Design and Implementation of E-Mail Agent

Project

submitted by
MOHIUDDIN SYED FAISAL
Information and Communication Systems
Masters Program

Supervisors:
Prof. Dr. Florian Matthes
Dipl.-Inf. Ulrike Steffens
Kiwi Logic: Mr. Olaf Voss

Technische Universitaet Hamburg-Harburg
Arbeitsbereich Softwaresysteme

Contents

List of Figures

Acknowledgement

I am thankful to KIWI LOGIC, which incidentely has won 1 million DM for being the best start up internet company, for offering me an opportunity to undertake my project, on the hottest topic in IT field.

It was of a tremendous experience to work in a jovial and interactive atmosphere, with prompt advice and help from all employees of Kiwi logic .

I am thankful to Mr. Rainer and Mr. Olaf Voss on their contributions in my completion of project.

I thank Mrs. Ulrike Steffens for having supervised my project and helping me in all possible ways and for allotting her time in first preference to me at times of dire need and also for having exercised patience at the time of my pause.

I thank Prof. Dr Florian Mathias for his support that he provided Me an opportunity to do project in industry related to E.commerce.

Mohiuddin Syed Faisal

M.S (ICS)

Chapter 1

Introduction

The pace of our daily life has been increasing for several decades. Our needs have multiplied as new products have appeared and then been replaced after a few years, or even months of existence by a more fashionable product or one of higher performance. the life cycle of technologies used in consumer and professional software products are becoming shorter. This acceleration is an inherent fact of our consumer society.

Natural language interpretation will transform the way people use the Internet. It will help companies to take full advantage of the potential of e-commerce by making it more natural and easy to communicate with websites. Natural language interpretation is essential for taking advantage of the full commercial potential of the Internet. Within a few years, the World Wide Web has developed into an economic force that is transforming virtually every business and is changing our day-to-day life dramatically. Still, most people have difficulty understanding this rapid development. They have trouble using the Internet to their advantage.

So far, three out of four people in the USA do not use the Internet at all, and those who do have a hard time finding the information they are looking for. Two out of three online shopping transactions get canceled. The largest part of the enormous potential e-commerce revenues seems still out of reach.

The reason for this is that using the Internet is still far from being easy and natural.

Many websites offer hundreds of pages of information, but can not answer user questions directly. They often have keyword search, but do not really understand what kind of information the customer wants. They have e-mail buttons, but can not respond to "instant messages", and users often have to wait for days until they get an answer to their e-mails, if they get it at all. Most websites feature nice graphics and animation, but still appear technical, cold and unemotional to most users. They provide detailed information on where users clicked, but still most companies do not know what their customers were looking for, let alone what they think of their products.

There are two ways of solving this problem. One involves human online help, for example through online call centers using voice-over-I p or "live chat"-techniques. This certainly is a good way to help users, but it is also very expensive. Often capacity is limited, forcing users to wait or denying them the service during peak times.

The other way relies on creating a natural language interpretation system that is able to answer typed user questions, using some kind of "artificial intelligence" technology. Such a system can deliver instant information about companies and products on demand, without forcing the user to specify the request in certain syntax. A well-designed natural language system can increase online sales by answering last minute questions, preventing users from abandoning their shopping carts. It can even act as a virtual salesperson, recommending products based on the user's individual needs and using cross-selling methods. It saves money by answering user questions directly, preventing unnecessary e-mails and help line calls. It also offers a lot of fun: By displaying a certain "personality", it can create a strong emotional relation to customers.

The motivation of Project was to improve the performance of Knowledge based agent " Lingubot " at Kiwi logic. Sometimes customer is in hurry and he does not want to wait for a long for his e-mail enquiry. Then After Implementing my application the user can write e-mail to the semi automatic agent and receive the reply without any delay, all the time and he also does not need any computer know how.

Chapter 2

Intelligent Agent

If an application performs an intelligent task automatically that is of the sort that one could imagine a person performing manually, then the application can be called an agent. So, an agent is now a label for an application belonging to a certain class, the class of applications that automate intelligent tasks people could perform manually[1].

An agent is delegated to perform some task for the user and is given constraints under which it can operate . The agents processing is attributable to knowledge of the task and the domain in which the task is situated. An agent can also learn to perform a given task . For example, an agent that uses some software tool to perform tasks could learn properties of the tool that would enable it to use the tool more effectively over time. Similarly, an agent that performs in some domain can learn properties of the domain that enable it to perform tasks more effectively over time.

The use of intelligent agent technology on the Internet is growing rapidly, the explosive expansion of information on the Internet, particularly, on the World Wide Web, and the lack of organization and quality control of that information has led to the urgent need for an intelligent and personalized way[2] to find and use only the specific Internet information in which one is interested. Intelligent agents provide the one possible solution for this growing problem.

There are several, partially overlapping ways in which intelligent agents are being utilized on the Internet. These include:

- Searching the Internet for information pertinent to the user
- Retrieving information in a desired frequency and manner
- Filtering information, selecting only the appropriate items of interest
- Present information in a desired form
- Alerting the user when information of interest is available
- Alerting the user when specific events occur or when a parameter reaches a particular value

- Speeding use of the Internet for the user. For example when an Internet user is reading one web page, intelligent agent software anticipates the next request by loading linked pages into the browser. Then when the user selects a link, the new page loads as if it were already in the cache.
- Aiding in the management of World Wide Web sites

Many additional types of agent applications are being developed and deployed as well, including applications in electronic commerce and business.

2.1 Electronic Commerce Agents

Intelligent agents enable new and improved approaches to electronic commerce. Agents can aid a user in searching for a product or service of interest. For example, agent searches online bookstores to find information and prices for a particular book, or to find a list of books on a particular topic. Also, agent systems have been developed to aid a user in the selection of Recorded music. By explicit input from the user and by monitoring the user's selections over time, they learn the user's music preferences and then suggest for purchase CDs that likely would be of most interest to the user.

Agent systems can provide personalized shopping, aiding both the shopper and the vendor. Agent-enabled electronic shopping services have been developed to learn an Internet shopper's browsing and shopping habits. Then they can assist the user by allowing each vendor's display to be customized in a way that is most convenient to the shopper, while also aiding the vendor by allowing the vendor to individualize the marketing of its products.

2.2 Business Application Agents

Intelligent agent technology is being used in a wide variety of applications related to business operations. Agent-based systems are in use in customer service, human resources, manufacturing, sales, and several other areas.

Intelligent agents are being used as automated customer service agents, providing information to customers without the need for human intervention. For example, intelligent agents are used in systems to provide banking information to bank customers and information on the status of shipments to customers of shipping companies. Customer service systems such as they are less expensive than human customer service personnel. They are often faster in providing standard requested information, and they allow the human customer service personnel to be used to provide answers to more difficult or unusual requests. Another major advantage of the agent systems is that they can offer some degree of intelligent service to customer 24 hours a day, 7 days a week.

Figure 2.1 , illustrates a customer support agent that carries out a dialog with users on the Web to solve routine hardware and software problems. For example, a user may be having trouble with a printer or a word-processing

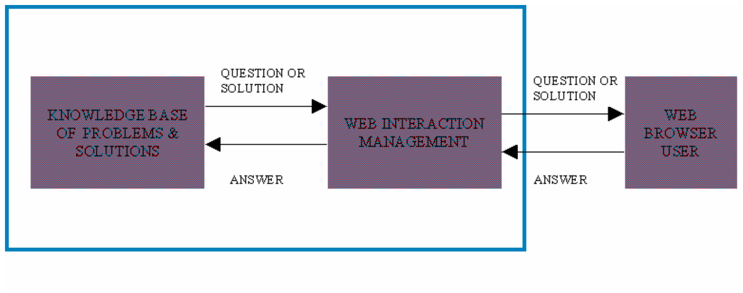

Figure 2.1: Customer Support Agent

application. Instead of calling a help desk about the problem, the user could have a dialog with the support agent on the Web. The support agent becomes the first line of response to customer problems. The agent is meant to only address relatively simple problems that are easy for the user to specify and can be resolved without complex steps.

The agent starts the dialog by asking the user a question. The set of possible answers to the question are also providing. Since the goal is to keep the dialog simple, questions have three to four possible answers. For example, the question may ask the type of printer being used. The user selects one response from the available choices. The user's response tells the agent a single fact about the problem. The user can also indicate that the question is irrelevant to the problem.

The response to each question is added to the representation of the user's problem being assembled by the agent. The agent's knowledge base contains additional knowledge about the class of problems it can resolve. This domain knowledge allows the agent to infer additional knowledge about the user's problem. For example, knowing what printer the user has, the agent could infer that this type of printer has been having ribbon problems. Then the agent could ask the user to check if the ribbon is properly installed.

The agent's representation of problems consists of a set of problems attributes with specified values. The solution to a problem is represented as a text file describing the solution. Essentially, the purpose of the dialog is to enable the agent to understand the user's problem. This understanding is represented as the values of attributes that underlie the questions he user Is asked. When the user's problem matches the stored representation of a problem in the knowledge base, the agent retrieves the corresponding solution file and provides the contents of the file to the user.

2.3 Interface Agents

There are two general types of interface agents. One class of interface agents helps users by filtering various data sources. For example, consider an interface agent that intelligently filters a user's electronic mail. If the mail agent is any good, it weeds out messages that the user is unlikely to find useful. The main benefit provide by this type of an interface agent is a reduction of the amount of data that users need to process.

Interface agents for data reduction generally are built as rule-based programs that apply profile to the items in the data sources. The profile is a specification of the attributes of desirable items in the data source. For example, a mail agent can use a set of rules that is applied to an incoming message to determine its usefulness for a user.

Interface agents for mediating in the use of applications are usually built as knowledge-based systems. A mediating agent has knowledge of the application that it uses and knowledge of the tasks that it solves using the application. The agent reduces the input task into a set of actions that can be performed using the application. The essential idea is to have the mediating agent automate the activity of manually using and application to perform a task. In the general case, a mediating agent constructs two different types of plans. One plan corresponds to the set of operations required to achieve the task. The second plan corresponds to the actions performed using the applications.

Chapter 3

Knowledge and Human Problem Solving

Knowledge System may be roughly defined as

"Computer programs which use knowledge represented explicitly to solve problems[3]." Based on this definition, one can argue that the two most important focal points of Knowledge System are knowledge and the process of solving problems. For this reason, before describing Knowledge System characteristics, it is important to understand the meaning of knowledge as well as its involvement in the problem solving process and to recognize the existence of some important activities underlying this process in order to comprehend the working method and the architecture of knowledge system.

3.1 Knowledge

"Knowledge is the sum of perceptions of an individual about aspects of some named universe of discourse at a particular time [4]". As such, we regard knowledge as being everything that an individual (e.g., a person) knows about a specific universe at a given time. The universe of discourse (also denoted application domain or simply domain) may be any part of the actual universe depicted in fiction, a universe existing in an individual's beliefs, etc. Thus, a "particular knowledge" must always be associated with these three different things (an individual, a domain, and a time) in order to be distinguishable.

This is quite an obvious observation. Two different persons generally have different knowledge of the same domain. Distinct domains generate clearly distinct knowledge. And the domain knowledge of an individual usually changes with the time since people commonly observe new things, make conclusions, etc. so that they change their perceptions of the aspects of this domain.

The above definition of knowledge ignores some interesting philosophical considerations as, for example, what we really mean by saying that one " knows" something. Is " knowing" associated with a kind of individual's beliefs? Does it translate only conscious things. However, these and other philosophical considerations are beyond the scope of this work. The definition of knowledge chosen here is in some sense not a philosophical one but a working definition, which is in reasonable agreement with the meaning ascribed to it in everyday usage. And it can serve as a basis for the design of computer systems working with this understanding of the notion of knowledge. Knowledge is of primary importance in expert system. According to Wirth's classic expression

Algorithms + Data Structures = Programs
For expert system is
Knowledge +Inference=Expert Systems.

If we describe in form of hierarchy, then knowledge is a part of this hierarchy, Figure 3.3. The bottom part is the noise, consisting of items that are of no interest and which obscure data. The next higher level is data, which are items of potential interest. Processed data are information, which is of interest. Next is knowledge, which represents very specialized information. For example knowledge in rule based systems are defined as the rules that are activated by facts to produce new facts or conclusions. This process of inferring is the second essential part of an expert system. Above knowledge is Meta knowledge where one meaning of the prefix "meta" means above. Meta knowledge is knowledge about knowledge and expertise. An expert system may be designed with knowledge about several different domains. Meta knowledge would specify which knowledge base was applicable.

