certainty factor for a specific hypothesis is not high enough to reflect absolute confidence in it (even though it may be the highest ranked).

A more complex structure was realized for the prototype of *Decisions To Go*. The idea was to make it possible that each piece of evidence can play multiple roles in the reasoning process. In the real world, a certain fact not only can support or reject a specific hypothesis, it usually has a much broader sphere of influence with varying degrees. That is to say that a single piece of evidence can increase belief in a hypothesis to a great extent, but also

increase the belief in a different hypotheses to a lesser amount and even decrease belief in other hypotheses. How this sphere of influence is incorporated has to be determined by the expert knowledge about the domain itself. An example for this regarding the cell phone domain could look like this:

```
(defrule age
    "If user young, then usage pattern A is likely with certainty 0.8"
    (fact (obj user-age) (val young) (cf ?x1))
    =>
    (assert (stmt (desc usage-pattern-A) (cf (* ?x1 0.8)))
    (assert (stmt (desc usage-pattern-B) (cf (* ?x1 0.5)))
    (assert (stmt (desc usage-pattern-C) (cf (* ?x1 -0.5)))
)
```

In this example, the user turns out to be young in age, which indicates that the times he or she is using the phone fall into a certain pattern that can be observed very often among young people (e.g., the hypothetical usage pattern A: 50% weekend, 30% weekday nights and 20% daytime). Hence the system will see this fact as evidence to support the hypothesis that this customer will most likely need a plan that has most of its unlimited airtime during the weekend and weekday nights. In addition to that, it is also taken as evidence against the hypotheses for which the usage pattern is different. In this case, the usage pattern B, which is quite similar to the first one, still receives some support by this piece of evidence. However, a completely different usage pattern is assigned a negative number to reflect the unlikelihood that it is the right hypothesis for this particular case. This helps to balance out the reasoning process a bit and is supposed to yield results that are more realistic and closer to human reasoning after all.

4.4 Reasoning Process

The reasoning process is based on two rule sets. Which one is used depends on the type of reasoning needed. The intelligent information retrieval module uses the rule set dealing with product research. This includes general rules about what information is important, rules

about the product domain, its market and consumer behavior as well as rules that try to analyze the public opinion about the particular product. The main goal of this reasoning is to determine how to prioritize the information that is available, such that the summary report will be most helpful to the user. Moreover, certain basic facts, such as product features, don't tell the user much right away. This wouldn't be an issue at home, when sufficient time is available to think about the implications that come with certain pieces of information, but in a mobile environment, the user expects to be presented with *all* the facts necessary to make an informed decision. Therefore, additional knowledge was incorporated that allows the system to examine the product's specifications more profoundly. For example, a cell phone comes with a camera. An expert shopper in the domain of cell phones would immediately look for connectivity features like Bluetooth, Infrared or USB, since the availability of a camera implies that the user will want to transfer the pictures to a notebook or desktop computer. The fact of whether or not the cell phone comes with such features suddenly becomes very important, climbing higher in the priority list. This translates to a rule that is fired upon encountering the fact that the cell phone ships with a camera and is missing all major means of transferring pictures to another device. The conclusion is that this should be seen as a major disadvantage as well as acknowledging the fact that the case has increased importance over other information:

```
(defrule product-research-camera-and-missing-connectivity
    "if cell phone has camera, then connectivity important. [cf: 0.9]"
    (stmt (has-camera) (cf ?x))
    (fact (obj feature) (val usb) (cf ?f))
    =>
    (bind ?cf_all (* (arule-min ?x ?f) 0.9))
    (assert (stmt (important-connectivity) (cf (* -1 ?cf_all))))
    (assert (stmt (advantage) (cf ?cf_all)))
)
```

In case the information retrieval module couldn't find any evidence for a Bluetooth, Infrared or USB feature, they are asserted to the knowledge-base with negative certainty factors, expressing strong belief in their absence from the feature list. This in turn leads to a negative

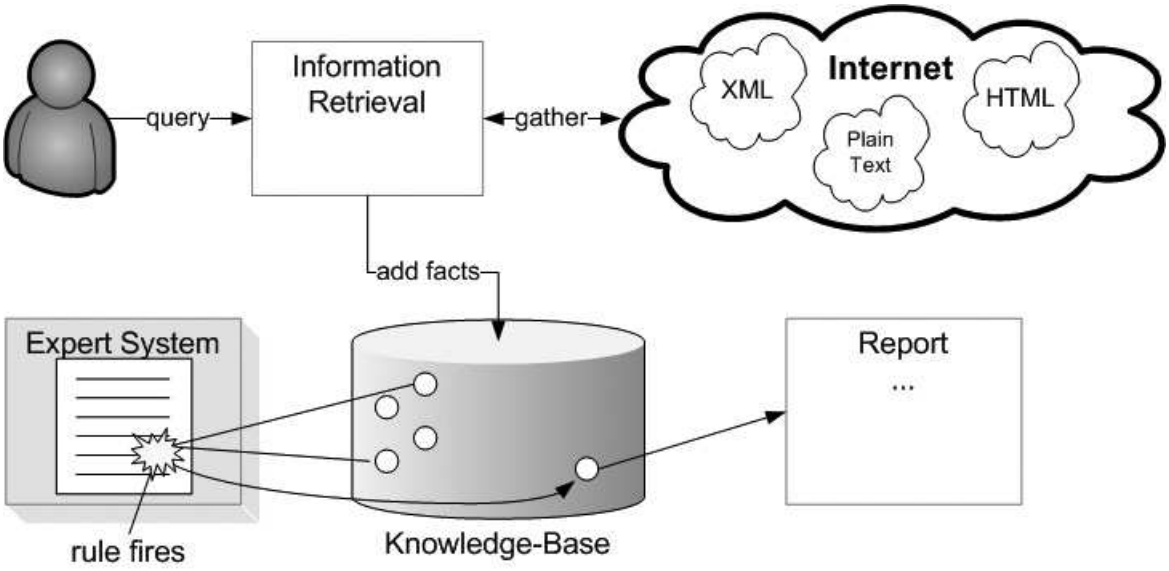

Figure 4.1: Expert System - Product Research

overall certainty factor (i.e. `cf_all`). The statement about being an advantage is therefore negative, meaning that the system considers this to be a major disadvantage. Consequently, the statement about the importance of this piece of information would be positive, making it a candidate to appear on the summary report. If, on the other hand, there was a connectivity feature readily available, the certainty factor for "advantage" would be positive, since the user would get a camera and the ability to easily transfer the pictures to another device. The importance of the information itself would be lower compared to other information, since there is no real issue anymore. Of course many facts are competing against each other in this process, for example price, rating, prominent feature, major disadvantage or other customer's opinions. Eventually, after all information is in and the inference engine comes to a stop, a hierarchy will have materialized regarding the importance of the individual pieces

of information, which can then be used by the report generator to lay out the summary report page.

The second rule set deals mainly with knowledge about consumer behavior and domain-specific criteria for product recommendations. In other words, it incorporates the knowledge that a well-trained salesperson would have. The goal is to gather additional information about the customer in order to derive a list of products that suits this customer's needs best. This was realized with a *stacked* forward chaining approach, where the global reasoning process is split into three tiers. Each tier has a number of different local goals that are derived by forward chaining. Once one of them has enough support by evidence, the reasoning process is switched over to the next tier using the results from the preceding tier as a basis for further reasoning. Finally, the system will come to a conclusion in the third tier, yielding a list of recommended phones, which are ranked according to their certainty factors.

In the first stage, the main focus lies on clues about the general or basic preferences of the user, which depend on the product category the system is dealing with. Based on the user input and the facts the system already knows from the user profile, the system will come to conclusions about certain general concepts like activities, mobility, general purpose and target group[2]. For example, if the user is of a certain age, some interests and hobbies are usually more likely to be important to him or her than others. Therefore, the system evaluates the current knowledge about the situation and carefully decides which question should be asked in the next step. If the certainty factors for one of the given concepts is high enough to be conclusive, no further questions have to be asked about it. However, if certainty factors remain low for an important concept due to ambiguous answers or lacking information, the system keeps asking in that direction in order to be able to make a meaningful guess about it.

The second tier is then narrowing down the solution space based on the previously derived general concept of the user. For example, certain activities lead to product features that are

[2]These concepts are considered important in the cell phone category

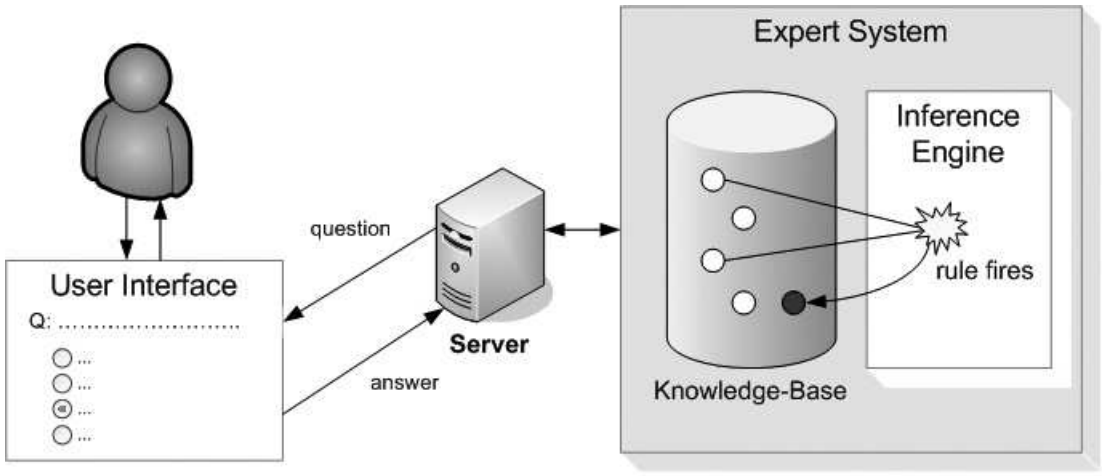

Figure 4.2: Expert System - Virtual Salesperson

most likely to be of interest for this particular user. The same is true for mobility and the target group the user most likely belongs to. The aim is to determine the main characteristics of the product, which for cell phones is mostly the set of features that come with it. Hence the next set of questions will be about finding out about the specifics of the customer's preferences. In this regard, questions are asked until the system is confident enough to make an accurate guess on the customer's needs. In the last step, we are left with a small selection of cell phones. In order to present the right recommendations to the customer, the system might ask one or two more specific questions in case it is not confident enough already. For example, after finding out that "messaging" is the most important feature for the customer's future cell phone, it could ask a more specific question on this kind of feature due to the fact that two of the phones that could be recommended are very similar. The final question would enable the system to cast a clear vote on which of these two would be better. Finally, a list of recommended products is displayed, ranked according to their certainty factors. Selecting one

of the proposed products will activate the automated product research on top of the already derived knowledge and gather all necessary information for the summary report page.

4.5 Creating a report

Once all priorities are assigned to the individual pieces of information, the report generator will be activated to create the summary report. For a query originating from a PDA or Smartphone, it will create a Web page that contains all the information and redirect the user to it. The report page has room for the product name and two columns holding information. The left column displays textual information such as customer reviews, feature descriptions and expert opinions. The right column holds three to four facts like price, average rating, major advantage or disadvantage and so forth. Each text or fact is generated as a link, directing the user to more detailed information on the respective topic if needed. In case the query came from a cell phone, the summary will be sent to the user's mobile device as a series of text messages with the same content as would have been displayed on the PDA (see figure 2.6). Each message contains a key code that can be used to retrieve more detail for the particular fact. The user would just send this key code to the server and receive another text message with the requested information.

The information is retrieved from the report generator using the Java API to access the list of facts stored in the knowledge-base. Since the "Expert Bean" is always available from anywhere in the program at any given time, the report can easily be generated by simple JSP code. However, the bulk of the work can be left to the Java program in the background by calling the appropriate methods from within the JSP code.

Chapter 5

Scenario

This chapter describes how to actually use *Decisions To Go* in a mobile environment. The general setting in which it should be used is any retail store that sells the products that are accessible via the Expert System. In other words, it is currently limited to the product category that was implemented for a prototype version of the software (i.e. cell phones). The user can use the system in two ways: "automated product research" and "virtual salesperson", both of which reflect a different kind of need for information. The former focuses on factual knowledge about the product that is gathered and analyzed by the intelligent information retrieval module. The latter requires the user to provide information about him- or herself in order to come up with a set of product recommendations, just as a human salesperson would do.

