Modelling Proactive Behaviour of Conversational Interfaces

Vom Fachbereich Informatik
der Technischen Universität Darmstadt
zur Erlangung des akademischen Grades eines
Doktor-Ingenieurs (Dr.-Ing.)
genehmigte

Dissertation

von

Diplom-Informatiker Marcello L’Abbate

aus Offenbach am Main

Referent: Prof. Dr.-Ing. Erich J. Neuhold
Koreferent: Prof. Dr.-Ing. Matthias L. Hemmje

Tag der Einreichung: 27.11.2006
Tag der mündlichen Prüfung: 18.12.2006

D 17

Darmstadt 2006
Abstract

Conversational Interfaces are software systems interacting with their users by means of a natural language based conversation. Their knowledge base consists of a collection of rules, whose triggering depends on text patterns recognized within the user input. The output is sometimes presented to the user in combination with a graphical visualization, e.g. a cartoon, of the related emotion. Even if they are an interesting and promising idea, recent statistics detected poor usage and a relatively short lifetime. In this approach, a conversational system is designed which is not simply reactive to a user's request, but capable of engaging in a collaboration with its users, by taking over the initiative in problematic and unclear situations. By comparing conversational interfaces to finite state machines a formal definition of proactivity is reached and a standard implementation methodology is suggested. Finally, a case study discussing an example implementation of a virtual risk management advisor is provided.
Zusammenfassung


Der im Rahmen vorliegender Arbeit besprochene Ansatz zielt auf die Modellierung eines natürlich-sprachigen Dialogsystems, welches nicht nur reaktiv zu einer Benutzereingabe ist, sondern die Kollaboration ermöglicht, indem es die Initiative in problematischen oder undeutlichen Situationen ergreifen kann. Wiederkehrende Sachverhalte und stereotypische Faktoren kooperativer Dialoge werden identifiziert, klassifiziert und für
die Bestimmung eines pro-aktiven Verhaltenmodells eingesetzt. Das Modell basiert auf
drei aufeinanderfolgenden Aufgaben, entsprechend dem menschlichen Dialogverhalten.
Zunächst soll ein passender Zeitpunkt für das Ergreifen der Initiative identifiziert wer-
den. Dafür wird die Dialoghistorie, d.h. alle bisher ausgetauschten Dialogbeiträge und
das Benutzermodell, d.h. alle bekannten und dialogrelevanten Aspekte und Eigenschaften
des Benutzers, nach jedem Interaktionsschritt analysiert und überprüft. In einem zweiten
Schritt wird der Inhalt der Systemausgabe ermittelt, der pro-aktiv dem Benutzer ange-
boten werden soll. Dabei wird unter Verwendung von Parametern, die aus dem bisherigen
Dialogkontext oder aus zusätzlichen Dialogschritten gewonnen werden, insbesondere auf
externe Datenquellen zugegriffen. Zuletzt werden beide Schritte unter Einbeziehung des
Benutzers und der allgemein gültigen Dialogziele evaluiert, damit über deren Gültigkeit
und Einfluss auf den restlichen Dialogverlauf bestimmt werden kann.

Eine formale Definition von Proaktivität wird ermittelt, indem natürlichsprachige
Dialogschmittstellen mit ausgewählten Aspekten aus der Automatentheorie verglichen
werden. Dabei wird eine standardisierte Implementierungsmethodik eingeführt und im
Rahmen einer Internet-Anwendung vorgestellt. Die Anwendung zielt auf die Umsetzung
eines virtuellen Versicherungsberaters, der im Dialog mit Geschäftsführer kleiner und
mittelgroßer Unternehmen das Risiko-Management durchführt und Unterstützung so-
wie Expertenwissen bei der Ermittlung und Bewältigung potentieller unternehmerischer
Risikofaktoren anbietet.

Die abschließende Evaluierung der Anwendung, durchgeführt unter Einbeziehung ech-
ter Unternehmer mit deren tatsächlichen Betriebsdaten, liefert positive Argumente für
den Einsatz des pro-aktiven Dialogmodells. Benutzer profitieren insgesamt von einer neu-
artigen und intuitiven Interaktionstheorie, basierend auf der Verwendung von natürli-
cher Sprache. Die Evaluierung beweist zusätzlich, dass komplexe und zufriedenstellende
Ergebnisse bei minimierten Interaktionsaufwand und erweiterter Systemunterstützung in
zumutbarer Zeit erreicht werden können.
## Contents

**Abstract** iii  
**Zusammenfassung** v

1 **Introduction** 1  
1.1 Aims and main contributions of the Dissertation 2  
1.2 Structure of the Dissertation 4  

2 **State of the Art and Proactivity** 7  
2.1 Artificial Intelligence 7  
2.2 Taxonomy of conversational systems 14  
2.2.1 Chatterbots 14  
2.2.2 Conversational Agents 20  
2.2.3 Spoken and Multi-Modal Dialogue Systems 28  
2.3 What is proactivity? 33  

3 **Solving Dialogue Problems with Proactivity** 39  
3.1 A Taxonomy of Dialogue Problems 40  
3.2 Product search with a virtual Shop Assistant 41  
3.3 Risk Management as a persistent dialogue problem 45  
3.4 Optimising Dialogues with Anticipated Problem Solving 50  
3.5 A comparison of the case studies 56  

4 **Conceptual Model and main approach** 59  
4.1 Introduction 59  
4.2 A stepwise refinement of chatterbots towards proactivity 61  
4.2.1 Data Types 61  
4.2.2 Basic Chatterbots 62  
4.2.3 Chatterbots handling variables 67  
4.2.4 Dialogue Plan-based Chatterbots 70  
4.2.5 Enabling context-based Information Retrieval 76  
4.2.6 Chatterbots with mixed-initiative 82
4.2.7 How to implement proactive Chatterbots ............................. 84
4.3 The Proactivity Model ....................................................... 87

5 Architecture and technologies used ........................................ 91
  5.1 Logical Architecture of a proactive Conversational System ........ 91
  5.1.1 Properties entailing robustness ..................................... 94
  5.2 Dialogue management with AIML ...................................... 96
  5.2.1 Categories .............................................................. 96
  5.2.2 Dialogue Plans and Topics ......................................... 99
  5.2.3 Parsing with Program D ............................................. 102
  5.3 Other relevant technologies ........................................... 103
  5.3.1 Risk Manager Online ............................................... 104

6 Implementation ................................................................. 111
  6.1 RMO with a virtual advisor ............................................ 111
  6.2 VIRMA ................................................................. 113
  6.2.1 The user interface ................................................... 114
  6.2.2 Dialogue Management ............................................... 116
  6.2.3 Confirmation .......................................................... 119
  6.2.4 Question Identifier ................................................... 120
  6.2.5 The Dialogue Plan ................................................... 122
  6.2.6 Conclusions .......................................................... 125
  6.3 Example Walkthrough ................................................... 125

7 Evaluation ................................................................. 135
  7.1 Functionality Tests with Domain Experts .............................. 136
  7.2 Usability Tests with Real Users ....................................... 137
  7.2.1 Usefulness of the system output .................................... 138
  7.2.2 Quality of system responses to user questions ................... 139
  7.2.3 VIRMA’s own initiative .............................................. 140
  7.2.4 Quality of results .................................................... 141
  7.2.5 Session length ....................................................... 142
  7.3 Lessons learned ....................................................... 143

8 Conclusions and future work ................................................ 145

Bibliography ................................................................. 147
1 Introduction

Research and development in the area of human-computer collaboration has not yet achieved completely satisfying results. The currently most widespread interaction mechanisms rely on the design of direct manipulation interfaces based on the desktop metaphor [Shn98]. A computer system is regarded as a tool or a virtual environment, offering interface objects as representations of formal objects or functions applied to them. To achieve efficiency and usefulness, the chosen metaphor needs to provide an appropriate set of source concepts to be used for the target objects and actions. Moreover, a high degree of intuitiveness is needed, allowing for usage by both novice and expert users. Such a complete metaphor is hard to find, especially if one considers the constantly growing complexity of the tasks to be performed on computer systems. Sophisticated interaction styles are required which allow both the user to clearly formulate his intentions and information needs and the system to react in a more flexible and useful way [SBC98]. The most powerful metaphor fulfilling these requirements appears to be a conversational interface [BCST95, TGMS95]. Through the use of natural language, specific learning skills are not needed on the part of the user. Moreover, the grammar of a natural language provides enough expressive power for formulating an interaction step even in many different alternative ways. And the combination of subsequent interaction steps (i.e. considered as a whole as a dialogue) can be intuitively utilized for describing and achieving very complex tasks. Obviously, the burden in the employment of conversational interfaces isn’t on the user, as he continues to adopt a means of communication used every day while interacting with a computer system (some authors in the current literature even claim that users of conversational systems automatically tend to employ a limited subset of language, due to situation dependency and perceived quality estimation [ZF91, Hi83]). The onus clearly falls on the interface implementation task. Since communication is defined to be a verbalization of thoughts, generally two major problems need to be addressed by designers: computers must be first taught to "think", and in parallel taught to know the rules of a particular language [Rei85]. Several disciplines like computational linguistics and artificial intelligence-oriented research on discourse modelling, dialogue management and planning need to be involved, since not only the interpretation of isolated and grammatically correct utterances is expected, but also other generally occurring features of natural language communication like for instance sentence fragments, anaphoric references and clarification subdialogues have to be mastered.

Many attempts at combining information systems with natural language based inter-
faces can be found in contemporary research [MM04, MMM05] and even in real world applications in various domains such as e-commerce, e-government and e-learning. They may range from simple question-answering systems [May03] to more complex conversational agent-based applications [HRJvGW99]. Most of them are capable of interpreting an explicit request formulated by means of natural language and thus providing a satisfying reaction based on the results of an underlying search or computational process. But in the majority of cases, a strictly reactive behavior is not enough: users prefer to engage in collaboration with the system rather than having to keep the initiative throughout the whole interaction. A more proactive system behaviour can improve the support of the user’s intentions especially in doubtful and problematic situations. Users expect to receive recommendations and suggestions, reflecting the naturalness of the communication as in the case of a human counterpart. While accessing electronic shopping portals or digital collections of various media users often don’t exactly know what they are looking for and the available catalogue browsing or stock searching services are not manageable enough to allow the expression of incomplete or even as yet unknown wishes. People rely on search engines because they perceive that they lack some knowledge to solve a problem or perform a task. This "anomalous state of knowledge" [Bel80, BK86] or "situation of irresolution" [WF87] in which information seekers must find something they know little or nothing about needs to be addressed by interfaces to Information Retrieval systems by offering user-friendly query refinement methods or feedback techniques.

A proactive conversational system can force the elicitation of information and combine the gathered data for generating suggestions of product categories or recommendations of suitable goods. By taking over the initiative during a conversation the system may also anticipate the solution to problems most likely to occur in subsequent steps. This behaviour, especially suitable for advisory systems, results in keeping interruptions to the flow of the interaction to a minimum. Solutions are suggested to the user beforehand, thus obviating the need for her to even think about how to formulate an adequate description of the problem.

1.1 Aims and main contributions of the Dissertation

Cooperative conversational systems are a current research topic of many universities and research institutions all over the world. The approaches developed are mainly based on effective combinations of natural language processing and sophisticated reasoning. Most of the systems provide for acceptable goal oriented dialogues based on mixed initiative and efficient problem solving capabilities. Nevertheless, the resulting applications are often restricted to a specific domain and are hard to transfer to other scenarios. Moreover, because of their rather complex implementation, they rarely go beyond the prototype stage, thus restricting their use only to academic and scientific environments. The task of processing natural language is often based on computational linguistic techniques
1.1 Aims and main contributions of the Dissertation

For the understanding of a user input, a semantic representation is first produced, according to the current dialogue task. By means of a semantic grammar, individual components of a user utterance are mapped to semantic entities, which encompass and represent a specific meaning. This can be achieved by rather simple context-free grammar parsing algorithms or by more elaborate recursive transition networks. The underlying grammars must be hand-coded, thus requiring an expensive and slow creation process. Within the framework of statistical approaches, their compilation is based on a predefined corpus of text, usually restricted to a specific domain (a more general and wide-ranging corpus may lead to more imprecise results). The entities produced by the semantic analysis of the user input are then further processed by a dialogue manager, which is responsible for the generation of an appropriate system reaction. This is usually achieved through different techniques ranging from frame-based strategies, in which the semantic entities are either used to fill slots or as originators for actions processing already filled slots, to more complex reasoning based on inference mechanisms. In every case, the implementation of a comprehensive dialogue manager is usually a tedious task and its porting and adaptation to other domains often more expensive than creation from scratch.

On the other hand, the demand for commercially available conversational interfaces is constantly increasing. As a result, several companies of all sizes operating on the internet agreed on employing chatterbots, which are conversational systems based on a more manageable technology. Especially in the context of web sites and internet applications, interaction with a virtual representative available 24/7, capable of understanding natural language and responding in a familiar and sometimes entertaining way (typically, an audio-visual presentation of the conversational interface in terms of a 3D or cartoon-like character is used for displaying emotions and highlighting relevant system responses) has become a trendy interface alternative and meets the requirement of a more self-service oriented and thus more widely accessed world wide web. Consequently, the need for a simpler technology enabling dialogue systems which can be easily adapted to different domains and various scenarios has arisen.

Chatterbots have their roots in the early sixties, in the context of the work carried out by Joseph Weizenbaum, a scientist from the Massachusetts Institute of Technology [Wei66]. He created ELIZA, a famous program that simulates a Rogerian psychoanalyst by taking excerpts from the user’s input and reusing them to formulate questions to pose back. Instead of performing a semantic analysis, the dialogue management of chatterbots is based on pattern matching: the user input is examined with the aim of recognizing cue words or phrases (the patterns) on the basis of which pre-prepared or pre-calculated responses are output. In this way, chatterbots can move the conversation on in an apparently meaningful way without requiring them to know what they are talking about. The success of a dialogue requires an active user who constantly takes the initiative during the conversation. As long as questions, assumptions or generic statements containing recognized patterns are input, the reaction of the chatterbot can be guaranteed by the output of the related response. Obviously, pure cooperative dialogues are harder to
achieve, as they require a system which is capable of acting on its own initiative and not only as a reaction to identified strings. Since the meaning of the user input is not available, other triggers for deciding on taking over dialogue control are missing.

The approach discussed in this work provides an enhancement of chatterbots with proactivity, which is an extension of the original reactive behaviour of chatterbots with more goal-oriented and thus cooperative dialogue contributions. To achieve this aim, recurring circumstances and stereotypical factors of cooperative dialogues are identified, classified and used as guidelines for the determination of a generic proactivity model. Specifically, three problem classes are identified, whose solutions require active initiative by the system:

- **Problems arising as a result of the last user input**, i.e. in the case of no applicable rule matching the patterns of the input

- **Problems subsisting through the whole dialogue session**, or in specific sub-dialogues, which require a systematic structuring of the entire rule base and a flexible adaptation strategy of the dialogue management

- **Problems which have not yet arisen, but are most likely to occur in the future**, whose anticipated solution prevents interruptions and deviations from the dialogue

On the operational level, the model tackles the execution of three subsequent tasks in trying to reproduce the behaviour adopted by humans in real dialogues. First of all, a suitable point of time for taking over the initiative has to be identified. The dialogue history, i.e. all utterances exchanged in previous interaction steps, and the user model, i.e. any known aspects of the user that may be relevant to the dialogue behaviour of the system, have to be analysed after every interaction step with the aim of determining the context of the conversation. The collected data is inspected in order to check whether the requirements of a proactive operation are fulfilled. In case of compliance, the second main step has to be undertaken, namely the determination of the content to be proactively transmitted. External information sources may be involved, using efficient methods for their access. Parameters to be used are extracted from the available contextual data or directly elicited by means of further interaction steps. Finally, both the decision on and the content of a proactive system output need to be evaluated. The main aim is to guarantee coherence and consistency by also considering the main goals to be reached during the interaction.

### 1.2 Structure of the Dissertation

This thesis is organised as follows. Chapter 2 starts with a brief overview of the history of conversational systems. Since different definitions and interpretations can be found in
the current literature, a taxonomy of conversational systems is also discussed, by pointing out relevant research areas in which they are considered. The current state of the art is introduced and relevant implementation variants for usual components of conversational systems analysed. By citing pertinent literature, a generic definition of proactivity is also provided. Chapter 3 aims at determining the requirements for the specification of a proactivity model. They are derived from both the analysis of the state of the art and general issues about cooperativity in dialogues with a chatterbot. The discussion and realisation of a generic proactivity model is contained in chapter 4. For this aim, the functionality and processing paradigms of chatterbots are compared to finite state machines and automata theory used for describing and proving the feasibility of the intended proactive enhancements. Chapter 5 analyzes a possible logical architecture for proactive chatterbots according to the model introduced previously. The individual behaviours of the required components are characterized and a possible integration methodology explained. A real implementation of a proactive chatterbot based on the proposed model and architecture is described in Chapter 6. The outcomes of usability and functionality tests, commenting the efficiency and usefulness of the implemented chatterbot are presented in Chapter 7. Finally, Chapter 8 provides a summary and conclusions.
2 State of the Art and Proactivity

The first steps towards natural-language based interaction between a human and a computer system date back to the early sixties. Conversational systems like "Eliza" (created by Joseph Weizenbaum of MIT’s Computer Science Department [Wei66]) and "Parry" (developed by Kenneth Colby of Stanford’s Psychiatry Department [CWH71, Col75]) emerged from artificial intelligence research as they were able to produce out of English input sentences other grammatical output sentences, sometimes giving the flavour of a conversation. The techniques adopted in these early systems are still used today in both academic and commercial environments. At that time other approaches for conversational systems, whose dialogue control was based on the semantics of the utterances exchanged, were developed within the field of computational linguistics [JM00]. Natural language processing techniques are used in the context of various applications, such as spoken dialogue systems, advisory systems, recommenders, expert systems and many others. Hybrid systems mixing pattern matching with natural language processing techniques can also be found occasionally.

In the remainder of this chapter, an overview of existing approaches for conversational systems is provided. First, their roots are analysed by briefly reviewing the history of artificial intelligence. Second, a classification of existing variants of conversational systems is introduced. The most recent achievements are then discussed by placing the focus on the components which typically form a conversational system. Finally, proactivity is defined and presented within the context of other application areas.

2.1 Artificial Intelligence

When the term "Artificial Intelligence" (AI) was first introduced by John McCarthy in 1956 to denote the various approaches aiming at creating machines with intelligent behaviour [MMRS55] an initial wave of enthusiasm was noticeable among researchers from different computational fields. The idea of reproducing human capabilities such as problem solving, learning and language stimulated many scientists to devise new theories and approaches. But instead of consolidating the efforts in a common research area, they gave birth to different and autonomous disciplinary fields concentrating only on the implementation of specific facets of human intelligence. For instance, the ability to solve problems like humans do was pursued in the fields of logical reasoning [NS57, Min00] and knowledge representation [Min85, DSS93]. As another example, researchers
in neuroscience concentrated on building models of the human brain through neural networks [Fau94]. Also the problem of understanding language soon defined its own
direction. The field known as computational linguistics or natural language processing
was based on the assumption that natural language is the most obvious way to express
intelligence. Tightly related to the issues addressed by cognitive science, a theoretical
branch of computational linguistics emerged. It dealt with the development of formal
models simulating aspects of the human language faculty and their implementation as
computer programs. On the other hand, applied computational linguistics focused on the
practical outcome of modelling human language use. The goal was to create methods,
tools and applications for improving human-machine interaction, which was seen as a
generic communication problem. Moreover, applied computational linguistics aimed also
at the implementation of fully automatic translation between human languages in order
to solve an even older communication problem between people with different mother
tongues [Wea57, Boo55].