3.2 An Overview of the Cognitive Cycle

Knowledge is for us, people, important because almost everything that we do is in some way based on the knowledge, which we have stored in our brains. In other words, people apply knowledge in order to achieve their goals. Hence, the process of problem solving is, for this reason, also a routine of applying knowledge, in this case, with the purpose of finding out problem solutions.

The ability of applying knowledge consciously is surely one of the most obvious skills that set people apart from the other creatures, which inhabit this planet. However, people can only apply the knowledge that they have, i.e., people are not able to execute tasks with which they are not familiar.

This observation suggests that application might be the most important but is not the only human activity involving knowledge, which is significant in this context. In truth, application is just one of four activities which together build what we are going to call a cognitive cycle a sequence of human activities through which knowledge passes when people apply it to solve problems. Because of this, in trying to understand this process, it is also necessary to analyse the phases of this cycle.

Furthermore, the above observations suggest that several kinds of knowledge exist and are used differently depending of the goal to be achieved. For example, the knowledge applied to solve mathematical problems is completely different

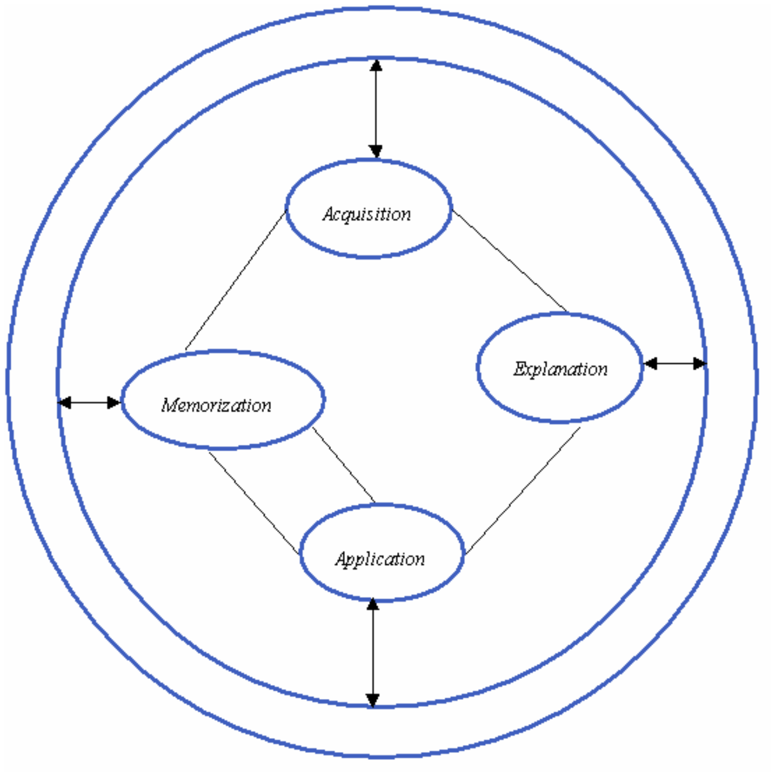

Figure 3.1: Knowledge-Cognitive Cycle

from that used to acquire the mathematical knowledge and even more to that utilized to bake a cake. Since we are focusing on knowledge system, only the kind involved in the human problem solving process is of interest for this work. For this reason, we disregard the several existing kinds of knowledge concentrating on those involved with problem solving.

As such, the first important phase in this context is the process of acquiring knowledge, i.e., the set of knowledge used to solve problems. Knowledge must be represented and organized in some form in our brains so that we are able to find and apply it when needed.

Once memorized, it is then ready to be applied in the next phase: the human problem solving processes itself. People basically solve problems by thinking. This means the active transformation of memorized knowledge to create new one that is, in turn, used to achieve a goal.

Once the goal has been achieved, people usually automatically assimilate the new knowledge that they have inferred during the solving process. However, assimilation is not enough to keep this information available over many years. It is also important to transmit it to other people.

The transmission of new knowledge generally involves explaining to people how one came to this acquaintance. Explanation is therefore the fourth and last phase in this context. As people are having things explained, knowledge is once again being acquired so that the whole cycle is then completed, Figure 3.1.

3.3 The Human Problem Solving Process

In order to understand the human problem solving process, we need to distinguish between two different types of operations that are applied during a thinking process. The first one can be translated as the human capability of reasoning, i.e., drawing inferences from current knowledge. In other words, making conclusions (i.e., generating new knowledge) by interpreting the knowledge stored in our brains.

Clearly, making conclusions is not sufficient to solve problems in an adequate way. It is above all necessary to make the right conclusions at the right time. This is achieved by using the second capability that we have, of guiding our reasoning processed. In some way, people also have the capability of determining a sequence of reasoning operations which, when applied over a domain, derive the expected solutions.

Surely, this second capability is the most important one for solving problems. By purely reasoning making, people would also come to the expected result, however, in a very inefficient way since many unnecessary conclusions would be also made. In order to solve problems in a fast and clear manner, it is therefore necessary to guide our reasoning process so that only he relevant conclusions are inferred Figure 3.2.

The task of solving problems in particular domains is actually the goal of any computer program. Most existing conventional programs (i.e., those out of Al fields) perform this task according to a form of decision-making logic embedded into their code, which cannot really accommodate significant amounts of knowledge. Generally, these programs manipulate two types of knowledge organized respectively in two different forms: data and algorithms. The latter determines the ways in which specific kinds of problems are worked out, and data defines values for parameters that characterize the particular problem at hand.

Firstly, people organize their knowledge in a separate entity (our brain) and access it to reason out specific problems. And secondly, the human problem solving process, organized on two or three levels, depending on the existence and use of Meta knowledge. In a philosophical sense, wisdom is the peak of all knowledge. Wisdom is the Meta knowledge of determining the best goal of life and how to obtain them.

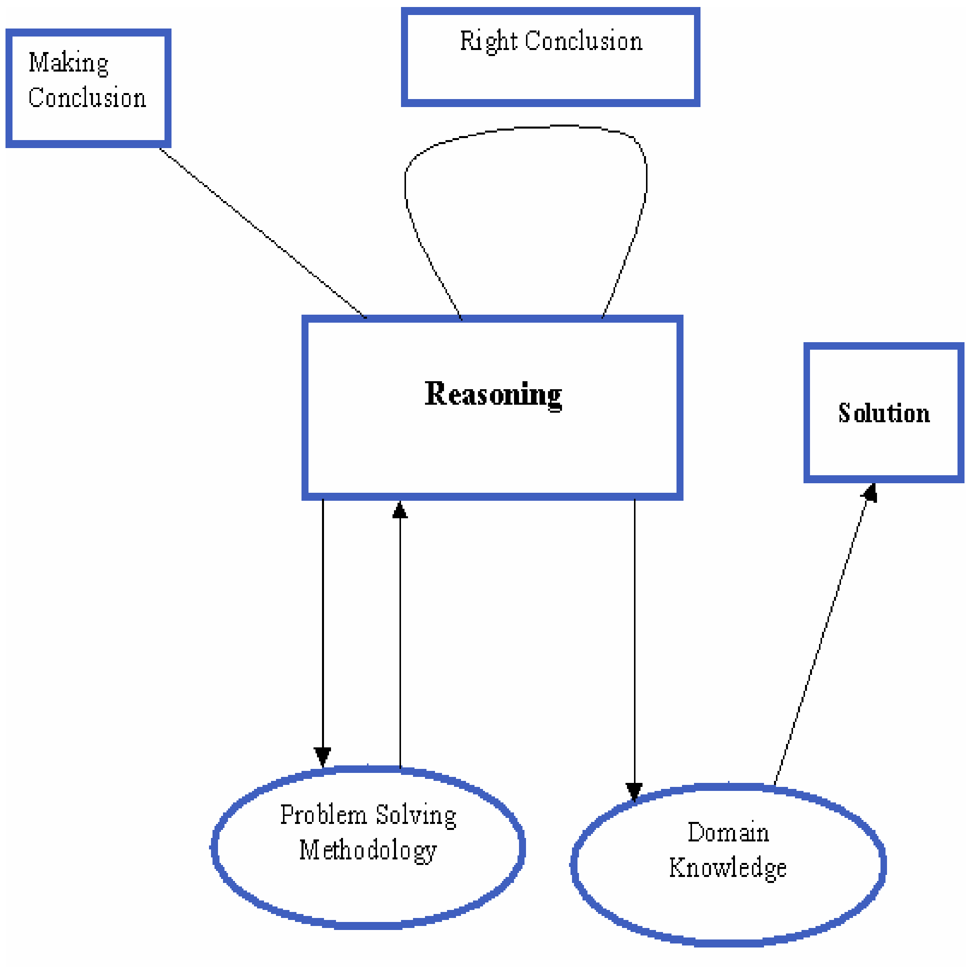

Figure 3.2: Human Capability of Reasoning

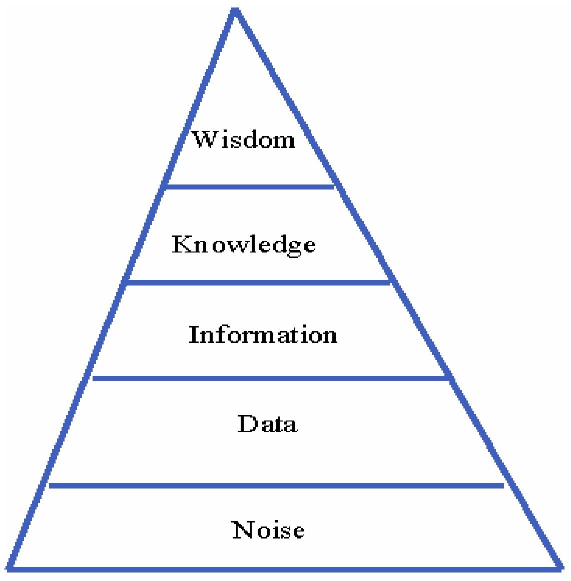

Figure 3.3: Hierarchy of Knowledge

Chapter 4

Expert Systems

Expert system is a branch of AI that makes extensive use of specialized knowledge to solve problems at the level of a human expert. An expert is a person who has expertise in a certain area. That is, the expert has knowledge or special skills that are not known or available to most people.

So we can say that

Expert Systems "Are computer programs which attempt to reach a level of performance by means of solving problems comparable to that of a human expert in some specialized application domain"[5].

The knowledge in expert systems may be either expertise, or knowledge, which is generally available from books, magazines, and knowledgeable persons. The term expert system, knowledge base system or knowledge based expert system are often used synonymously. Most people use expert system simply because it's shorter, even though there may be no expertise in their expert system, only general knowledge.

4.1 Components of an expert system

An expert system consists of the following components, Figure 4.1.

User Interface is the mechanism by which the user and the expert system communicate. Explanation facility explains the reasoning of the system to a user. Working memory is a global database of facts used by the rules.

Inference engine makes inferences by deciding which rules are satisfied by facts or objects, prioritizes the satisfied rules, and executes the rule with the highest priority. Agenda is a prioritized list of rules created by the inference engine, whose patterns are satisfied by facts or objects in working memory.