5.1 Using Automated Product Research

In the first case that this section is looking at, the user is interested in a particular product and wished to know more about it. In the traditional way, he or she would probably have done a good amount of online research before going to the store to see the real product. But instead of this, the user will make use of his or her cell phone or PDA to access the decision support system via one of the different ways as described in chapter 2. As a result, information is pushed to the mobile device, which allows the customer to make an informed decision on the purchase. Figure 5.1 shows the process of accessing the automated product research function on a PDA. After selecting this option, the user can either select a given item from the list or enter a product name to start the search. A few moments later, the user

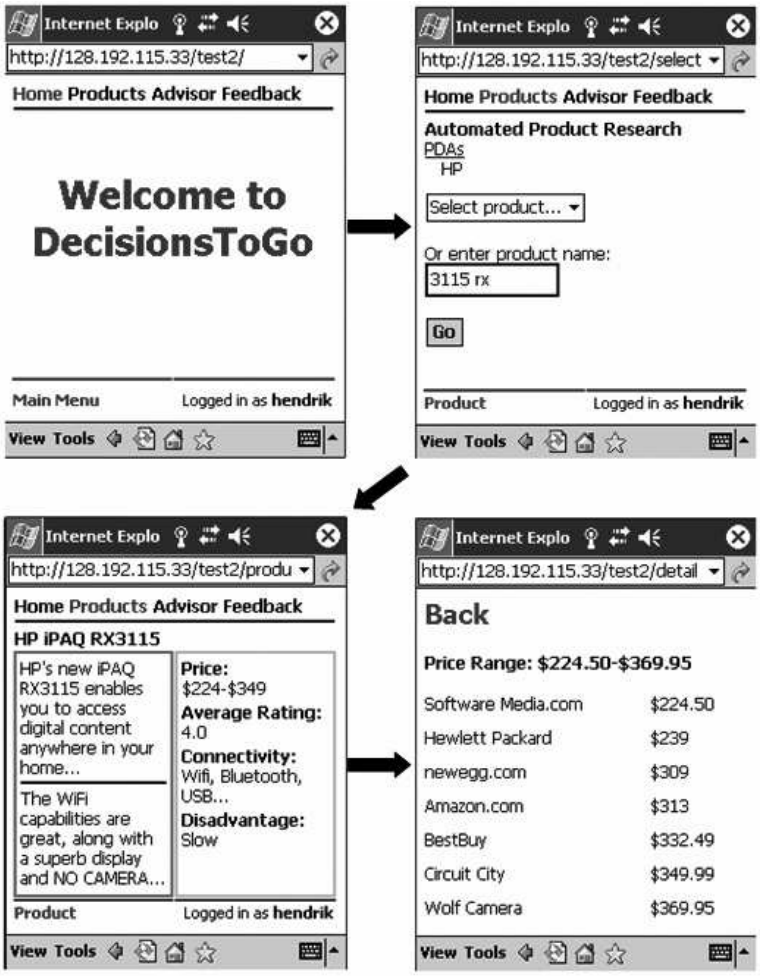

Figure 5.1: User Interaction with the Automated Product Research module on a PDA

will be redirected to the summary report page, where all the information that is important for the user is displayed. Moreover, he or she can click on any of the text boxes to get to more detailed information for that matter.

In the example depicted in figure 5.1, the user is looking for specific information on a PDA by Hewlett Packard. After having selected the right product category and brand, he or she is asked to select the product from a list or enter a name if it doesn't appear to be on the

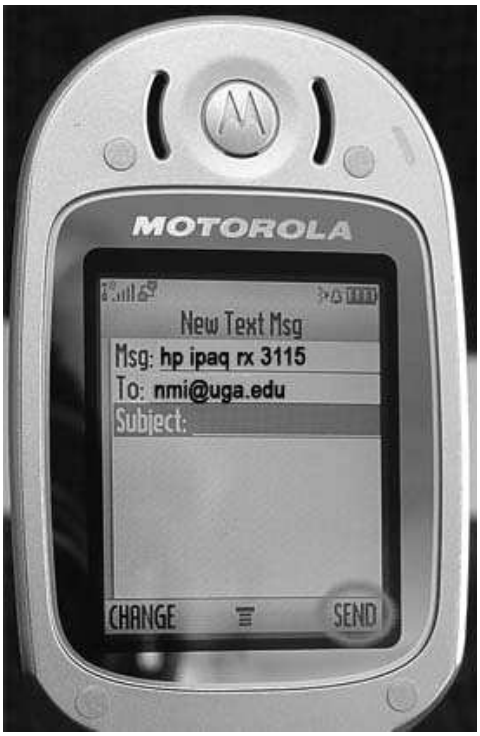

Figure 5.2: Sending a query to the server from a cell phone

list. Pressing the "Go" button activates the information retrieval module. More importantly, the summary report page is generated and displayed to the user. If the user wants to know more about a particular fact, he or she can click on any of the items in the report page to get to a more detailed explanation of it. In the example, the user clicked on "'price" in order to find out more about how the price range came about and which stores offer this product at what price. The list of stores in which the information retrieval module found the product on sale is sorted by price. Therefore, the customer can see all the offers and quickly decide whether or not it would be reasonable to make the purchase at an online store.

Another way to communicate with the decision support system is through text messaging. Instead of using a graphical user interface, queries are sent to the systems SMS interface and

processed like a regular query. The user sends a text message containing the product name to a dedicated email address that our server is listening to. Each incoming message is parsed and processed by the information retrieval module. Figure 5.2 shows a product query on a cell phone via text messaging. Moreover, the results are also sent back as a series of text messages. They inform the user about the desired product similar in the extent to what the Web interface provides. See figure 2.6 for an example of a summary report as it is sent to a cell phone.

5.2 Querying the Virtual Salesperson

The second case is dealing with the virtual salesperson. In this case, the customer doesn't know about what product to buy considering a general product category. Hence, he or she will use Decision To Go's virtual salesperson feature to get information about his or her options. The system will ask a couple of questions that can be easily answered and comes back with a list of three top recommendations for the user. Upon selecting any of these, the system will activate the information retrieval module for the selected product, in order to generate a summary report for it. Figure 5.3 shows how a customer answers a question on cell phone features and is presented with the list of recommended cell phones from which he or she can select.

Due to the restricted access to cellular networks, the SMS service is relatively slow. The server currently sends any messages to the user as emails over an email gateway, which transform emails into text messages and send them to the user's cell phone. The major disadvantage of this solution is the high latency encountered with this service. A faster data exchange between the decision support system and the users' cell phones would enable us to provide access to the virtual salesperson over this communication channel, which is currently not in place.

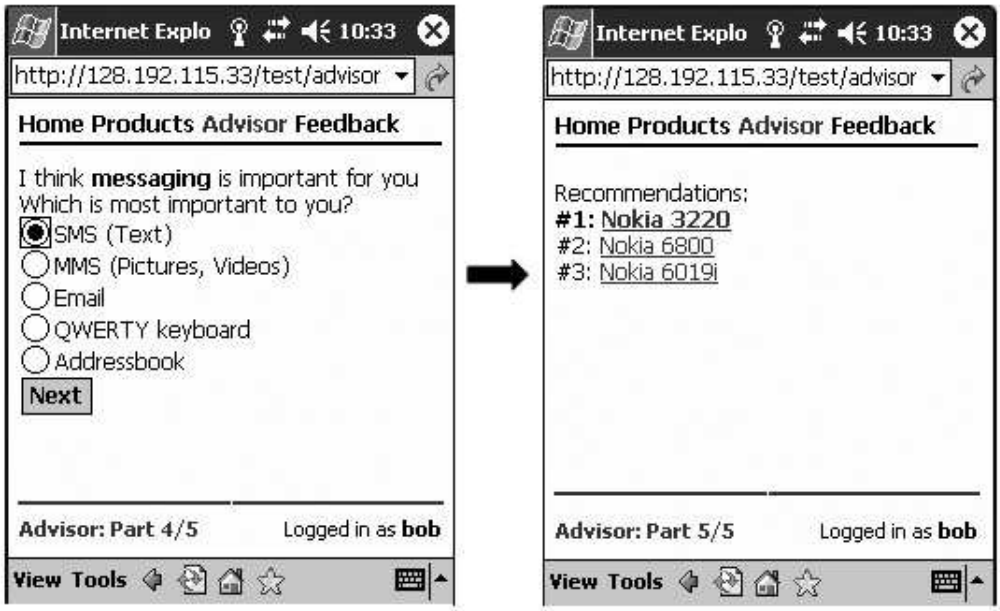

Figure 5.3: User Interaction with Virtual Salesperson on a PDA

Chapter 6

Conclusion and Future Directions

This paper describes the design and development of *Decisions To Go*, a mobile intelligent decision support system. *Decisions To Go* is an attempt to combine the capabilities of intelligent systems with the opportunities that lie within mobile communication. The main goal is to enable the customer to make informed decisions while in retail environments. Many people do online product research before their visit to a retail store in order to feel well-informed and ready to make a purchase decision. The prototype presented in this paper goes one step further, by providing this kind of information automatically and in condensed form on the user's mobile device. Therefore all necessary information can be accessed from within the retail store, making it possible to interact with the real product and at the same time knowing more about the product itself than an ordinary customer would know.

The two main pillars of this project are the information retrieval module and the expert system. The former currently accesses the product database of Amazon.com through a Web service provided by Amazon itself. In an attempt to broaden the spectrum of sources from which the system acquires its data, the system also parses the product evaluation Web pages from Epinions.com, even though as of now, this is just a basic keyword search for main elements within the report page for a specific product. The expert system incorporates a number of key concepts about the shopping domain. It has a set of basic rules that help determine which information is important and which product can be recommended to the advice seeking customer, taking into account any available knowledge about the user and the current situation in the product domain. In other words, the way summary reports

are generated or what recommendations are presented to the user heavily depends on the acquired information about the environment and the user.

Another big challenge was the communication between the server system on the one side and the mobile handsets on the other side. For a traditional Web browser, which can be found on most PDAs and Smartphones, the user interface was realized as a simple Web page, especially designed for the small screen resolution of these devices. Cell phones were a much bigger challenge, due to the lack of a traditional HTML Web browser. As a consequence, the system is currently using the proprietary text messaging features that most carriers incorporate into their networks.

The intelligent server system was completely developed under Java, and is run on Apache Tomcat 5.5 Servlet/JSP Container. This created an environment in which we could take advantage of the powerful features of both Java and Tomcat. For the information retrieval module, the HttpClient class library, developed under Apache's Jakarta Commons project, was used to access Web content through a Java-based software agent. Moreover, the JavaMail API version 1.3.2 was used to send text messages through the carrier's email gateway to the cell phones that were used for testing this application. Finally, the expert system module was completely realized with the Java Expert System Shell (JESS) version 7.0a5 by Sandia Corporation. This allowed us to interweave a traditional rule-based system with the server application. Moreover, it allowed us to establish a basic communication infrastructure between the individual modules and the client's mobile device, which is working considerably well.

In general, there seems to be a pressing need for intelligent applications on mobile devices. Just considering the continental United States, over 100 million cell phones are in use today. In addition to that, wireless Internet zones become more and more popular and are even expected by the customer in many places like coffee shops, airports and hotels. The infrastructure developed and deployed for the prototype of *Decisions To Go* enables most of these users to interact with the intelligent decision support system without having to install any addi-

tional software. Artificial Intelligence might just be the major driver of mobile applications development in the future.

From a business perspective, there are many positive results stemming from this research project. First of all, it creates a new type of market place, because it is bringing together otherwise separate market platforms (i.e. traditional and online stores). There seem to be benefits for every participant in such a market. Traditional "brick" stores could benefit, since they heavily rely on customer satisfaction and repeat business, both of which are generally increased if the customer is well-informed before making a decision. The customer on the other hand will embrace this technology as a new opportunity for immediate gratification, skipping the inconvenient delivery times that one is confronted with when buying products online. As a matter of fact, people do their product research anyway - why not provide a platform for this?

The main challenge for future versions of the prototype is to make the Artificial Intelligence work better. This means increasing the capabilities of the information retrieval module, as well as refining and extending the knowledge-base of the expert system. Currently, only two sources of information are ever accessed (besides some specific facts that are stored locally). It would be desirable to include more Web services and to parse other Web content in order to increase the overall quality of the product research. For example, Ebay and Google are granting more and more access to their Web facilities, which could lead to a greater variety of sources for prices, product ratings and customer reviews. On a different note, the expert system incorporates some basic knowledge about customer behavior, the market and certain product categories, but would have to be heavily extended in future versions of *Decisions To Go*.

In addition to that, there could be more profound additions and alterations to the system that possibly increase its power and usefulness significantly.

6.1 A Learning System?

A learning component of the system could encompass two major approaches to improve the system. The first idea aims at improving the way the current rule-base is maintained over time. The idea is that rules should be able to adapt to user feedback. Currently, each rule comes with a certainty factor attached to it, reflecting the importance of it in terms of the overall reasoning process. This factor is entered based on expert knowledge upon creation, but can also change later on. Instead of making these modifications manually, the rules could adjust automatically over time. This would require a way for the user to be able to give feedback on the decisions the expert system made, especially in terms of product recommendations. The user might not be satisfied with the alternatives the system came up with and instead go for a different product. If the system knew which product was chosen instead, it would be able to adjust the rules accordingly, if a great number of customers in similar situations decided in the same way.