As a result of this heterogeneous approach, the late 80s were characterized by a general
withdrawal of both academic and corporate funding. The so-called "AI Winter" arrived,
since the initial promises of AI that computers would begin to process language and
ideas in a human-like way - did not come true. As an attempt to categorize the outcomes
of this first unsatisfying phase of AI history, two main branches should be distinguished:
"strong" AI, dealing with the creation of techniques and methodologies aiming at true
reasoning and problem solving, and "weak" AI, aiming at a mere simulation of reasoning
processes. A weak AI machine acts as if it were intelligent, without really possessing
sentence or true intelligence.

Attempts to implement strong AI were initially stimulated by the hypothesis that
intelligent behaviour can be explained and therefore also guaranteed using a small number
of principles [NS61]. This assumption failed and further efforts in this direction were
kept to a minimum, and researchers moved into related areas with more modest goals
such as expert systems (early well-known systems are DENDRAL [BF78] and MYCIN
[Sho76]), machine learning (the first algorithms dealing with symbolic machine learning
are "Version Spaces" [Mit77] and ID3 [Qui79]) and robotics (the first robot featuring
"techniques in artificial intelligence applied to the control of a mobile automaton in a
realistic environment" was called SHAKEY [RN66]).

The most prominent example of weak AI are chatterbots, i.e. software systems which
try to simulate the conversation or "chatter" of a human being, often entertaining
the user with some "smalltalk". Chatterbots such as "Eliza" and "Parry" are well-
known early attempts at creating programs that can at least temporarily fool a real
human being into thinking he is talking to another intelligent person. By implementing
"Eliza", Weizenbaum aimed to prove that a computer system capable of entering into
a dialogue with users does not necessarily need to be "intelligent". In other words,
dialogues are possible to some extent even if no semantic representation of the contents
is used by the system. Instead of confirming his rather "provocative" assumption, public
opinion agreed in asserting that Weizenbaum’s efforts indeed created an effective weak AI system, simulating intelligence. Eliza models the behavior of a psychiatrist (or, more specifically, the "active listening" strategies of a Rogerian therapist). It largely consists of rephrasing many of the patient’s statements as questions and posing them back to the patient. The context of psychotherapy was chosen as it does not necessitate giving "the program a data base of real-world knowledge", as Weizenbaum asserted. Indeed, the therapeutic situation can be seen as one of the few real human situations in which a human being can reply to a statement with a question that indicates very little specific knowledge of the topic under discussion. For an example conversation with Eliza see Figure 2.1.

<table>
<thead>
<tr>
<th>WHO ELSE IN YOUR FAMILY TAKES CARE OF YOU?</th>
</tr>
</thead>
<tbody>
<tr>
<td>My father.</td>
</tr>
<tr>
<td>YOUR FATHER?</td>
</tr>
<tr>
<td>You are like my father in some ways.</td>
</tr>
<tr>
<td>WHAT RESEMBLANCE DO YOU SEE?</td>
</tr>
<tr>
<td>You are not very aggressive</td>
</tr>
<tr>
<td>But I think you don’t want me to notice that.</td>
</tr>
<tr>
<td>WHAT MAKES YOU THINK I AM NOT VERY AGGRESSIVE?</td>
</tr>
<tr>
<td>You don’t argue with me.</td>
</tr>
<tr>
<td>WHY DO YOU THINK I DON’T ARGUE WITH YOU?</td>
</tr>
<tr>
<td>You are afraid of me.</td>
</tr>
<tr>
<td>DOES IT PLEASE YOU TO BELIEVE I AM AFRAID OF YOU?</td>
</tr>
<tr>
<td>My father is afraid of everybody.</td>
</tr>
<tr>
<td>WHAT ELSE COMES TO YOUR MIND WHEN YOU THINK OF YOUR FATHER?</td>
</tr>
<tr>
<td>Bullying.</td>
</tr>
<tr>
<td>DOES THAT HAVE ANYTHING TO DO WITH THE FACT THAT YOUR BOYFRIEND MADE YOU COME HERE?</td>
</tr>
</tbody>
</table>

Figure 2.1: A dialogue with Eliza. The capitalized lines are the machine responses.

As a counterpart to Eliza, Kenneth Colby created the chatterbot Parry in 1971, simulating a paranoid patient in a clinical interview. Colby regarded his system as a useful tool to study the nature of paranoia. Parry engaged its partner in conversation by stating its "beliefs, fears, and anxieties" within a certain network of association in a question-answer mode, producing the appearance of a paranoid report. Parry’s success was mainly due to the analogy between pattern matching based language interpretation and the usual dialogue behavior of paranoid patients. Indeed, the decision about how to
respond to an interviewer does not depend on a deep and highly nuanced understanding of a sentence, but rather on "partial, idiosyncratic and idiologic meaning" [Koe81]. The answers generated by Parry are based on isolated terms and simple syntactic constructs recognized within input sentences. In the same way, paranoiacs react to their interviewers by reading content into signs and symbols and discovering a meaning that is not really there. When run Parry merely classifies the input by a set of superficial metrics and its current emotional state (set within variables such as Anger, Fear, Mistrust, as well as some superficial memory of what has happened in the conversation so far). Some psychiatrists who carried out a conversation with Parry were not able to recognize its intrinsic synthetic nature. They believed they were talking to a human paranoid patient, whose bizarre, stereotyped behaviour could usually be expected. Despite its more complex implementation (the knowledge base contained almost 6000 rules), Parry did not match the popularity of Eliza.

Meanwhile, approaches to conversational systems based on natural language processing techniques appeared on the scientific horizon. After being deployed mainly in the context of machine translation tasks (based on the work on Generative Grammars introduced by Chomsky in his book *Syntactic Structures* [Cho57]), two significant developments led to the consideration of natural language processing within dialogue applications. First, Schank developed his "conceptual dependency" theory which enabled computers to make more plausible inferences about the meaning of the "semantic primitives" in sentences [Sch69, Sch72]. Second, Woods conceived a parsing scheme called the "Augmented Transition Network" [Woo70]. By mixing syntax rules with semantic analysis, the scheme could discriminate between the meanings of ambiguous sentences. The key systems developed on the basis of these new achievements were SHRDLU, LUNAR and LIFER/LADDER.

Terry Winograd, a doctoral student at MIT, implemented the first widely accepted natural language understanding program called SHRDLU [Win72]. By means of English terms, a user could instruct SHRDLU to move various objects around in a small "blocks world" containing basic objects such as blocks, cones, balls, etc (see Figure 2.2). What made SHRDLU unique was the combination of simple ideas that added up to make the simulation of "understanding" far more convincing. Indeed, the restricted set of managed objects and locations could be described by including as few as perhaps 50 words (including the verbs referring to feasible actions). Moreover, the required "memory" power, i.e. the faculty of remembering the previously processed instructions and modifications to the "blocks world", was very low, even if learning capabilities (such as of user defined names for the objects) were provided. SHRDLU understands its world in terms of the relation between semantics and the physical properties of the objects and locations. It consists of subsystems that parse, interpret and construct sentences, carry out dictionary searches and semantic analyses and make logical deductions. Despite its great acceptance, it has gradually been realized that the techniques employed in SHRDLU would not work beyond artificially defined toy worlds or restricted areas of expertise because, to do so,
the computer would have to possess vast amounts of knowledge that humans regard as common knowledge or common sense.

![Image of a robot with a block]

**Figure 2.2: SHRDLU Original screen display (shown on a DEC-340 display screen attached to a PDP-6 computer)**

The LUNAR system answered questions about rock samples brought back from the moon [Woo77]. The program used an Augmented Transition Network parser and Woods "Procedural Semantics" approach with the aim of building queries to two databases, one containing chemical analysis of the moon rocks and one collecting literature references. The parser was able to identify the basic elements of English sentences such as subject and object, nouns, verbs, adjectives, adverbs, prepositions, and so on. By using augmented transition networks, improvements in terms of performance and efficiency with respect to earlier rule-based approaches could be reached. On the other hand, the network was not able to parse slightly ungrammatical sentences that a human would easily understand. Moreover, correct sentences from the grammatical point of view but without a clear meaning were also accepted. The parsed questions were used as input to an understanding process. Its aim was to re-cast the parsed entities into a simple vocabulary of "semantic primitives", i.e., basic units of meaning arranged to a network (e.g. the primitive "Stone(X)" encompasses the meaning that X is a stone). The primitives were then translated into database queries. A preliminary evaluation of LUNAR provided impressive results (almost 90% of the questions in the test set were answered successfully). This was due to the fact that the people involved in the evaluation were all geologists, who instinctively knew the kind of information that was likely to be in a geographical database. Indeed, further tests with students and professionals from other fields showed far less convincing results.

LIFER/LADDER was designed as a natural language interface to a database of information about US Navy ships [Hen77]. The system used a semantic grammar (that is, a grammar based on labels such as "ship" and "attribute" rather than syntactic labels such as noun and verb) to parse questions and query a distributed database. Moreover,
LIFER/LADDER provided first attempts to handle problems with reformulation ellipsis. Fragmentary inputs could be successfully parsed, if similarities with previously parsed inputs were found. For instance, consider the following piece of dialogue:

User: What is the length of the Kennedy?
System: 200 meters
User: The fastest aircraft carrier?

Since both "the Kennedy" and "the fastest aircraft carrier" match the semantic category "ship", the latter phrase is allowed to substitute for the former and an answer can be provided. The importance of ellipsis handling increases if one thinks that one of the main findings during the evaluation of the LIFER/LADDER system was that humans quickly adapt to the machine and attempt to use very incomplete sentences, even to the point of entering input that looks rather like informal database query language statements. Systems using semantic grammars are closely tied to the domains for which they were designed, and they can be easily adapted to suit new terms or phrases. On the other hand, an already designed semantic grammar can only be ported to different domains with difficulty. Implementation from scratch is preferable, in this case. Today, semantic grammars are widely used in most natural language processing systems, but there are many variations and new approaches are continually being developed.

At the beginning of the 90s software agents came into play: their faculty of perceiving their environment and of acting upon it somehow resembles the behaviour of human beings, and if rationality can be assumed for steering the agent's internal processes, a possible framework for a strong AI system can be achieved. A rational or intelligent agent is one that does the right and most successful thing, given that an effective way for measuring success can be identified. A traditional criterion is language: dialogues with a rational agent can be compared to dialogues with human counterparts and if no clear difference is noticed, rationality can be assumed.

One of the first conversational systems designed as an intelligent software agent was HOMER [VB90]. A modular architecture allowed for the integration of separate components, such as a memory, a natural language processor, a learning module and so on, managed by a central control module called the planner (see Figure 2.3). The underlying philosophy behind the creation of HOMER was the ambitious intention to integrate available artificial intelligence research results into a single working agent. The application simulates a robot submarine which takes instructions from a user in a limited subset of English (with about an 800 word vocabulary) and performs the related activities in a two-dimensional "Seaworld" (which typically relate to collecting and moving items around). The memory can be divided into two sections, general knowledge, including world knowledge and lexical knowledge, and episodic knowledge, which is constantly updated during the interaction by means of the learning module. It gives the agent the ability to answer questions about its past experiences. The planner is responsible for the performance of
the actions requested by the user. Scheduling allows for the execution of actions at a certain time. When the current action plan becomes obsolete (e.g., it is fulfilled or the current circumstances do not allow its processing) replanning is applied and a new plan selected out of a local episodic memory. A natural language generator is responsible for providing textual feedback to the user in response to requests and instructions.

![Diagram](image)

Figure 2.3: The architecture of the HOMER agent

HOMER had a number of drawbacks. The system performance suffered from the episodic memory growing too large and from relatively slow plan generation and execution processes. The limited simulation of the "Seaworld" environment produced a behaviour which strongly differs from typical behaviour patterns in the real world. Combined with the overall lack of scalability of most of Homer's modules, the system was not convincing enough to be considered as an effective prototype for the solution of real world problems.

In sum, artificial intelligence can be regarded as a common denominator for the outcomes of this past 50 years of research on conversational systems. Nevertheless, a broad range of technologies and mechanisms previously associated with AI are nowadays found in both academic and commercial environments. Depending on the kind of language understanding, dialogue modelling or means of interaction utilised, a variety of different approaches can be currently found for the creation of systems capable of communicating with their users through natural language. The following section provides an overview of the state of the art in conversational systems categorised by the most relevant technologies and application areas.
2.2 Taxonomy of conversational systems

The following review of currently active conversational systems starts with a more detailed examination of chatterbots. We will look at their principal implementation frameworks and the established companies successfully commercialising the technology. Approaches based on agent technology are subsequently analysed and the major research institutions dealing with conversational agents introduced. Finally, we will consider conversational systems enhanced with additional or alternative interaction channels. Question-answering systems, which provide responses to isolated user questions without the context of a dialogue, are not considered in this overview.

2.2.1 Chatterbots

A chatterbot (i.e. "chatting robot") is a software system capable of engaging in a conversation with users in natural language. Its knowledge base consists of a collection of dialogue management rules, which mainly use text pattern matching for processing the user's input. Applying a rule has the effect of determining the output text by taking into account some keywords extracted from the preceding user's input sentence. Some chatterbots are also able to perform a first elementary user modeling. Answers to direct questions such as "What is your name?" or "Where do you come from?" are decomposed to extract strings which are stored in variables for further usage. Rule bases of more powerful chatterbots may be subdivided into different subsets, each of which is concerned with a particular topic. The management of sentences having different meanings depending on the subject of conversation becomes more flexible and efficient. Indeed, the application of a rule also causes a change to the chatterbot's internal state, providing more appropriate knowledge for satisfying the user's requests.

The efficiency of pattern matching approaches is often legitimised by applying the famous Zipf's Law [Zip49] to the set of all possible user sentences appearing in a dialogue. The law states that even if humans have a theoretically infinite number of different formulations at their disposal, they tend to use only a restricted subset of natural language during conversations. Moreover, a distribution can be assumed, which provides an ordering of the sentences depending on the frequencies by which they are used. Zipf's Law can also be applied to subsets of natural language, for instance comprising all the sentences which start with a specific combination of words (such as "What is"). This feature is often used by pattern matching algorithms for improving the performance of the pattern identification task [KJP77]. The patterns can be arranged into parse trees, whose traversing order is determined by the beginning words of a sentence (in depth) and by the frequencies of usage (in breadth). The branching factor of parse trees decreases rapidly with each successive word of a sentence. Indeed, given a set of words starting a sentence, the overall average of possible words following the first one is usually about two. It becomes about one if we consider the third words of a sentence and so on.
2.2 Taxonomy of conversational systems

Chatterbots (sometimes also called "chatbots", "chatterboxes", "chat agents", "AI-bots" or just "bots") are primarily deployed in the entertainment domain. The aim of their dialogues is to provide some amusement for the user, as if she were talking to a friend during her spare time and just for fun. The textual output of chatterbots is sometimes presented to the user in combination with a graphical visualisation, e.g. a cartoon, of the related emotion. This feature especially enhances the prevalent entertainment function of the majority of existing chatterbots. The visualisation is implemented by using a variety of technologies, which are briefly summarised in section 2.2.3. As regards its control, rules are usually extended with specific instructions, updating the visualisation simultaneously with a related output sentence. A model for automatic generation of chatterbot expressions is discussed in [PL01].

Noteworthy examples for entertaining chatterbots can be mainly found on the Internet. Some of them regularly participate in annual competitions such as the "Chatterbox Challenge" (http://www.chatterboxchallenge.com) or the "Loebner Prize Contest" (http://www.loebner.net/Prizef/loebner-prize.html), which aim to award prizes for the best chatterbots in terms of popularity, learnability and of course the most human-like conversational behaviour. The winner of the Chatterbox Challenge 2005 was JABBEROCK designed by Juergen Pirner (http://www.abenteuermedien.de/jabberwock). Named after a creature invented by the fantasy writer Lewis Carroll, JABBEROCK is defined to be a "chat-beast", without profound real world knowledge and showing a rebellious and stubborn personality. The rule interpreter is implemented in RealBasic, which is a variant of the Visual Basic programming language for the Apple Macintosh platform. The knowledge base, comprising about 1,000,000 rules (and thus, responses), is subdivided into several discourse topics. A multiple-topic handling mechanism as well as techniques for synonym management and rule marking (to avoid providing the same answer twice during a conversation) contribute to an improvement of the dialogue quality and fluency. Here is an example chat:

Jabb: Hello you at the other side of the screen, my name is Jabberwock.
What's your name?
User: Hi There. Why do you want to know my name?
Jabb: What’s that? Entering my lair without invitation, and the first thing is a bluntly question? That’s not the best way to introduce yourself, dude, so let me ask again - what’s your name?
User: My name is unimportant
Jabb: Okay, calm down, sweety. Your name is not that important. I will call you Toddler until we will get a better name
User: How's the wheather?
Jabb: Cool, at least here in my cave
User: Where do you live?
Jabb: I live in a little Fairy Tale Land near by. Want to plan to meet there?
User: Sure. Tell me how to get there.
Jabb: Alright then. By the way: Do you prefer books or TV?

One of the most successful participants in chatterbot contests is ALICE, designed by Richard Wallace of the A.L.I.C.E. AI Foundation (http://www.alicebot.org). Its popularity is also due to the underlying open-source technology AIML (see section 5.2). The knowledge base was used by many chatterbot designers as a starting point for further implementations, creating ALICE clones with their own personality and knowledge focus. ALICE has been subject to numerous studies and evaluations, aiming at a characterisation of its social impact. For instance, the authors of [DJC01] have analysed the dialogue behaviour of users chatting with ALICE, by observing the effects of anthropomorphism and social role of virtual conversational partners.

Many web portals exist, offering free of charge hosting services for AIML based chatterbots. Particularly worthy of mention is the portal of PANDORABOTS (http://www.pandorabots.com). Based on a selection of mostly free and open source technologies, such as GNU Common Lisp and MySQL, the website offers tools for editing and improving self-made chatterbots which do not require particular programming skills. In May 2003, PANDORABOTS hosted about 21,000 bots, some of them having more than 100,000 interactions (i.e. exchanged utterances) per day.

A further multiple-award winning entertainment chatterbot is JABBERWACKY, created by Rollo Carpenter (http://www.jabberwacky.com/). The system is designed to achieve and improve its knowledge during the interaction with its users. By means of appropriate learning mechanisms, rules are created at run time and applied to further conversations whose contexts are similar. This strategy, of course, can cause the system to also provide inappropriate responses if the learned rule contains a wrong knowledge fact, due to a previous misleading user message. An additional feature of JABBERWACKY is to provide an emotional state of the chatterbot which influences the dialogue behaviour. The chatterbot's answers are based on happiness, annoyance, irritation, anxiety, and so on, determined by initial settings or by the current dialogue progression. The project's website offers the opportunity of creating individual chatterbots for registered users. One of them, called GEORGE, won the Loebner Prize Contest in 2005.

Another interesting chatterbot approach based on machine learning is undertaken by the "Ai Research" group (http://www.a-i.com). Based in Tel Aviv, Israel, an international team of scientists, researchers, engineers, and language specialists work together towards the creation of "a new form of life". A reinforced learning algorithm is the core technology for HAL, the "virtual child", i.e. a chatterbot which is supposed to acquire language skills in a similar way like children do. The project web site, in the context of a dedicated area called "The HAL Nursery", offers the opportunity for researchers and for those who express an interest to join the training activities of already created virtual personalities. Custom virtual children can also be created and educated by following particular strategies. The first phase of the learning process consists of the conveyance of
2.2 Taxonomy of conversational systems

basic knowledge units such as elementary shapes, colours, mathematical skills, behaviour patterns and so on. The next phase deals with more complex entities, which are composed of the basic units conveyed before. A set of heuristics is responsible for controlling and enabling the learning process during a conversation. For instance, if a new term is introduced by the user (let's say "dog"), a heuristic foresees asking about its meaning (i.e. "What is a dog?"). A following user answer ("A dog is an animal") is used for creating new relationships or updating data classes previously learned (the value "dog" is added to the data class "animal" which possibly already contains "cat, bird, lion" and so on).