Knowledge acquisition facility is an automatic way for the user to enter knowledge in the system rather than by having the knowledge engineer explicitly code the knowledge. The knowledge acquisition facility is an optional feature on many systems. In some expert systems tools the tool can learn by rule induction through examples and automatically generate rules. Depending on the

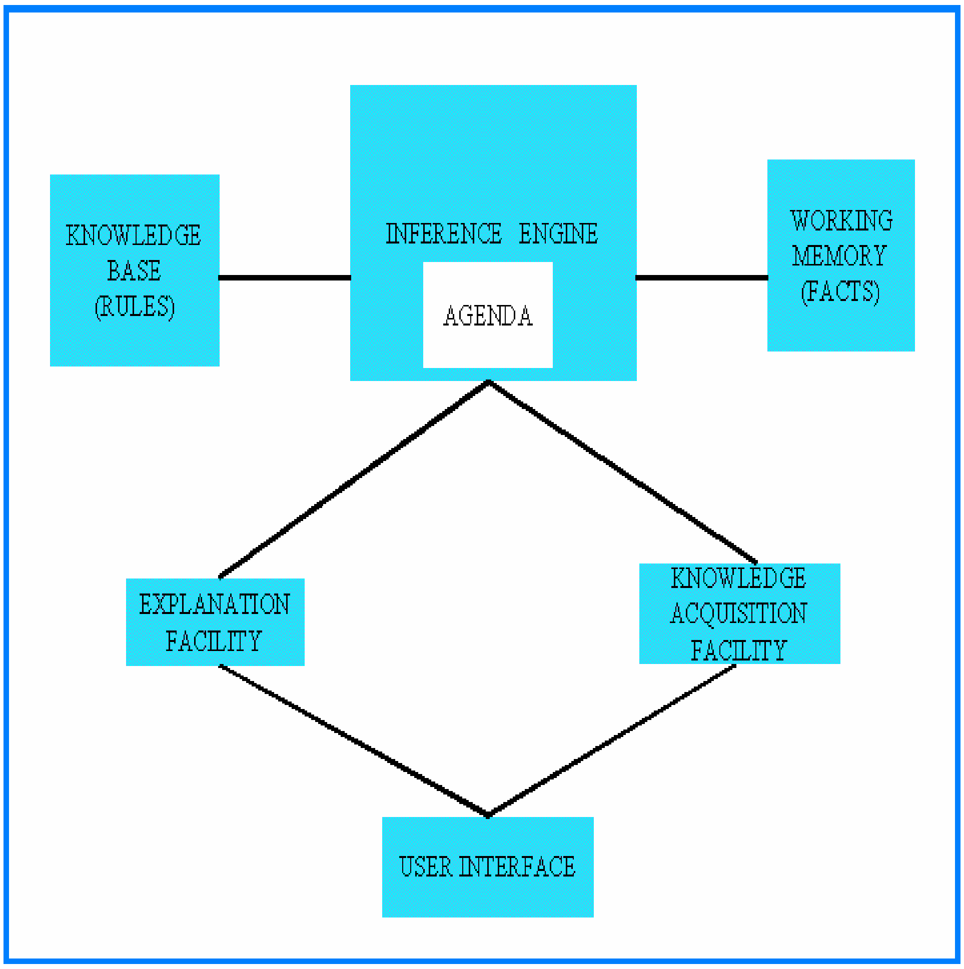

Figure 4.1: Components of Expert System

implementation of the system, the user interface may be a simple text oriented display or a sophisticated high resolution, bit mapped graphical display. The knowledge base is also called the production memory in a rule based expert system . As a very simple example, consider the problem of deciding to cross a street . the production s for the two rules are as follows, where the arrows mean that the system will perform the actions on the right of the arrow if the conditions on the left are true.

The light is red –> stop
The light is green–> go
The productions rules can be expressed in an equivalent pseudo code IF THEN format as:

```
Rule: Red_ light    IF
The light is red
THEN
Stop

Rule: Green_ light   IF
The light is green
THEN
Go
```

Each rule is identified by a name. Following the name is the IF part of the rule . The section of the rule between the IF and THEN part of the rule is called by various names such as antecedent, conditional part or left hand side (LHS): The individual condition

The light is green Is called a conditional element or a pattern. The logical combination of conditional element is called antecedent

In a rule-based system, the inference engine determines which rule antecedents, if any are satisfied by the facts . two general methods of inferencing are commonly used forward

Chaining and backward chaining as the problem solving strategies of expert systems. Other methods used for more specific needs may include means-end analysis, problem reduction, back tracking, plane generate test, hierarchical planning and the least commitment principle, and constraint handling. [6] Forward chaining is reasoning from facts to the conclusions resulting from those facts. For example if we see that it is raining before leaving home (the fact), then we should take an umbrella.

Backward chaining involves reasoning in reverse from a hypothesis, a potential conclusion to be proved, to the facts which support the hypothesis. For example, if we have not looked outside and someone enters with wet shoes and an umbrella your hypothesis is that it is raining. In order to support this hypothesis, we could ask the person if it was raining . if the response is yes, then the hypothesis is proven true and becomes a fact. Hypothesis can be viewed as a fact whose truth is in doubt and needs to be established. Depending on design, an inference engine will do either forward or backward chaining. The working memory may contain facts regarding the current status of the traffic light such as the light is green or the light is red. Either or both of these facts may be in working memory at the same time . if the traffic light is working normally, only one fact will be in memory. However it is possible that both facts may be in working memory if there is a malfunction in the light. Notice the difference between the knowledge base and the working memory. Facts do not interact with one another. The fact the light is green has no effect on the fact the light is red. Instead our knowledge of traffic light says that if both facts are simultaneously present then there is a malfunction in the light. If there is a fact the light is green in working memory, the inference engine will notice that this fact satisfies the conditional part of the green light rule and put this rule on the agenda. If a rule has multiple patterns, then all of it patterns must be simultaneously satisfied for the rule to be placed on the agenda. Some patterns may even be satisfied by specifying the absence of certain facts in working memory. A rule whose patterns are all satisfied is said to be activated or instantiated. Multiple activated rules may be on the agenda at the same time . in this case the inference engine must select one Rule for firing . The term firing comes from neuron physiology, the study of the nervous system. An individual nerve cell or neuron emits an electrical signal when stimulated . no amount of further stimulation can cause the neuron to fire again for a short time period. This phenomenon is called refraction. Rule based expert systems are built using refraction in order to prevent trivial loops. That is if the green light rule kept firing on the same fact over and over again, the expert system would never accomplish any useful work. Various methods have been invented to provide refraction. Each fact is given a unique identifier called a time tag when it is entered in working memory. After a rule has fired on a fact, the inference engine will not fire on that fact again because its time stamp has been used.

4.1.1 Knowledge Base

From technical perspective, no other factor is more important when selecting a shell than its knowledge base coding facility. This facility defines how you can represent the knowledge (e.g., rules, frames, decision trees). It is also important, however, to consider other knowledge base utilities that may be available, such as inexact reasoning and procedural processing capabilities.

Knowledge Representation: Shells for building knowledge-based systems can be classified according to the way they represent knowledge. The most popular categories are: rule-based Frame-based, case examples for induction or CBR, and fuzzy logic. Some shells offer multiple ways of representing knowledge.

Inexact Reasoning [7]: One of the trademarks of expert system technology is the ability to solve problems involving uncertain or unknown information, and inexact knowledge, This requires that the expert system be equipped with some inexact reasoning mechanism, such as certainty factors, the Shafer-Dempster method, and in rare occasions a Bayesian approach.

Procedural processing capability: In some applications there is a need to write procedural code. Functions might be need, or in a frame-based system, methods required to support message passing. Most of the shells, with the exception of frame-based ones, provide limited procedural processing capability. Frame based shells, the larger ones usually provide a Rich environment for creating procedural code to support the knowledge processing activity.

4.2 Technical characteristics of expert systems

4.2.1 Developer Interface

The various shells offer different levels of capabilities for the expert system designer to develop and refine the system. The smaller tools usually provide limited development features, while the more sophisticated ones, though more difficult to learn, provide a wider choice of knowledge representation methods, inference techniques, and user interface design alternatives. Various levels of debugging methods are also provided. Come shells also provide extensive on-line help that can greatly assist in system development.

Knowledge Base Creation. There are two broad ways of creating a knowledge base. Some shells require you to type the entire knowledge base, much in the same way as using a word processor. One advantage with using this approach is that you can print and review the knowledge. Other shells provide an incremental approach, where individual pieces of knowledge are created and added to the knowledge base. To create a rule, for example, the designer might select from a menu a create-rule item, and be presented with a rule-entry form to be filled out. The advantage of this approach is that the shell aids with system development. However, it can be difficult to determine the new rule's relationship with existing knowledge, unless the shell provides a good rule-browsing utility.

4.2.2 Debugging Utilities

Debugging utilities allow the designer to check and debug the system. One of the more common and valuable utilities is rule tracing. Following a session with the system, this utility provides a trace through all of the rules processed, including their processing order and information provided by the user.

On-line Help: Having a good manual that provides the instruction needed to build the expert system is obviously a valuable resource for the designer. Some shell vendors have gone a step Further and provide system development help on-line. While developing the system, the designer can access this utility to obtain aid in the effort. This aid might simply be in the form Of text that provides help on the requested item. In more advanced on-line help utilities, you can paste a template of a knowledge element (e.g., a function) directly into the system, and then edit it as needed.

4.2.3 User Interface

Display Type: The early expert systems all relied on a text-based display. Questions were posed to the user in text and answers were provided via keyboard typing or by selecting from a menu. Today, most shells provide a graphical-user-interface (GUI) that permits the user to interact through a point-and-click method. Since most computer users today are more accustomed to a GUI, and computer novices find the interaction more intuitive, a shell offering a GUI is usually preferable.

Information Entry: There is a wide range of methods by which the user can enter information into the system. Ti may be as simple as typing an answer to some question, or as extensive as filling out an entire form. A question might be posed requiring a single answer, or one that permits the user to select multiple answers from a list. Graphical input components typically available are radio buttons, check boxes, forms, scroll bars, and pushbuttons. Some shells provide limited default ways for entering information, while others permit the designer to choose from a variety of methods while creating the display from scratch.

Information Display: The expert system must be able to display its findings to the user. This information might be the final conclusion, or intermediate findings discovered during the session. Typical ways of displaying information are message boxes, value boxes, radio buttons, check boxes, forms, and graphics.

Interface Control: The user must always feel in control of the session. This includes an easy way of starting and ending a session, and readily accessible ways of activating needed control during the session. Some shells offer default methods for managing these tasks, while others require the designer to develop the techniques. For a graphical interface, this control is typically done with pushbuttons or a menu-select scheme. A user may also request other special features, such as save-case and what-if. The save-case feature permits the user to save the context of the present session, either for reporting requirements or for later what-if system testing. The what-if feature allows the user to change an answer to some question from a completed session; in order to see what impact the new answer might have on the final results.

Chapter 5

Types Of Tools

5.1 Rule-Based Tools

Rule-based tools use if then rules to represent knowledge. These rules are processed through a backward or forward chaining process, or a combination of the two (called bi-directional inference). Some tools permit the coding of inexact rules and inexact inferring, possibly using confidence factors and processing methods found in the certainty theory.

5.2 Frame-Based Tools

Frame-based tools represent knowledge in frames. Each frame captures descriptive and behavioral information on some object, in slots and methods, respective. Knowledge processing may be accomplished through an interaction of the frames using a technique such as message passing, or through an interaction of frames with a set of rules using a shell that is often referred to as a hybrid tool.

5.3 Fuzzy Logic Tools

Fuzzy logic expert system tools represent knowledge in fuzzy rules and fuzzy sets. Fuzzy rules, like ones found in the rule-based systems, are if-then structures. However, they include statements that contain fuzzy variables with corresponding fuzzy values that are represented mathematically in a fuzzy set. It is the term ”mathematically” that separates fuzzy logic tools from the symbolic rule-based ones. Processing of these rules relies upon a mathematical approach based in the principles of fuzzy logic.

5.4 Induction Tools

Induction tools generate rules from examples. They are products of AI research in machine learning. A developer enters a large set of examples from the domain under consideration, where each example contains values for a set of domain features, and a single result characterizing the example. The induction tool then uses an algorithm, such as ID3, to generate a rule or a decision tree. In operation, a user enters information about a current problem and the system then determines the probable result.

5.5 Cased-Based Reasoning Tools

Cased-based reasoning tools are somewhat recent additions to the toolbox of the knowledge engineer. In practice, they serve a role similar to induction tools in that they use past experiences to solve current problems. Given an input specification of a problem, the system will search its case memory for an existing case that matches the input specification. It may Find an exact match and immediately go to a solution. Even if an exact match can not be found, the system applies a matching algorithm in order to find a case that is most similar to the input specification.

5.6 Domain-Specific Tools

The early shells were general-purpose tools and were offered to address a broad range of problem solving activities. In many instances, however, it was found that although the tool was fine for one activity, it was inadequate for another one. To address this situation, the AI community, including both software vendors and researchers, turned their attention to developing domain-specific tools. Domain specific tool is designed to be used to develop an expert system for a particular problem-solving activity. It provides special features to the developer that is tailored for producing an expert system for the activity. For example G2 is a Graphical object-oriented environment for building intelligent process management solutions. Its natural language editor allows users to enter rules, models and procedures that describe real time operations.