The second idea addresses the need for a large number of sources during the information retrieval phase. The hypothesis is that if more sources are accessed and evaluated, the more accurate, detailed and bias-free the resulting report will be. But not all sources are as well-defined as the information obtained from Amazon's product database. In fact, many Web sites that could be used for product research don't even have a numeric rating system that could help guide the system at all. In general, product reviews and opinions by other customers are a valuable source of information. There are plenty of Web sites that specialize in bringing people together by providing a platform for information exchange and discussion (Prasad, 2003). However, it is hard to extract information from plain texts with no semantic markup. A possible solution could be a naive Bayesian classifier system that predicts the numeric product rating for any product review given the body of the evaluation text. This "machine learner" could be trained on a large number of available product evaluations that are rated by the customer by assigning a crisp number to it. The classifier would depend on the question of whether or not the way humans write reviews bears a general

concept of how good or bad a certain product is. If that was the case, then it should be able to identify or predict the numeric rating for any given descriptive evaluation text with some accuracy. As a consequence, the number of Web pages that can be used as sources of information increases drastically. For example, we could use Google to retrieve relevant Web sites that contain certain keywords or phrases and then analyze the given text segments to check if they classify the product as good or bad.

6.2 Incorporating Belief Structure

The main issue with simple rule-based expert systems is that they don't necessarily model the human reasoning process very well. A rule-based system usually consists of a set of rules which, evidence provided, contribute to support a certain hypothesis. At any intermediate step, all we can do to check out the prospects of achieving such a goal is to add more supporting knowledge, which is not always desirable (Doyle, 1985). Humans probably don't think in terms of these "strict" rule systems, but rather form beliefs and develop theories over time.

An expert system that has a belief system as its foundation would probably be able to keep better track of the reasoning process in many ways. One important step is to improve the reasoning system by incorporating dependencies between rules or pieces of evidence for certain hypotheses. That way, newly added evidence not only supports or rejects a hypothesis, but can also have an impact on other hypotheses via a "dependency network". More importantly, in contrast to pure rule-based systems, belief-based expert systems provide facilities to maintain and revise their current knowledge about the world, based on the currently known facts. Doyle's model of non-monotonic justifications (Doyle, 1985) would allow us to incorporate commonsense reasoning. It can detect inconsistencies and blatant contradictions in interdependent hypotheses and can make justified assumptions or retract justifications for pieces of knowledge already available in the working memory in order to bring forward the reasoning process – something one cannot achieve with a simple rule-based system. Finally,

such systems usually propagate this kind of information, such that the inference procedure is not necessarily a straight-forward process anymore, but builds an "opinion" while working through the known facts.

This approach might be a better and more intuitive way to model and incorporate the required knowledge into our expert system, since the line of reasoning regarding the domain of shopping for goods is rather fuzzy and often escapes strict logical thinking.

Bibliography

Maristella Agosti and Alan Smeaton. *Information Retrieval and Hypertext.* Kluwer Academic Publishers, 1996.

Annenberg. *Surveying the Digital Future, Year Four.* USC Annenberg Center for the Digital Future, 2004.

John Doyle. *A Truth Maintenance System.* Artificial Intelligence, Vol.11:231-272, 1985.

Forrester. *The US Consumer 2004: Multichannel and In-Store Technology.* Forrester Research, 2004.

Joshua Freed. *The customer is always right? Not anymore.* Associated Press Business, 2004.

Peter Jackson. *Introduction to Expert System, Third Edition.* Addison-Wesley Publ. Company, Reading, MA, 1998.

Jupiter. *In-store Pickup, Implement Last Mile Policies to Optimize Technology Investments.* Jupiter Research Group, 2002.

Craig A. Knoblock. *Modeling Web Sources for Information Integration.* Proceedings of the National Conference on Articial Intelligence (AAAI), 1998.

F. Wilfrid Lancaster. *Information Retrieval Systems: Characteristics, Testing and Evaluation.* Wiley, New York, 1968.

John McCarthy. *Some Expert System Need Common Sense.* Computer Culture: The Scientific, Intellectual and Social Impact of the Computer, Vol. 426, Annals of the New York Academy of Sciences, 1984.

Bhanu Prasad. *Intelligent Techniques for E-Commerce.* Journal of Electronic Commerce Research, Vol.4, No.2, 2003.

David S. Prerau. *Selection of an Appropriate Domain for an Expert System.* AI Magazine, Vol.6, No.2:26-30, 1985.

Kiefer Sims. Interview with Best Buy Customer Sales Manager, 2004.

Hans van der Heijden. *Mobile Decision Support for In-Store Purchase Decisions.* Decision Support Systems, Vol.34:139 156, 2002.

Soe-Tsyr Yuan. *A Personalized and Integrative Comparison-Shopping Engine and its Applications.* Decision Support Systems, Vol.34:139 156, 2002.

Appendix A

Overview

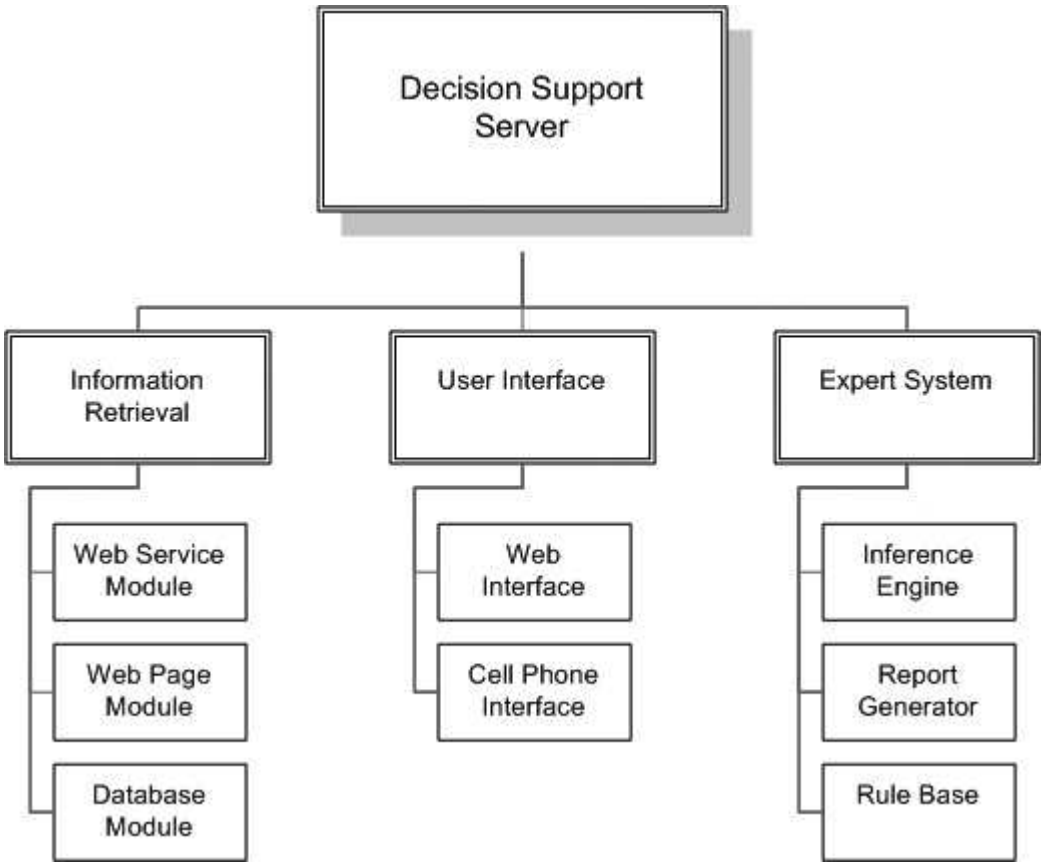

Figure A.1: Overview of the application

Appendix B

Code

B.1 Decision Support Server

The DecisionsToGo main class serves two purposes. For one, it initializes the cell phone interface class that interacts with a user on a cell phone. Secondly, it serves as the entry point for user that want to access the system from a PDA or Smartphone

```
public class DecisionsToGo {

    private InfoRetrieval ir;
    private TempContainer tc;
    private Report r;
    private ExpertSystem esref;

    public DecisionsToGo() {
        System.out.println("*** Starting up Decision Support Server");
        new CellInterface(this).start();
    }

    public static void main(String[] args) {
        // Initialization
        DecisionsToGo dtg = new DecisionsToGo();
    }
```

The following illustrates the decision support server processing a user query. First, the product is determined by locating it in the database. After the profile is retrieved, the information retrieval module can be started. This will access Amazon.com and Epinions.com for gathering data on the particular product. The object holding all the information is then passed on to the Expert System. Once the reasoning process is complete, the report is generated and can be sent back to the user.

```
// Analyze Content (Retrieve product name from message)
String key = content.toLowerCase();

// Retrieve product from DB
String[] product = getProduct(content);

// Retrieve profile from DB
Profile p = getProfile(phone);

// Start Information Retrieval Module
InfoRetrieval ir = new InfoRetrieval(product[0], product[1]);
ir.getAmazonData();
ir.getEpinions();
```

```
TempContainer tc = new TempContainer(ir.apref, ir.epref, ir.category);

// Start Expert System
ExpertSystem es = new ExpertSystem(tc, p, product[0], product[1]);
es.reason();

// Start Report Generator
Report r = es.generateReport(product[0]);

String report1 = r.getFactsReport();
String report2 = r.getTextReport1();
String report3 = r.getTextReport2();
String report4 = r.getTextReport3();

// Send the report to "phone"
send(1, phone, report1);
send(2, phone, report2);
send(3, phone, report3);
send(4, phone, report4);
```

In order to access the system through the Web interface, the main class is initialized and the user's profile is retrieved as soon as he or she loads the main page of the Web portal.

```
<jsp:useBean id="d2go" class="dec.DecisionsToGo" scope="session" />

<!-- Retrieve Profile -->
<%
Cookie[] cookies = request.getCookies(); String uname="";

    for(int i=0; i<cookies.length; i++) {
      Cookie cookie = cookies[i];
      if ("uname".equals(cookie.getName()))
        uname = cookie.getValue();
    }
    d2go.getProfile(uname)
%>
```

If the user is accessing the system via cell phone, the profile is retrieved as soon as the query is processed. More important, the user is identified by the phone number from which he or she sent the query message.

```
public Profile getProfile(String phone) {
        Profile p = new Profile();
        try {
            Class.forName("com.mysql.jdbc.Driver").newInstance(); // MySQL
            Connection conn = (Connection) DriverManager.getConnection(connstring);
            int userError = -1;
            try {
                // User is identified by his or her cell phone number
                String query = "SELECT uname FROM user WHERE cell = '" + phone + "'";
                stmt = (Statement) conn.createStatement();
                rs = (ResultSet) stmt.executeQuery(query);
                while(rs.next()) {
                    p.setUsername(rs.getString("uname"));
                }
                String sql = "SELECT * FROM profile WHERE uname='" + p.getUsername() + "'";
                stmt = (Statement) conn.createStatement();
                rs = (ResultSet) stmt.executeQuery(sql);
                while (rs.next()) {
                    p.setAge(rs.getInt("age"));
```

```
                p.setEthnicity(rs.getInt("ethnicity"));
                p.setJob(rs.getInt("occupation"));
                p.setState(rs.getString("state"));
                p.setZip(rs.getString("ZIP"));
                p.setGender(rs.getString("gender"));
                htemp = rs.getString("hobbies");
            }
            // Hobbies
            p.setHobbies(parseHobbies(htemp));
            conn.close();
        } catch (Exception ie) {
            // handle the error
        }
    } catch (Exception ex) {
        // handle the error
    }
    return p;
}
```

Finally, the product itself is retrieved from a local database. Only the product name and the product category it falls into are stored. This is to make sure the query is valid and will yield results before accessing the Web.

```
public static String[] getProduct(String query) {
    String[] p = new String[2];
    int userError = -1;
    try {
        Class.forName("com.mysql.jdbc.Driver").newInstance(); // MySQL
        Connection conn = (Connection) DriverManager.getConnection(connstring);
        stmt = (Statement) conn.createStatement();
        rs = (ResultSet) stmt.executeQuery(
            "SELECT * FROM products WHERE productname LIKE '%" + query + "%'");

        while(rs.next()) {
            p[0] = rs.getString("productName");
            p[1] = rs.getString("productType");
        }

    } catch (Exception e) {
        // Exception
    }
    return p;
}
```

B.2 Information Retrieval

The information retrieval module can currently access two sources of information: Amazon.com and Epinions.com. The former is queried through a Web service, whereas the latter is only available in plain HTML code, which we have to parse manually.

```
public void getAmazonData() {
    String asin = this.searchItem();
    this.makeAmazonRequest(asin);
}

public void getEpinions() {
    this.makeEpinionsRequest();
}
```