The first commercialised entertaining chatterbot was the virtual storyteller RACTER (http://www.mobygames.com/game/ racter) released by Mindscape Inc. in 1984. It was first available for DOS systems and then ported to Apple and Amiga platforms. RACTER was based on a computer program written by William Chamberlain and Thomas Etter in 1984, which was apparently able to generate English language prose at random. The program itself never became public, but the authors published a book, called The Policeman's Beard is Half Constructed, which was entirely written by RACTER. Sentences were composed by putting words together on the basis of "syntax directives" given by the authors. The illusion of coherence of the generated text was increased by repeated re-use of patterns, which mainly corresponds to the dialogue management process of chatterbots.

A more contemporary commercialisation of entertaining chatterbots is provided by the company Artificial Life (http://www.artificial-life.com). The focus is on mobile computing, allowing also for interactions by means of cellular phones and personal digital assistants (PDAs). Their main product, called the "virtual girlfriend" (available on a separate site, http://www.v-girl.com) or the virtual boyfriend (http://www.virtual-boys.com), allows customers to create a virtual chatting partner. By choosing it out of a catalogue of predefined personalities which can be personalised according to individual wishes and preferences, the user can establish a persistent relationship with the chatterbot created. The friendship is cultivated by means of different interaction channels, such as online chat, electronic mail and mobile messaging. The topics and contents of the conversations influence the emotional state of the virtual partner and the addiction to the user in the long run.

On the internet, chatterbots are frequently used as interactive guides. While showing the different pages of a web site, they tell the user about the contents, functionality and the main controlling mechanisms of the currently displayed page. They also take over the role of a virtual shop assistant or salesperson, trying to establish a trusting and confident relationship with the customer (see Figure 2.4 for some examples).

By acting as a company's representative, they provide quick answers to frequently asked questions and an alternative means for satisfying information and orientation needs. In the setting of a conversation, configuration tasks can become a more natural process, and the user can benefit from a familiar and intuitive interaction. Virtual assistants also
support order management, by collecting relevant user data, such as delivery address and payment methods, and answering questions about shipment tracking and delivery status. By using chatterbots, companies are able to reduce the costs of online customer support services. Indeed, the acquisition and maintenance of chatterbots are usually cheaper than the operating costs of call centres. Thus, chatterbots provide a valuable alternative in the field of electronic customer relationship management (eCRM).

Many companies have succeeded in commercialising improved chatterbots especially as user interfaces for websites. One of the leading European companies in this field is Kiwilogic (http://www.kiwilogic.com). Their product, called Lingubot, has been adopted by several top European companies, such as Daimler Chrysler (Smart), SAP and Siemens. The architecture of Lingubots is kept very simple allowing for an easy integration with existing websites. A web engine located on a server computes an answer to a user’s input using a predefined knowledge base. The user input is first divided into single words and then compared to patterns of decreasing complexity and relevance. The patterns of the knowledge base consist of logically combined required words and at least one answer is associated with each pattern. The most relevant fitting pattern determines the answer to deliver back to the user. By the use of a set of communication interfaces the Lingubot Web Engine is additionally able to address external components such as visual representations of emotions, databases and other applications, but access is hard coded
2.2 Taxonomy of conversational systems

within the rules of the knowledge base and therefore triggered only in the case of matching patterns. Another worldwide leading company commercialising chatterbots is Native Minds (http://www.nativeminds.com). Their product, called virtual Representative (vRep), can be found on the websites of the Coca Cola Company, Ford Motor Company, American Express and many others. As in the case of Lingubots, vReps operate by pattern matching, comparing a user’s question with possible answers and selecting the answer most appropriate for the question asked. Instead of searching for information from within a company’s database, a predefined question taxonomy is used to map similar questions back to a core answer, thus making it possible to answer a higher number of questions than the preloaded ones. Particular care has been given to the implementation of a memory. If the conversation comes to a dead end, a vRep recalls what the user said in previous dialogue steps trying to offer further support.

In the case of more sophisticated service-oriented web sites chatterbots are also employed to provide users with support during the operation of integrated data processing tools or expert systems. Their complexity can be often compared to reasoning in intelligent systems, thus augmenting the strictly conversational-based nature of pure chatterbots. This is the case of the CyN Project (http://www.daxtron.com) which merges an AIML-based chatterbot with the OpenCyc general knowledge base and inference engine [Cou04]. By means of a natural language based conversation, the user can interrogate the OpenCyc ontology, thus leading to a chatterbot with enhanced knowledge. Moreover, CyN can also be used for updating the contents of OpenCyc. This entails a different view on the system, i.e. one of an inference engine providing an extended user interface based on natural language.

Different open source frameworks for building chatterbots can be found in current research. They are based on both proprietary developed and well-known technologies, such as Java or XML. Among them, the JFRED chat server [NG97] is realised as an extensible Java package and can be installed on different platforms. The server supports a variety of front-end/client interfaces, ranging from telnet-based applications to Java applets embedded in HTML pages. The responses of chatterbots built in the context of JFRED are determined by a pattern matching approach based on fuzzy logic. Besides the representation of patterns in terms of regular expressions, fuzzy rules are used for matching similar strings on the basis of a given probability. A frame-based learning technique allows for the management of contextual information acquired during a conversation.

AIML (the Artificial Intelligence Markup Language) is definitively the most prominent open source chatterbot implementation language currently available. Its rule set and syntax is SGML-compliant. Several open source parsers based on different technologies such as Java, C++, Perl, etc. can be found. They allow for a free customisation and extension of both rule set and response behaviour. The available parsers can be easily embedded in various common server environments, thus resulting in maximum portability.

Many researchers have focused on extensions of AIML. Available approaches provide an expansion of the basic rule set for various goals, such as the access to external information
sources and expert systems [KKP03] or an improvement of the level of dialogue fluency [NB03]. Since the personality of AIML-based chatterbots is hardcoded within the rules of its knowledge base, attempts to provide a more flexible approach based on exchangeable personality models can be found in [Rus02] and [GBNR04].

Generic improvements to pattern matching techniques employed in chatterbots can also be found. For instance, the authors of [DV04] propose an approach based on information retrieval for generating system responses by querying an appropriately indexed repository of documents. The repository is constantly updated with new sentences based on the ones that were already present. Their generation is carried out by means of a genetic algorithm [Hol92], which applies the uniform crossover operation on two sentences having the same number of genes. An attempt to mix pattern matching techniques with aspects from computational linguistics is described in [Zdr00]. The aim is to allow for a deeper understanding of the user input in order to detect spelling errors, remember preceding context for the resolution of anaphora and learn new linguistic knowledge during a conversation. Details about techniques from the domain of computational linguistics are discussed in the following section.

### 2.2.2 Conversational Agents

Software agents were introduced in the late 80s for describing complex software entities which act with a certain degree of autonomy in order to carry out predefined tasks on behalf of their user. Nowadays, an adequate definition of the notion of agency delimiting the field from other technologies such as object-oriented programming is still a controversial issue. According to the so-called "weak" definition [WJ95b], agents are software systems which exhibit the properties of autonomy, social ability, reactivity and pro-activeness. In the context of more specific application areas the initial set is extended with other attributes like mobility, rationality, persistence and so on. Given the vast usage in which agents can be applied, several characterisations of the notion of agency have arisen. For instance, Intelligent Agents were introduced as particularly related to the field of artificial intelligence as they are able to learn knowledge, skills and attitudes and adapt to the environments in which they are employed [WJ95a]. In the frame of Information Agents the focus is more on data-specific aspects, since facilities for the discovery, acquisition, management, synthesis, and presentation of information are provided and coupled with automated assistance and support mechanisms for users [Khu01, KAR+05]. Agents communicating with users directly through the input and output channels of a given user interface are called Interface Agents. In their primary assistive and advisory role they act on the person’s behalf for the achievement of tasks and goals [Lie97]. A further classification of agent systems can be found in [FG97].

Conversational agents are a specification of interface agents also embracing attributes and functionalities of intelligent and information agents. Communication with users is mainly based on natural language. In contrast to chatterbot-based solutions, research on
2.2 Taxonomy of conversational systems

Conversational agents has taken a different direction. The restrictions of string matching approaches were recognized soon, and the focus placed on computational linguistics. Conversational agents (called "embodied" if a 2D or 3D visual representation is also provided [CSPC00]) are characterised by a different approach in the language interpretation and dialogue management strategy: instead of using a set of pattern matching rules, semantic analysis and dialogue planning are adopted in order to allow for a more goal oriented dialogue behaviour.

A generic processing cycle of conversational agents can be summarised as follows. The user input is first analysed by a language understanding module. By using different parsing approaches, the semantics of the input are extracted. This task results in the identification of a user's intention to perform an action and the determination of the meaning of the utterance's contents (first studies on human communication came to the conclusions that "speaking is a way of performing actions" and every message contains "atomic pieces with a well-defined meaning" [Aus62, Sea69]). The achieved internal representation of the input is further processed by a decision maker module, which identifies and carries out the most pertinent reaction of the system. It may consist of a reasoning task, producing an output message and/or an update of the current system's state and context. The output message needs sometimes to be adapted or completed before being conveyed to the user. In this case, natural language generation tasks are performed. If the application goals are not reached, the sequence is started again and the next user input processed.

Parsing procedures [GJ90] are usually related to an underlying formal grammar. It describes a formal language by means of a set of rules, delineating a (usually infinite) set of finite-length strings over a (usually finite) alphabet. Formal grammars can have a generative nature, consisting of a basic algorithm which generates strings in the language by successively rewriting strings starting from a designated start symbol. Parsers usually apply a more analytic approach. Given an arbitrary string as input, a set of rules is used for successively reducing or analysing the input string towards a final boolean "yes/no" result indicating whether or not the input string is a member of the language described by the grammar. For this aim, the input string is first subdivided into tokens (during a process called lexical parse phase) each of which is a meaningful symbol with respect to the grammar used. Afterwards, a syntactic analysis of the generated tokens tries to determine whether the sequence is an allowable expression of the underlying grammar (i.e. the tokens correspond to valid components appearing in a valid order). The result of this phase is represented in terms of a parse tree which is further processed in the frame of a semantic analysis. It forms the basis for a decision about the implications of the input such as the appropriate system actions to perform.

Different parsing techniques are currently available and in use [BT96]. They range from approaches limited only to lexical parsing (such as shallow parsing, [Osb00]) to probabilistic-based procedures (such as statistical parsing, [MM91]). Other relevant approaches are based on chart parsing [Kay86], deep parsing [RVS95] and conceptual
parsing [Dye83, Man97], which address the task of achieving successful parsing also in the case of ambiguous or mispelled user inputs.

The semantic analysis of a parsed user message yields to an interpretation of the user’s intentions expressed within her input. By the message conveyed the user aims at performing a specific action which is usually called the speech act in the literature. Austin [Aus62] identifies a taxonomy of three kinds of acts for classifying the meaning of a sentence. For instance, the message "I don’t understand" contains a locutionary act represented by the performative verb "to understand" (i.e. the description of a speaker’s action). Nevertheless, the sentence also contains an illocutionary act consisting of the speaker’s protest (i.e. the description of the speaker’s request for clarification). Finally also a perlocutionary act can be found, which refers to the effects produced on the addressee of the message (namely the hearer’s action to repeat or reformulate his contribution).

A further classification is provided by Searle [Sea69]. He suggests that all speech acts can be classified into one of five major classes (i.e. assertives, directives, commissives, expressives and declarations). More recent work, especially carried out in the context of the development of dialogue systems, provides an expansion of the models introduced by Austin and Searle. Worth mentioning are the dialogue acts defined by Bunt [Bun94] and the conversational moves introduced by Power [Pow79] and Carletta [CH97]. Still based on Searle’s speech act theory, the Foundation for Intelligent Physical Agents (FIPA, http://www.fipa.org) has identified a set of communicative acts which had the aim of creating an international standard agent communication language (ACL). The assumptions of FIPA were mainly based on the fact that communication among agents is purely syntactic. It consists of matching strings which carry no meaning as distinct from natural language.

Speech acts are also often related to underlying discourse models or grammars, which define allowable sequences and structures of speech acts with respect to a given goal. Grosz and Sidner argue about the presence of a discourse purpose that is held by the person initiating the discourse [GS86]. Likewise, also segments of a discourse may be characterised by their own discourse segment purposes, which contribute to the achievement of the related discourse purpose. By means of two relation types, namely dominance and satisfaction-precedence, discourse purposes and discourse segment purposes are related together with the aim of forming a coherent arrangement of discourse goals. It defines the order and structure to be followed for the satisfaction of goals in the frame of a conversation.

Frequent examples of discourse purposes can be found in the context of information agents and more generally in the context of user interfaces supporting information retrieval. Typical applications are based on dialogues initiated by a human information seeker, whose purposes are possibly satisfied by the system (the information provider). A user’s specific information need can be assumed and set as the current discourse goal to reach. As discussed by Belkin [Bel80, BK86], the user’s information behaviour can be
either characterized by a goal-oriented browsing strategy or by a rather aimless search, requiring more support by the information provider. In both cases a refinement of the primary goal into interrelated discourse segment purposes allows for the stepwise determination of system actions, aiming at the retrieval of information which possibly satisfies the user’s need. The system takes over the role of a Mediator [Ing92, Ing94] in charge of translating the identified information need into an internal representation which can be further processed by the underlying retrieval tasks. The mediating function is part of a more extensive methodological framework adopted by interactive information systems, in which user activities and system functions are mapped to underlying behaviour models [HSB96]. As in the case of visual query interfaces, in which a given workflow describes the required steps for the configuration of a final instruction in a machine-readable language (such as for instance SQL), conversational systems can be based on a discourse model, defining the mediator’s strategy for the dialogue guidance and parameter collecting task.

![Diagram](image.png)

**Figure 2.5: The CONversationsal Roles (COR) model**

For instance, Sitter and Stein introduce a model of conversational roles (called COR) to be used in human-computer information seeking dialogues [SS92, SGT99]. In line with the theory of speech acts, the COR model categorises dialogue acts on the basis of their illocutionary force. The model is represented as a recursive (state-)transition network (RTN - see Figure 2.5) that determines all of the legitimate types and possible sequences of dialogue acts, focusing on the negotiation and temporary assignment, acceptance or
refusal of conversational roles during interaction.

A discourse model, such as the COR model, provides directives (also called scripts [BCST95]) for supporting the semantic analysis phase. The semantic analysis consists of a mapping between the generated parse tree and a given set of speech acts. Different approaches can be found, the most relevant being the plan-inferential approach (based on inference and deductive mechanisms applied to logical predicates, [GL71, Sea75]) and the cue-based approach (in which different sources of knowledge are used in support of the mapping, such as lexical, collocational, syntactic or prosodic cues, [JBC+97]).

Once the semantics of the user input have been interpreted, the reasoning module of conversational agents assumes control by determining an appropriate agent’s reaction. It may consist of messages to be delivered back to the user, which have the form of natural language sentences or audio-visual representations, and of environmental changes, such as contextual updates or rearrangements of objects of the agent’s world. The reasoning module (also often called the dialogue manager) is therefore responsible for controlling the flow of the dialogue, as it decides at a high level how the agent should contribute to the dialogue continuation. The reasoner tries to accomplish a predefined goal by taking into account the user’s messages and the directives coming from a specific discourse plan chosen from a predefined set [MR91]. The plans consist of sequences of dialogue rules, mostly related to an underlying discourse grammar or model. Plans also contain rules which address external computational components or expert systems, providing further support for the determination of appropriate answers to user enquiries. If the current dialogue goal changes during a conversation, a new plan is determined and used for the forthcoming steps.

Software agents often base their plan management strategy on the BDI (Belief, Desire and Intention) model [BIP91, RG92]. The agent’s domain knowledge (i.e. the beliefs), can be statically set up at design time and/or dynamically updated at run time. The intentions represent the plans to be used for achieving the desires, i.e. the goals of the system. Typical for BDI agents is a balanced behaviour between the properties of reactivity and pro-activeness. Indeed, environmental changes which are perceived by the agent’s sensors (i.e. input means) are reflected by updates to the currently adopted plan (as a “reaction”) and influence the decision on persisting or replacing it (as a “pro-action”). Plans need to be managed efficiently in order to avoid conflicts and redundancies among the goals. Desires which cannot be satisfied in the current environment are "dropped" and the related plans discarded. Specifically in the setting of conversational agents desires can be represented by a set of discourse obligations. Both the user’s intentions and the agent’s task-related goals are modelled on the basis of discourse obligation rules, defining priorities and relations which lead to obliged actions and other conversational commitments (such as promises, acknowledgements, coordination, and so on) [TA94]. In sum, the dialogue manager of a BDI-based conversational agent bases its task of generating the system’s response speech act on the agent’s current beliefs about the domain, on a model of discourse obligations and on the application of a plan’s rule.
2.2 Taxonomy of conversational systems

A flexible plan-management and replacement approach allows for dialogues based on mixed-initiative, which is a typical behaviour property of agent-based systems. Mixed-initiative is defined as a flexible interaction strategy in which each agent (human or computer) contributes what it is best suited to at the most appropriate time [All99, Hor99]. In this way, at every interaction step in a dialogue, the agent who currently knows best how to proceed coordinates the other agent, by taking over the initiative and thus setting a new dialogue goal. Through mixed-initiative, the collaborative aspect of dialogues is enabled. Effective collaboration takes place when each participant depends on the other in solving a common goal or in solving a goal more efficiently. It is the synergistic effect of the two problem-solvers working together that makes the collaboration beneficial for both parties [Gui98].

The system's message determined by the reasoning module sometimes needs to be adapted before being finally output. In this case, natural language generation processes are applied, which mainly consist of completing given output templates with the outcomes of the preceding reasoning step. Alternative approaches are based on inferential and abductive processes, such as Reiter and Dale's referring expressions [RD00].

In the remainder of this section, we will briefly give an overview of some conversational agents built upon the technologies mentioned above. For instance, the TRAINS-95 system developed at the University of Rochester is a collaborative mixed-initiative planning tool, supporting a manager to solve routing problems in the transportation domain [All95]. The core of TRAINS-95 is a set of modules performing language understanding and dialogue processing based on speech act theory which activate a domain reasoner module. The parsing mechanisms applied are capable of achieving a result even in the case of incomplete or incorrect sentences [AMRS96]. By the use of a knowledge base, route-planning problems are solved by considering the parameters extracted from the dialogue. The reasoning module is incremental, in the sense that constraints expressed in further dialogue steps are used for modifying the currently achieved routing plan. As a result, very robust natural language understanding has been accomplished, but due to the set of specialized reasoner modules, the system cannot feasibly be ported to other domains.

Further developments at Rochester are aimed at an integrated AI system, called TRIPS (The Rochester Interactive Planning System), concentrating on the transportation and logistics domain [FA99]. TRIPS uses an abstract problem solving model to represent the user and the system contributions to the task to be performed. These contributions are represented as actions operating on problem solving objects (such as objectives, goals, resources, atomic actions, constraints, etc.). Collaboration between user and agent takes place as they work together to achieve a common goal, in a way that takes advantage of their complementary capabilities (such as speed, memory, accuracy, competence, creativity and so on). During problem-solving, initiative may change back and forth between participants depending on which goals the two participants are working on. TRIPS' logical architecture is comprised of a complex network of agents (as shown in figure 2.6) which
communicate by exchanging KQML (the Knowledge Query and Manipulation Language, [CL95]) messages. Each agent has a well-defined linguistic or planning role in the overall dialogue interaction. The modules can be classified into three areas of functionality (i.e. interpretation, behaviour and generation), which correspond to the processing steps of a generic conversational agent discussed above. More details about the architecture can be found in [AFS01]. TRIPS has been applied to several planning scenarios such as the evacuation of the population of regions affected by natural disasters or the control of robots for locating mines in an underwater area.