Chapter 6

Expert System Concepts

The characteristics of expertise include fast and accurate performance, usually in narrow domain of knowledge. In addition, an expert can explain and justify the recommendation or result, as well as explain the reasoning process leading to the result. Further, experts quickly learn from experience, resulting in improved performance. Expertise implies the ability to solve unique and unusual cases-often reasoning from basic principles or a model, or from a body of experience structured into cases or rules. Finally, experts often must reason under uncertainty and apply common sense and general world knowledge to the situation. In order to identify potential ES applications, it is useful to distinguish between types of ES. Very often one type of ES may be a much better choice depending on several factors. The situation particulars, degree of structure in the domain knowledge, the knowledge representation schema, and user characteristics. There are three general types of ES.

1. Case based
2. Rule based
3. Model based

6.1 Case-Based Reasoning (CBR)

6.1.1 Introduction

No matter what type of problem-solving task it is, people usually have more confidence in you when you say that you have done that particular task many times before, i.e., you have the experience. Why? Because by having done that task before, it means that you are able to do another similar task. It is also a well-known fact that human experts solve problems by relying on their past experiences in solving similar problems. This is especially true in areas such as law and medicine. It was from this idea of reusing past experiences to solve new or current problems that an Artificial Intelligence approach called Case-Based

Reasoning was born. CBR is sometimes classified under Machine learning, and supports knowledge acquisition and problem solving.

CBR Concepts - Figure 6.1

1. A new problem or case is analyzed and represented in a form such that the CBR system can retrieve relevant past cases. The goal is to retrieve useful cases, that is, those that have the potential to provide a solution to the new problem at hand.

2. Once relevant cases are retrieved, they are ranked and the best subset, or most promising cases, is returned to the user for browsing.

3. Very often, an old case does not fit the new one exactly, hence, it is necessary to modify or make changes to an old solution to fit the new problem situation. The process of making these changes may range from a minor substitution of values, to structural changes. What adaptation, and how it is to be done, depends on domain knowledge.

4. The initial solution to the new cases is then proposed to the user.

5. The proposed solution is tested or evaluated, and improvements made on it. Feedback is obtained and analyzed. If it does not perform as expected, and explanation of the anomalous results is given. Follow-up procedures include explaining failures and attempts to repair them are stored, so that future failures can be predicted and avoided.

6. The new case is updated into the case library for future use. By adding new situations into the case library, the system is actually carrying out an incremental learning process. This is especially useful for dynamic domains that need to keep up with the times.

6.1.2 Representing a Case

A case could resemble a database entry, with a list of characteristics or features describing a particular situation. A case is usually complete by itself, or may be connected to a set of sub cases. How are cases defined and represented'? There are many ways of representing a case. Including predicate representations, frames, or even entire resembling database record. In general, there are three features that need to be captured: the problem-situation description, the solution, and the outcome. The problem-situation is essentially a description of the characteristics of the problem, the context, or situation in which it occurs. The representation must match the reasoning goals of the system.

6.1.3 Indexing Cases

In order for the case-based reasoning system to retrieve similar cases, labels must be assigned to a case to identify it as an appropriate case to be retrieved,

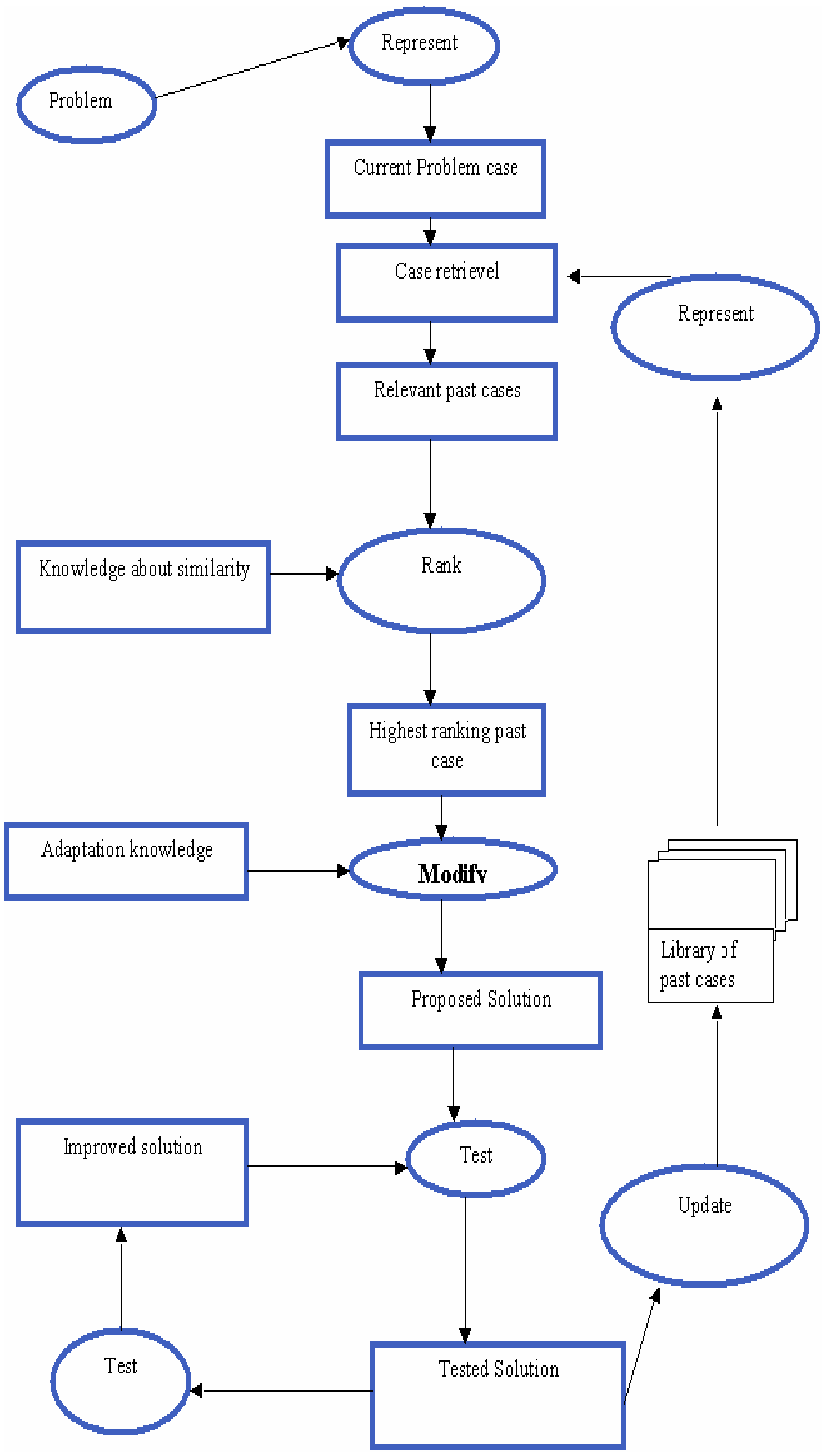

Figure 6.1: CBR Architecture

given a particular situation. Hence, indexes signify which cases are relevant and are potentially able to provide a solution to a problem.

6.1.4 Similarity Matching

Given the particulars of a new case, the case-based reasoning system looks at all the existing cases and retrieves those very similar ones. Matching each individual feature of the new case with all the existing cases identifies most similar ones. One such matching process is known as nearest-neighbor matching.

6.1.5 CBR and Rule-Based Reasoning (RBR)

Rules of the form IF-THEN-ELSE cover a single aspect of knowledge, whereas a case cover a particular problem-solving situation. It is sometimes difficult for a domain expert to come up with rules to problem solving, whereas cases are actually examples of a particular situations. Hence in general, acquiring case is easier than rules.

RBR has some inherent weaknesses:

- Past experiences are not learned.
- New, novel problems that deviate from the norm cannot be solved.
- Maintaining a large rule based is difficult and time consuming

Wrong information in rules must be individually edited, and every time a new rule is added, there is a risk of redundancy or contradiction. Deleting a rule may interrupt the reasoning process, whereas removing an individual case does not have the same extent of consequences. Hence, RBR systems are more difficult to maintain, and more difficult to update to keep up with rapidly changing domains.

6.1.6 CBR and Model-Based Reasoning

Model-Based reasoning emphasizes the usage of large chunks of general knowledge, based on models that cover the normative situations. Model-Based reasoning is normally used for well-understood domains that could be accurately represented in a formal language, and which tends to be static. Hence, building a Model is time consuming, and so is maintaining it. In combining Model-Based reasoning with CBR, the model-based reasoning handles the well-understood components.

Case based reasoning should be used when

- The solution alternatives can be explicitly enumerated
- Numerous examples of worked cases exist that cover domain knowledge

- No domain model or theory exists
- Expert represent domain in terms of cases
- Human expert is not available
- Situational information are conflicting, or missing.
- Knowledge is volatile and dynamic
- Domain knowledge and expertise already captured by past cases
- Workforce experience and performance are low
- Need a fast way to acquire domain knowledge
- Want to illustrate an outcome or explanation with an example

6.2 Model-Based Reasoning (MBR)

MBR provides a representational and conceptual framework for knowledge that defines both knowledge structures and inferring methods. MBR defines and structure relevant domain objects/concepts, their attributes, and their behavior in order to organize work in complex domains and perform simulations. MBR also defines the relationship between the objects in case of class hierarchies, composition and causation. The most basic knowledge structure, governing all types of knowledge, is the "Object Attribute Value" triple.

MBR encompasses, represents, and organizes all type of knowledge, including CBR and RBS, as well as databases, text, images, and other media. Types of MBR are object-based, frame-based and domain specific. MBR models can also be categorized as quantitative or qualitative. MBR requires a well-understood and well-structured domain theory. In simulation, MBR components are often so tightly linked together that MBR has a limited value without a completed model. MBR is also very useful for organizing and structuring complex domains and processes.

Model-Based reasoning should be used when:

- A consensus framework of concepts and domain theory exists
- Business processes, methods, and events needed to be represented and modelled
- Want to represent and organize large scale, complex system
- Want to simulate performance and side effects from future work system design
- Want to control, monitor, and measure informational workflows

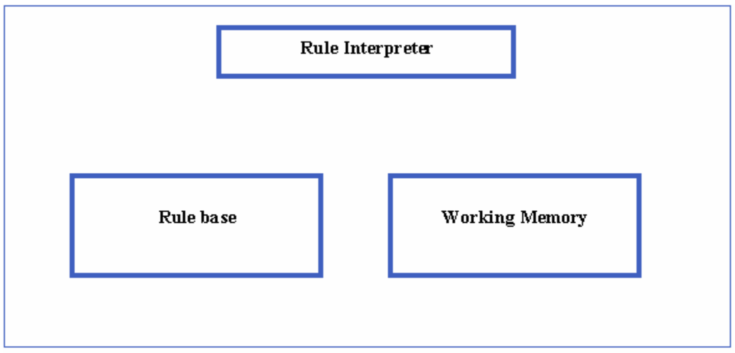

Figure 6.2: Rule Based System

- Want to represent, organize and integrate elements of knowledge repositories and related performance support systems
- System navigation and presentation is important
- Environment and data are relatively dynamic
- IT infrastructure uses client/server architecture
- Results from knowledge elicitation and acquisition need to be organized.

6.3 Rule Based System

At kiwi logic the rule base reasoning system exists. The system was evaluated and used as a platform for my application. Consider the following example of rule-based system (Figure 6.2).

If the antenna to transmitter coaxial cable has been disconnected from the antenna port, then examine the coaxial cable connecter for any signs of breakage, discoloration, or fraying . if such such signals are detected, then remove and replace the connecter via the procedure. Otherwise connect the RF generator to the antenna port and initiate the test sequence. If the RF generator is connected to the antenna port, then turn it on and set the frequency to 220.5 Megacycles. If the generator has been on for at least 2 minutes and the frequency is stable then initiate the test.

Examining these few steps carefully, we may note that the contents of each step represent extracts from the knowledge base of an electrical engineer with specific expertise in the testing of RF systems. Note carefully that each step consists of one or more IF -THEN type Of statements. That is if a particular situation exists, then take certain action. Such a statements form the basis of rule-based systems. Which in turn used as foundation of existing expert systems.