B.2.1 Web Service Module

The first method searches for the ASIN number by issuing the appropriate command to the Web service

```
public String searchItem() {
    String product = this.product;
    String cat = this.category;
    String searchUrl = amzBaseURL
        + "&Operation=ItemSearch"
        + "&Keywords=" + product
        + "&SearchIndex=" + searchIndex
        + "&MinimumPrice=" + minPrice;
    URL url;
    String ASIN = null;
    try {
        // Replace white space with escape sequence "%20"
        url = new URL(convertWhiteSpace(searchUrl));
        URLConnection connection = url.openConnection();
        ByteArrayOutputStream result = new ByteArrayOutputStream();
        InputStream input = connection.getInputStream();
        byte[] buffer = new byte[1000];
        int amount=0;
        while(amount != -1) {
            result.write(buffer, 0, amount);
            amount = input.read(buffer);
        }
        String response = result.toString();

        // Parse response
        ASIN = AmazonParser.getAsin(response);
    } catch (MalformedURLException e) {
        e.printStackTrace();
    } catch (IOException e) {
        e.printStackTrace();
    }
    return ASIN;
}
```

Retrieve the ASIN number from the XML response:

```
public static String getAsin(String xml) {
    int i0 = xml.indexOf("<ASIN>");
    int i1 = xml.indexOf("</ASIN>");
    return xml.substring(i0 + 6, i1);
}
```

Next, the item is looked up by its ASIN number and the results are stored

```
public void makeAmazonRequest(String asin) {

    String tmpUrl = amzBaseUrl
        + "&Operation=ItemLookup"
        + "&ItemId=" + asin
        + "&ResponseGroup=Small,ItemAttributes,EditorialReview,SalesRank,Reviews";

    URL url;
    try {
        url = new URL(convertWhiteSpace(tmpUrl));
        URLConnection connection = url.openConnection();
        ByteArrayOutputStream result = new ByteArrayOutputStream();
        InputStream input = connection.getInputStream();
        byte[] buffer = new byte[1000];
        int amount=0;
        while(amount != -1) {
            result.write(buffer, 0, amount);
            amount = input.read(buffer);
        }
    // Parse [Price, Salesrank, Avg Rating, Editorial, Total Reviews, Features]
    apref.parse(result.toString());

    } catch (MalformedURLException e) {
        e.printStackTrace();
    } catch (IOException e) {
        e.printStackTrace();
    }
}
```

The response, in the form of an XML file, is parsed with Xerces' SAX Parser, a simple but powerful XML parser.

```
public void parse(String xml) {
    XMLReader xr;
    try {
        xr = XMLReaderFactory.createXMLReader();
        AmazonHandler handler = new AmazonHandler();
        xr.setContentHandler(handler);
        xr.setErrorHandler(handler);

        StringReader s = new StringReader(xml);
        BufferedReader in = new BufferedReader(s);

        xr.parse(new InputSource(in));

        // Storing the results
        this.setAvgrating(handler.getAvgrating());
        this.setEditorial(handler.getEditorial());
        this.setFeatures(handler.getFeatures());
        this.setPrice(handler.getPrice());
        this.setReviewstotal(handler.getReviewstotal());
        this.setSalesrank(handler.getSalesrank());
        this.setPriceNum(handler.getPriceNum());
    } catch (SAXException e) {
        e.printStackTrace();
    } catch (IOException e) {
        e.printStackTrace();
    }
}
```

The handler retrieves the information by getting called every time a start or end tag was found. It then sets a flag indicating that a particular information is located at this point and calls a method for storing the information found between those tags accordingly.

```
public void startElement(String uri, String name, String qName, Attributes atts) {
    if (qName.equals("SalesRank"))
        sr = true;
}

public void endElement(String uri, String name, String qName, Attributes atts) {
    if (qName.equals("SalesRank"))
        sr = false;
}

public void characters(char ch[], int start, int length) {
    if (sr) {
        StringBuffer sb = new StringBuffer();
        for (int i = start; i < start + length; i++) {
            sb.append(ch[i]);
        }
        this.salesrank = Integer.valueOf(sb.toString()).intValue();
    }
}
```

B.2.2 Web Page Module

Parsing a Web page is more of a multi-stage process, since the information retrieval module has to load each page containing valuable information separately and search it for keywords. This method uses Apache's HTTP client for Java, which allows us to access a Web page just as a human user would do. This is important, because the report pages are not directly accessible, but have to be looked up through Epinions.com's search mask.

```
   public void makeEpinionsRequest() {
       HttpClient client = new HttpClient();
       String link = "http://www.epinions.com";

       // Find the right summary report page through search
       HttpMethod m = new PostMethod(link + "/search/?submitted_form=searchbar&search_string="
                                           + product.replace(" ", "+"));
       try {
           int statusCode = client.executeMethod(m);
           if (statusCode != HttpStatus.SC_OK) {
               System.err.println("Method failed: " + m.getStatusLine());
             }

             // Read the response body.
             ByteArrayOutputStream result = new ByteArrayOutputStream();
             InputStream input = m.getResponseBodyAsStream();
             byte[] buffer = new byte[1000];
             int amount=0;
             while(amount != -1)
             {
                 result.write(buffer, 0, amount);
                 amount = input.read(buffer);
             }

             // Read the result and search for the link to the summary report
```

```
      String searchPage = result.toString();
      StringReader sp = new StringReader(searchPage);
      BufferedReader in = new BufferedReader(sp);
      String read;
      while (!(in.readLine()).contains("in the following categories")) {
          // skip
      }
      while ((read = in.readLine()) != null) {
          if (read.contains("a href=\"")) {
              int linkIndex = read.indexOf("<a href=\"") + 9;
              link = link + read.substring(linkIndex,read.indexOf("\">", linkIndex));
              break;
          }
      }
} catch (HttpException e1) {
    // TODO Auto-generated catch block
    e1.printStackTrace();
} catch (IOException e1) {
    // TODO Auto-generated catch block
    e1.printStackTrace();
}
finally {
    m.releaseConnection();
}
```

After having retrieved the correct URL of the summary report page, the information retrieval module can download it and parse it.

```
// Retrieve summary report page
String sortlink = link + epExtension1;
HttpMethod m2 = new PostMethod(sortlink);
try {
    int statusCode = client.executeMethod(m2);
    if (statusCode != HttpStatus.SC_OK) {
        System.err.println("Method failed: " + m2.getStatusLine());
      }

      // Read the response body.
      ByteArrayOutputStream result = new ByteArrayOutputStream();
      InputStream input = m2.getResponseBodyAsStream();
      byte[] buffer = new byte[1000];
      int amount=0;
      while(amount != -1)
      {
          result.write(buffer, 0, amount);
          amount = input.read(buffer);
      }

      // Retrieve the result page
      String resultPage = result.toString();
      //System.out.println(resultPage);
      epref.parseSummary(resultPage, false);
    ...
```

Some products are offered by many stores. If more than 20 online shops are having the product on sale, Epinions.com splits the list of stores into multiple parts. Therefore, the information retrieval module checks the number of stores and downloads the other parts as well if needed.

```
      ...
      if (epref.getNumstores() > 20) {
          String morelink = link + epExtension2;
          HttpMethod m3 = new PostMethod(morelink);
          int statusCode2 = client.executeMethod(m3);
          if (statusCode2 != HttpStatus.SC_OK) {
              System.err.println("Method failed: " + m3.getStatusLine());
            }

            // Read the response body.
            ByteArrayOutputStream result2 = new ByteArrayOutputStream();
            InputStream input2 = m3.getResponseBodyAsStream();
            byte[] buffer2 = new byte[1000];
            int amount2=0;
            while(amount2 != -1)
            {
                result2.write(buffer2, 0, amount2);
                amount2 = input2.read(buffer2);
            }

            // Rertieve the result page
            String resultPage2 = result2.toString();
            epref.parseSummary(resultPage2, true);
            m3.releaseConnection();
      }

} catch (HttpException e1) {
    e1.printStackTrace();
} catch (IOException e1) {
    e1.printStackTrace();
}
finally {
    m2.releaseConnection();
}
```

Finally, the "customer reviews" page is downloaded and parsed for reviews, pros and cons, as well as an average rating on more specific properties within the general product category (e.g. "clarity" for cell phones).

```
String custlink = link + "/display_~reviews";
HttpMethod m4 = new PostMethod(custlink);

try {
    int statusCode = client.executeMethod(m4);
    if (statusCode != HttpStatus.SC_OK) {
        System.err.println("Method failed: " + m4.getStatusLine());
      }

      // Read the response body.
      ByteArrayOutputStream result = new ByteArrayOutputStream();
      InputStream input = m4.getResponseBodyAsStream();
      byte[] buffer = new byte[1000];
      int amount=0;
      while(amount != -1)
      {
          result.write(buffer, 0, amount);
          amount = input.read(buffer);
      }

      // Read the customer review page
      String custPage = result.toString();
      epref.parseCustomerReviews(custPage, category);
      for (Iterator iter = epref.getCons().iterator(); iter.hasNext();) {
        String pro = (String) iter.next();
        System.out.println(pro);
```

```
            }

        } catch (HttpException e1) {
            e1.printStackTrace();
        } catch (IOException e1) {
            e1.printStackTrace();
        }
        finally {
            m4.releaseConnection();
        }
    }
```

B.3 Expert System

B.3.1 Inference Engine

After the Expert System is invoked, it will compute the information given by the profile and add this knowledge to the knowledge-base. Since the boundaries between age groups are fuzzy, multiple assertions are made for each age.

```
// Constructor
 public ExpertSystem(TempContainer tc, Profile p, String item, String cat) {
    // Store query data
    this.product = item;
    this.category = cat;
    this.user = p;
    this.tcref = tc;

    // Initialize Expert System
    r = new Rete();
    try {
        r.executeCommand("(batch C:/Java/eclipse/workspace/Decisions/D2GO.clp)");
        r.reset();

        // Put content of temporary container object into knowledge-base
        this.processTempContainer(tc);
        this.processProfile(p);

        // Set initial goal and category
        r.assertString("(category " + cat + ")", r.getGlobalContext());

    } catch (JessException e) {
         // TODO Auto-generated catch block
        e.printStackTrace();
    }
}

public void computeProfile() {
    // AGE
    if (this.getAge() < 19) {
        r.executeCommand("(assert (fact (obj user-age) (val teen) (cf 0.9)))");
        r.executeCommand("(assert (fact (obj user-age) (val young) (cf 0.3)))");
        r.executeCommand("(assert (fact (obj user-age) (val senior) (cf -0.8)))");
        r.executeCommand("(assert (fact (obj user-age) (val old) (cf -0.9)))");
    } else if (this.getAge() < 36) {
        r.executeCommand("(assert (fact (obj user-age) (val teen) (cf 0.3)))");
        r.executeCommand("(assert (fact (obj user-age) (val young) (cf 0.9)))");
        r.executeCommand("(assert (fact (obj user-age) (val senior) (cf -0.2)))");
        r.executeCommand("(assert (fact (obj user-age) (val old) (cf -0.8)))");
    } else if (this.getAge() < 46) {
        r.executeCommand("(assert (fact (obj user-age) (val teen) (cf -0.9)))");
        r.executeCommand("(assert (fact (obj user-age) (val young) (cf 0.5)))");
        r.executeCommand("(assert (fact (obj user-age) (val senior) (cf 0.8)))");
```

```
    r.executeCommand("(assert (fact (obj user-age) (val old) (cf 0.1)))");
} else if (this.getAge() < 56) {
    r.executeCommand("(assert (fact (obj user-age) (val teen) (cf -0.9)))");
    r.executeCommand("(assert (fact (obj user-age) (val young) (cf 0.2)))");
    r.executeCommand("(assert (fact (obj user-age) (val senior) (cf 0.9)))");
    r.executeCommand("(assert (fact (obj user-age) (val old) (cf 0.3)))");
} else if (this.getAge() < 66) {
    r.executeCommand("(assert (fact (obj user-age) (val teen) (cf -0.9)))");
    r.executeCommand("(assert (fact (obj user-age) (val young) (cf -0.3)))");
    r.executeCommand("(assert (fact (obj user-age) (val senior) (cf 0.9)))");
    r.executeCommand("(assert (fact (obj user-age) (val old) (cf 0.6)))");
} else {
    r.executeCommand("(assert (fact (obj user-age) (val teen) (cf -0.9)))");
    r.executeCommand("(assert (fact (obj user-age) (val young) (cf -0.5)))");
    r.executeCommand("(assert (fact (obj user-age) (val senior) (cf 0.6)))");
    r.executeCommand("(assert (fact (obj user-age) (val old) (cf 0.9)))");
}
// Occupation
switch (this.job) {
    case 0:
        r.executeCommand("(assert (fact (obj job) (val other) (cf 0.9)))");
        break;
    case 1:
        r.executeCommand("(assert (fact (obj job) (val student) (cf 0.9)))");
        break;
    case 2:
        r.executeCommand("(assert (fact (obj job) (val self) (cf 0.9)))");
        break;
    case 3:
        r.executeCommand("(assert (fact (obj job) (val gov) (cf 0.9)))");
        break;
    case 4:
        r.executeCommand("(assert (fact (obj job) (val worker) (cf 0.9)))");
        break;
    case 5:
        r.executeCommand("(assert (fact (obj job) (val retired) (cf 0.9)))");
        break;
    case 6:
        r.executeCommand("(assert (fact (obj job) (val school) (cf 0.9)))");
        break;
     }
// Homeground
if (this.getState().equals("Georgia")) {
    r.executeCommand("(assert (fact (obj user-area) (val georgia) (cf 0.9)))");
} else if (this.getState().equals("Various")) {
    r.executeCommand("(assert (fact (obj user-area) (val nation) (cf 0.9)))");
} else if (this.getState().equals("Utah")) {
    r.executeCommand("(assert (fact (obj user-area) (val utah) (cf 0.9)))");
}
// City
if (this.getZip().startsWith("30")) {
    // Atlanta Metro Area, GA
    r.executeCommand("(assert (fact (obj user-home) (val georgia) (cf 0.9)))");
} else if (this.getZip().startsWith("12")) {
    // Various
    r.executeCommand("(assert (fact (obj user-home) (val nyc) (cf 0.9)))");
} else if (this.getZip().startsWith("19")) {
    // Pennsylvania
    r.executeCommand("(assert (fact (obj user-home) (val penn) (cf 0.9)))");
} else if (this.getZip().startsWith("84")) {
    // Utah
    r.executeCommand("(assert (fact (obj user-home) (val utah) (cf 0.9)))");
}
```