Conversational agents are also frequently employed in the eLearning domain. For instance, the AUTOTUTOR application developed by the Institute of Intelligent Systems (http://www.iismemphis.org) of the University of Memphis (http://www.memphis.edu/) is a web-based intelligent tutoring system based on extensive analyses of human-to-human tutoring, pedagogical strategies, and conversational discourse [GWHWH99]. AUTOTUTOR supports students’ learning by engaging them in a natural language conversation about a particular subject matter. Its user interface consists of an animated agent that acts as a dialogue partner for the learner and of graphical displays and animations completing the output in case of particular topics (Figure 2.7). A tutoring session is characterized by frequent initiative takeovers since both the student and the tutor pose questions or start new topics of discussion [LGOtTRG02]. The tutor knowledge is contained in glossaries and document corpora which are arranged in a repository of curriculum scripts, representing the contents associated with a question or problem. Besides the ideal answer, each script provides additional information such as a set of keywords, synonyms, misconceptions and corrections, expectations, hints, summaries for shorter answers as well as metainformation addressing the audio-visual behaviour of the
2.2 Taxonomy of conversational systems

tutor while outputting the related utterances. A set of authoring tools has been made available, in order to allow subject domain experts with limited technical and programming skills to add knowledge and content to the system [SAE+03]. By means of a set of computational linguistic modules [COG05], the user utterances are interpreted and the contents are extracted and classified. Based on the theory of speech acts and on a set of processing modules and dynamic storage units, the dialogue management modules of AUTOTUTOR adapt the system’s response behaviour to specific attributes of the student, such as her ability, initiative, verbosity and so on. The student answers to questions are evaluated by applying latent semantic analysis (LSA) [GHO+02, HCL+03], which is a high-dimensional statistical technique for measuring the conceptual similarity of any two pieces of text, such as words, sentences, paragraphs, or lengthier documents. Several evaluations of the AUTOTUTOR system have been carried out, addressing both the technical quality of the computational modules and the success in producing learning gains [GJM+03, GMM+03]. The mean ratings achieved were positive but there is room for improvement in the naturalness and pedagogical effectiveness of the dialogues.

In addition to one-to-one dialogue systems, in which a human actor engages in a dialogue with exactly one conversational agent, approaches to multi-party applications can also be found [TR02, RAB03]. A main characteristic of these systems is the availability of immersive or non-immersive virtual worlds or chatrooms in which several humans (visually represented by their avatars) and virtual characters meet with the aim of performing specific tasks. Mostly, their communication is based on natural language, carried out within appropriate multi-agent architectures [Woo01].
Research on conversational agents carried out in Europe can be mainly found in the context of projects sponsored by public institutions like the European Commission (http://europa.eu.int) and the German Federal Ministry of Education and Research (http://www.bmbf.de). Research institutions like the German Research Center for Artificial Intelligence (http://www.dfki.de) and the French National Institute for Research in Computer Science and Control (http://www.inria.fr) take part in several international and national projects by sometimes establishing competence centres and special interest groups. Worth mentioning are the projects VERBMOBIL (http://verbmobil.dfki.de), concentrating on automatic and mobile translation of spontaneous speech; SMARTKOM (http://www.smartkom.org), focusing on the creation of intelligent user interfaces which accept and support the natural communication style of non-expert computer users; VIRTUAL HUMAN (http://www.virtual-human.org) aiming at an improvement of the credibility and realism of virtual dialogue partners with the focus on animation and graphical representation of synthetic characters; SAFIRA (http://gaiva.inesc.pt/safira), addressing the need to enrich human-computer interaction with an affective and emotional dimension; SONG (http://www.octaga.com/ Song-Web) building portals of the next generation featuring intelligent agents embodied in realistic avatars as new user-friendly interfaces and real-time audiovisual communications; VIP-ADVISOR (http://vip-advisor.fi.upm.es) aiming at a virtual and independent advisor for personal insurance and finance risk management; VICTEC (http://www.victec.org) targeting the emotional and empathic aspects of the learning process, in particular focusing on believable synthetic characters in a virtual environment; DIALOG (http://www.ags.uni-sb.de/~chris/dialog) which investigates flexible natural language tutorial dialogue on mathematical proofs (the project websites provide links and references to publications and reports containing more detailed information).

2.2.3 Spoken and Multi-Modal Dialogue Systems

Besides typed and written text, the interaction means of conversational systems are frequently extended with additional input and output channels defining new application cases and usage opportunities [And03]. In the setting of multimodal interaction (which has become an established area of human-computer interaction) multiple modes of interfacing with a system going beyond the traditional keyboard and monitor display are considered. Such interfaces combine visual modalities (e.g. a display, keyboard and mouse) with voice modality (speech recognition for input, speech synthesis and recorded audio for output) and even other modalities, such as pen-based input or haptic input/output (i.e. the technology which interfaces the user via the sense of touch). Multimodal dialogue systems are characterised by the joint use of a basic input channel conveying the user’s natural language message (i.e. the keyboard or a microphone) and at least one additional input device completing or substituting the pure language input. For instance, the WITAS dialogue system [LBGP03] can be used to control a small autonomous heli-
copter by briefing mission goals and discussing about automatically planned waypoints. The human operator can use a pointing device (such as the mouse) on an interactive map for supporting routing tasks and real-time navigation of the helicopter. As another example, project MIAMM (http://www.miamm.org) deals with tangible (i.e. haptic) interfaces in the field of multimodal dialogue systems accessing multimedia databases. In the context of a testbed scenario, a special device equipped with keys that can actively exert forces to resist or aid the movement of the user’s fingers and provide vibrotactile signals to the fingertip, is used for the selection and exploration of music on a portable device. The technology enables both the creation of tactile sensations on the skin of the user as well as adding the sensation of weight, since active resistance can be provided to the movement of a user’s hand (see Figure 2.8).

![Figure 2.8: A haptic user interface employed in project MIAMM](image)

Particularly relevant for the field is also the work carried out by André and Rehm [RA05a, RA05b]. One of their research prototypes, called GAMBLE, implements a simple game of dice among two human players and an embodied conversational agent. By the use of a camera mounted cup of dice and a speech recognition system, the synthetic player joins the group as if it were a third human player. Complex communicative behaviours carried out by means of both verbal and non-verbal output produce highly emotional situations, leading to an increased realism of the application.

A very frequent multimodal output enhancement of conversational systems consists of a graphical representation by means of a cartoon-like or three-dimensional character.
The effects of anthropomorphism increase, and the understanding of the system messages can be improved if accompanied by gestures and animated emotions. Different techniques can be used, ranging from a simple set of pictures displayed according to the kind of utterance to output, to more complex animations requiring additional visualisation components or widgets. Specific frameworks allowing for the implementation of graphical presentations for conversational agents can also be found, such as the Multimodal Presentation Markup Language (MPML) [ZDI00] based on the Macromedia Flash model (http://www.macromedia.com/flash) or Cassell's Behaviour Expression Animation Toolkit (BEAT) [CVB01].

The use of speech recognition and synthesis technologies mentioned above allows for dialogue applications running on specific platforms which do not provide the standard input and output means of a computer (e.g. keyboard, mouse, display, printer and so on). For instance, telephony-based interfaces allow only for speech input and output. As another example, ticketing machines at bus or train stations often offer a reduced keyboard and limited display capabilities. Furthermore, access to conversational systems can also be extended to physically disabled users. Thus, if natural language based interaction is wished for these applications, speech capabilities are appropriate and preferable.

Conversational systems involving speech input and output are often also called Spoken Dialogue Systems [McT02, NGA+99, DA05].

Several approaches for recognition and synthesis techniques exist and are used in conversational systems. A deep analysis would go far beyond the scope of this work. Nevertheless, a generic overview can be found in [HHH02] and specifically for dialogue systems in [JM00]. In the following, we will concentrate rather on particular aspects of dialogue management in spoken dialogue systems and analyze and discuss some example applications.

Currently available speech recognition systems are still at an early phase of development. Due to specific human voice features such as prosody and intonation completely satisfying results are still not achievable. Better performing recognition systems require extensive and time-consuming training processes, in which the user is asked to feed in samples of spoken text also related to the current noise level of her environment. Even context-based approaches, which try to infer incorrectly recognized sentence fragments from the preceding input or the current topic still do not deliver satisfactory results [MH95]. For this reason, dialogue manager modules of spoken language systems need to extend their processing with an additional step called verification, in which the user is asked to confirm his entered message [Smi97]. In most cases, the recognized user input is displayed for comparison or repeated in the system message. If a misrecognized input prevents the dialogue from continuing (because of ungrammatical or incomplete sentences), the user is asked to repeat the whole message or only the problematic part. In order to avoid frequent interruptions to the conversation, the dialogue manager must evaluate the circumstances in which a verification subdialogue should be initiated. This is normally based on a distance value expressing the difference between the recognized
2.2 Taxonomy of conversational systems

and the *expected* sentence for a given dialogue step [All95]. The comparison is an even harder process in systems allowing for more frequent user-initiative, since an expected sentence cannot always be identified.

In the context of spoken dialogue systems, three main types of dialogue management approaches are identified by current literature [McT02, JM00]. Besides the already discussed agent- (or BDI-) based systems, we have finite state- (or graph-) based systems and frame- (or template-) based systems.

The dialogues produced by state-based systems consist of a sequence of predetermined steps or states. At every dialogue state a system message is generated and the user’s input used for the determination of the next state. The required vocabulary and grammar for each state are specified in advance (i.e. during design phase), thus resulting in a more simplified language understanding task. On the other side, the user’s input is restricted to a predefined set of words or phrases, preventing her from asking questions or introducing new topics. Thus, the initiative is continuously maintained by the system. State-based approaches are mainly suitable for dialogues related to well-structured tasks which are characterised by a unique and recurring sequence of steps. Indeed, corrections and rearrangements are difficult to achieve as they require a restructuring of the underlying graph. As an example for a state-based spoken dialogue system, the NUANCE telephony banking application (http://www.nuance.com), allows callers to carry out transactions over the telephone such as paying a bill and managing a bank account. Each transaction requires a fixed number of values, which are elicited from the caller in a predefined order.

Frame-based approaches are characterised by system questions aiming at filling slots in a frame or template. Once a frame is complete, a dialogue goal is reached and the results of a computation can be presented to the user. The order in which the slots are filled is not fixed, thus allowing the user to input relevant data at any time of the dialogue by her own initiative. A slot-filling operation is carried out by a so called *production rule*, which is responsible for recognizing relevant data from the user input to be used for slots. If not driven by the user’s initiative, the dialogue management has to select the following question out of a given set. This is achieved by considering priorities and evaluating a set of preconditions related to each question. For instance, in the context of a train timetable information system, specific attributes, such as the starting and destination station, the time and date, the kind of preferred train and facilities during the trip (such as the availability of a restaurant, sleeper cabin, etc.) are required for querying a database of train connections and providing an appropriate result. Generic questions such as "When do you want to leave?" aim at the elicitation of required data, but leave the user the freedom of choosing the information to convey. The answer could be restricted to only a date, expressed in terms of a weekday or a day-month-year combination, or could also contain time data. An appropriate production rule is responsible for processing the user’s answer and extracting all provided information. As for the preconditions used for selecting a system output, the question "At what time would you like to leave?" would be
asked only if a preceding user input did not provide the requested data (in other words, only if the related slot is still empty). The same application can be based on several templates to be completed, thus requiring a more effective management of production rules.

A frequently used implementation language for frame-based spoken dialogue systems is VoiceXML (http://www.voicexml.org). It consists of a set of XML-compliant tags whose syntax is described by an XML schema. An application built with VoiceXML is based on a three-tier architecture. A client application (often called the implementation platform) provides facilities for recording the user's audio input within a predetermined detection interval and for the audio output using recorded audio files or a text-to-speech system. The VoiceXML-documents are stored in a document server and processed by a document interpreter according to the recognized user input. VoiceXML-Documents contain forms which define interactions collecting values for a set of field item variables (thus implementing the behaviour of a frame-based dialogue manager). Each field may specify a grammar that defines the allowable inputs for that field. According to the complexity of the grammar used, several fields can be filled from one utterance.

Many research institutions and universities have set up competence centres and research forums dealing with spoken dialogue systems. For instance, a division of the Research Triangle Institute (RTI - http://www.rti.org) called Responsive Virtual Human Technology (RVHT - http://www.rvht.info) has been working since 1998 on a suite of computer technologies called AVATALK [HFG00], whose aim is to create spoken language conversational agents for the learning and marketing domain. Based on a three-tier architecture, involving a dialogue processor, a behaviour engine, and a virtual environment adapter, the technology enables synthetic characters providing "context-specific, content-appropriate behaviour, including dynamic speech, facial expressions, gestures, and body movements, as well as changing emotional states". In the context of predefined scenarios, virtual humans are employed for training and practice purposes providing a realistic learning environment prior to the challenge of actual engagement. Several application cases can be found, such as the JUST-TALK project [FHG02], in which law enforcement personnel learn to recognize and respond appropriately to subject language and behaviours during encounters with the mentally ill and the "Virtual Customer Experience" allowing users to practice customer service interaction skills with a number of different virtual customers and receive feedback from a virtual coach. Other scenarios include the health domain (see Figure 2.9), with applications allowing users to practice administering informed consent [HGS+04] and pediatric medical educators to face instructional and assessment challenges regarding interaction skills [HGK+00].

Researchers at the Massachusetts Institute of Technology (MIT) have founded their own group called Spoken Language Systems (http://www.sls.lcs.mit.edu). They are engaged in various projects about conversational systems all based on a proper set of core technologies. Applications range from a weather information system called JUPITER [Zue00] to an airline flight planning system called MERCURY [Sen02], as well as a
2.3 What is proactivity?

The term "proactivity" was first introduced by the Jewish psychiatrist Viktor Frankl during World War II. In the context of his invented philosophy of logotherapy, which literally means "healing through reason" and focuses on challenging clients to search for meaning in life, Frankl identifies four main human properties, representing the key difference to the strictly instinctive nature of animals: Self-awareness, which leads a

conversational system interfacing to a database of electronic automobile classified ads called WHEELS [MBG+96]. They are all based on an open source architecture called GALAXY [SHL+98], which provides for a client-server based integration of individual core components. Usually, these components are a speech recognition module (called SUMMIT), a natural language interpreter (TINA), a frame-based dialogue manager, a language generator (GENESIS) and a speech synthesis module (ENVOICE). As an optimisation of the verification step, the speech recognition phase produces a ranked list of candidate sentences to be individually parsed. For this aim, a context-free grammar, augmented with syntactic and semantic features is used [Sen92]. The dialogue manager takes over the highest scoring recognition hypothesis which produces a full parse tree and converts it into a semantic frame.
human to isolate himself from his bodily experiences and become aware of the situation as it is; *discretion*, which is used to assess whether what is happening is good or bad; *imagination*, supporting a human in finding other ways in which things could be done, to remove the bad part and to enhance the good part of the situation; and *willpower*, which implements what has been decided, in spite of hardships. As Frankl argues in his book called *Man's Search for Meaning* [Fra70], proactivity refers to the freedom to choose a response to a stimulus, using the four above-mentioned properties, instead of just reacting to the stimulus (see Figure 2.10). The origins of his thesis can be found in Frankl’s own experiences while he was held captive in a concentration camp along with his family members. He realised that the freedom to choose not to feel humiliated and insulted could not be taken away by his torturers. Instead, by being proactive, one can attribute a higher message to the situation, taking the pain away and providing motives for a new beginning.

![Diagram](image)

**Figure 2.10: Viktor Frankl’s definition of Proactivity**

Even if not contained in most dictionaries, the term proactivity is nowadays frequently used in the context of management literature. Proactivity is the first of the seven habits of highly effective people, the title of a popular book by Stephen Covey providing a useful and sequential framework for understanding much about the process of personal development [Cov89]. According to Covey, being proactive means focusing efforts and attentions on the long-term and to think in terms of the long-term consequences of actions. In contrast to a reactive behaviour, which is characterised by a resignation to the truth that whatever is done in the present can have no effect on circumstances and chances, proactivity is based on the conviction that there are always choices and that every effort is worthwhile for influencing the future. Covey moves from Frankl’s theory for motivating the human’s
need and duty of taking the initiative. In a passage of his book, he argues that "Our basic nature is to act, and not be acted upon. As well as enabling us to choose our response to particular circumstances, this empowers us to create circumstances". Taking the initiative means "recognizing our responsibility to make things happen".

In the context of computer science, proactivity is mainly used to define a property of software agents. Given the variety of different agent definitions, several interpretations of proactivity can be found. They are mostly related to the specific application fields in which agents are employed. Probably the most generic definition can be found in the frame of the "weak" notion of agency. As stated by Wooldridge and Jennings, by means of pro-activeness

"agents do not simply act in response to their environment, they are able to exhibit goal-directed behaviour by taking the initiative." [WJ95b]

The focus is clearly put on the agent’s initiative in pursuing specific goals. For other authors, proactivity is the key property differentiating software agents from other technologies such as object oriented programming or direct-manipulative interaction frameworks. For instance, Negroponte claims that

"Proactivity, in the context of software agents, is basically the spontaneous creating of subgoals that belong to larger task delegated by a human user. It is one of the core questions in the ongoing paradigm shift from direct manipulation to delegation." [Neg97]

Similarly, O’Connel legitimises the intelligence of software agents by describing proactivity as follows:

"The common paradigm for human-computer interaction is that the user issues commands and the program obeys. However, the autonomous intelligent agent can operate proactively, both within the scope of goals the user has expressed and in pursuit of goals unknown to the user which support his stipulated goals. So sometimes, without first consulting the user, the agent proactively performs actions to achieve goals that will benefit the user. The agent does this independently as the opportunity arises. It can proceed proactively both in response to new information and without new information." [O’C00]

Specific applications of software agents lead to even more specific definitions of proactivity. For instance, the JITIR (just in time information retrieval) agents introduced by Rhodes and Maes are information agents which focus on the discovery and presentation of information by means of IR-techniques. Their definition of proactivity reflects the usefulness of JITIR agents for information seeking users:
"Search engines and structured knowledge bases such as Yahoo! are inherently interactive: an information seeker has some query in mind and directly interacts with the program to obtain the desired information. JITIR agents, in contrast, are proactive. The user need not have a query in mind, or even know that information relevant to his or her situation exists. This proactivity has ramifications for the information retrieval techniques that can be used, because the query utilized is limited to what can be sensed in the environment. It also has ramifications for interface design, because proactively displayed information can be far more distracting than requested information." [RM00]

Agents applied in the robotics and cybernetics domain are proactive as they autonomously select their actions and general behaviour by sensing their environment. Indeed, Pack, Wilkes, and Kawamura refer to proactivity by stating that:

"Agents act locally, based on their own internal state, resources and observations of other agents through specific relationships. The core of the agent's operation is a decision or action selection process." [PWK97]

Finally, agents specifically applied in the problem solving domain, exhibit proactivity as they are:

"able to foresee a problem and act before it happens." [Bra97]

The different views on proactivity thus far analysed may converge to a definition in the context of conversational systems:

A conversational system exhibits a proactive behaviour if it is able to take the dialogue initiative during a conversation.

By system dialogue initiative we mean the output of dialogue contributions which initiate a new subdialogue. Its aim is a different goal, with respect to the one of the preceding user input. The previous goal is either completely dropped or restored after the achievement of the new one. Relations between the older and the newer goal may exist. For instance a newer goal may be used for improving the chances of achieving the substituted one, or it may be an alternative goal, provided that evidence for the preferability or impossibility of reaching the original one is known.