The transformation of one pattern to another in a rule-based language is understood to represent an IF-THEN implication. IF There is rain And we are in Spain THEN We must be on the plain

Or Operations to be performed: IF Block A is on block B And A has nothing on top of it THEN Take block A off block B

6.3.1 Forward and Backward chaining

The previous section covered only one of the possible uses of rules. We made the assumption that the current pattern is matched to the left hand side of the rule and is transformed into the new pattern on the right hand side. But what would happen if our initial data were the sentence

The chilly bird flew south And we wanted to find out whether the sentence was grammatically correct?

To answer this question, we can use the same set of rules that we used to generate grammatical sentences. One way to do this to start generating all possible grammatical sentences from the symbol S until we have generated the sentence in the question. Or we can imagine the arrow in the rules pointing in the opposite direction and thus make the reverse implications to reduce the sentence first to grammatical structures such as Noun and adjective and finally to the symbols shows the process of proving the sentence to be grammatically correct by using the rules in the opposite directions. The ways we have used the grammar rules from left to right to generate a sentence and from right to left to prove the correctness of sentence are simple examples of forward and backward chaining.

Forward-chaining problems usually have a large number of data in their initial state from which a solution must be constructed. There is neither single nor optimal goal state-there is only a set of constraints to which the goal must conform. At each step in a forward chaining process the question is "what is the next step to take to move closer to the goal".

Backward-chaining problems start with a hypothesis or goal and derive the substantiating evidence for that conclusion . at each step in backward chaining process the question is "what must be true in order for the current state to be true. Backward chaining problems are those for which a set of solution is known and for which the current case must be classified as one of The known solutions. Diagnostic problems are the most common examples of backward chaining problems.

One way to tell whether a problem should be solved by forward or backward chaining is to look at its initial and goal states . if the initial states contains many facts that must be synthesized into a solution, then forward chaining is appropriate. If there is a description of a current state that must be analyzed to find the facts in the database that support the state, then backward chaining is more suitable. Some problems are best solved using a combination of backward and forward chaining. OPS5 execute rules in a forward-chaining fashion, and can Also be programmed for backward chaining . An OPS5 program can create

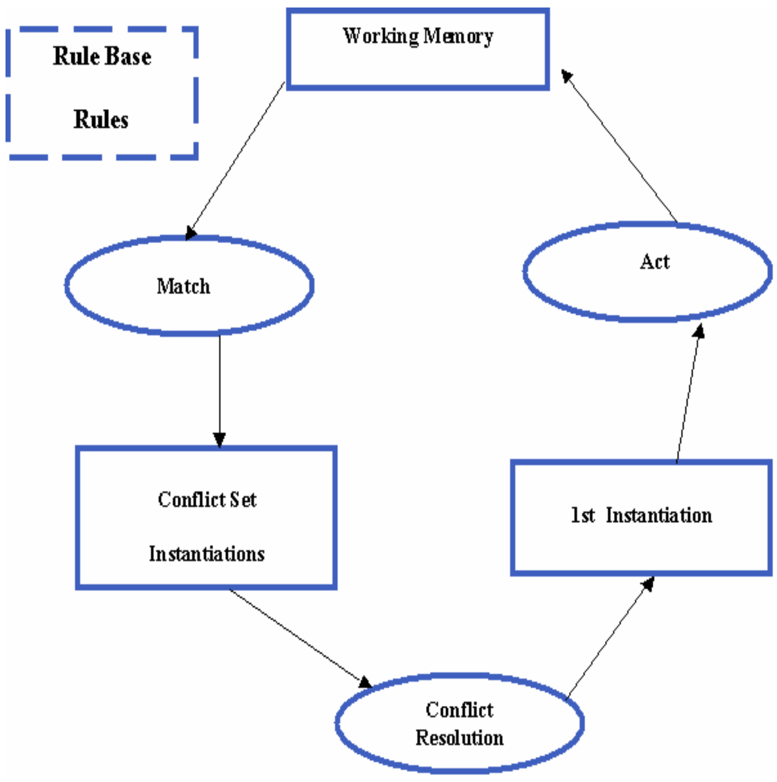

Figure 6.3: Recognize-Act Cycle

data that represent a goal (forward chaining) and can nothen determine whether that goal can be confirmed (backward chaining).

6.3.2 The recognize act cycle

Until now we referred rule interpreter as the black box that is responsible for driving the execution of a rule based program. now we take more precise an close look at rule interpreter. The act cycle has three steps match, select and act. Time tags are assigned to each working memory element by the rule interpreter to identify WMEs uniquely. Time tags are used in the recognize -act cycle. Especially during conflict resolution. A rule is considered successfully matched when all of its positive condition elements are matched by WMEs and there are no WMES in working memory that match the rules negated condition elements . the result of a successful match is an instantiation. The result of a match step is a conflict set that contains all the instantiations that are eligible for execution. During the conflict resolution step a single instantiation is chosen to execute. The decision criteria and the method used to choose a rule are called a conflict resolution strategy. There are many strategies a rule based system can employ in choosing an instantiation from the conflict set, such as the first rule to enter the conflict set or the rule with the highest priority number. During the act phase the actions on the RHS (right hand side) of the rule are executed in the order they are written, (Figure 6.3). Any changes to working memory are immediately reflected in the conflict set.

6.3.3 Writing a rule base

To write a rule base we can start by writing rules in English. From the English rules we can write experimental rules. Alternating between the two representations gives us a sense of the correct element class representation for the problem. The rule base contains the rules; attribute declarations and an optional start up statements. Then we should provide a set of commands with which we can interact with our program and with recognize act cycle.

Chapter 7

Bot

7.1 Introduction

A bot is a software tool for digging through data. You give a bot directions and it brings back answers. The word is short for robot of course, which is derived from the Czech word robota meaning work. The idea of robots as humanoid machines was first introduced in Karel Capek's 1921 play "R.U.R.," where the playwright conceived Rossum's Universal Robots. Sci-fi writer Isaac Asimov made them famous, beginning with his story I, Robot (1950) and continuing through a string of books known as the Robot Series On the Web, robots have taken on a new form of life. Since all Web servers are connected, robot-like software is the perfect way to perform the methodical searches needed to find information. For example, Web search engines send out robots that crawl from one server to another, compiling the enormous lists of URLs that are the heart of every search engine. Shopping bots compile enormous databases of products sold at online stores. The term bot has become interchangeable with agent, to indicate that the software can be sent out on a mission, usually to find information and report back. Strictly speaking, an agent is a bot that goes out on a mission. Some bots operate in place; for example, a bot in Microsoft Front Page automates work on a Web page. Bots have great potential in data mining, the process of finding patterns in enormous amounts of data. Because data mining often requires a series of searches, bots can save labour as they persist in a search, refining it as they go along. Intelligent bots can make decisions based on past experiences, which will become an important tool for data miners trying to perfect complex searches that delve into billions of data points.

Bots were not invented on the Internet, however. Robotic software is generally believed to have been created in the form of Eliza, one of the first public displays of artificial intelligence. Eliza is a computer programmer that can engage a human in conversation: Eliza asks the user a question, and uses the answer to formulate yet another question. Artificial intelligence is an advanced form of computer science that aims to develop software capable of processing

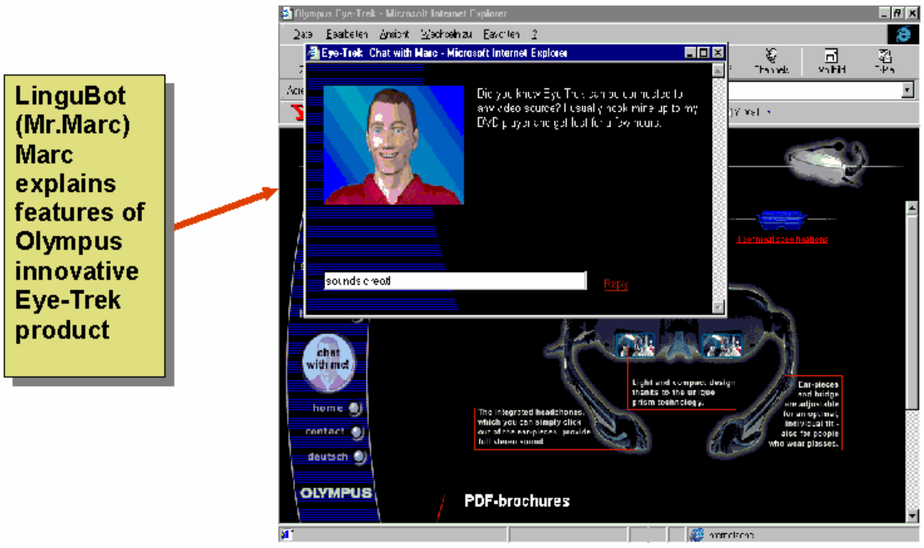
LinguBot (Mr.Marc) Marc explains features of Olympus innovative Eye-Trek product
Eye-Trek Chat with Marc - Microsoft Internet Explorer
Reply
chat with me!
home
contact
deutsch
OLYMPUS
PDF-brochures

Figure 7.1: A LinguBot

information on its own, without the need for human direction.

7.1.1 Technical applications of a Bot

The advantages of including a Bot on our Web site are numerous. A short list would include: assisting in Web site navigation, conducting e-commerce, offering quick and efficient responses to customers' Frequently Asked Questions, and providing entertaining and unique customer support.

7.2 Lingubot

A LinguBot, Figure 7.1, is a virtual personality that chats online with users. It recognizes and responds appropriately to natural language-that is, the way people talk. In addition to understanding natural language, a LinguBot uses graphics to reveal emotions appropriate to the conversation.

7.2.1 Structure of LinguBot

The system consists of the Lingubot Web Engine and the Lingubot Creator (Figure 7.2). The Lingubot Web Engine parses the user input and, by applying pattern-matching technology, looks up the appropriate answers in the knowledge base. The rules, which make up the knowledge base, are defined using the Lingubot Creator. All dialogues are recorded in a transcript database. The Lingubot Creator also provides tools for conveniently analyzing these transcripts, enabling the author to continuously improve the performance of the Lingubot.

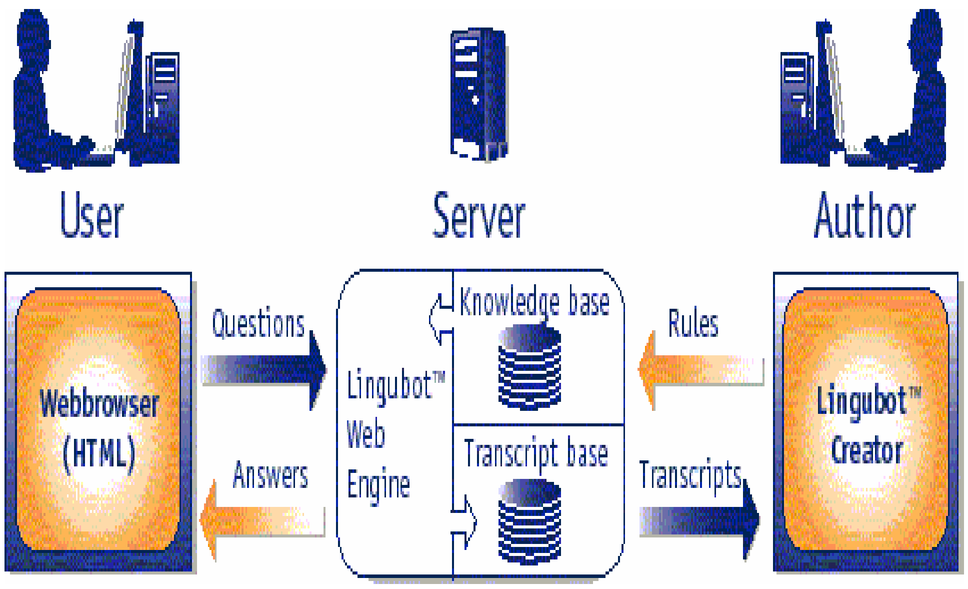

Figure 7.2: LinguBot Architecture

Knowledgebase have three essential elements: Recognitions, Answers, and Macros. Together they form, a file with . LBF as an extension. The LinguBot Engine breaks user input into sentences and the sentences into words. The software checks to see which parts of that input it recognizes, then gives the user an appropriate Answer. It is the author's task to anticipate and write these Recognitions and Answers.