The hobbies are handled in a way that reflects their impact on other activities besides the main activity. For example, somebody who likes playing golf may also like sports in general and, to a lesser extent, may also be interested in hiking.

```
    // Hobbies
    for (int i = 0; i < this.hobbies.length; i++) {
        switch (hobbies[i]) {
            case 1: // golf
                r.executeCommand("(assert (fact (obj hobby) (val golf) (cf 0.9)))");
                r.executeCommand("(assert (fact (obj hobby) (val sports) (cf 0.5)))");
                r.executeCommand("(assert (fact (obj hobby) (val hiking) (cf 0.3)))");
                break;
            case 2: // movies
                r.executeCommand("(assert (fact (obj hobby) (val movies) (cf 0.9)))");
                r.executeCommand("(assert (fact (obj hobby) (val friends) (cf 0.4)))");
                break;
            case 3: // hiking
                r.executeCommand("(assert (fact (obj hobby) (val hiking) (cf 0.9)))");
                r.executeCommand("(assert (fact (obj hobby) (val traveling) (cf 0.3)))");
                r.executeCommand("(assert (fact (obj hobby) (val sports) (cf 0.3)))");
                break;
            case 4: // friends
                r.executeCommand("(assert (fact (obj hobby) (val friends) (cf 0.9)))");
                r.executeCommand("(assert (fact (obj hobby) (val food) (cf 0.3)))");
                break;
            case 5: // gaming
                r.executeCommand("(assert (fact (obj hobby) (val gaming) (cf 0.9)))");
                r.executeCommand("(assert (fact (obj hobby) (val friends) (cf 0.5)))");
                r.executeCommand("(assert (fact (obj hobby) (val sports) (cf 0.4)))");
                break;
            case 6: // traveling
                r.executeCommand("(assert (fact (obj hobby) (val traveling) (cf 0.9)))");
                r.executeCommand("(assert (fact (obj hobby) (val hiking) (cf 0.3)))");
                break;
            case 7: // reading
                r.executeCommand("(assert (fact (obj hobby) (val reading) (cf 0.9)))");
                r.executeCommand("(assert (fact (obj hobby) (val movies) (cf 0.3)))");
                break;
            case 8: // good food
                r.executeCommand("(assert (fact (obj hobby) (val food) (cf 0.9)))");
                r.executeCommand("(assert (fact (obj hobby) (val friends) (cf 0.5)))");
                r.executeCommand("(assert (fact (obj hobby) (val traveling) (cf 0.3)))");
                break;
            case 9: // sports
                r.executeCommand("(assert (fact (obj hobby) (val sports) (cf 0.9)))");
                r.executeCommand("(assert (fact (obj hobby) (val friends) (cf 0.5)))");
                r.executeCommand("(assert (fact (obj hobby) (val hiking) (cf 0.3)))");
                break;
            }
        }
```

In general, evidence is added simply by executing the appropriate command from within Java

```
public void addEvidence(String obj, String val, double cf) {
    try {
        this.r.executeCommand("(assert (fact (obj " + obj + ")
            + (val " + val + ")
            + (cf " + cf + ")))");
    } catch (JessException e1) {
        e1.printStackTrace();
    }
}
```

The getNextQuestion method is used to access the knowledge-base in order to find out which question should be asked next. It searches for "question"-facts that have been previously asserted during the reasoning process and indicate that one of the previous questions or a global goal has not been completely satisfied yet.

```
public Question getNextQuestion() {
    Vector tmpQ = new Vector();
    float[] cfs = new float[100];
    int i = 0;
    for (Iterator i = this.r.listFacts(); i.hasNext();) {
        Fact f = (Fact) i.next();
        if (f.getFactId() != 0) {
            try {
                Value v1 = f.getSlotValue("obj");
                Value v2 = f.getSlotValue("val");
                Value v3 = f.getSlotValue("cf");
                eobj = v1.stringValue(this.r.getGlobalContext());
                eval = v2.stringValue(this.r.getGlobalContext());
                ecf = (float) v3.floatValue(this.r.getGlobalContext());

                // Check goals of the current level
                if (eobj.startsWith("goal") && eval.equals(this.getCurrentGoal()) {
                    if (ecf < 0.5) {
                        tmpQ.add(this.getGoalQuestion(this.getCurrentGoal()));
                        cfs[i++] = ecf;
                    }
                // Check if a rule has entered a question
                } else if (eobj.startsWith("question")) {
                    tmpQ.add(this.getQuestionByName(eval));
                    cfs[i++] = ecf;
                }
            } catch (Exception e) {
                //
            }
        }
    }

    // Sort the open questions by certainty factor
    Vector qs = new Vector();
    int j = 0;
    currentcf = -2.0;
    for (Iterator i = tmpQ.iterator(); i.hasNext();) {
        Question q = (Question) i.next();
        if (cfs[j] > currentcf) {
            qs.add(q,0);
            currentcf = cfs[j];
        } else {
            qs.add(q);
        }
        j++;
    }

    // return the question that is either the most important
    // or is part of a goal that is very close to be satisfied
    return qs.get(0)
}
```

B.4 User Interface

B.4.1 Web Interface

In order to access the system via Internet, the user has to login first. This is necessary, since part of the reasoning process relies on the information about the user stored in his or her profile.

```
<%@ page import="java.util.*" %>
<jsp:useBean id="idCheck" class="dec.Login" scope="request"> <jsp:setProperty name="idCheck" property="*"/>
</jsp:useBean>
<%
    String username = idCheck.getUserName();
    String userpass = idCheck.getUserPass();

    if (idCheck.validate(username, userpass)) {
    session.setAttribute("user", username);

    Cookie userCookie = new Cookie("uname", username);
    response.addCookie(userCookie);
%>
<jsp:useBean id="d2go" class="dec.DecisionsToGo" scope="session" /> <jsp:forward page="home.jsp"/>
<%
    }
    else {
%>
<jsp:forward page="retry.jsp"/>
<%
    }
%>
```

The automated product research is started by selecting one of the products that are listed for each of the categories (HDTV, cell phones and PDAs). Submitting the query activates the information retrieval module, which gathers information from Amazon.com and Epinions.com.

```
<%@ page import="java.util.*" %>
<%@ page import="dec.InfoRetrieval" %>
<%@ page import="dec.TempContainer" %>
<%@ page import="dec.Profile" %>
<%@ page import="dec.UserQuery" %>
<%@ page import="dec.Report" %>
<%@ page import="dec.ExpertSystem" %>

<jsp:useBean id="d2go" class="dec.DecisionsToGo" scope="session" />
<jsp:useBean id="query" class="dec.ProductResearchBean" scope="request">
<jsp:setProperty name="query" property="*"/>
</jsp:useBean>

...

String pname = query.getProductName();
String pcat = query.getProductCategory();
d2go.setProductName(pname);

// Start information retrieval module
InfoRetrieval ir = new InfoRetrieval(pname,pcat);

// Perform searches
ir.getAmazonData();
ir.getEpinions();
```

```
// Create container for query
TempContainer tc = new TempContainer(ir.getApref(),ir.getEpref(), pcat);

// Save results as reference in bean
d2go.setIr(ir);
d2go.setTc(tc);

// Get Profile
Cookie[] cookies = request.getCookies(); String uname="";

    for(int i=0; i<cookies.length; i++) {
      Cookie cookie = cookies[i];
      if ("uname".equals(cookie.getName()))
        uname = cookie.getValue();
    }
Profile p = UserQuery.getProfileByName(uname);

// Start expert system module w/ tc and profile
ExpertSystem es = new ExpertSystem(tc, p, pname, pcat);
es.reason();

// Create report for query
Report r = es.generateReport(pname);
d2go.setES(es);
d2go.setReport(r);
response.sendRedirect("product_result.jsp");
```

The reasoning process of the Expert System will then yield a report that is used to dynamically create the summary page to which the user is redirected.

```
<jsp:useBean id="d2go" class="dec.DecisionsToGo" scope="session" />
<%
InfoRetrieval ir = d2go.getIr();
TempContainer tc = d2go.getTc();
ExpertSystem es = d2go.getES();
Report r = d2go.getReport();

...

<!-- text column -->
<table>
    <tr>
        <td>
        <b>
        <a class="hlink" href="details.jsp?page=<%= r.getTextTitle(1) %>">
            <b><%= r.getTextTitle(1) %></b>
            <br>
            <%= r.getTextItem(1) %>
        </a>
        </td>
    </tr>
    <tr>
        <td>
        <b>
        <a class="hlink" href="details.jsp?page=<%= r.getTextTitle(2) %>">
            <b><%= r.getTextTitle(2) %></b>
            <br>
            <%= r.getTextItem(2) %>
        </a>
        </td>
    </tr>
...
```

```
<!-- facts column -->
<%
System.out.println("  > Summary Report: Adding facts");
Vector fact = r.getFactTitles();
Vector items2 = r.getFactItems();

%>

<table>

<%
for(int i=0; i<fact.size(); i++) {
    String ft = (String) fact.get(i);
    String it = (String) items2.get(i);
%>
    <tr>
        <td>
        <a class="hlink" href="details.jsp?page=<%= ft %>">
            <b><%= ft %></b>
            <br>
            <%= it %>
        </a>
        </td>
    </tr>
<% } %>
</table>
```

The virtual salesperson mode works the other way round. The Expert System is started first, and will continue to ask the user questions about the product domain he or she has selected from the menu. Once enough information is available, the Expert System will make a product recommendation. It is important to note that the answers that are provided with each question are pre-ordered, reflecting the way the Expert System currently thinks about the user's needs.

```
<!-- Consume Profile Data and Retrieve Questions -->
<%
ExpertSystem expref = d2go.startExpertSystem();

expref.computeProfile();

expref.setGoal(0);

Question q = expref.getNextQuestion();

String qshort = q.getShortText();

%>
<b>Q:<%= q.getQuestionText() %></b><br><br>
<form action="advisor.jsp" method="post">
<%
for (Iterator iter = q.getAnswers().iterator(); iter.hasNext();) {
    String a = (String) iter.next();
    out.println("<input type=\"radio\" value=\"" + a + "\" name=\"" + qshort + \">" + a + "<br>");

}
%>
<br><center><input type="submit" name="submit" value="Next"></center><br> </form>
```

Subsequent posts will add the new evidence that was acquired through the previous question to the knowledge-base, as well as ask a new question.

```
<!-- Consume previous answers -->
<%
Question q = expref.getCurrentQuestion();

String qshort = q.getShortText();

if (request.getParameter(qshort) == null) {
    expert.addEvidence(qshort, false);
} else {
    expert.addEvidence(request.getParameter(qshort), true);
}
```