But why should a conversational system be proactive and what are the reasons for initiating a new subdialogue? An answer to this question will be discussed in the following chapter. Here, we will briefly look at the purposes for taking the initiative in human to human conversations. Without the assumption of completeness, the following cases can be identified. The initiative is sometimes taken for making suggestions and recommendations, as for instance in the case of a shop assistant, who tries to interpret the client's preferences and offers her a potentially suitable product. Initiative can also be taken for
2.3 What is proactivity?

influencing and convincing the counterpart, i.e. a shop assistant may introduce an additional product, which may be more profitable than the one already chosen. In the case of a customer insisting on apparent flaws of a chosen product, the seller may be interested in changing the topic of the conversation in order to minimise the risk of a sudden shift of opinion. Thus, the initiative can be sometimes taken for catching a discussion going in a wrong or risky direction. Finally, a frequent and efficient case in which the initiative is taken consists of anticipating problems and wishes. A shop assistant may recognize and address specific requirements which the client has deliberately or subconsciously not yet mentioned. In all cases, the decision of taking the initiative is motivated by the recognition of a problem, to be solved in the context of the following dialogue steps. Mostly, also the counterpart takes an active role for its solution, by answering questions or placing counterquestions, or simply by expressing acknowledgement or rejection of the preceding utterances.

In the following chapter, we will analyse and classify the circumstances in which a synthetic conversational partner may identify a suitable point in time for being proactive.
3 Solving Dialogue Problems with Proactivity

The analysis of the theoretical and practical results achieved so far may lead to the specification of a set of requirements for a proactive conversational system. First of all, employing techniques from computational linguistics, additionally based on a predefined semantic grammar, may place restrictions on portability and reuse issues of the intended model. The utilisation of a pattern-matching based approach allows us to keep the complexity at a reasonable level, nevertheless enabling us to address a wider application range in the context of different scenarios and domains. Second, a goal-oriented behaviour which enables user guidance and advanced problem solving can only be reached by employing dialogue management plans, which need to be based on a grammar or dialogue model. Third, the triggering of external reasoners or expert systems has to be made available, both in direct response to a user input and as a consequence of the system’s dialogue initiative. For an improvement of the overall system’s knowledge, data repositories and efficient access methods need to be defined, as it would become particularly cumbersome to code all the knowledge within dialogue rules. For their access, contextual information including the history of the dialogue and user attributes and properties need to be involved in the dialogue management. Finally, the multi-modal improvement of a conversational system by means of enhanced technologies such as speech recognition and synthesis or haptic input/output may not be advisable at present. These technologies are not yet sufficiently mature, resulting in increased difficulty for other tasks involved such as language interpretation and generation.

Before providing a possible conceptual model and architecture addressing these requirements, a deeper specification of proactivity in conversational systems will be discussed. The aim is to identify a set of problem classes which require a systematic approach for their solution. Two case studies of conversational systems addressing the identified problem classes will be introduced. They both meet the requirements mentioned above, thus falling into the category of chatterbot systems employing pattern matching or shallow parsing based goal orientated dialogue management with plans. The systems access external information sources and make extensive use of collected contextual information [LT03]. One of the observed applications includes the option to use speech processing features for the input and output of text messages. It can be turned off during interaction, without affecting the application’s aims and functionality.
3.1 A Taxonomy of Dialogue Problems

The essential task of conversational systems consists of providing a suitable response to the user's utterance. Mainly, this is achieved by the application of a dialogue rule, and is strictly based on the last input sentence. Most of the existing chatterbot systems are designed with the aim of reaching only this kind of goal. Their success is measured upon the ability to interpret the last user input and their power relies on the size of the rule base: the more keywords can be recognized and processed, the more appropriate answers can be generated. In case of no rule to apply, standard sentences are output, such as "I don’t understand you" or "please rephrase your last sentence".

On the other hand, conversational systems which do not simply react in response to their environment, but are instead able to exhibit a goal-directed behaviour, can have a more widespread range of use. Applications are more efficient and powerful if they address overarching goals and use persuasive skills during the dialogue. While aiming at a given goal, several problems may arise, which cannot be solved by the unique application of a dialogue rule. Solutions often require an adaptation of the interaction flow, since details need to be collected and intermediate steps have to be presented and evaluated. Moreover, problems arising can be explicitly mentioned by the user or autonomously recognized by the system. Their solutions can also be either directly requested by the user or initiated by the system, as a consequence of an underlying proactive behaviour.

These considerations gain even more in importance if we apply the technology to a specific domain. For instance, within the e-commerce domain, advertisements and product offers can have a more positive impact if carried out proactively. Providing an offer of the most suitable product at the most suitable time and with the most convenient conditions has always been a high priority within economic contexts. But this can only be fully achieved if the application is capable of identifying and reaching a complex goal or solution to a problem by its own initiative, i.e. not as a consequence of an explicit request.

Generally, the solution of dialogue problems that have arisen can be represented as an additional goal to reach. Conversely, a goal to reach requires procedures and approaches similar to those used for solving problems. Thus, both concepts are intercommunicable and used as synonyms in the remainder of this thesis.

Problems can be classified according to the following scheme:

- **Problems arising from the last user input.** Particular user requests can only be satisfied by accessing external data sources, such as catalogues of products or continuously updated information boards. A dialogue management rule in this case manages to activate parallel processes searching for a set of possible results. A weighting mechanism provides a ranking, in order to find the most relevant record to deliver to the user. As we will see in the example of a chatterbot acting as a virtual shop assistant (section 3.2) a proactive behaviour can be considered for
improving the quality of the retrieval task, by exploiting any available contextual information to include in the information search.

- **Problems subsisting through the whole dialogue session** or in specific sub-dialogues. The solving of these problems is the main objective of the entire system. The dialogue management rule base is conceived in such a way that the relevant information to be used is elicited from the user during the dialogue, combined and processed by an expert subsystem and then presented as a result at the end of the session. These systems concentrate on a specific domain and are usually able to understand a limited vocabulary of input terms. A virtual advisor discussed in section 3.3, concentrates on the business risk management domain with the aim of taking the user through a Q&A session with predefined questions before producing a risk analysis matrix.

- **Problems which have not yet arisen, but are most likely to occur in the future.** Users of conversational systems are sometimes clustered into categories according to classes of profiles and patterns of usage. By examining a structured recording of the already known dialogue, the system tries to assign a new user to one of the predefined classes and predict his behaviour by considering past dialogue sessions of other users pertaining to the same category. In this way, possible problems can be anticipated and tried to be solved in advance. These problems may range from requests for clarifications to vaguely defined product searches. A discussion about the proactive recognition and solution of anticipated problems is given in section 3.4.

The following sections provide approaches of conversational systems specifically addressing the problem classes introduced. A comparison of the approaches will conclude the chapter.

### 3.2 Product search with a virtual Shop Assistant

The main aim of the chatterbot Susanna, prototypically implemented in the context of the EU project COGITO [TLP+03], is to act as a virtual salesperson and search assistant for the Internet e-commerce site of an international bookselling company. Besides providing for a natural language based interaction, an additional component called "prompter" implements an intelligent search functionality. Depending on the current dialogue rule and context, the prompter checks whether the circumstances are suitable for generating a query to a product database. The set of attributes of the query takes relevant data into account, such as keywords from previous utterances. Thus, it is possible to retrieve not only product information, but also highly relevant meta-information from sources like manuals and FAQs [LT01a]. The prompter employs general search strategies with the aim
of transparently collecting information to also be used for recommending a potentially interesting product [LT01b]. Depending on the chosen search strategy, the query to be generated will consist of a generic template, updated with the values of variables gathered from previously interpreted dialogue sentences. The generated query may lead to unsatisfactory results, e.g. because too few or too many items are retrieved. Therefore, once the system has evaluated the result set, it may be necessary to refine the query in order to enhance the quality of the set of results. For the query refinement task a set of query expansion methods relying on information to be systematically extracted from external sources can be used (see Figure 3.1). A query expansion process consists of

![Diagram of query expansion process]

Figure 3.1: A query expansion model used in project COGITO

an improvement of the criteria used for the specification of a query. This is usually achieved by adding search terms to an already defined query. The intent is to improve precision and/or recall which are well known and commonly used measures to evaluate the performance of information retrieval systems. Precision is the number of retrieved items that were relevant. It is computed as the proportion between the set of the retrieved relevant texts and the set of all texts the system has retrieved for a specific inquiry. Recall is the number of relevant items that were retrieved. It is defined as the proportion between the set of the retrieved relevant texts again and the set of relevant texts for a specific inquiry [GS83]. Expansion mechanisms can be applied several times subsequently until the desired quality of search results is reached. The search terms can be obtained from a variety of information sources. In the model shown in Figure 3.1, three sources are considered, namely the dialogue, the user profile and a thesaurus of terms.

Both user and system utterances are stored within a special kind of log file called Structured Dialogue History (SDH). It is accessed for dialogue optimisation and expa-
sion purposes. For instance, further questions aiming at the elicitation of data used for the refinement of a query can be avoided if similar questions can be found within the dialogue history. Moreover, the information stored in the SDH is usually augmented by a reference to the context in which the utterance originally took place. In this way it is possible to further classify the meaning of a sentence, in order to decide on its relevance to the expansion process (A sentence like "I like Jaguars" can have a different interpretation if pertaining to a context dealing with animals or to a context about the automotive industry).

Personalisation facilities are employed to collect user properties gathered during the dialogues. The user profiles generated are accessed in case of users visiting the site at different times and in different sessions. In this way the recurrence of some basic questions can be avoided, and the dialogue kept more interesting and up-to-date for the user. The profiles may contain both generic user attributes like age, country of provenance and gender, as well as details about their hobbies, activities, likes and dislikes. This information can be used for extending the set of attributes of a query.

Products like books or other kinds of media are usually characterised by a textual description. The most relevant words contained in these descriptions can be clustered according to their relation to the most frequently appearing ones, with the aim of generating a "thesaurus" of terms. In this way it is possible to look up, for a given term, a set of other probably related terms to be used for expanding a query. The user expressing his interest in a result within the proposed list may be also interested in alternative books, retrieved by adding search terms related to the representative keywords of the selected book to the original query. For instance, if a query is intended to retrieve manuals about a given topic expressed by a sequence of terms (e.g. "Unix Shell"), the result quality can be improved by adding also the terms resulting from the intersection of the determined sets of related terms (such as "scripting" or "Linux"). The generation of clusters can be kept more successful if applied to previously defined product categories, in order to be able to assign different relevance values to terms appearing simultaneously in different categories. In the frame of the COGITO project a product thesaurus was built by means of the Layer-Seeds term clustering method, described in [CLTN04, CLTN05].

Additional sources for expansion purposes can also be considered, such as for instance online available portals and data warehouses, ontologies and catalogues of meta-information. Even the user himself/herself can be involved in the expansion process, by means of specific questions aiming at collecting additional details or resolving ambiguous issues. The user is probably the most efficient choice, but also the most costly solution, as it requires her willingness and time to spend on a prolonged dialogue.

An example dialogue presented in the following will elucidate how an interaction between Susanna and a customer can profit from a proactive query expansion process. The user here is visiting the web site for the first time. After an initial "smalltalk" phase, in which some basic properties are collected, the user expresses his intention to search for a book about Java.
Susanna.1: Hello visitor! Welcome to our web store. I am Susanna, who are you?"
User.1: Hello. My name is Peter.
Susanna.2: Nice to meet you Peter. Where are you from?
User.2: I live in London.
Susanna.3: Interesting! I already had some visitors from London in the past. What is your occupation, Peter, if I may ask?
User.3: I am a junior programmer at Dell.
Susanna.4: I see...You’re addicted to lots of bits and bytes, just as I am! Do you like to read in your spare time?
User.4: Yes.
Susanna.5: Then this is the right place for you, Peter! What kind of books do you prefer?
User.5: I like to read about computers and technology.
Susanna.6: Good Choice! My origins and roots can be very inspiring.... Can I help you with something specific, Peter?
User.6: Indeed! I am searching for a book about Java.

The sentence contained in "User.6" is an explicit indicator for an user adopting the browsing search strategy, which is characterised by a lack of requirements as regards document as well as information specificity. The search can be part of a process of needs recognition, of search question formulation and of a learning process, where the strategy consists of acquiring new knowledge or new associations and ideas within a familiar or a new subject domain. The need is usually vaguely defined and, when browsing, user and assistant intuitively explore the stocks for good ideas. Thus, a standard continuation of the dialogue would foresee further system questions aiming at the elicitation of additional details. They would be used for presenting intermediate search results, to be commented on and refined in several subsequent dialogue steps until a desired result is achieved.

Instead, the prompter initiates a proactive step, with the aim of directly presenting a probably suitable recommendation and to avoid going over additional dialogue passages. First, the last user input is transformed to a query, consisting of the terms "book" and "Java". As a first try, the prompter executes the query, getting back a high number of results (about 450 different books). Since this is inappropriate for determining a unique item to recommend, query expansion on the basis of the model discussed above is performed. The first source to consider is the SDH. Particularly, the entry related to the sentence "User.3" can be used, as it provides a reference to a topic addressed during the dialogue, namely the occupational field of "information technology". The
3.3 Risk Management as a persistent dialogue problem

For the proactive solution of problems arising in subdialogues or subsisting during the entire dialogue we consider a virtual personal insurance and finance assistant capable of natural language interaction. It was developed in the framework of another EU-funded project, called VIP-ADVISOR. This personal assistant is specialised in risk management counselling for small and medium enterprises but can be extended towards general insurance counselling for private individuals. The interface supports speech recognition and synthesis in order to make the advisor easier and, thus, more convenient to use. Through online translation mechanisms it is possible to use the advisor in different languages although the knowledge base has been built only in one language - the set up of a good knowledge base is usually a difficult and tedious task.

Based upon an existing static tool (the Risk Manager Online), the user is taken through a Q&A session with predefined questions before producing a risk analysis matrix. The virtual advisor supports the Risk Manager’s functionality by offering zero-level-like support. It consists of capturing user requirements about identifying risks as well as providing help
and expert advice on risks. The standard interaction means of the Risk Manager Online are augmented by the usage of natural language for both input and output. The virtual assistant guides the user during the elicitation of the required data and provides help in unclear or problematic situations. Therefore, the user can always take over the initiative, by asking questions or expressing his uncertainty during the fulfillment of a specific task (Figure 3.2).

![Image of the VIP-Advisor system](image)

Figure 3.2: A screenshot of the VIP-Advisor system

Internally, system and user actions are described by a semantic language based on communicative acts (see section 2.2.2) for representing both the meaning and the intention of an interaction step. Actions performed by means of the pointing device or English sentences exchanged between user and system are all represented by a communicative act, such as "request", "inform", "confirm", "authorise" and so on. All acts are extended by a set of parameters which specify the meaning and contents of the interaction step performed. For instance, the signature for "request" is:

\[
\text{request (type, matter, subject, content)}
\]

This communicative act is used to represent a situation in which one of the speakers requires the other speaker to provide some information. The parameter "type" is used for determining the kind of information requested. Usual values are "data", "explanation" or "comparison". The parameter "matter" specifies what exactly is being requested. For instance, common values are "definition", "process", "feature" or "lastAction". The last two parameters provide a deeper characterisation of the communicative act: "subject" is used for specifying the topic of conversation and "content" refers to the object of the request. Analogously, the communicative act "inform" provides the same signature as "request". It is used when information (not necessarily new in the dialogue) is provided
3.3 Risk Management as a persistent dialogue problem

in a sentence. In the case of a user utterance, the choice of a communicative act, as well as the assignment of values to its relevant parameters is carried out by both analysing the grammatical structure of the sentence and by considering contextual information. For instance, during the advisory process, the user may post the question "What is meant by missing liquidity?". The sentence is indeed a "request" for an "explanation" (typically expressed by the formulation "What is meant by"). Particularly, a "definition" is sought, referring to a "risk" (as possibly defined by the context, i.e. the currently processed interaction step in which identified risk factors are presented), namely "missing liquidity". As a result, the communicative act

request (explanation, definition, risk, missing liquidity)

is generated.

The main aim of the dialogue management components of VIP-Advisor is to control the evolution of the conversation. This is achieved by following the active dialogue strategy, defined within the currently adopted dialogue management plan. A plan can be represented in terms of a decision tree whose nodes stand for the system’s output communicative acts and the edges represent user inputs (also annotated by communicative acts). Therefore, if at a specific state one edge can be applied (i.e. the act annotating it matches the user input) the destination node is set as the new current state, and the related system utterance is outputted. If, instead, no matching edge can be identified, the user utterance is analysed by a Problem Identifier component. Its aim is to check whether the user input can be interpreted as a dialogue problem.

<table>
<thead>
<tr>
<th>Problem Class:</th>
<th>Information Source:</th>
<th>Solution method:</th>
</tr>
</thead>
<tbody>
<tr>
<td>1) concept definition</td>
<td>FAQ-like</td>
<td>Case Based Retrieval</td>
</tr>
<tr>
<td>&quot;what is a fire risk?&quot;</td>
<td>dialogue Cases</td>
<td>Case Based Retrieval</td>
</tr>
<tr>
<td>2) process description</td>
<td>Manual-like</td>
<td>Case Based Retrieval</td>
</tr>
<tr>
<td>&quot;how can I prevent a fire risk?&quot;</td>
<td>dialogue Cases</td>
<td>Case Based Retrieval</td>
</tr>
<tr>
<td>3) concept clarification</td>
<td>Portal, Web</td>
<td>General IR</td>
</tr>
<tr>
<td>&quot;why should I prevent a fire risk?&quot;</td>
<td>manual-like dialogue Cases</td>
<td>Case Based Retrieval</td>
</tr>
<tr>
<td>4) arbitrary knowledge request</td>
<td>User</td>
<td>ask for reformulation</td>
</tr>
</tbody>
</table>

Figure 3.3: Classification of problem definitions

As reported in Figure 3.3, a set of dialogue problem classes can be identified. Concept
definition may consist of a straightforward question from the user seeking a definition e.g., "What is a fire risk?". Process description entails the user asking for explanation of a process, e.g., "How can I prevent a fire risk?". Concept clarification is more complex than the first two problem classes since the solution consists of more than a straightforward sequence of questions and answers. A user may, for instance, ask "Why should I prevent a fire risk?". More information needs to be elicited from the user so that the system can present the user with a process as its answer. An arbitrary knowledge request is not part of the dialogue, for example, a user may enquire as to possible payment modes or ask for the location of an office. The solution is provided by general information retrieval, information sources being either the enterprise portal or the Web. The final case in which the dialogue breaks down completely means that the system has to request reformulation.

Whenever an unexpected dialogue situation occurs, consisting of a problem causing a deviation from the original dialogue plan, a new one has to be selected and temporarily substituted for the no longer valid plan. As a key technique for the selection of a new plan to apply, Case Based Reasoning is used: solutions that were used to solve old problems are adapted for solving new ones [Lea96]. The system relies on a continuously updated set of successful dialogue plans, whose abstractions are indexed by a retrieval engine in order to identify dialogues (i.e. cases) similar to the current situation. The estimation of the suitability of a new plan is based on pragmatic relevance: a dialogue plan is relevant to a certain problem if its application helped solving the problem in similar situations happened before. Cases establish a repository of problems which occurred previously, together with the solutions which were applied to solve the according problem. A problem solution contained in a case was useful (and used) in the past; hence, the more similar the current problem is to the old one, the more likely it is that the old solution will apply [LFTN04].