A Knowledgebase also includes images, scripts, and a variety of Web deployment attributes. It is only after being published that a Knowledgebase becomes a Bot.

As in any pattern-matching system, the "intelligence" of a Lingubot lies in its rule base, which has to be specifically designed for a certain purpose. That means that the performance of the system is defined by how many rules it has and how well those rules are written

Basically, the author of a Lingubot has to think of all the different questions users might ask the system, turn those possible questions into recognition patterns and write the appropriate answers. This is a highly creative process, comparable more to writing a book than to writing a software program.

Recognition

Recognition is one of the three essential elements comprising a Knowledgebase. It is by means of Recognitions that a Bot "understands." Recognitions identify specific user input and select appropriate responses.

Upon receiving user input, the Web Engine checks it against all Recognitions to determine the Recognition best suited to process the input and provide an Answer. If the user input satisfies Recognition's condition, but that Recognition can not provide an Answer, the search continues. If the conditions of two or more Recognitions are met by the user input, the Recognition with the higher Rank is given precedence.

An appropriate Answer is selected from those in the Answer box. (A given Recognition may have up to 9999 Answers.) The Answer selection process takes Answer Rank, Additional Conditions, and any relevant scripts into consideration.

The central recognition: There will always be unexpected user inputs. To make it possible for our Lingubot to give an answer, there should be a recognition that is always true and has a very high ranking in accordance with its inexactness. If this recognition is fulfilled, we know that all of the other recognitions have not taken effect. At this point, we should give the dialog a new turn and address one of our active topics or of the Lingubot's special area. The number of answers on this may and should be very large.

The basic Recognitions The basic Recognition Module (recognitions.kef) contains hundreds of useful Recognitions. This Module relies upon the basic Macros, and therefore requires that macros. kef is imported into the same Knowledgebase. It does not include Answers.

Description The Description field is used to illustrate or summarize the Recognition's contents or context. Descriptions can be used to state in plain English what user input the Recognition is intended to anticipate. When you use the Wizard, it's a good idea to state the anticipated user input exactly the way in which it is most likely to be stated.

EXAMPLES

Good Descriptions	Bad Descriptions
How old are you?	Age question
Tell me about Widget Works.	They want to know about our co.
Please send me a catalogue.	Catalog Request

Comment. In this field you may explain or comment on the Recognition. Creator will process nothing in this field, the Comment field serves as a place for your own notes. You may leave this field blank if you prefer. It's possible to search the Comment field, so it's a good place to record information that identifies the Recognition in ways not covered by the Description.

EXAMPLES OF POSSIBLE COMMENT FIELD USAGE

Comment : User inquiries about new products

Comment : Follow up recognition to Answer #55

Comment : Only active if User_Age is under 13.

Comment : Identifies user age and stores in User_Age variable

In the Navigation Box
Navigation

You may navigate through Recognitions using the arrow buttons (< >) in the upper right corner of the Recognition window. The buttons (|<) and (>|) jump to the first and last Recognitions of the Knowledgebase, respectively. You may also enter the Recognition # of a known Recognition into the numeric field between the arrows and strike Enter to navigate directly to a known Recognition.

User can write his own Knowledge Bases In order to ease the introduction into working with the LinguBot Creator, the software includes prefabricated Macro, Recognition, and Answer Modules (data files with the extension . KEF). They can be imported into our own Knowledge Bases, then adapted and expanded according to our requirements.

Engine The Web Engine is the "brain" of the Bot. Its operation is summarized below.

One	The Engine divides the user input into sentences, then into words. Words go through automatic spelling correction and are identified within spelling tolerance parameters.
Two	The Engine searches all Recognitions and identifies all Recognitions whose Conditions are satisfied by the user input.
Three	The Engine compares the Recognition Ranks of all satisfied Recognitions. Beginning with the highest-ranked Recognition and proceeding to the lowest-ranked, it identifies the best Answer. In doing so, it takes Answer Rank, repeatability, Additional Conditions, and any relevant scripts into consideration.
Four	The appropriate Answer is issued.

Macro Macros are one of the three essential elements of each Knowledgebase, along with Recognitions and Answers. A Macro is a summary of a complex Condition. Macros are typically lists of synonyms, figures of speech, idioms, or thematically connected keywords. Macros can be used in any Condition field. The Wizard tries to generalize user input examples using Macros. Macro Variables can be defined to catch and store that part of the user input recognized by the Macro.

Macros can be accessed by choosing Macros from the View menu, or by double-clicking on any Macro Name in a Condition field.

A Macro is shorthand for Condition writing. Each Macro is itself a Condition that can be used like any other Condition. Macros make the life of the Bot author much easier. They help you efficiently anticipate the wide variety of user input.

The basic Macros The basic Macro Module (macros.kef) contains hundreds of useful Macros.

Macro description When building a new Macro, the Wizard will use the text in the Description field as the user input example from which to generalize.

Any given Recognition should recognize as many user input variations as possible. (A "how old are you" Recognition, for example, should recognize variants of "how old are you," "what's our age," "how many years have you been alive," and so on.) The Wizard helps build Recognitions and built in Macros from user input examples by searching for existing Macros that appear to match them, whether whole or in part. Macros are only included in this search process if they have been marked "Suitable for Generalization" in the Macro window. The Wizard will not consider macros that are not marked "Suitable for Generalization".

The Macro Descriptions in the English language Knowledgebase provided by Kiwi logic follow a very specific format, as illustrated in the following examples:

A	Syns for good, nice, pleasant... See also BAD (a), NICE (a)
n	List of New York Neighbourhoods... See also other geographical macros
V	Syns for eats, consume, devour... See also DRINK (v)
pv	Vars of He will

The first part of each Description is its part of speech.

a	Adjective or adverb (hot, cold, big, small)
art	Article (the, a)
conj	Conjunction (and, or, but)
excl	Exclamation (Wow!, Listen!, Shut up!)
int	Interrogative (Who is, What are, When will)
n	Noun (house, animal, computer)
p	Preposition (near, toward, from)
pn	Pronoun (you, he, they)
pv	Pronoun-Verb (he will, you may, we can)

v	Verb or action (to run, to eat, to laugh)

The second is its category.

Syns	for – Synonyms for..
Vars	of – Variants of..
List	of –, erg
List	of... for example...
List	of – incl Syns e.g. –
List	of... including Synonyms, for example...

Lastly, there are generally cross-references referring you to other related Macros within the Kiwi logic English language Knowledgebase. These are generally synonyms, antonyms, or related keywords.

Memory Lingubot Creator will issue each Answer only once, unless it is explicitly marked as repeatable (in the Answer window). Once issued, non-repeatable answers will not be issued again until the Bot's Memory lapses-in other words, until the Bot "forgets" that it has already used that answer. The Bot's "Memory" is set in the Misc tab of the Options window, in number of "Submits" (user inputs). The larger the number of Submits, the longer the Bot will converse before "forgetting" having issued a particular answer.

EXAMPLE

Recognition: how+old+are+you

Answer: I'm old enough to know better.

Repeatable: No

Memory: 60 Submits

For the following example the default is 60

If the user's fifth input is "How old are you," they'll receive the Answer: "I'm old enough to know better." If their fortieth or sixty-fifth input is again "How old are you," this recognition will be passed over because the Bot will "remember" having issued its only Answer to this Recognition less than 60 user inputs ago. (Recognitions without valid, active Answers are treated the same as Recognitions without any Answers at all: they are ignored.) If the sixty-sixth user input is "How old are you," however, the Answer will be issued again, because the Bot will have "forgotten" that it issued the Answer 61 user inputs ago.

Please note that this has no impact on transcripts, which will record the entirety of the dialog regardless of the Memory setting.

Answer window An Answer is LingurBot's response to the user input. Specifically, it's the response to the user input recognized by the Recognition with which it's associated. While a single Recognition may have thousands of Answers associated with it, only one Answer will be provided to the user each time the Recognition is triggered.

Answer # Each Answer is assigned a unique ID number, which is located in the upper right corner of the Answer window. As in the case with Recognitions, these numbers cannot be changed and once deleted will not be re-used. Understanding the assorted uses of Answer-IDs will save you time when authoring, so that we know the identifier of every Answerand their use can help the Bot converse more naturally.

Dialog History In the upper left corner of the Answer page, this box documents the conversation history up to this point, using the text from the Recognition and Additional Condition windows (as briefed below). Description fields to represent user input, and Answer text for the Bot's responses. The Dialog History is purely informative, and cannot be edited. It is modified automatically when the Description or Answer fields themselves are edited.

Answer Text This field contains the Answer that will be presented to the user. You may format texts with the aid of HTML tags. You may also include variables. Line feed is automatic when the Answer is issued through the Web server. Manually inserted line feeds (carriage returns) will be ignored.

Additional Condition Additional Conditions are "sub conditions" that can fine-tune the selection of an Answer within a given Recognition. The button text will be red if there is an Additional Condition assigned to that Answer.

Answer Rank It is possible to generate several different Answers to the same Recognition. The Answer Rank determines the order in which Answers will be given in successive uses of a given Recognition. This parameter can take on any whole-number value between 1 and 9999. The value 1 has the highest priority and the value 9999 has the lowest. When multiple Answers for the same Recognition are assigned the same Rank, the Answers are given to the user in a random order.

Repeatable Checking Repeatable on a particular Answer allows that Answer to be given again the next time the Bot recognizes the Condition. By default, Answers are not repeatable, and will only be given once. Recognitions that have only one Answer therefore become "inactive" once that answer has been given. Exception: If the number of user inputs entered in the Memory Option expires after the Answer was given, even non-repeatable Answers can be re-issued

7.3 Working of LinguBot

The Web Engine breaks the user's input into sentences and the sentences into words. It only checks the recognitions for each sentence in the order of its ranking. Sentences are generally ranked according to their priority. The higher-ranking sentences have high and low ranking sentences have low priority. Recognition is correct if the sentence fulfils the conditions that were formulated in the recognition window. After completing the process for all sentences, it issues the answer that is assigned the recognition with the lowest ranking. The number of recognitions and the answers assigned form the knowledge base of the Lingubot. This is prepared by the text composer. It is his/her task to write recognitions and answers that match.

We can Compose Texts for a Lingubot.

A Lingubot

- has a character and an area of knowledge
- has "active" topics that it initiates on its own
- Carries on dialogs in its very personal tone and in its own personal fashion
- Learns through dialogs

Character and area of knowledge:

First of all, we jointly establish the reactions of the Lingubot with regard to its tone and content. In this process, we have to carefully plan ahead of time in order to be able to write recognitions and answers to frequent or expected user inputs. What is necessary here is not only an exact recognition of sentences, but also less specific conditions such as those that are fulfilled when certain words appear.

Active topics:

The dialog with the user is made more vivid by the fact that the Lingubot actively takes part in developing the conversation and not only answers questions. The Lingubot states something on its own or asks the user questions in the hope of being able to react to the user inputs it prompts in this fashion. In order to be able to compose the recognitions for inputs such as these ahead of time, it is necessary to remember some exemplary dialogs.

Dialogs:

Test the Lingubot as early as possible and as frequently as possible with a lot of different human testers. If the group of testers is too small, there is the danger the testers can not think of anything anymore or only something very special. Then the Lingubot does not learn anything or it requires a lot of effort to learn to adapt to the dialog behaviour of a special group. Beyond this, guessing at user inputs as mentioned naturally involves suppositions so that we may have to invest a lot of work in writing recognitions that do not come to bear because the users react differently than expected. Tests are also a helpful corrective measure here.

Function of knowledge base

The recognitions and the answers assigned to them form the basis for the knowledge base of the Lingubots. Basically, we only require recognitions and answers for

- Beginning the dialog
- Saying hello and goodbye
- The area of knowledge
- Knowledge on the person and character of the Lingubot
- Small-talk
- The central recognition (refer section 7.2).