B.4.2 Cell Phone Interface

The heart of the cell phone communication is the mail scanner. It constantly checks a dedicated mail server for incoming messages. Once a message is received, the sender's phone number is extracted and the message is forwarded to the UserQuery class, which processes the query.

```
    public void run() {
        Properties props = new Properties();
        props.put("mail.smtp.host", "mail.com");
        Session session = Session.getDefaultInstance(props, null);
        try {
            boolean loop = true;
            int oldLength = 0;
            int newLength = 0;
            System.out.println("> Starting Mail Scanner");

            while (loop) {
                // Connect to Mailbox
                Store store = session.getStore("pop3");
                store.connect(host, user, pass);

                // Open Folder
                Folder folder = store.getFolder("Inbox");
                folder.open(Folder.READ_WRITE);

                // Get latest message
                Message message[] = folder.getMessages();
                newLength = message.length;
                if (newLength > oldLength) {
                    Message last = message[message.length - 1];

                    // Retrieve Sender and Message Content
                    Address sender = last.getFrom()[0];
                    String s = parseSender(sender.toString());
                    Object o = last.getContent();
                    String msg = o.toString();
                    String phone;

                    if (s.startsWith("1"))
                        phone = "001" + s.substring(1, s.indexOf("@"));
                    else
                        phone = "001" + s.substring(0, s.indexOf("@"));
                    if (!startup) {

                        // Analyze Content (Retrieve product)
```

```
                    String input = msg.toLowerCase();

                    // Execute User Query
                    new UserQuery(input, phone).start();
                } else
                    startup = false;
                oldLength = newLength;
            }
            folder.close(false);
            store.close();

            // Timeout
            sleep(20000);
        }
    } catch (NoSuchProviderException e) {
        e.printStackTrace();
    } catch (MessagingException e) {
        // TODO Auto-generated catch block
        e.printStackTrace();
    } catch (IOException e) {
        // TODO Auto-generated catch block
        e.printStackTrace();
    } catch (InterruptedException e) {
        // Go Back To Work
    }
}
```

After the query has been processed, the results are sent to the user's cell phone through a SMS gateway. The messages are sent as an XML file that contains all necessary data (including credentials that allow access to the gateway itself).

```
public static String send(int idsms,
                          String number,
                          String message,
                          int flashing,
                          String URLDeliveryNotification,
                          String URLNonDeliveryNotification) {

    // Setting up the XML file
    String content = "<?xml version=\"1.0\" encoding=\"ISO-8859-1\"?>\r\n"
        + "<aspsms>\r\n"
        + "<Userkey>" + userkey + "</Userkey>\r\n"
        + "<Password>" + password + "</Password>\r\n"
        + "<Originator>" + originator + "</Originator>\r\n"
        + "<Recipient>\r\n"
        + "<PhoneNumber>" + number + "</PhoneNumber>\r\n"
        + "<TransRefNumber>" + idsms + "</TransRefNumber>\r\n"
        + "</Recipient>\r\n"
        + "<MessageData>" + message + "</MessageData>\r\n"
        + "<FlashingSMS>" + flashing + "</FlashingSMS>\r\n"
        + "<URLDeliveryNotification>"
        + URLDeliveryNotification
        + "id=</URLDeliveryNotification>\r\n"
        + "<URLNonDeliveryNotification>"
        + URLNonDeliveryNotification
        + "id=</URLNonDeliveryNotification>\r\n"
        + "<Action>SendTextSMS</Action>\r\n"
        + "</aspsms>\r\n";

    InetAddress inetAddr = null;
    String xmlResult = "";

    try {
        URL aspsmsURL = new URL(xmlURL);
        URLConnection aspsmsCon = aspsmsURL.openConnection();
```

```
            aspsmsCon.setRequestProperty("Content-Type", "text/xml");
            aspsmsCon.setDoOutput(true);
            aspsmsCon.setDoInput(true);

            // Prepare the output stream that contains the XML data
            PrintWriter out = new PrintWriter(aspsmsCon.getOutputStream());

            char[] buffer = new char[1024 * 10];
            buffer = content.toCharArray();

            out.write(buffer, 0, content.length());
            out.close();

            // Read the answer from the SMS gateway in order to see if it was sent successfully
            BufferedReader in = new BufferedReader(
                                new InputStreamReader(aspsmsCon.getInputStream()));
            String inputLine = null;

            while ((inputLine = in.readLine()) != null) {
                xmlResult = xmlResult + inputLine;
                System.out.println(inputLine);
            }
            in.close();

        } catch (Exception ex) {
            System.out.println(ex.getMessage());
        }
        return xmlResult;
    }
```

The report is much shorter for cell phones. Initially, just a shortened version of most important textual information and a short list of the most significant facts are sent to the user. All messages can have at most 160 characters. Hence longer texts are shortened and the number of facts is limited dynamically

```
public void sendTextReport(Report r, String phone) {

    // If messages get too long, more reports are generated
    boolean report3 = false;
    int r3index = 0;
    boolean report4 = false;
    int r4index = 0;

    String report2 = ""; // First text report (reviews, pro/con, etc.)
    for (Iterator iter = r.getTextTitles().iterator(); iter.hasNext();) {
        String element = (String) iter.next();
        String proposal = "";
        String item = "";
        if (!(c>=r.getTextItems().size())) {
            item = (String) r.getTextItems().get(c);
        }
        if (item.length() > 0) {
            proposal = Report.cellshorten((element + "\n" + item));
            if (((report2.length() + proposal.length()) >= 160)) {
                report3 = true; // Overflow - create 3rd report
                r3index = c;
            }
            else report2 = report2 + proposal;
        }
        c++;
    }
```

```
    // If report3 is required, repeat:
    if (report3) {
        Vector remainingItems = r.getTextTitles().copyAll(r3index);
        String report3 = "";
        for (Iterator iter = remainingItems.iterator(); iter.hasNext();) {
            String element = (String) iter.next();
            String proposal = "";
            String item = "";
            if (!(c>=r.getTextItems().size())) {
                item = (String) r.getTextItems().get(c);
            }
            if (item.length() > 0) {
                proposal = Report.cellshorten((element + "\n" + item));
                if (((report3.length() + proposal.length()) >= 160)) {
                    report4 = true; // Overflow - create 4th report
                    r4index = c;
                }
                else report3 = report3 + proposal;
            }
            c++;
        }
    }

        // If report4 is required, repeat:
    if (report4) {
        Vector remainingItems2 = remainingItems.copyAll(r4index);
        String report4 = "";
        for (Iterator iter = remainingItems2.iterator(); iter.hasNext();) {
            String element = (String) iter.next();
            String proposal = "";
            String item = "";
            if (!(c>=r.getTextItems().size())) {
                item = (String) r.getTextItems().get(c);
            }
            if (item.length() > 0) {
                proposal = Report.cellshorten((element + "\n" + item));
                if (((report4.length() + proposal.length()) >= 160)) {
                    // SKIP
                }
                else report4 = report4 + proposal;
            }
            c++;
        }
    }
}
```

Appendix C

Rule-Base

C.1 Templates

```
(deftemplate stmt "Holds a statement as evidence"
    (slot desc)
    (slot cf (type FLOAT))
)

(deftemplate fact "Holds a object-value pair as evidence"
    (slot obj)
    (slot val)
    (slot cf (type FLOAT))
)

(deftemplate goal
    "Holds a goal"
    (slot val)
)

(deftemplate tier
    "Holds tier number"
    (slot num)
)
```

C.2 MYCIN-Style Certainty Factor Functions

The following functions are implementations of the antecedent and parallel combination rule for the MYCIN-style certainty factors. The antecedent rules combine the confidence that the input is accurate with the confidence to which extent a particular rule has an impact on the reasoning process. The parallel combination rule is activated whenever two hypotheses exist in parallel and have to be merged into one, taking their respective certainty factor into account.

```
;; Define functions for calculating the certainty factor
;; a) Antecedent Combination Rules
(deffunction arule-min ($?a)
   (bind ?x 2.0)
   (foreach ?y $?a
           (if (< ?y ?x) then
                   (bind ?x ?y))
   )
   (return ?x)
)

(deffunction arule-max ($?a)
   (bind ?x -2.0)
   (foreach ?y $?a
           (if (> ?y ?x) then
```

```
                    (bind ?x ?y))
    )
    (return ?x)
)

(deffunction arule-neg (?x)
    (return (* -1 ?x))
)

;; Combination rules for facts and statements

;; FACTS (defrule combination-facts
    "Apply parallel combination rule to facts"
    (declare (salience 50))
    ?fact1 <- (fact (obj ?fname) (val ?fval) (cf ?f1))
    ?fact2 <- (fact (obj ?fname) (val ?fval) (cf ?f2&:(neq ?fact1 ?fact2)))
    =>
    (if (and (>= ?f1 0) (>= ?f2 0)) then
        (modify ?fact1 (cf (- (+ ?f1 ?f2) (* ?f1 ?f2))))
    else
        (if (<= (* ?f1 ?f2) 0) then
            (modify ?fact1 (cf (/ (+ ?f1 ?f2) (- 1 (min (abs ?f1) (abs ?f2))))))
        else
            (if (and (< ?f1 0) (< ?f2 0)) then
                (modify ?fact1 (cf (+ (+ ?f1 ?f2) (* ?f1 ?f2))))
            )
        )
    )
    (retract ?fact2)
)

;; STATEMENTS (defrule combination-stmts
    "Apply parallel combination rule to statements"
    (declare (salience 50))
    ?fact1 <- (stmt (desc ?fname) (cf ?f1))
    ?fact2 <- (stmt (desc ?fname) (cf ?f2&:(neq ?fact1 ?fact2)))
    =>
    (if (and (>= ?f1 0) (>= ?f2 0)) then
        (modify ?fact1 (cf (- (+ ?f1 ?f2) (* ?f1 ?f2))))
    else
        (if (<= (* ?f1 ?f2) 0) then
            (modify ?fact1 (cf (/ (+ ?f1 ?f2) (- 1 (min (abs ?f1) (abs ?f2))))))
        else
            (if (and (< ?f1 0) (< ?f2 0)) then
                (modify ?fact1 (cf (+ (+ ?f1 ?f2) (* ?f1 ?f2))))
            )
        )
    )
    (retract ?fact2)
)
```

C.3 Sample Rules

The following is an example of a set of rules determining the importance of the picture quality for a HDTV type user query. The more evidence is available that the quality is high, the higher the certainty factor is.

```
(defrule arp-picture-quality-1
    "If picture quality high and pq keywords, then picture quality important 0.6"
    (fact (obj picture-quality) (val ?x&:(> ?x 3)) (cf ?c1))
    (stmt (desc all-digital) (cf ?c2))
    (stmt (desc accurate-film-frames) (cf ?c3))
```

```
    (stmt (desc digital-noise-reduction) (cf ?c4))
    =>
    (bind ?conf (* (arule-min ?c1 ?c2 ?c3 ?c4) 0.6))
    (assert (stmt (desc important-picture-quality) (cf ?conf)))
)

(defrule arp-picture-quality-2
    "If picture quality high and pq keywords, then picture quality important 0.4"
    (fact (obj picture-quality) (val ?x&:(> ?x 3)) (cf ?c1))
    (stmt (desc all-digital) (cf ?c2))
    (stmt (desc accurate-film-frames) (cf ?c3))
    =>
    (bind ?conf (* (arule-min ?c1 ?c2 ?c3) 0.4))
    (assert (stmt (desc important-picture-quality) (cf ?conf)))
)

(defrule arp-picture-quality-3
    "If picture quality high and pq keyword, then picture quality important 0.2"
    (fact (obj sound-quality) (val ?x&:(> ?x 3)) (cf ?c1))
    (stmt (desc all-digital) (cf ?c2))
    =>
    (bind ?conf (* (arule-min ?c1 ?c2) 0.2))
    (assert (stmt (desc important-picture-quality) (cf ?conf)))
)

(defrule arp-picture-quality-4
    "If picture quality high, then picture quality important 0.1"
    (fact (obj picture-quality) (val ?x&:(> ?x 3)) (cf ?c1))
    =>
    (bind ?conf (* ?c1 0.1))
    (assert (stmt (desc important-picture-quality) (cf ?conf)))
)

(defrule arp-picture-quality-5
    "If picture quality low, then picture quality important 0.6"
    (fact (obj picture-quality) (val ?x&:(< ?x 3)) (cf ?c1))
    (fact (obj eps-number-of-reviews) (val ?y) (cf ?c2))
    =>
    (bind ?conf (* (arule-min ?c1 ?c2) 0.6))
    (assert (stmt (desc important-picture-quality) (cf ?conf)))
)
```