The initial case base utilized for VIP-ADVISOR contained about 500 different entries [LKT04]. For the collection of the cases, several knowledge sources were used, including the FAQs and glossaries of various insurance and financial web portals as well as interviews carried out with professional advisors. The interviews had the aim of recognizing problems and questions that users normally have during a risk management advisory session. The cases are divided into different categories, according to the topic they refer to. For instance, cases about pertinent laws, risk factors, and insurance types are included, but also FAQs about business sectors and activities, general financial and insurance concepts and explanations of specific processes and functions of the Risk Manager Online tool can be found. The case base was regularly updated by examining the system log files: problems which could not be solved were considered as a new case after the identification of an appropriate solution. As depicted in Figure 3.4, cases are coded using the extensible markup language (XML). The problem definition part consists of a communicative act and a representative user utterance causing the problem. The problem solution part contains the dialogue plan to apply in order to solve the problem. The figure shows a simple problem solution, consisting of a straight answer to the user’s request. The case
of Figure 3.4 would be relevant for the user mentioned previously, asking for the meaning of "missing liquidity". It solves a problem pertaining to the Concept Definition class.

The problem solutions of cases solving problems pertaining to the concept clarification class have a more complex structure than simple FAQ-based cases. Instead of a single answer, they require the processing of an entire dialogue plan before producing a final answer. Within the plan, the system may ask counter questions, evaluate the user’s answer and finally generate a dynamic output, based on the data collected during the processing of the plan. For instance, consider the following user question: "Should I care about a fire risk in my stock?" (coded by the communicative act "request(clarification, feature, risk, fire in stock)"). A direct answer to this question would be inappropriate, as it depends on whether the user’s business is trading with flammable goods or not. Therefore, before delivering a final statement, the system should investigate the inflammability of
the goods. This is achieved by processing the plan contained in Figure 3.5. The actual branch of this plan is processed according to the user’s response to the counter question. For the sake of simplicity in this example we considered a counter question allowing only for a simple answering scheme (e.g. a confirmation or a rejection). This approach can also be applied in the case of counter questions enabling for multiple responses.

![Diagram](image)

Figure 3.5: A case addressing a concept clarification problem

The kind of proactivity considered in the case of the VIP-ADVISOR application addresses the solution of persistent dialogue problems. The dialogues are characterised by a main management plan, which guides the whole interaction towards a given aim. If the plan cannot cope with specific concerns, it is temporarily substituted by a new plan, providing for subdialogues which handle the problematic issue. Their determination is carried out autonomously by the system. Therefore, initiative is taken for searching and applying an alternative dialogue management strategy, instead of simply announcing the inability to fulfil the request. The following section will discuss the solution of not yet occurring but expected problems, with the aim of optimising the dialogue.

### 3.4 Optimising Dialogues with Anticipated Problem Solving

The impact and efficiency of conversational systems are strongly dependent on the number of dialogue steps required for the achievement of the predefined goals. If the same
results can be reached by means of shorter dialogues, both user and system are allowed to save resources and reuse them in other tasks. Goals are characterised by a minimum set of information to collect, required for the computation of a result. The expected duration for their achievement can be shortened during the design phase by implementing optimised management plans and dialogue strategies. Other factors causing delays and interruptions to dialogues are determined by the occurrence of technical and conceptual problems which the user may have in interpreting or understanding the system’s requests. Technical problems can be addressed by employing efficient hardware and robust implementation frameworks. As regards conceptual problems, solving methodologies can be adopted similar to the ones discussed in the previous sections. They still require additional dialogue steps, thus prolonging the conversation. An empirical analysis of dialogue log files may reveal the presence of repetitive problems, exhibiting stereotypical attributes and based on contextual details amenable to clustering. Such problems can be assumed to occur in the case of recurring circumstances. An early detection of these circumstances may lead to dialogue contributions aiming at the avoidance or anticipated solution of the related problem. In this way, interruptions to the dialogue flow can be prevented. The user is thus allowed to concentrate on the main task instead of being engaged with a problem and sometimes costly formulation of its definition. If the anticipated solution can be integrated without altering the original dialogue flow, the user is not distracted even if the problem does not effectively occur.

For an exemplification of these concepts, let’s consider again the shop assistant introduced in section 3.2. The user "Peter", whose IP-address was stored during his previous visits, connects again with Susanna, having the following initial dialogue:

Susanna.1: Nice to meet you again, Peter. I am pleased that you have found your way back to our web site. What can I do for you?
User.1: I'm looking for a book.
Susanna.2: Then this is the right place for you, Peter! We have hundreds of thousands of books in stock. Are you looking for a book by a specific author?
User.2: Yes
Susanna.3: What is the name of the author?
User.4: King
Susanna: I shall check whether we have a book by the author King. Please wait ...

Out of this short piece of dialogue the system recognizes again the user’s willingness to perform a search based on the browsing strategy. As a first step, a query to the product database is generated, consisting of the search for books written by an author named "King". Without further processing the query, the first ranked books of the search result
are by the author of horror literature "Stephen King" (see Figure 3.6), while books by other authors with the same name are found further down the list (the ranking is based on attributes such as popularity or frequency). They are displayed on the main web site and proposed for the user’s attention. If Peter were not looking for a book by Stephen King, he would either scroll down the long result list or proceed in his browsing strategy, by mentioning additional criteria which refine the current query.

Figure 3.6: Searching for books by the author "King"

A proactive expansion of the initial query about books by the author King, may avoid these additional steps, thus abbreviating the dialogue before reaching the aimed goals. Let’s assume that the profile of the user Peter corresponds to the one shown in Figure 3.7. It has been generated by considering his past experiences and actions performed with the online shop and by applying a machine learning approach [SDL+03]. The books offered by Susanna are organised into 10 book categories typically representing a genre or thematical area. A training set of 500 examples, each one representing a different user of the web site, was used with the aim of inferring proper classification rules. For each book category and user, specific transaction properties of past interactions, such as the number of searches, frequency of searches, number of purchases and frequency of purchases, have been considered. The training sets were used for inferring the user’s attitude towards each category, expressed in a coefficient between 0 (dislike) and 1. The table of Figure 3.7, shows that the category Computer and Internet is the favourite one.

The query aiming at books written by the Author "King" is therefore first constrained to Peter’s preferred category, producing the result set displayed in Figure 3.8.

The decision to expand the initial query by the system’s own initiative was based on the assumption that gathered profile information can be used for providing a more
3.4 Optimising Dialogues with Anticipated Problem Solving

Figure 3.7: A COGITO User Profile

personalised service. Using the user's preferred book category for a specification of the search process rather than usual weighting criteria of retrieval systems may be more appropriate for meeting the user's expectation. If this assumption ends in failure (i.e. Peter is this time not looking for computer science literature), the predefined dialogue duration is not affected, since the following steps can still be used for constraining and improving the set of items to be browsed.

By considering transactions carried out in the past, general usage patterns can be generated, and used for predicting the behaviour of users providing similar profiles. In the context of COGITO, a set of usage patterns was generated by applying unsupervised machine learning techniques in order to induce a theory in form of rules from pre-classified previous interactions. A rule has a generic form consisting of a list of criteria - the pattern - and a related goal, which can only be reached if the pattern is matched. Thus, if the current user fulfils most of the criteria of a specific rule, she can be considered to be aiming at the related goal, which is usually expressed in the form of an interest in a product or service. For instance, consider the following lines showing a generated rule represented in an "if-then" form:

If "ComputerUndInternet" == "yes" and "Kinderbuecher" == "no" and "SchuleUndBildung" == "no" then "WissenschaftUndTechnik" := "yes"

It expresses the fact that users liking the "Computers and Internet" category and disliking the categories "Kinderbücher" and "Schule und Bildung" (German translation for "Children's literature" and "School and Education", respectively) are most likely to be interested in the category "Wissenschaft und Technik" (i.e. "Science and Technology"). Since this rule can be applied to the profile contained in Figure 3.7, the query generated for the user Peter can also be expanded with the category "Science and Technology", providing a different set of results.
Within systems employing plan-based dialogue management, it is even likelier to identify recurring problems. Indeed, since they are based on the same main plan, dialogues with different users are similar in their structure, and differ only with respect to introduced properties or arisen problems. For instance, the dialogue steps of the risk management application described in section 3.3 always refer to the same sequence of questions, which is not interrupted if no problems occur. Specific content provided by the user’s answers may produce deviations and alternative paths in the management plan, but they always converge to the same final dialogue steps, producing the risk analysis matrix. Plan instances can be grouped together with the aim of forming clusters of users exhibiting the same properties. Some of these clusters may be characterised by the occurrence of a problem at the same dialogue step. It can be assumed that a new user pertaining to this cluster may face the identical problem which had occurred previously. Thus, it may be appropriate to provide an anticipated solution some steps before the problem usually occurs, even more if it obviates additional dialogue steps (such as at least the one required for expressing and defining the problem). A temporarily applied plan, sometimes producing a subdialogue, may be required for the solution of the problem. In this case, the dialogue management needs to be extended with an anticipated, and thus proactive, plan substitution strategy, like the one described in the following.

The ongoing dialogue is monitored in real time by examining its extended log file. When enough properties for determining the user’s classification in terms of a cluster are available, an initial set of alternative plans is selected and processed in the background by using the actual user input. As the dialogue moves on, the set is refined since additional user properties are known. By examining the dialogue context, an appropriate point
of time may be identified for exchanging the current plan with a new one following an alternative strategy. This process can be reversed whenever an utterance expressing a rejection of the proposed alternative is identified. In this case the original plan is applied again. Figure 3.9 sketches the procedure described so far.

![Diagram of the proactive switching of plans](image)

**Figure 3.9: Proactive switching of plans**

The thick line in the figure represents the flow of the current dialogue. Plan 1 guides the dialogue and after every step $i$ the context is determined. It contains both static profile information about the user ($P_n$) as well as dynamic information describing the current dialogue step ($T_n$). The updated context is continuously monitored and compared with the ones from the alternative plans. In case of a correspondence the control of the dialogue switches from plan 1 to the plan having the corresponding context. Step $n+1$ of plan 2 may contain the solution to a problem which may be raised by the user in some further steps. After step $n+1$ of plan 2 the control of the dialogue is passed back to the original plan 1.

For instance, the concept clarification problem discussed at the end of the previous section may be observed to occur in a given class of users and during a specific dialogue phase. In the context of a conversation with a new user pertaining to this cluster, the plan handling the solution can be applied before entering the risk discussion phase. By posing the question about the trade with inflammable goods, the user’s answer can be exploited to provide the adapted clarification during the presentation of the identified risk factors. The problem is therefore solved even before its possible realisation.
3.5 A comparison of the case studies

The two applications analysed so far offer a system user interface, based on the natural language interaction paradigm. In addition to the capability of being reactive to the users’ requests, by handling their input appropriately and providing suitable responses, both systems adopt proactive behaviours for improving problem solving efficiency and therefore user satisfaction in the interaction process. The common aim is an enhanced cooperation offered by the system while fulfilling the tasks. Users feel more guided and heed as the responses received appear more accurate and relevant in meeting their expectations. While the shop assistant concentrates on more locally arising problems, the support through case-based reasoning of the virtual advisor has a wider scope, affecting upcoming stages of the dialogue. Indeed, the new plan achieved may be extended for several steps, including subdialogues and alternative paths towards the tasks to reach. In the case of clarification subdialogues, it is more prohibitive to decide whether the new plan is more appropriate than the previous one, especially in the case of the system trying to forecast problems. This entails augmenting the system’s transparency, communicating to the user that a change of the plan has happened, and that she could easily go back to the original one without losing the trail. On the other hand, product offers can be evaluated immediately, as the user can provide direct feedback expressing his acceptance or denial of the proposed solutions.

A different dialogue management approach characterises the two applications. The assistant used for the bookselling domain relies on a structured organisation of the rule base, allowing for a broader choice of the topics of discussion. The dialogue planning task is implemented by constraining the application of some rules to the currently adopted search strategy. Dialogues with the insurance advisor follow predefined plans, restricting the user’s freedom to change the topic of discourse, but allowing for a more efficient steering of the conversation. This diversity is required because of the different nature of the support offered by the assistants in their respective systems: the cooperation achieved by the virtual advisor refers to the usage of the Risk Manager tool, which encompasses a predefined sequence of interaction steps. The initiative has to remain on the system’s side, as the user may not deliberately change the order of the tasks to perform. In the product search domain, the user has to elaborate her own strategy in terms of dialogue requests for achieving her goals and therefore needs to frequently take the initiative for formulating aims and desires. As a result, the kind of proactivity offered by a conversational system has to take into account which side of the interaction process carries the burden of a dialogue leader: if it stays on the system’s side, proactive planning may be helpful, but if it stays on the user’s side, proactivity can only enhance the quality of the system’s responses in short-term objectives.

The general effectiveness of their proactive behaviour is achieved by the use of the same underlying technology. The area of Information Retrieval provides a robust theoretical framework to rely on as well as widely accepted criteria for the quality assessment of
search results. Performance issues can be overlooked, as the field’s maturity already provides enough experience and solutions. The systems need only to identify the correct point of time at which a retrieval step is necessary and accurately specify the query to be posed by including available contextual information. The underlying retrieval engine as well as its weighting mechanisms will do the rest of the job, extracting the most relevant record out of the available information corpus, even if the contained data is not carefully prepared or formatted.

The following chapter will provide a formalisation of the pattern-matching technology employed by chatterbots. Three successively applied extensions will address the modelling of a proactive behaviour, required for solving the problem classes introduced above. The implementation of dialogue management plans as well as a strategy for their dynamic switching will be achieved by the introduction of an effective structuring schema for pattern-matching rules. By means of an information retrieval based access to external repositories, the solution of problems which have arisen locally will be made possible, also by enabling predefined query expansion models. Finally, a proactive consideration of alternative dialogue flows will be made possible by introducing a dialogue monitoring process, which uses heuristics for the suggestion of different system reactions to a given user input. The aim is to enhance cooperativity and provide anticipated solving of dialogue problems.
4 Conceptual Model and main approach

4.1 Introduction

A typical conversation between a human and a chatterbot system always has a sequential nature. Both actors have a dedicated time slot at their disposal in which their contribution to the dialogue is put together and then submitted to the counterpart. Even if humans are to some extent able to synchronously listen to a sentence and output some words, chatterbots aren’t. Therefore, a dialogue with a chatterbot system can be more appropriately compared to one-directional communication means such as walkie-talkies rather than bi-directional means like a telephone or a face-to-face conversation. In the following, the basic steps undertaken by a chatterbot during a dialogue are explained (see a summary in Figure 4.1). After outputting an utterance and possibly updating other output channels such as its visual representation or an underlying website, a Chatterbot moves into a "waiting" state, in which the user is in charge of communicating his message. This is usually done via a keyboard or other input means such as a microphone. By

![Diagram of chatterbot steps](image)

Figure 4.1: Basic steps undertaken by a chatterbot

confirming his input the user initiates the chatterbot’s interpretation process in which an applicable rule is selected. For this aim, the input sentence is compared to all known patterns of the rule base and in case of a matching pattern the actions defined by the related rule are performed. These actions include the generation of the output text and possible updates of other output means. If no match can be found, a standard system response, asking the user to reformulate his input, is generated. By moving back to step one, the generated system response is output thus waiting again for the user’s turn. This process is repeated until the user decides to interrupt the conversation.
A further analysis of the steps identified so far leads to a comparison of chatterbots with finite state machines. Indeed, steps one and two describe the typical behaviour of a "state", in which an output is produced and the machine is halted until a new input is available. Steps three, four and five, implement the "transition" to a new state, by means of the application of an appropriate rule. The aim of this chapter is to provide a formal definition of proactive chatterbots based on automata theory. By starting with the description of standard chatterbots in terms of a deterministic finite state machine the requirements for proactivity identified in the previous chapter are applied stepwise. Figure 4.2 contains a graphical representation of the enhancements applied at every step. Circles represent state-of-the-art available chatterbot technology, whereas boxes stand for the enhancements applied in this chapter. Extensions with a darker background refer to improvements of more content-specific aspects of the system responses, whereas extensions with a lighter background colour refer to enhancements of the dialogue management and control. Basic chatterbots are first extended by the capability of managing elementary storage locations such as variables and predicates. Although this kind of extension is already commonly used in current chatterbot implementations, we introduce it for clarifying our decision to adopt a more specific kind of finite state machines in our arguments, namely augmented transition networks. The second enhancement refers to the chatterbot’s ability to pursue dialogue goals during conversations. For this aim, a model of dialogue planning will be introduced. Instantiations of the model, i.e. the dialogue plans, are used as guidelines for a goal-oriented structuring of the rule base. In the context of the third extension, external data sources are accessed for improving the contents of the chatterbot’s responses. Finally, the fourth enhancement aims at

Figure 4.2: Extending basic chatterbots towards proactivity
achieving a more flexible dialogue behaviour based on the system’s initiative to select an alternative dialogue goal whenever appropriate. In order to guarantee the feasibility of each applied enhancement the definitions produced at every intermediate step are still based on finite state machines. Since the finally achieved proactive chatterbot is defined as a non-deterministic finite state machine, a proposed implementation methodology is discussed (see also [LTK05]).

4.2 A stepwise refinement of chatterbots towards proactivity

4.2.1 Data Types

Before being able to provide a complete definition of chatterbots it is necessary to first introduce the basic data types used. The aim is a specification of the functions describing the input and output behaviour in terms of their domain and range. Basically, the object of the interaction between a chatterbot and its user is natural language. Nevertheless, the communication is not constrained to an individual language or to a predefined grammar. Since the usual input means is a keyboard, the user is allowed to also enter grammatically and syntactically incorrect sentences or meaningless texts. Therefore, every combination of characters which can be internally represented as ASCII code are allowed for both input and output. A first set of standard data types can now be defined as follows:

- **integer**
  - domain: *the set of integers*
  - operations: +, −, ∗, /, mod, div, etc.: integer × integer → integer

- **real**
  - domain: *the set of real numbers*
  - operations: +, −, ∗, /, etc.: real × real → real

- **boolean**
  - domain: {true, false}
  - operations: and, or: boolean × boolean → boolean
  - not: boolean → boolean

- **character**
  - domain: *every symbol of the ASCII table*

- **string**
  - domain: *every combination of character*
  - operations: concatenate: string × string → string
  - compare: string × string → boolean
The data type string provides two operations. Concatenate is used for merging two strings into a new one. In this way, as we will see in the following sections, chatterbots are able to construct their responses at run time. The operation compare is used for assessing if two strings contain exactly the same sequence of characters. For this aim, the data type boolean, which provides the usual logical operators, had to be included. In the rest of this chapter other data types will be introduced and defined where needed.

4.2.2 Basic Chatterbots

The knowledge base of standard chatterbots contains a finite number of rules, resulting in a finite number of different responses. As discussed above, every different response can be considered as a state, in which a user input is expected after the output process. Thus, all states considered as a whole form a finite set, which is a prerequisite for a finite state machine. Moreover, if there is no rule to apply to a specific user input, a standardised answer is given, inviting the user to reformulate his sentence. This solution guarantees the determinism of a chatterbot allowing us to formulate the following:

Definition 1: A chatterbot $C$ is a deterministic finite state machine, described by the tuple

$$C = (X, Y, Z, \alpha, u, g)$$

in which:
- $X$ is the input alphabet,
- $Y$ is the output alphabet,
- $Z$ is a set of states,
- $\alpha \in Z$ is the starting state,
- $u$ is the transition function, and
- $g$ is the output function.