The more precisely we recognise a user input, the better we can react to it. However, the number of inputs that are not recognized based upon a deviating formulation if they have almost the same content increases with the preciseness of the recognition. Therefore, apart from recognizing whole sentences, we should also write recognitions for the important areas of knowledge that react to a special word. The answers assigned have to adapt themselves to the preciseness of the recognition. The Lingubot Creator has a "base of basic knowledge" that contains a series of "ready-to-use" recognitions. As an option, it also contains patterns for important recognitions in the areas of product and company information that only have adapted to the focal point of our Lingubot.

Beginning the dialog: We require several sentences for the beginning of the dialog so that users can be greeted in a different fashion in further dialogs. Our Lingubot will say something at the beginning of a dialog without having to react to the user input. We should take advantage of this by getting the user in the mood of our topic with our greeting and/or setting the basic direction and tone for the dialog. We will soon note that the users of the Lingubot will ensure that the mood and topic change abruptly.

Saying hello and goodbye: Is a part of most dialogs.

Area of knowledge: The Lingubot has a special area and active topics. Therefore, we require recognitions for the questions and remarks that the users enter (either spontaneously or prompted).

Knowledge on the person and character of the Lingubot: Our Lingubot will be asked a number of questions on its person, its task and technical background.

Small-talk: Users not only want to talk about the Lingubot's special area, but they also just want to talk to it (especially if it has an interesting personality). Of course, there are almost an uncounted number of potential topics here. Therefore, we should ensure that there is the right mixture of exact and inexact recognitions and co-ordinate our answers with this. In this fashion, our Lingubot can make more exact and detailed statements on some topics while only treating others rather shortly.

Chapter 8

Design and implementation of E. mail agent

This project describes further efforts for enhancement of the e-mail. The mail agent is autonomous, customized, personalized and long lived.

The agent fetches itself regularly after time from POP3 mailbox using post office protocol. After processing "reading, scanning the receive mail, the agent created in each case new e-mail reply which consists of asking questions from client and send to a human user, who if necessary complete and modify it and send back to the client, Figure 11. The agent is universally applicable and it is knowledge-based application. The above goal was achieved using the Rule based knowledge base systems and the application was implemented using the C++ Borland builder.

Natural language interpretation is essential for taking advantage of the full commercial potential of the Internet.

8.1 The "intelligence" of a Lingubot lies in its rule base.

A pattern usually consists of strings and logical operators. A string represents a keyword or part of a word. For example, the pattern [I+love+you] would match with any sentence that contains those three words in the same order, for example "I do not love you" or "I do not care whether you love me, you fool!" Of course, there are many different operators, like the not-operator [!] that can be used to exclude certain words (for example the "do not" in the example above). It is also possible to specify how many words are allowed in a sentence that are not contained in the pattern, in order to prevent misunderstandings like in the second example.

For example, the pattern [(I+love+you)&! (do not/not)] With a specified maximum "no hit"-value of 2 would match with the sentences "I love you", "I

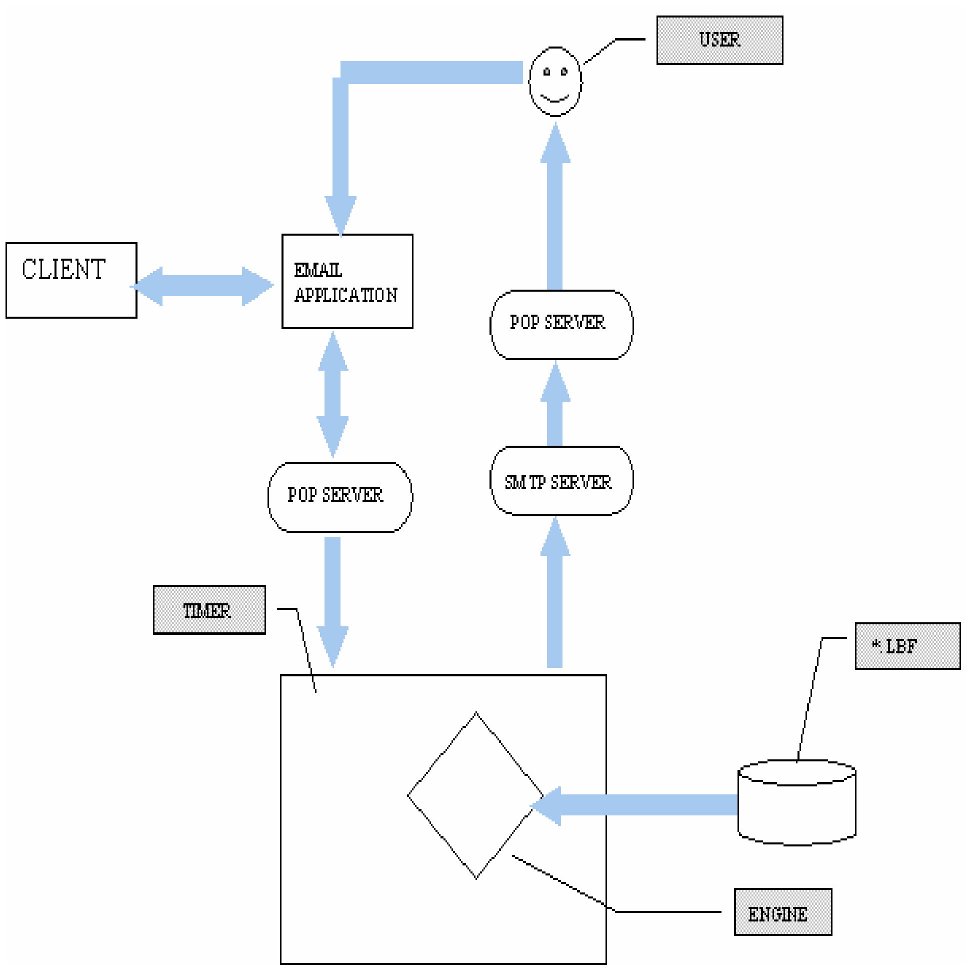

Figure 8.1: E-Mail Application

really love you" and "I love you very much", but not with "I do not love you" or "I love Paul instead of you".

One problem that authors often encounter is that users may put questions in many different ways. For example, the question "What is the price of product XYZ" can also be put as "How much does product XYZ cost" or "What do I have to pay for product XYZ" or even "How much bucks is product XYZ". Since the rule has to match all of those (and more) questions, the resulting pattern can be quite complex. Here is a real world example, taken from the "Marc" Lingubot Kiwi logic developed for Olympus:

```
((((((((how/hwo/hou/hows/howz/how's/hows'))+
(much/muhc))/howmuch/hwomuch)&
(money/cash/dough/bread/pesetas//quid/cents/euro/euros/dm/{$}
/usd/penny/quarter/dime/nickle))
/((((how/hwo/hou/hows/howz/how's/hows')
/(what/wot/waht/wat/waddya/whut)
/(((what/wot/waht/wat/waddya/whut)+is)
/whats/what's/wats/wat's/wots/wot's/wahts/waht's/wotz))&
(expensive/expencive/much/muhc/cost/costs/(does+cost)
&glasses/shades/sunnies/sunglasses)/((it/one)&^eytrek)/{e-t})!{e.t.})
```

Note that a large part of the complexity of this pattern results from the fact that it also recognizes common misspellings, for example "hwo" instead of "how".

8.2 UML

User will refer to the person talking to Lingubot Botnic will stand for the person or company Pop 3 it fetches mail and using fetch mail use case and calling class kstring function divide mail in sentences. Engine input stores the question in local variable and then call the engine. Engine reply question by calling react function and then call engine output Engine output stores all questions given by engine and finally call smtp. That concatenates all answers in the form of mail and sends it back to client. Bot consisting of a web engine that loads a standard knowledge base (i.e.lbf file containing the identification and standard answers) the answers may be saved in as files in database.

Activity Diagram - Figure 8.5

Here the pop3 fetches the mail in the beginning of activity. Divide sentences function is called to split mail into sentences. Set user input is calling in order to store sentences and then get user input is called. C Engine calls the React method to provide the answer to each question. CEngineoutput calls the set replies function to store and then get replies method . if all the questions replied then it send to smtp through send mail, otherwise it sends again back to Get user Input to get answer.