The following is a rule about shopping. It expresses the idea that if the product is available at well-known stores and the price is close to the lower trusted prices found in other web shops, then the customer might be more interested in this fact. Three rules of this type express the varying degree depending on the price.

```
(defrule arp-wellknown-rule-1
    "If product available at well-known store and not more than 15% more than lowest trusted 0.5"
    (fact (obj well-known-store) (val ?x) (cf ?c1))
    (fact (obj lowest-trusted-price) (val ?l) (cf ?cl))
    (fact (obj ?x) (val ?y&:(< (/ ?y ?l) 1.15)) (cf ?c2))
    =>
    (bind ?conf (* (arule-min ?c1 ?cl ?c2) 0.5))
    (assert (stmt (desc important-well-known-store) (cf ?conf)))
)

(defrule arp-wellknown-rule-2
    "If product available at well-known store and not more than 33% more than lowest trusted 0.1"
    (fact (obj well-known-store) (val ?x) (cf ?c1))
    (fact (obj lowest-trusted-price) (val ?l) (cf ?cl))
    (fact (obj ?x) (val ?y&:(< (/ ?y ?l) 1.33)) (cf ?c2))
    =>
    (bind ?conf (* (arule-min ?c1 ?cl ?c2) 0.1))
    (assert (stmt (desc important-well-known-store) (cf ?conf)))
```

```
)

(defrule arp-wellknown-rule-3
    "If product available at well-known store and not more than 5% more than lowest trusted 0.8"
    (fact (obj well-known-store) (val ?x) (cf ?c1))
    (fact (obj lowest-trusted-price) (val ?l) (cf ?cl))
    (fact (obj ?x) (val ?y&:(<= (/ ?y ?l) 1.05)) (cf ?c2))
    =>
    (bind ?conf (* (arule-min ?c1 ?cl ?c2) 0.8))
    (assert (stmt (desc important-well-known-store) (cf ?conf)))
)
```

Other examples of rules:

```
(defrule arp-general-rating-rule
    "If rating is less than 3.5, important 0.75"
    (fact (obj total-rating) (val ?x&:(<= ?x 3.5)) (cf ?c1))
    (fact (obj total-reviews) (val ?y&:(>= ?y 5)) (cf ?c2))
    =>
    (bind ?conf (* (arule-min ?c1 ?c2) 0.75))
    (assert (stmt (desc important-warning-total-rating) (cf ?conf)))
)

(defrule arp-general-rating-rule-2
    "If rating is more than 4.5, important 0.75"
    (fact (obj total-rating) (val ?x&:(>= ?x 4.5)) (cf ?c1))
    (fact (obj total-reviews) (val ?y&:(>= ?y 5)) (cf ?c2))
    =>
    (bind ?conf (* (arule-min ?c1 ?c2) 0.75))
    (assert (stmt (desc important-high-rating) (cf ?conf)))
)

(defrule arp-eps-vs-amazon-ratings-1
    "If eps more significant than amazon, then eps-rating important 0.5"
    (fact (obj eps-number-of-reviews) (val ?x) (cf ?c1))
    (fact (obj amazon-number-of-reviews) (val ?y) (cf ?c2&:(>= ?c1 ?c2)))
    =>
    (bind ?conf (* (arule-min ?c1 ?c2) 0.5))
    (assert (stmt (desc important-eps-rating) (cf ?conf)))
)

(defrule arp-eps-vs-amazon-ratings-2
    "If amazon more significant than eps, then amazon-rating important 0.5"
    (fact (obj eps-number-of-reviews) (val ?x) (cf ?c1))
    (fact (obj amazon-number-of-reviews) (val ?y) (cf ?c2&:(<= ?c1 ?c2)))
    =>
    (bind ?conf (* (arule-min ?c1 ?c2) 0.5))
    (assert (stmt (desc important-amazon-rating) (cf ?conf)))
)

(defrule arp-total-reviews-1
    "If # of reviews > 10, then important 0.25"
    (fact (obj total-reviews) (val ?x&:(> ?x 10)) (cf ?c1))
    =>
    (bind ?conf (* ?c1 0.25))
    (assert (stmt (desc important-total-rating) (cf ?conf)))
)

(defrule arp-total-reviews-2
    "If # of reviews > 20, then important 0.6"
    (fact (obj total-reviews) (val ?x&:(> ?x 20)) (cf ?c1))
    =>
    (bind ?conf (* ?c1 0.6))
    (assert (stmt (desc important-total-rating) (cf ?conf)))
)
```

```
(defrule arp-total-reviews-3
    "If # of reviews > 50, then important 0.8"
    (fact (obj total-reviews) (val ?x&:(> ?x 50)) (cf ?c1))
    =>
    (bind ?conf (* ?c1 0.8))
    (assert (stmt (desc important-total-rating) (cf ?conf)))
)

(defrule arp-total-reviews-4
    "If # of reviews <5, then important 0.1"
    (fact (obj total-reviews) (val ?x&:(<= ?x 5)) (cf ?c1))
    =>
    (bind ?conf (* ?c1 0.1))
    (assert (stmt (desc important-total-rating) (cf ?conf)))
)

(defrule arp-negative-positive-reviews-1
    "If negative credibility > positive credibility, then important 0.8"
    (fact (obj amazon-negative-reviews) (val ?x) (cf ?c1))
    (fact (obj amazon-positive-reviews) (val ?y) (cf ?c2&:(< ?c2 ?c1)))
    (fact (obj amazon-number-of-reviews) (val ?z) (cf ?c3))
    =>
    (bind ?conf (* (arule-min ?c1 ?c2 ?c3) 0.8))
    (assert (stmt (desc important-warning-neg-pos-cred) (cf ?conf)))
)

(defrule arp-negative-positive-reviews-2
    "If negative credibility < positive credibility, then important 0.4"
    (fact (obj amazon-negative-reviews) (val ?x) (cf ?c1))
    (fact (obj amazon-positive-reviews) (val ?y) (cf ?c2&:(> ?c2 ?c1)))
    (fact (obj amazon-number-of-reviews) (val ?z) (cf ?c3))
    =>
    (bind ?conf (* (arule-min ?c1 ?c2 ?c3) 0.4))
    (assert (stmt (desc important-neg-pos-cred) (cf ?conf)))
)

(defrule arp-negative-positive-reviews-3
    "If negative credibility > 90, then important 0.4"
    (fact (obj amazon-negative-reviews) (val ?x) (cf ?c1&:(>= ?c1 0.9)))
    (fact (obj amazon-number-of-reviews) (val ?z&:(>= ?z 5)) (cf ?c3))
    =>
    (bind ?conf (* ?c1 0.4))
    (assert (stmt (desc important-warning-negative-cred) (cf ?conf)))
)

(defrule arp-negative-positive-reviews-4
    "If positive credibility > 90, then important 0.4"
    (fact (obj amazon-positive-reviews) (val ?x) (cf ?c1&:(>= ?c1 0.9)))
    (fact (obj amazon-number-of-reviews) (val ?z&:(>= ?z 5)) (cf ?c3))
    =>
    (bind ?conf (* ?c1 0.4))
    (assert (stmt (desc important-positive-cred) (cf ?conf)))
)



(defrule arp-base-price-1
    "If base price is close to lowest trusted price, then trusted price important 0.4"
    (fact (obj lowest-base-price) (val ?x) (cf ?c1))
    (fact (obj lowest-trusted-price) (val ?y&:(> (/ ?y ?x) 0.95)) (cf ?c2))
    =>
    (bind ?conf (* (arule-min ?c1 ?c2) 0.4))
    (assert (stmt (desc important-low-trust-price) (cf ?conf)))
)

(defrule arp-price-0
    "If Lowest Trusted Price is less than 95% of amazon, then importance -0.2"
    (fact (obj amazon-price) (val ?x) (cf ?c1))
```

```
    (fact (obj lowest-trusted-price) (val ?y&:(< (/ ?y ?x) 0.95)) (cf ?c2))
    =>
    (bind ?conf (* (arule-min ?c1 ?c2) -0.2))
    (assert (stmt (desc important-low-trust-price) (cf ?conf)))
)

(defrule arp-price-1
    "If Lowest Trusted Price is less than 90% of amazon, then importance 0.3"
    (fact (obj amazon-price) (val ?x) (cf ?c1))
    (fact (obj lowest-trusted-price) (val ?y&:(< (/ ?y ?x) 0.9)) (cf ?c2))
    =>
    (bind ?conf (* (arule-min ?c1 ?c2) 0.3))
    (assert (stmt (desc important-low-trust-price) (cf ?conf)))
)

(defrule arp-price-2
    "If Lowest Trusted Price is less than 80% of amazon, then importance 0.6"
    (fact (obj amazon-price) (val ?x) (cf ?c1))
    (fact (obj lowest-trusted-price) (val ?y&:(< (/ ?y ?x) 0.8)) (cf ?c2))
    =>
    (bind ?conf (* (arule-min ?c1 ?c2) 0.6))
    (assert (stmt (desc important-low-trust-price) (cf ?conf)))
)

(defrule arp-price-3
    "If Lowest Trusted Price is less than 70% of amazon, then importance 0.9"
    (fact (obj amazon-price) (val ?x) (cf ?c1))
    (fact (obj lowest-trusted-price) (val ?y&:(< (/ ?y ?x) 0.7)) (cf ?c2))
    =>
    (bind ?conf (* (arule-min ?c1 ?c2) 0.9))
    (assert (stmt (desc important-low-trust-price) (cf ?conf)))
)

(defrule arp-salesrank-1
    "If salesrank < 500, then importance 0.9"
    (fact (obj amazon-salesrank) (val ?x&:(< ?x 500)) (cf ?c))
    =>
    (bind ?conf (* ?c 0.9))
    (assert (stmt (desc important-salesrank) (cf ?conf)))
)

(defrule arp-salesrank-2
    "If salesrank < 1000, then importance 0.8"
    (fact (obj amazon-salesrank) (val ?x&:(< ?x 1000)) (cf ?c))
    =>
    (bind ?conf (* ?c 0.8))
    (assert (stmt (desc important-salesrank) (cf ?conf)))
)

(defrule arp-salesrank-3
    "If salesrank < 5000, then importance 0.5"
    (fact (obj amazon-salesrank) (val ?x&:(< ?x 5000)) (cf ?c))
    =>
    (bind ?conf (* ?c 0.5))
    (assert (stmt (desc important-salesrank) (cf ?conf)))
)

(defrule arp-salesrank-4
    "If salesrank < 10000, then importance 0.2"
    (fact (obj amazon-salesrank) (val ?x&:(< ?x 10000)) (cf ?c))
    =>
    (bind ?conf (* ?c 0.2))
    (assert (stmt (desc important-salesrank) (cf ?conf)))
)

(defrule arp-salesrank-5
    "If salesrank > 10000, then importance 0.1"
    (fact (obj amazon-salesrank) (val ?x&:(> ?x 10000)) (cf ?c))
```

```
    =>
    (bind ?conf (* ?c 0.1))
    (assert (stmt (desc important-salesrank) (cf ?conf)))
)

(defrule arp-salesrank-6
    "If salesrank > 30000, then importance 0.5"
    (fact (obj amazon-salesrank) (val ?x&:(> ?x 30000)) (cf ?c))
    =>
    (bind ?conf (* ?c 0.5))
    (assert (stmt (desc important-salesrank) (cf ?conf)))
)

(defrule arp-salesrank-6
    "If salesrank > 80000, then importance 0.6"
    (fact (obj amazon-salesrank) (val ?x&:(> ?x 80000)) (cf ?c))
    =>
    (bind ?conf (* ?c 0.6))
    (assert (stmt (desc important-warning-salesrank) (cf ?conf)))
)


;; Virtual Sales Person Rules

;; Cell Phones (Examples)

(defrule tier0-range-georgia
    "If area=georgia AND home=georgia, then stationary 0.8"
    (goal 0)
    (fact (obj user-area) (val georgia) (cf ?x1))
    (fact (obj user-home) (val georgia) (cf ?x2))
    =>
    (bind ?cf_mob (* (arule-min ?x1 ?x2) 0.8))
    (assert (stmt (desc (user-stationary) (cf ?cf_mob))))
)

(defrule tier0-range-nyc
    "If area=nyc AND home=nyc, then stationary 0.8"
    (goal 0)
    (fact (obj user-area) (val nyc) (cf ?x1))
    (fact (obj user-home) (val nyc) (cf ?x2))
    =>
    (bind ?cf_mob (* (arule-min ?x1 ?x2) 0.8))
    (assert (stmt (desc (user-stationary) (cf ?cf_mob))))
)

(defrule tier0-range-penn
    "If area=penn AND home=penn, then stationary 0.8"
    (goal 0)
    (fact (obj user-area) (val penn) (cf ?x1))
    (fact (obj user-home) (val penn) (cf ?x2))
    =>
    (bind ?cf_mob (* (arule-min ?x1 ?x2) 0.8))
    (assert (stmt (desc (user-stationary) (cf ?cf_mob))))
)

(defrule tier0-range-utah
    "If area=utah AND home=utah, then stationary 0.8"
    (goal 0)
    (fact (obj user-area) (val utah) (cf ?x1))
    (fact (obj user-home) (val utah) (cf ?x2))
    =>
    (bind ?cf_mob (* (arule-min ?x1 ?x2) 0.8))
    (assert (stmt (desc (user-stationary) (cf ?cf_mob))))
)

(defrule tier0-range-nation
    "If area is nation, then stationary -0.8"
```

```
    (goal 0)
    (fact (obj user-area) (val nation) (cf ?x1))
    =>
    (bind ?cf_mob (* ?x1 -0.8))
    (assert (stmt (desc (user-stationary) (cf ?cf_mob))))
)

(defrule tier0-range-travel
    "If user likes to travel, then stationary -0.3"
    (goal 0)
    (fact (obj hobby) (val traveling) (cf ?x1))
    =>
    (bind ?cf_mob (* ?x1 -0.5))
    (assert (stmt (desc (user-stationary) (cf ?cf_mob))))
)

(defrule tier0-range-hiking
    "If user likes to hike, then stationary -0.3"
    (goal 0)
    (fact (obj hobby) (val hiking) (cf ?x1))
    =>
    (bind ?cf_mob (* ?x1 -0.3))
    (assert (stmt (desc (user-stationary) (cf ?cf_mob))))
)
```