Furthermore:
- $X, Y \neq \emptyset$;
- $u$: $X \rightarrow Z$;
- $g$: $Z \rightarrow Y$;
- $\forall x \in X$ : the data type of $x$ is string
- $\forall y \in Y$ : the data type of $y$ is string

The transition function $u$ implements the if-part of a chatterbot rule as it analyses the user input and determines how to proceed. It can be described by a table representing a mapping between all possible input patterns and the related states, which are responsible for the output of the system reaction (see Figure 4.3). At the beginning of
a new conversation, the system automatically sets the input pattern to "CONNECT" to move the chatterbot into the starting state \( \alpha \). If the user inputs the string "Hello" at any time during the conversation, the chatterbot will move into state \( z_1 \). Similarly, the input "Who are you?" will lead the chatterbot to state \( z_2 \). Any other input pattern will produce a transition to state \( z_3 \). The task of analysing a text with the aim of finding the occurrence of a given pattern is a well-known problem in computer science often referred to as the "string matching problem". Several implementations and algorithms can be found in the related literature, offering different performance values in terms of the time needed to find a mapping and the preprocessing time, needed for setting up the system before starting the search. Four interesting approaches for string matching problems are discussed in [CLR97]. One of them is based on a finite automaton, featuring accepting states which are reached in the case of a successful mapping. Such an automaton can be easily integrated as part of a finite state machine describing a chatterbot, thus implementing the function \( \sigma \). In order to avoid increasing the level of complexity, the finite state machines considered here won’t be extended with implementations of the string matching problem. The algorithms used for string matching purposes can also be adapted for solving possible conflicts as in the case of two or more rules reacting to the same pattern. For instance priorities can be set or rules marked as "processed" in order to avoid a repeated response in the case of the same user input during a dialogue.

The output function \( \gamma \) implements the then-part of a chatterbot rule, consisting of the communication of a system response to the user. This is usually done by printing out a string or a sentence on the standard system output channel, such as the CRT or the loudspeakers. The string to output is defined by the state in which the automaton currently is. Therefore, the output alphabet of a basic chatterbot always contains a finite number of elements which corresponds at most to the number of states (i.e. \( |Y| \leq |Z| \)). Since function \( \gamma \) implements a one-to-one mapping between the states of \( Z \) and the strings contained in \( Y \), a table can be drawn, mapping all states to output strings (Figure 4.4).
### Figure 4.4: An example output function

By merging the tables describing the transition function and the output function into a new table, the complete behaviour of a chatterbot can be summarised (see Figure 4.5).

### Figure 4.5: A Table describing an example Chatterbot

Alternatively, finite state machines are often depicted by state-transition diagrams, in which circles are used to represent the states and directed edges are used for the representation of the transitions. Figure 4.6 contains the state-transition diagram of the chatterbot introduced in the example above. The starting state $\alpha$ produces the output of the greeting message "Hello there!". No edges lead back to $\alpha$, which means that this output won't be given again in the same conversation (the pattern "CONNECT" is only considered as a system instruction for starting the conversation). Instead, three edges leave the state $\alpha$. They refer to the three input alternatives defined by the transition function table, namely the string "Hello", the string "Who are you", and the placeholder UNDEF, which stands for all other possible strings. Depending on the input sentence, one of the states $z_1$, $z_2$ or $z_3$ is reached, and the related system response is output. Once again, by means of three leaving edges, the next user input can be processed and the following state determined. If the user repeats his input during the conversation, the same rule is applied again, producing the same transition and thus the same output. This behaviour is represented in the diagram of Figure 4.6 by the edges leading back to the state in which the transition originated.
A possible conversation between a user and the example chatterbot described so far is reported in the following:

**Chatterbot (C):** "Hello There!"

**User (U):** "Hello"

**C:** "Pleased to meet you!"

**U:** "I’m pleased too"

**C:** "Pardon me?"

**U:** "What are you?"

---

Figure 4.6: A state-transition diagram of a standard chatterbot
C: "Pardon me?"

U: "Who are you?"

C: "I am a chatterbot"

U: "What is a chatterbot?"

C: "Pardon me?"

The input patterns "I'm pleased too", "What are you?" and "What is a chatterbot?" in the example dialogue above are treated as unrecognized input (UNDEF), since no specific transitions (i.e. rules reacting upon these patterns) are provided. Obviously, the quality of conversations with chatterbots strongly depends on the number of different input patterns that can be processed. Nevertheless, a complete anticipation at design time of all patterns that may be input during a conversation is an impossible task, mainly because of the infinite number of sentence formulations that can be achieved with natural language. As a trade off, most chatterbot implementation frameworks allow the usage of wildcards or placeholders in the specification of a pattern to be recognized. For instance, the pattern " * are you?" contains the wildcard "**" which can be substituted by any possible string. In this way, one single rule or transaction can be implemented for reacting to both the sentences "What are you?" and "Who are you?". Of course, as a collateral effect, other values of the wildcard "***" may also lead to the same response, such as "How are you?", "Where are you?", and so on. The answer "I am a chatterbot" may not be appropriate in these cases. Additional rules need to be implemented and a prioritisation mechanism must be available for handling rules accepting patterns without wildcards first. Therefore, placeholders can be considered as a powerful but at the same time dangerous feature, to be handled with care. As another example for efficient wildcard usage consider the case in which a sentence may contain words whose meaning does not influence the response to give. For instance, for the pattern "My favourite movie is *" a rule may be implemented which produces the chatterbot response "I like it too". Alternatively, the answer "I don't know that movie" could be provided, and rules added for movies which the rule designer prefers to let the chatterbot comment on differently. By considering wildcards we can now extend the basic data type string to the data type pattern, which is defined as a string containing an arbitrary number of wildcards.

- **wildcard**
  - domain: *metacharacters which are used in place of other characters or strings*
  - operations: *valueOf: wildcard → string*
4.2 A stepwise refinement of chatterbots towards proactivity

- **pattern**
  - domain: *every combination of strings and wildcards*

The data type *wildcard* provides the operation *valueOf* for determining the actual string input by the user in place of the wildcard (including the empty string).

Another interesting application of wildcards is the reuse of the words they stand for in the immediately following response. Consider for instance the pattern "My name is *" and the related system output "Pleased to meet you, *". Of course, chatterbots pertaining to definition 1 are not able to output system responses constructed at run time. Indeed, the output function $g$ can only print out strings as they are specified in the current state. The concatenation of strings with the values of wildcards can be carried out during a transition, in order to let the output function carry out its usual job when the destination state is reached. For this aim, a new data type has to be introduced, allowing for the output of strings to be completed at run time.

- **template**
  - domain: *output strings containing an arbitrary number of wildcards*
  - operations: *substitute: template $\times$ wildcard $\times$ string $\rightarrow$ template*

Also the states of a finite automaton can be considered as a basic data type. They support the operation $o$, which delivers the output template associated with a specific state.

- **state**
  - domain: *the finite set of states of a finite automaton*
  - operations: $o$: *state $\rightarrow$ template*

The following chapter introduces the definition of chatterbots in terms of augmented transition networks, which are a variant of finite state machines, allowing for various actions to be performed during transitions.

### 4.2.3 Chatterbots handling variables

A specific aspect of programming languages is the ability of storing values in dedicated memory locations to be used and modified in further processing steps. The usage of variables may come in handy also for chatterbots. Indeed, a first elementary personalisation can be achieved by "remembering" and reusing specific user aspects learned during the dialogue. For instance, as already shown in the example at the end of the previous section, patterns such as "My name is *", "I am * years old" or "My preferred colour is *", can be considered as containers for user properties to be possibly exploited during the rest of the conversation. Even if a semantic analysis is not performed, the meaning of the wildcard can be assumed, based on the rest of the pattern containing it. Of course, rules
must be provided, reacting to specific cases in which the explicit part of a pattern may be used for other aims. In the name example, patterns such as "My name is unimportant", "My name is secret", and so on should be processed by appropriate rules for avoiding addressing the user with a wrong name.

The learned user properties can be stored in variables and used for the generation of system responses. Their values can be either used as constituents of the output or in comparisons and conditional statements. For instance, if the pattern "I am a male" or the pattern "I am a female" could be recognized, a specific variable called "gender" may be set accordingly and inspected to provide more gender-related system responses during the rest of the dialogue (questions like "Do you prefer wearing skirts or trousers?" could be avoided in conversations with a male user...). Thus, a new data type variable can be introduced. As in the case of wildcards, variables support the operation valueOf, which is used to determine the value contained by a variable:

- variable
  domain: storage locations with a name that can be assigned a value
  operations: valueOf: variable → string
               =: variable × string → variable (used for assigning values to variables)

To allow chatterbots to deal with variables, an extension of the definition provided in the previous section is needed. Indeed, simple finite state machines do not provide enough processing facilities. Instead, augmented transition networks (ATNs) seem to be more appropriate. This is described in the following:

Definition 1.1: A chatterbot $C_{var}$ is an augmented transition network, described by the tuple

$$C_{var} = (X, Y, Z, V, \alpha, u, g)$$

in which:
- $V$ is a set of variables (possibly empty),
and
- $\forall v \in V :$ the data type of $v$ is variable

Everythihg else corresponds to Definition 1.

Besides allowing for variables, in ATNs each transition can have associated with it also some actions to be performed during the transition. The formal definition of an ATN basically corresponds to the one of a finite state machine. Merely the actions have to be specified separately for each transition. This can be done for instance in the context of the table used to describe a finite state machine. A column can be added providing entries only for transitions in which specific actions apply. The actions may correspond
to the operations defined for the basic data types. For instance, the table displayed in Figure 4.7, foresees the execution of two actions in the case of a user input corresponding to the pattern "My name is \( *_p \)" and leading to state \( z_3 \).

<table>
<thead>
<tr>
<th>Pattern</th>
<th>State</th>
<th>Actions</th>
<th>Output</th>
</tr>
</thead>
<tbody>
<tr>
<td>CONNECT</td>
<td>( z_0 )</td>
<td></td>
<td>&quot;Hello there!&quot;</td>
</tr>
<tr>
<td>&quot;Hello&quot;</td>
<td>( z_1 )</td>
<td></td>
<td>&quot;Pleased to meet you!&quot;</td>
</tr>
<tr>
<td>&quot;Who are you?&quot;</td>
<td>( z_2 )</td>
<td></td>
<td>&quot;I am a chatterbot.&quot;</td>
</tr>
<tr>
<td>&quot;My name is ( ^*_p ).&quot;</td>
<td>( z_3 )</td>
<td>user_name:=valueOf(&quot;p&quot;); substitute(o(z3), ( ^*_t ), valueOf(user_name));</td>
<td>&quot;Pleased to meet you, ( ^*_t ).&quot;</td>
</tr>
<tr>
<td>&quot;I am a ( ^*_p ).&quot;</td>
<td>( z_4 )</td>
<td>substitute(o(z4), ( ^<em>_t ), ifThenElse( ( ^</em>_t )</td>
<td>&quot;I always love to chat with a ( ^*_t ) like you&quot;</td>
</tr>
<tr>
<td>UNDEF</td>
<td>( z_u )</td>
<td></td>
<td>&quot;Pardon me?&quot;</td>
</tr>
</tbody>
</table>

Figure 4.7: A table describing an example chatterbot managing variables

The first action is responsible for storing the value of the wildcard \( *_p \) in a variable called user_name. The second action performs a substitution of the wildcard \( *_t \) of the output template of state \( z_3 \) with the value of variable user_name.

Other actions can also be specifically introduced when a chatterbot is implemented, thus extending the set of operations defined for the basic data types. This is the case of the action ifThenElse used in another example in Figure 4.7:

\[
\text{ifThenElse: boolean \times string \times string \rightarrow string}
\]

Like an if-then-else statement of a standard programming language, it returns the value of the first or second string depending on the boolean value of the first parameter. In the example, the input "I am a \( *_p \)" produces a transition to state \( z_4 \). The system response is constructed by substituting the wildcard in the output pattern with the result of a comparison action. If the user had entered the string "I am a female", the comparison would have returned the string 'girl' leading to the final output "I always love to chat with a girl like you". In the example, the else-part of the comparison returns the contents of the wildcard of the input pattern, but it could have been substituted with another comparison (returning for instance the string "boy" in case of the user input "I am a male").

Generally, the complexity of chatterbots increases with the number of recognizable
patterns and thus related states providing responses. But even if a higher number of rules enables the chatterbot to "understand" more natural language, true dialogues extending over several steps are still not possible. Indeed, a standard chatterbot is not capable of "remembering" the sentences exchanged in previous dialogue steps. While generating a system output only the directly preceding user input is taken into account and the contents and dialogue topics mentioned beforehand are disregarded (if not stored in variables). For instance, the reaction to the user input "yes" cannot be made dependent on a preceding chatterbot question or sentence. Moreover, more complex linguistic features such as anaphoric references cannot be resolved. Some chatterbot implementations provide facilities for creating dependences between two or more rules (for instance, the applicability of a specific rule can be constrained to the application of another one in the preceding step). But this basic enhancement is still not sufficient for achieving complete dialogues. Consider, for instance, the case of a configuration task, in which the whole dialogue or a subdialogue are used for negotiating a product variant on the basis of specific needs and preferences. This can only be achieved in several subsequent and interrelated dialogue steps. Therefore, an improvement to basic chatterbot technology is desirable, consisting of expanding the dependency of the transition function \( u \) on entities which are capable of simulating dialogue memory, such as the states of a finite automaton. This will be the aim of the following section.

4.2.4 Dialogue Plan-based Chatterbots

Chatterbots featuring goal-oriented behaviour need to follow a dialogue strategy based upon which they can guide the user during the conversation. As we have seen in section 3.2, dialogue goals are sometimes already known at the outset or can be defined dynamically during the interaction, depending on a user’s request or on a specific combination of circumstances derived from the current context. In both cases the system’s utterances to generate have to be based on a predefined plan, such as a dialogue management plan, which defines the sequence of the necessary steps to undertake towards the intended goal. For instance, for reaching the goal of a successful bookselling transaction, the following steps may be necessary:

- **Identification of the book.** Within this step the book to be sold is identified. For this aim the user may either directly name or select the intended book or find it in stock by means of different search strategies (based on her preferences or on specific book attributes, such as name, author, genre, and so on).

- **Negotiation of the transaction properties.** In this phase details about the transaction such as the payment and shipment method are negotiated.

- **Identification of relevant user attributes.** User attributes, such as delivery address or other relevant profile information are collected.
• **Confirmation.** Finally, the transaction is confirmed by the user and the shipment procedures authorised.

Of course, other scenarios and classifications of required steps which differ from this example are conceivable. In order to build a dialogue management plan, the listed steps have to be arranged in a sequence, determining the order in which the steps are processed. For instance, a possible sequence is given in the following:

1. **Determination of the book**
2. **Identification of relevant user attributes**
3. **Negotiation of the transaction properties**
4. **Confirmation**

Every step can be seen as a subgoal and thus as a goal of its own to be reached. It can be also divided into several substeps. For instance, the subgoal "Negotiation of transaction properties" defined above, can be further subdivided in the following sequence:

3. a) **Determination of the payment method**
   b) **Choice of the dispatch mode**

In most cases, the subgoals required for the achievement of a main goal are not linearly interrelated. Depending on specific attributes of an achieved goal, there may be different alternative subgoals to be reached and thus steps required for the achievement of the main goal. For instance, if we consider two different payment methods for the example discussed above, such as bank transfer or credit card, two alternative successive goals may be provided. In case of a bank transfer, it may be necessary to collect the user's bank account details, whereas the card details may be determined for users who have chosen to pay via credit card. Thus, an alternative plan for the achievement of the subgoal "Negotiation of transaction properties" can be provided:

3. a) **Determination of the payment method**
   b) i. Payment method: bank transfer $\rightarrow$ collect bank account details
      ii. Payment method: credit card $\rightarrow$ collect credit card details
   c) **Choice of the dispatch mode**

The process of dividing subgoals into several substeps can be recursively repeated until steps describing atomic actions are reached. In the setting of a natural language based conversation, atomic actions consist of the utterances exchanged between the dialogue partners. In the case of system outputs, the goal defined by an atomic action is reached when a sentence is successfully output and thus conveyed to the user. In the case of
user inputs, an atomic action refers to a set of alternative text messages which are expected to be input at a specific point of the dialogue. Depending on the input dialogue contribution, a successive atomic action defining the system output is processed. The list of expected user inputs also contains a default entry (i.e. the "UNDEF"-pattern), which is processed if none of the other alternatives match. Therefore, the goal related to a user atomic action is reached whenever the user confirms a sentence which has been input. Regardless of the contents of the confirmed user input, a dialogue management plan always provides for a continuation of the dialogue, either in form of an advancement towards the intended main goal or in form of a request for reformulation.

Since sequences of atomic actions are usually ramified, their description is more appropriately visualised by means of a directed graph. For instance, the subgoal "collect credit card details" introduced above can be broken down to the atomic actions defined in Figure 4.8. Boxes contain the system outputs, while the arrows leaving a box represent the possible dialogue contributions of the user.

![Diagram of atomic actions](image)

Figure 4.8: Atomic actions of a subgoal
4.2 A stepwise refinement of chatterbots towards proactivity

All arrows leaving a specific box (including the arrow marked with "UNDEF") refer to the user atomic action of that particular point of the dialogue plan. Every plan of a main goal or a subgoal contains an entry point which is initiated by a specific user intention. In the example of Figure 4.8, the subgoal of collecting the credit card details is only pursued if the user had previously entered the text pattern "I want to pay by credit card" (or similar patterns expressing the same intention). The first step in the plan consists of the elicitiation of the credit card type. For this aim, the system output consists of the question "Which credit card are you going to use?". The expected user inputs at this time can be divided into three groups. Answers referring to the user owning either a "VISA" or a "Mastercard" credit card are considered as appropriate. They lead to a positive advancement in the plan towards the main goal. In the case of users owning an "American Express" card a specific arrow in the plan is provided. It leads to a system output explaining the user that this kind of credit card is not accepted. The user can now either select another card or a different payment method (which may require use of a different plan). Finally, the last group of expected user inputs refers to the "UNDEF" case. It stands for all input variants which lead to a request for reformulation (i.e. the original question about the credit card type is formulated differently, based on the assumption that the user did not understand the previous message). This output is repeated, until an acceptable input is given. Similar behaviour applies to the following questions of the plan, namely the request for the credit card number and expiry date. System outputs which provide no successive atomic user actions represent the terminating states of a subgoal (see the boxes with a double line border in Figure 4.8). Their achievement corresponds to a successful accomplishment of the goals aimed by the current plan.

Figure 4.9 shows the state-transition diagram of a plan based chatterbot designed on the basis of the dialogue management plan discussed in the previous example (shown in Figure 4.8). Indeed, since augmented transition networks (ATN) describing chatterbots and directed graphs representing dialogue management plans (DMP) are based on similar structures, a merger of both entities can be achieved, with the aim of producing plan based chatterbots. The nodes of a DMP-graph can be mapped to the states of a chatterbot automaton, and the edges between nodes can be represented as transitions between states. For every edge linking two boxes in a DMP several transitions in the ATN may be necessary. Indeed, the user's turn in a dialogue plan is described by a concept or an intention, whereas the implementation of chatterbots requires explicit patterns as annotations of transitions. The same intention (such as a positive answer to a question) can be expressed by the user with different patterns (such as "yes", "yeah", "of course", and so on) which all need to be coded as rules and thus represented by their own transition. The generated ATN maintains the same structure of its source DPM-graph providing for extended transitions between some of the states. For instance, the transition between states $z_1$ and $z_{11}$ may take place by means of both the pattern "* Mastercard" (which stands for the inputs "I own a Mastercard", "I am going to use a Mastercard", and so on) and the pattern "* Visa" (which stands for the inputs "I own a Visa", "I am going
to use a Visa", and so on).