Classes and member functions used in e-mail agent

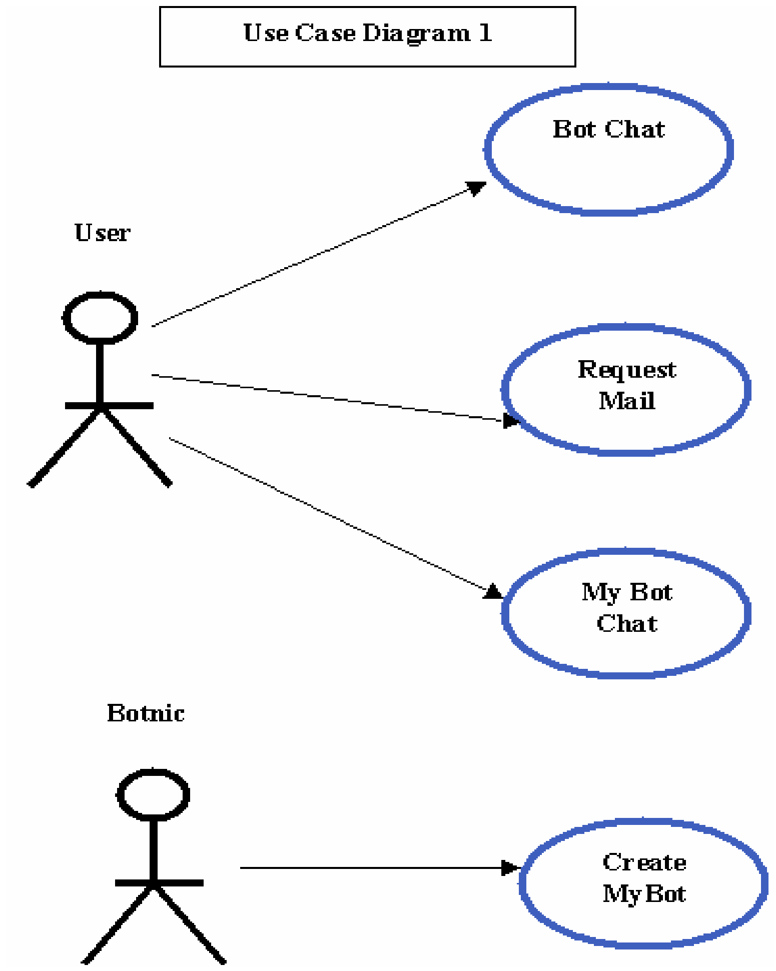

Figure 8.2: Use Case Diagram 1

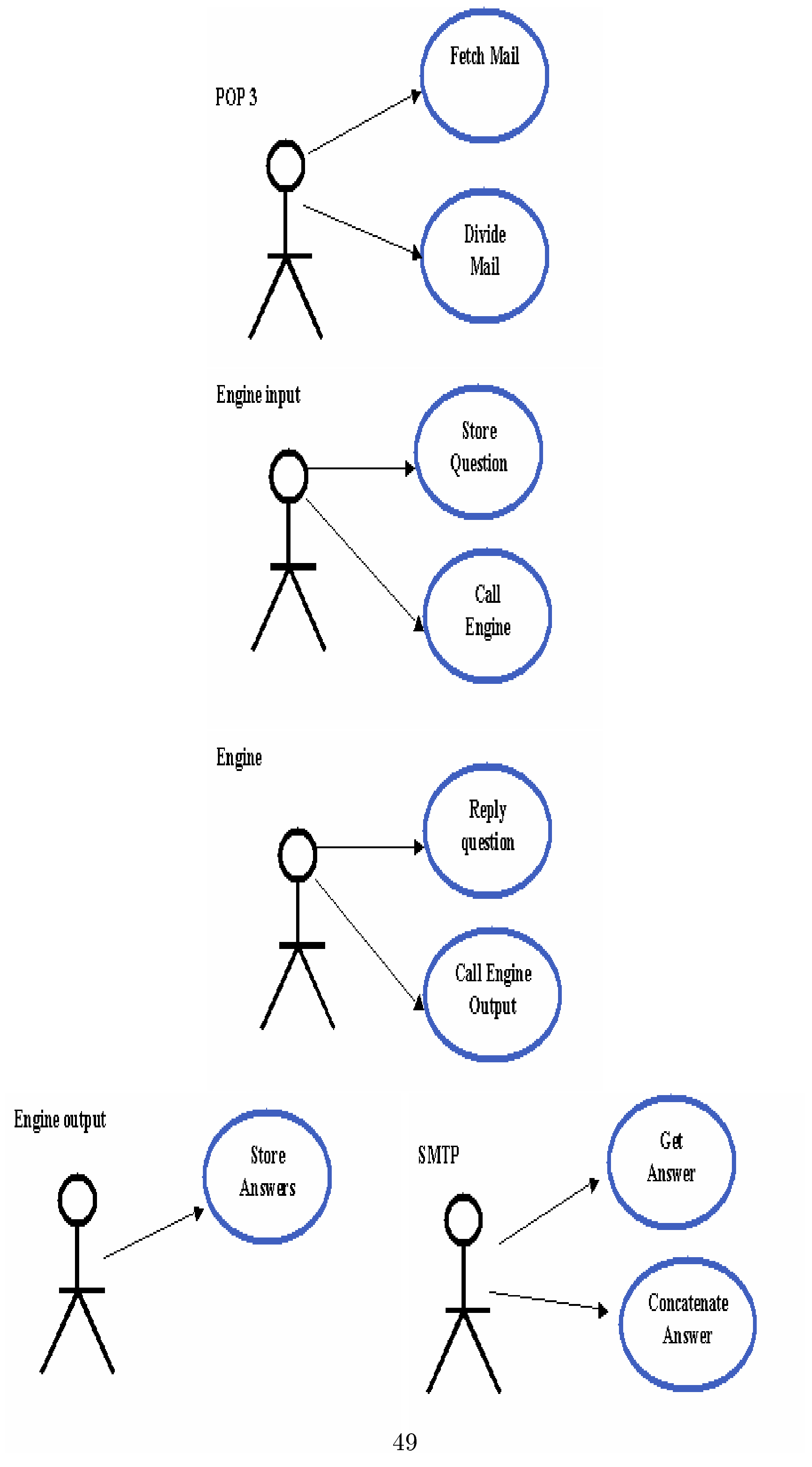

Figure 8.3: Use Case Diagram 2

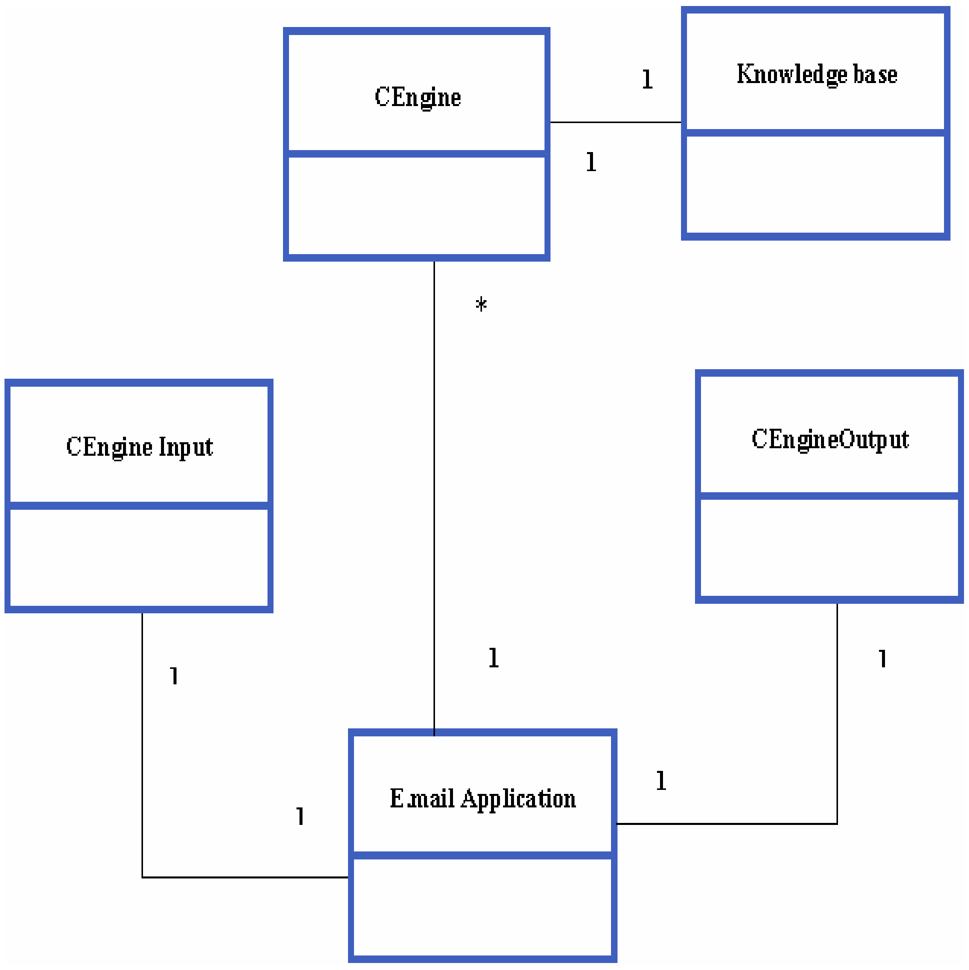

Figure 8.4: Class Diagram

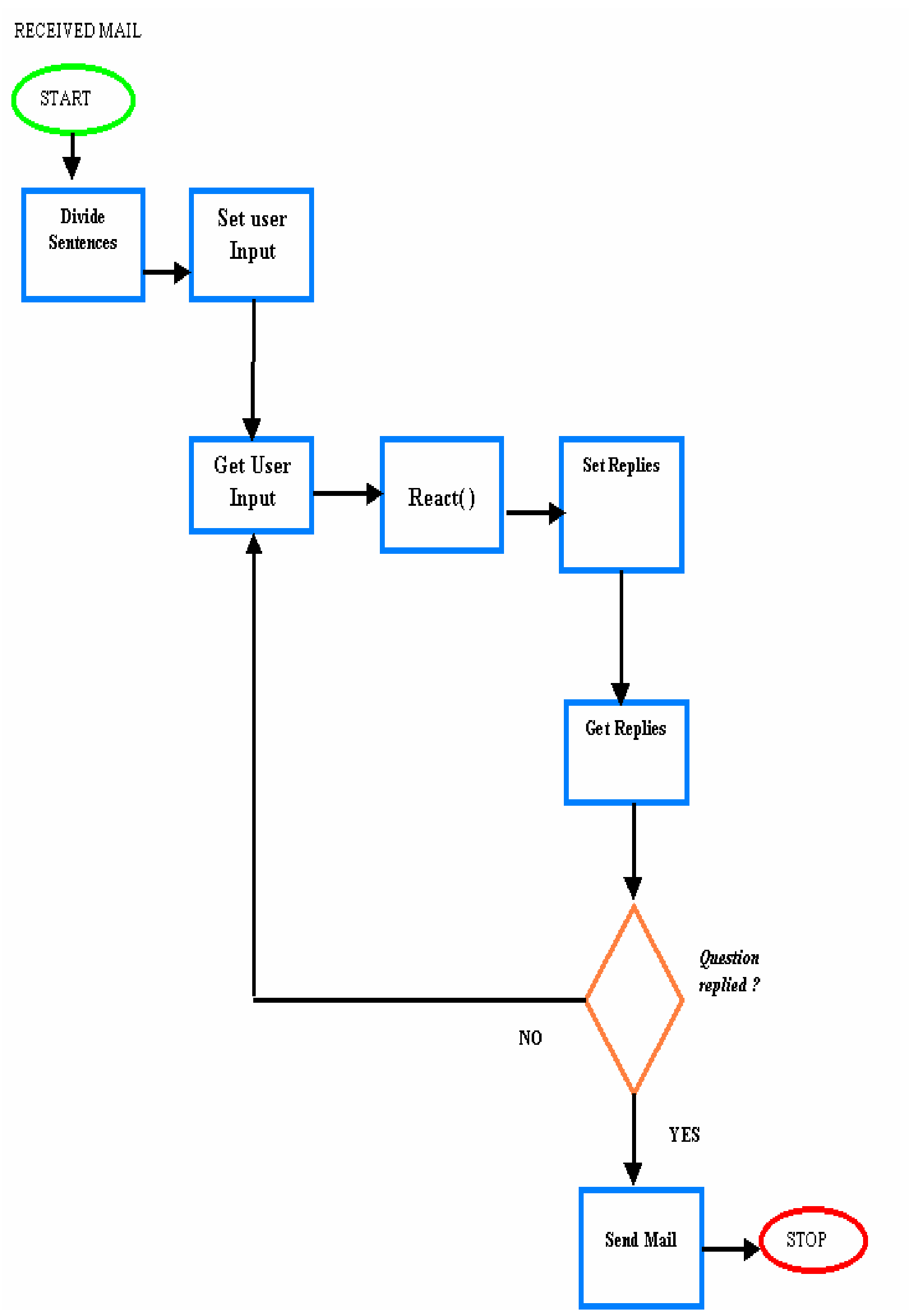

Figure 8.5: Activity Diagram

Kstring.cpp To split up the incoming e-mail into sentences
Divide sentences(Kstring sx Input,const CKStr Vec &vAbk)

CEngine

React( ) To get an answer to sentence. Priority of the return xChosen Recognition has to be compared to the threshold to check if the recognition was good enough.

VoidCengine::React(CengineInput*xEngineInput,

CengineOutput*xEngineOutput, CsessionData*xSessionData, CabstractEngineEnvironment*xAbstractEngineEnvironment, Crecogntion*&xRecogntion, bool xbls Mybot).

Cengineinput Input sentences of mail have to be put here.
Request Mode has to be set to RM_NORMAL
Created before call of a Cengine::React

CEngineOutput Answer can be found here
Created before call of a Cengine::React

CSessionData,CAbstract Engineenvironment Created before call of Cengine::React

Dummy objects only created before calling Cengine::React, After getting answer they will be destroyed.

Chapter 9

Conclusion

Natural language search engine technology has years to go to fit comfortably in a space that aggregates structured and unstructured information. Conventional natural language processing technology is simply not up to the task. Not yet now, may be not yet ever. For one thing, just performing an exhaustive syntactic parse of a document (even a sentence) is still an inexact science.

It's also language dependent in a very deep way that is English as an example can be sliced at many different angles to reveal all manner of different types of variation . we can say that there is technical English, business English, legal English and so on, as well as American English, British English, Australian English, Jamaican English, Asian English. Furthermore we can say that there are hundreds of distinct variants within each of those types of categories, and we can add that there are various (defective) forms of a language that we also need to address. For instance, just because our colleague is not a native English speaker and might make occasional grammatical errors in e-mail does not mean that we cannot understand what he writes. A smart bot /mail agent should be independent of the rules of any specific human language.

Natural language interpretation system has fully taken the charge of human being, the agent is Still not completely automatic. We still need some customer services personal to handle acute problems.

I suggest that case based reasoning could be applied as an alternative, because it can handle a particular Problem. Despite of the limitations of Natural language system the advantages are so great that it transform way people interact with websites quite fundamentally. Combined the upcoming speech recognition technology, it will soon enable people to speak to machines and use mobile Internet devices.

Bibliography

[1] David Prerau,Mark Adler, Intelligent Agent Technology, The Handbook of applied expert systems/Jay Lebowitz,editor, 1998 by CRC press LLC.

[2] Pablo Noriega,Carles Sierra, Lecture Notes in Artificial Intelligence, Subseries of Lecture notes in Computer Science, First International workshop May 1998.

[3] Giarratano Rile, Expert Systems Principles and Programming, Second Edition, 1993

[4] Thomas A Cooper, Nancy Wogrin, Rule based Programming with OPS5, 1988, by Morgan Kaufman Publishers Inc.

[5] Edited by Gerhard Weiss, A Modern Approach to Distributed Artificial Intelligence, 1999.

[6] Gheorghe Tecuci, Building Intelligent Agents, 1998 by Acedemic press.

[7] James P.Ignizio, An Introduction to Expert Systems, 1991 by McGraw-Hill.

[8] Tom Swans, Mastering Borland C++, Second edition, Sams Premier.

[9] Win Runner, Users guide version 6.0.

[10] Kent Reisdorph, Teach yourself Borland C++ Builder 4, Borland Press.

[11] An approach to Knowledge base Management.

[12] Martin Fowler, UML Distilled edition Addison Wesley Longman, Inc.