The following is an excerpt from the rule set of the virtual salesperson module. It deals with the question whether or not the user can be considered mobile.

```
;; ============================================
;; Virtual Salesperson Knowledge
;; ============================================

;; Profile (Tier-0)
;; Job, Age, Hobbies get asserted through Java API from DB

;; Mobility (defrule tier0-range-georgia
    "If area=georgia AND home=georgia, then stationary 0.8"
    (tier (num 0))
    (fact (obj user-home) (val georgia) (cf ?x2))
    =>
    (bind ?cf_mob (* ?x2 0.8))
    (assert (fact (obj mobility) (val stationary) (cf ?cf_mob)))
)

(defrule tier0-range-nyc
    "If area=nyc AND home=nyc, then stationary 0.8"
    (tier (num 0))
    (fact (obj user-home) (val nyc) (cf ?x2))
    =>
    (bind ?cf_mob (* ?x2 0.8))
    (assert (fact (obj mobility) (val stationary) (cf ?cf_mob)))
)

(defrule tier0-range-penn
    "If area=penn AND home=penn, then stationary 0.8"
    (tier (num 0))
    (fact (obj user-home) (val penn) (cf ?x2))
    =>
    (bind ?cf_mob (* ?x2 0.8))
    (assert (fact (obj mobility) (val stationary) (cf ?cf_mob)))
)

(defrule tier0-range-utah
    "If area=utah AND home=utah, then stationary 0.8"
    (tier (num 0))
```

```
    (fact (obj user-home) (val utah) (cf ?x2))
    =>
    (bind ?cf_mob (* ?x2 0.8))
    (assert (fact (obj mobility) (val stationary) (cf ?cf_mob)))
)

(defrule tier0-range-travel
    "If user likes to travel, then stationary -0.3"
    (tier (num 0))
    (fact (obj user-hobby) (val traveling) (cf ?x1))
    =>
    (bind ?cf_mob1 (* ?x1 -0.5))
    (assert (fact (obj mobility) (val stationary) (cf ?cf_mob1)))
)

(defrule tier0-range-hiking
    "If user likes to hike, then stationary -0.3"
    (tier (num 0))
    (fact (obj user-hobby) (val hiking) (cf ?x1))
    =>
    (bind ?cf_mob2 (* ?x1 -0.3))
    (assert (fact (obj mobility) (val stationary) (cf ?cf_mob2)))
)

(defrule tier0-mobility-01283
    "If user stationary, then mobile -0.9"
    (tier (num 0))
    (fact (obj mobility) (val stationary) (cf ?x1))
    =>
    (bind ?conf (* -1 ?x1))
    (assert (fact (obj mobility) (val mobile) (cf ?conf)))
)
```

Another example of virtual salesperson rules comes from the HDTV specific rules:

```
;; Assumptions (defrule tier0-screen-size-assumption-1
    "If user is teen and likes sports, then screen size big 0.4"
    (tier (num 0))
    (goal (val hdtv))
    (fact (obj user-hobby) (val sports) (cf ?c1))
    (fact (obj user-age) (val teen) (cf ?c2))
    =>
    (bind ?conf (* (arule-min ?c1 ?c2) 0.4))
    (assert (fact (obj goal-screen) (val large) (cf ?conf)))
)

(defrule tier0-screen-size-assumption-1
    "If user is young and likes sports, then screen size big 0.5"
    (tier (num 0))
    (goal (val hdtv))
    (fact (obj user-hobby) (val sports) (cf ?c1))
    (fact (obj user-age) (val young) (cf ?c2))
    =>
    (bind ?conf (* (arule-min ?c1 ?c2) 0.5))
    (assert (fact (obj goal-screen) (val large) (cf ?conf)))
)

(defrule tier0-purpose-assumption
    "If user is teenager, then purpose is bed room 0.4"
    (tier (num 0))
    (goal (val hdtv))
    (fact (obj user-age) (val teen) (cf ?c1))
    =>
    (bind ?conf (* ?c1 0.4))
    (assert (fact (obj goal-purpose) (val bedroom) (cf ?conf)))
```

```
    (assert (fact (obj goal-device) (val compact) (cf ?conf)))
    (assert (fact (obj goal-screen) (val small) (cf ?conf)))
)

(defrule tier0-purpose-assumption-2
    "If user is young, then purpose is living room 0.4"
    (tier (num 0))
    (goal (val hdtv))
    (fact (obj user-age) (val young) (cf ?c1))
    =>
    (bind ?conf (* ?c1 0.4))
    (assert (fact (obj goal-purpose) (val livingroom) (cf ?conf)))
    (assert (fact (obj goal-device) (val mid-sized) (cf ?conf)))
    (assert (fact (obj goal-device) (val compact) (cf (* 0.5 ?conf))))
    (assert (fact (obj goal-screen) (val medium) (cf (* 0.8 ?conf))))
    (assert (fact (obj goal-screen) (val small) (cf (* 0.6 ?conf))))
)

(defrule tier0-purpose-assumption-3
    "If user is senior, then purpose is livingroom 0.4"
    (tier (num 0))
    (goal (val hdtv))
    (fact (obj user-age) (val senior) (cf ?c1))
    =>
    (bind ?conf (* ?c1 0.4))
    (assert (fact (obj goal-purpose) (val livingroom) (cf ?conf)))
    (assert (fact (obj goal-device) (val mid-sized) (cf ?conf)))
    (assert (fact (obj goal-device) (val big) (cf (* 0.4 ?conf))))
    (assert (fact (obj goal-screen) (val medium) (cf (* 0.9 ?conf))))
    (assert (fact (obj goal-screen) (val large) (cf (* 0.9 ?conf))))
)

(defrule tier0-purpose-assumption-4
    "If user is old, then purpose is livingroom 0.4"
    (tier (num 0))
    (goal (val hdtv))
    (fact (obj user-age) (val old) (cf ?c1))
    =>
    (bind ?conf (* ?c1 0.4))
    (assert (fact (obj goal-purpose) (val livingroom) (cf ?conf)))
    (assert (fact (obj goal-device) (val mid-sized) (cf ?conf)))
    (assert (fact (obj goal-device) (val big) (cf (* 0.9 ?conf))))
    (assert (fact (obj goal-screen) (val medium) (cf ?conf)))
    (assert (fact (obj goal-screen) (val large) (cf ?conf)))
)

(defrule tier0-sizes-assumption
    "If user is mobile, size should be smaller 0.3"
    (tier (num 0))
    (goal (val hdtv))
    (fact (obj mobility) (val mobile) (cf ?c1))
    =>
    (bind ?conf (* ?c1 0.3))
    (assert (fact (obj goal-screen) (val small) (cf ?conf)))
)

(defrule tier0-sizes-assumption-2
    "If user hobby movies, size be big 0.3"
    (tier (num 0))
    (goal (val hdtv))
    (fact (obj user-hobby) (val movies) (cf ?c1))
    =>
    (bind ?conf (* ?c1 0.3))
    (assert (fact (obj goal-screen) (val large) (cf ?conf)))
)
```

Once enough evidence for the first tier is gathered, the expert system can make a guess on which HDTV type would be most suitable for the customer. This allows other rules in the second tier to fire and will lead to more specific questions regarding the particular type of TV.

```
(defrule tier1-purpose-solution
    "If purpose is living room and mid-sized, then plasma"
    (tier (num 1))
    (goal (val hdtv))
    (fact (obj goal-purpose) (val livingroom) (cf ?c1))
    (fact (obj goal-device) (val mid-sized) (cf ?c2))
    =>
    (bind ?conf (* (arule-min ?c1 ?c2) 0.8))
    (assert (fact (obj goal-class) (val plasma) (cf ?conf)))
)

(defrule tier1-purpose-solution-2
    "If purpose is living room and big, then projection"
    (tier (num 1))
    (goal (val hdtv))
    (fact (obj goal-purpose) (val livingroom) (cf ?c1))
    (fact (obj goal-device) (val big) (cf ?c2))
    =>
    (bind ?conf (* (arule-min ?c1 ?c2) 0.8))
    (assert (fact (obj goal-class) (val projection) (cf ?conf)))
)

(defrule tier1-purpose-solution-3
    "If purpose is living room and compact, then lcd"
    (tier (num 1))
    (goal (val hdtv))
    (fact (obj goal-purpose) (val livingroom) (cf ?c1))
    (fact (obj goal-device) (val compact) (cf ?c2))
    =>
    (bind ?conf (* (arule-min ?c1 ?c2) 0.8))
    (assert (fact (obj goal-class) (val lcd) (cf ?conf)))
)

(defrule tier1-purpose-solution-4
    "If purpose is bed room and mid-sized, then plasma"
    (tier (num 1))
    (goal (val hdtv))
    (fact (obj goal-purpose) (val bedroom) (cf ?c1))
    (fact (obj goal-device) (val mid-sized) (cf ?c2))
    =>
    (bind ?conf (* (arule-min ?c1 ?c2) 0.5))
    (assert (fact (obj goal-class) (val plasma) (cf ?conf)))
)

(defrule tier1-purpose-solution-5
    "If purpose is bed room and big, then projection"
    (tier (num 1))
    (goal (val hdtv))
    (fact (obj goal-purpose) (val bedroom) (cf ?c1))
    (fact (obj goal-device) (val big) (cf ?c2))
    =>
    (bind ?conf (* (arule-min ?c1 ?c2) 0.5))
    (assert (fact (obj goal-class) (val projection) (cf ?conf)))
)

(defrule tier1-purpose-solution-6
    "If purpose is bed room and compact, then lcd"
    (tier (num 1))
    (goal (val hdtv))
    (fact (obj goal-purpose) (val bedroom) (cf ?c1))
```

```
    (fact (obj goal-device) (val compact) (cf ?c2))
    =>
    (bind ?conf (* (arule-min ?c1 ?c2) 0.5))
    (assert (fact (obj goal-class) (val lcd) (cf ?conf)))
)

(defrule tier1-purpose-solution-7
    "If purpose is kitchen and mid-sized, then lcd"
    (tier (num 1))
    (goal (val hdtv))
    (fact (obj goal-purpose) (val kitchen) (cf ?c1))
    (fact (obj goal-device) (val mid-sized) (cf ?c2))
    =>
    (bind ?conf (* (arule-min ?c1 ?c2) 0.5))
    (assert (fact (obj goal-class) (val lcd) (cf ?conf)))
)

(defrule tier1-purpose-solution-8
    "If purpose is kitchen and big, then plasma"
    (tier (num 1))
    (goal (val hdtv))
    (fact (obj goal-purpose) (val kitchen) (cf ?c1))
    (fact (obj goal-device) (val big) (cf ?c2))
    =>
    (bind ?conf (* (arule-min ?c1 ?c2) 0.5))
    (assert (fact (obj goal-class) (val plasma) (cf ?conf)))
)

(defrule tier1-purpose-solution-9
    "If purpose is bed room and compact, then lcd"
    (tier (num 1))
    (goal (val hdtv))
    (fact (obj goal-purpose) (val kitchen) (cf ?c1))
    (fact (obj goal-device) (val compact) (cf ?c2))
    =>
    (bind ?conf (* (arule-min ?c1 ?c2) 0.5))
    (assert (fact (obj goal-class) (val lcd) (cf ?conf)))
)
```

Rules can also assert question markers. In certain situations, there may be a need for further information about the user's needs. If that is the case, then a rule like the one below will indicate that it would be helpful to know more about the size of the living room, in order to make a better decision regarding screen size. However, all questions are in competition with each other in each step, so that it is not guaranteed that all questions get asked eventually (e.g. if the system moves to another tier, the remaining questions from the lower tier are discarded).

```
;; Question Example (HDTV)

(defrule vs-projected-use
    "if sound importance high, then ask about price"
    (goal 0)
    (fact (obj importance) (val picture) (cf ?c&:(> ?c 0.5)))
    =>
    (assert (fact (obj question) (val price) (cf ?c)))
)
```