Figure 4.9: An excerpt of a state-transition diagram of a plan based chatterbot

By looking at a state transition diagram of a plan based chatterbot it should become clear that the main advancement with respect to basic chatterbots handled in the previous sections consists of the way in which a system response is generated. Indeed, not only the last user input is considered, but also the directives defined by the adopted dialogue management plan. The system reactions to a specific user input are therefore adapted to the current position within the plan. On the other hand, the same user in-
4.2 A stepwise refinement of chatterbots towards proactivity

put can produce different responses depending on the node preceding it. Therefore, the transition function of a plan-based chatterbot has also to be based on the current state. This is reflected in the following:

Definition 2: A plan-based chatterbot $C_{PB}$ is a deterministic augmented transition network (Moore-Type), described by the tuple

$$C_{PB} = (X, Y, Z, V, \alpha, u', g)$$

in which

- $u' : Z \times X \rightarrow Z$,

and everything else corresponds to Definition 1.1.

Once again, a table can be used for the description of $u'$, including an additional column for the specification of the current states $z$ used for the mapping. For instance, Figure 4.10 contains an excerpt of the description of a transition function of a plan-based chatterbot. For ease of readability of the table, input variants leading to the same transition have been grouped together in one table entry. The input "yes" (as well as synonyms such as "yeah", "of course" and so on) produces a transition to state $z_{11}$ or to state $z_{111}$ depending on the current state in which the input was entered.

<table>
<thead>
<tr>
<th>Current State</th>
<th>Pattern</th>
<th>New State</th>
</tr>
</thead>
<tbody>
<tr>
<td>$z_i$</td>
<td>&quot;yes</td>
<td>yeah</td>
</tr>
<tr>
<td>$z_i$</td>
<td>&quot;no</td>
<td>nope</td>
</tr>
<tr>
<td>$z_i$</td>
<td>UNDEF</td>
<td>$z_{13}$</td>
</tr>
<tr>
<td>$z_{111}$</td>
<td>&quot;yes</td>
<td>yeah</td>
</tr>
<tr>
<td>$z_{111}$</td>
<td>&quot;no</td>
<td>nope</td>
</tr>
</tbody>
</table>

Figure 4.10: An excerpt from a table describing a transition function for a plan-based chatterbot

For every state, the case of an unrecognized input needs to be handled by implementing the transition to a state in which an appropriate system reaction is output. In this way, it becomes possible to avoid a standardised and generic message (for instance limited to "I did not understand you!"). By considering the context, i.e. the preceding dialogue steps and particularly the last message output by the chatterbot, a more adapted and pertinent request for reformulation can be expressed. Usually, the previous system output is
repeated, explained more explicitly or extended with examples, based on the assumption that a misunderstanding on behalf of the user caused him to provide an input which deviates from the expected reaction. Moreover, the user’s attention is set back to the previous dialogue step originating the unrecognized pattern. Obviously, there may be the risk of entering a cycle of unexpected inputs and requests for reformulations, which hinders the progress of the predefined dialogue plan. As we will see in the following, other techniques such as the invocation of external information sources or the dynamic identification of a new dialogue goal can be used to avoid such cycles.

The complexity of plan based chatterbots is obviously higher than that of standard chatterbots. In terms of number of rules, plan based chatterbots may theoretically use \(|Z|n\) rules if we consider \(n\) as being the number of rules of a normal chatterbot and assuming that every rule is responsible for processing exactly one pattern. Indeed, for every state, the reaction to all known patterns has to be specified by coding an according number of rules. In fact the set of expectable patterns, i.e. user input variants which may be appropriate to the current chatterbot output, may be different at every step of the dialogue. Thus, not every pattern needs to be considered at every state. Furthermore, there may be patterns which can be handled independently from the current state. They usually do not refer to previously exchanged utterances, producing transitions which always lead to the same destination state.

Even if plan based chatterbots allow for more comprehensive and coherent dialogues, the risk of frequent requests for reformulations is still high. Increasing the number of recognizable patterns, i.e. by implementing a larger set of rules, may only solve the problem partially. Indeed, if the topics discussed refer to contents which frequently change and are updated, the rule set has to be redesigned every time. More advantageous is to use an external repository of up to date data, which can be accessed at run time for generating a system response. This alternative will be discussed in the following section.

### 4.2.5 Enabling context-based Information Retrieval

The way in which the determinism of \(u\) and \(u'\) is made possible also makes a limitation of chatterbots clear: only user input variants foreseen at design time can be processed. Chatterbots featuring dialogues with a high frequency of requests for reformulations soon lose attractiveness, resulting in frustrated users. Moreover, personalised answers, which take into account specific user properties or other contextual parameters, are rather difficult to achieve, since a complete anticipation at design time is too complex. These considerations lead us to head for another extension of chatterbots, in which the output function assumes a dynamic nature, allowing for a more appropriate output generation at runtime. For this aim, information sources such as databases, document repositories or case bases must be made available. Moreover, appropriate access methods have to be defined, such as, for instance Information Retrieval engines. This required expansion of chatterbots, can be more formally described in the following:
4.2 A stepwise refinement of chatterbots towards proactivity

Definition 3: A context-IR-chatterbot $C_{CIR}$ is a deterministic augmented transition network (Mealy-Type) with a non-deterministic output function, described by the tuple:

$$C_{CIR} = (X, Y, Z, V, I, \alpha, u', g')$$

in which
- $I$ is a set of external information sources (whose contents are determined at run time),
- $g': Z \times X \times V \rightarrow Y \cup I$,

and everything else corresponds to Definition 2.

After every transition, the system output is either defined as usual by the contents of the target state or determined by the result of a retrieval step accessing external repositories. For this aim, the last user input is converted to a query and possibly expanded in order to involve contextual parameters in the search process. Thus, a more extensive handling of variables is needed. Text patterns which explicitly mention a user property (such as "I prefer the genre horror") or direct answers to system questions (for instance the pattern "horror", input after the question "Which genre do you prefer?") are further analysed in order to identify a value to be set for a contextual variable (such as "preferredGenre"). To guarantee consistency the recognized values are usually checked against a given set of already known values.

The success of the retrieval step may be used as a criterion for the choice of the kind of output to provide. For instance, consider a chatterbot acting as a shop assistant for an online book store. The books in stock are described by the usual metadata such as title, author name, abstract and so on. At a certain interaction step, a user may request the availability of a preferred book type by mentioning a summary of its contents (for instance "I am looking for a book featuring chainsaw murderers"). An applicable rule may not be present in the chatterbot’s rule base (obviously, it would be a hard task to code rules about contents of all existing books of a shop), therefore the request is used for querying the books database. Specifically the title and the abstract are suitable sources to consider and textual retrieval may deliver a first result set. Moreover, an expansion process may be useful for restricting the result set by determining more suitable results. For instance, user properties may include its language, preferred genre, preferences of paperbacks or hard covers, and so on. These properties may be learned from previous dialogue steps or known in advance (some users agree to disclose their profile when connecting to a web site or online shop). An IR-engine usually returns a weighting of the achieved results as in the case of probabilistic retrieval. A threshold can be used for guaranteeing quality: if no result weight exceeds the threshold, a standard answer to the user’s request can be still delivered (such as "Sorry, but currently we cannot offer you a book featuring chainsaw murderers"). Results having weights greater than the threshold can be directly provided as a response, possibly embedded in an
output sentence which informs the user about the positive search outcome (for instance "I looked through our stock and found out that following titles may be interesting for you..."). Both weights and thresholds are expressed as numerical values between 0 and 1. Suitable values for thresholds are determined depending on the quality and complexity of the used repositories. More comprehensive databases may provide higher probabilities for successful retrieval outcomes, while sparsely populated information sources will certainly provide less satisfying results. In both cases empirical approaches are necessary, to be possibly repeated during the chatterbot’s lifetime. Indeed, the contents of the used repositories may dynamically change with time, due to frequent updates and extensions. A single threshold may be used for all dialogue steps or different values adopted for each of the chatterbot’s states in which an IR-process is performed.

For a possible implementation of context-IR-chatterbots the features of augmented transition networks can be used again. Indeed the set of actions to perform after a transition can be extended with the aim of determining the output message. In the following, some actions are defined and explained. The first action considered consists of the transformation of (part of) the last user input into a query. As usually done in IR-environments, the raw input text is processed for both the elimination of stopwords (i.e. common words such as "a", "of", "the" and so on which have little or no meaning by themselves) and stemming procedures (i.e. the attempt to reduce a word to its stem or root form). The result of this action is a list of words to be used as search terms in a retrieval process.

\[ \text{buildQuery: } \text{string} \rightarrow \text{string} \text{ (build a list of search terms out of a given text)} \]

As introduced before, the generated list of search terms can be expanded with other terms coming from the collected dialogue context. The aim is to restrict the set of retrieved results in order to provide a more concise system response. Different query expansion models can be applied, defining strategies, sequences and the most suitable sources from which to get the terms to be used for expansion. The models to use are chosen depending on the application’s domain and on the kind of information sources to be queried. Section 3.2 discusses a query expansion model suitable for the bookselling domain. For the expansion process, the \texttt{concatenate} operation defined for the basic type \texttt{string} may be appropriate, as it merges a first string (i.e. the query) with a second string (i.e. the expansion). It may be advantageous to first expand a text before generating a query out of it. In this way, also the added expansions can be stemmed and cleared of stopwords.

Once that the query is complete, retrieval can be carried out by means of the action \texttt{IR}. It returns an empty string if no records could be found or if no record exceeds the given threshold. Otherwise, records with the highest relevance weight are delivered back. For their handling an additional data type needs to be introduced:
4.2 A stepwise refinement of chatterbots towards proactivity

- **list**
  - domain: ordered lists of string elements
  - operations: select: list \( \times \) integer \( \rightarrow \) string
    choose: list \( \times \) string \( \rightarrow \) string
    \( s: \) list \( \rightarrow \) string

A list encloses an ordered set of strings, whose elements are referred to by their position within the list or by specific keywords. By the use of the operation select, an item can be chosen out of the list by indicating its position. The operation choose performs the same as select but requires a string instead of a number. The used string should match one of the keywords representing the list item. Finally, a list can be converted to a string by means of the operation \( s \). It consists of a concatenation of the list items in which the original order is maintained. Different types of separators can be used between the items, such as numbers or punctuation marks. Their choice can be defined during configuration and set up of the chatterbot.

The maximum number of records to be returned by the operation \( IR \) is defined by a given parameter, as well as the identifier of the information source to use. The retrieval decision is made by comparing the terms of the query with the index terms (important words or phrases) appearing in the records of the information source. The relevance weights can be calculated in different ways, taking into account, for instance term frequencies and/or adjacencies of the query terms in the records of the information source. The specification of the operation \( IR \) is:

\[
IR: \text{string} \times \text{real} \times \text{integer} \times \text{string} \rightarrow \text{list}
\]

Thus,

\[
IR(sID, \text{th}, \text{max}, q):
\]

can be read as follows:

"post the query \( q \) on the source with identifier \( sID \) returning at maximum \( \text{max} \) records whose weights exceed the threshold \( \text{th} \)"

The records of the information source must of course provide a suitable format for being enclosed as part of a chatterbot response. Usually, only a specific field of the record is delivered back. It contains one or more text sentences which can be directly output by the chatterbot without the need to apply a transformation or reformulation.

Finally, an action is needed to substitute the output of the target state with the results of the retrieval process (if any). An introductory sentence can be also added. It is specified as follows:
\[ \text{replaceOutput: state} \times \text{string} \times \text{string} \rightarrow \text{template} \]

For instance,

\[ \text{replaceOutput}(z_i, iText, rText): \]

can be read as:

"if \text{rText} is not the empty string replace the output of state \( z_i \) with the value of \text{iText} followed by the value of \text{rText}"

Figure 4.11 shows the entry of a transition table of a context-IR-chatterbot implementing the book search example discussed previously.

<table>
<thead>
<tr>
<th>Pattern</th>
<th>State</th>
<th>Actions</th>
<th>Output</th>
</tr>
</thead>
<tbody>
<tr>
<td>&quot;I am looking for ( \ast _p )&quot;</td>
<td>( z_1 )</td>
<td>substitute(s(z_i), ( \ast _p ), \text{valueOf}(( \ast _p ))); iText=&quot;I looked through our stock and found out that following titles may be interesting for you: &quot;; rList=IR(&quot;isSource:Books&quot;, 0.2, 2, \text{valueOf}(\text{buildQuery(\text{concatenate(valueOf}(( \ast _j ), \text{valueOf(preferredGenre})))); \text{valueOf(rList))}); rText=s(rList); replaceOutput(z_i, \text{valueOf(iText)}, \text{valueOf(rText))};</td>
<td>&quot;Sorry, but currently we cannot offer you ( \ast _1 )&quot;</td>
</tr>
</tbody>
</table>

Figure 4.11: A transition enabling context-IR

The input "I am looking for a book featuring chainsaw murderers" produces a transition to state \( z_1 \) since it matches the pattern "I am looking for \( \ast \_p \)". The first action performed consists of the substitution of the placeholder "\( \ast \_1 \)" in the standard output of \( z_1 \) with parts of the input text, composing the output message "Sorry, but currently we cannot offer you a book featuring chainsaw murderers". Depending on the outcomes of the following actions, this output will remain unaltered or will be substituted by a retrieval result. The predicate \( iText \) is used to ease the readability of the figure. It contains an introductory sentence which will precede the description of possible retrieval results.
Afterwards, a query is constructed by first extending the relevant part of the user input (i.e. the substring "a book featuring chainsaw murderers") with terms describing the user's preferred book genre (if not void). We assume for this example that in a previous dialogue step the predicate preferredGenre may have been set to "horror". After applying stopword elimination and stemming the resulting search terms are "book featuring chainsaw murder horror". The stopword "a" has been eliminated and the root forms "featur" and "murder" are derived from the terms "featuring" and "murderers" respectively (the achieved root forms may vary depending on the kind of stemming algorithm used). As an information source for the retrieval step a repository with the identifier "isource:Books" is used. It consists of a database of book descriptions. Every record contains standard fields such as author, title, abstract, genre, price and so on, as well as a specific field, called shortDescription used for the chatterbot output. The record contents may have been indexed previously after having been cleared from stopwords and stemmed to root forms, as in the case of the query terms. One of the records may be the horror book "The Texas Chainsaw Massacre Companion" written by "Stefan Jaworzyn". Due to its title, genre and abstract description the book may turn out to be particularly relevant to the generated query, providing a relevance weight of 0.4. Since this weight exceeds the required threshold of 0.2, it will be considered as part of the system response. No other search result may provide an acceptable weight, so that only one book is delivered back and its shortDescription field is used as an item of the result list. The list is converted to a string and added to the system response, which finally consists of the text: "I looked through our stock and found out that following titles may be interesting for you: The Texas Chainsaw Massacre Companion by Stefan Jaworzyn".

In the case of result lists consisting of more than one record, specific rules must be implemented for allowing the user to select an item out of the list. He may address his choice by either mentioning a position (such as by sentences like "I'd like to buy the third book" or "Which is the price of book number 2?") or by including keywords of the chosen item in his input ("The book by Stephen King sounds interesting to me"). The implemented rules must allow for the extraction of relevant information out of the input sentence, to be used with the appropriate select or choose operation.

Even if context-IR-chatterbots may provide for more flexibility in the determination of a system response, truly cooperative dialogues are still not achievable. Conversations based on a fixed plan may only achieve satisfactory results if the targeted goals are valid throughout the entire dialogue. But if the current dialogue plan becomes obsolete (the user may lose her interest in the current goal or a new goal may arise), a strategy for substituting the plan with a new one must be made available. Mainly, two alternative cases can be identified, in which the need to initiate a new dialogue plan is recognized. The first case, based on an explicit user request for an alternative goal, can be implemented through standard pattern matching of chatterbots: dedicated rules need to be added, which provide transitions to starting states of new dialogue plans independent of the current state. Already known values of contextual parameters can be still made available
within the new plan or cancelled for a new collection process. The second case, entirely based on the system’s initiative, is discussed in the following section. Another extension of chatterbots is provided, mainly consisting of the introduction of a non-deterministic transition function.

4.2.6 Chatterbots with mixed-initiative

A crucial property of cooperative dialogues is the ability for both parties to take the initiative during the conversation. It mainly allows the alternation of turn-taking in dialogue-guidance, needed for a variety of reasons. For instance, it may become necessary for one speaker to catch a discussion going in a wrong or risky direction. Influencing and convincing the counterpart, as well as making suggestions and recommendations are also frequently occurring reasons which lead to an interruption of the current dialogue flow. Finally, the initiative can also be taken for anticipating problems and wishes, given that enough prerequisites are known.

In the area of conversational agents, several approaches deal with turn-taking in dialogue-guidance. For instance, Allen defines four different levels of mixed initiative [All99]: unsolicited reporting, which is used to notify the counterpart about critical information which has suddenly become available; subdialogue initiation, used whenever a new subdialogue is introduced for clarifying, correcting, and so on; fixed subtask initiative, which occurs for solving a predefined subtask; and negotiated mixed initiative, for solving tasks if specific properties apply (based on the answers to questions like "Do I have the resources to solve the task?" or "Am I best qualified for its solution?" and so on). Irrespective of the kind of initiative to take over, a chatterbot needs to identify the correct point of time for being proactive. This may depend on a rather complex arrangement of conditions and constraints, thus it cannot be coded within rules. A specific subset of proactive states needs to be introduced, i.e. states that entitle the chatterbot to take the initiative for a given aim. Usually, proactive states represent an entry point for a new dialogue plan. As happened in Definition 3 with the output function, the transition function also needs to become dynamic: at every state it must be checked whether to react normally to the user’s input or to move to a proactive state. Since this decision cannot be taken at design time the related finite state machine is non-deterministic.

This is described by:

Definition 4: A proactive context-IR-chatterbot is a non-deterministic augmented transition network with a non-deterministic output function, described by the tuple:

\[ C_{PCIR} = (X, Y, Z, Z_P, V, I, \alpha, u', g') \]

in which
- \( u' : Z \times X \rightarrow \mathcal{P}(Z) \), with \( \mathcal{P}(Z) \) meant as the power set of \( Z \);
- \( Z_P \subseteq Z \), contains the proactive states.

and everything else corresponds to definition 3. The subsets of \( Z \) considered for the transition relation \( u' \) contain one \( z \in Z \setminus Z_P \) and possibly one or more \( z_p \in Z_P \). This means that at every user input, the transition may either lead to a normal state as defined by the underlying dialogue management plan or to a proactive state, in which a new dialogue plan is initiated.

In the context of the bookshop example introduced before, let us consider some cases in which the chatterbot may take the initiative during a dialogue. Figure 4.12 shows a state transition diagram whose starting state \( z_0 \) informs the user about a successfully retrieved book.

![Figure 4.12: An excerpt from a state-transition diagram of a proactive context-IR chatterbot. Double-lined circles represent proactive states.](image-url)

At this point only one kind of user input is considered in the diagram, namely a question about the shipment costs for the book retrieved. As a standard reaction, the
current dialogue plan foresees to output the amount of the fee applied normally for the shipment of goods (state z1). But specific circumstances may require an alternative dialogue strategy. For instance it may be the case that the delivery fees vary according to different criteria such as selected book attributes (price, format and so on) or user properties (location, special privileges and so on). A direct system response in which all alternatives are listed may lead to confusion and misunderstandings on behalf of the user. A more appropriate and effective support consists of initiating a new subdialogue, in which further properties and preferences are elicited with the aim of generating a more specific and thus concise answer (state zp1). Unsolicited reporting is performed in the case of state zp2. The exemption from delivery costs is communicated to the user in terms of a system response.

These examples should make clear why a non-deterministic approach is needed. At every step, the last user input and possibly some basic contextual parameters may not be enough for determining the best way to proceed. They may suffice for a simple and plan-based reaction, but the decision of being proactive cannot be taken within chatterbot rules. A possible implementation approach for proactive chatterbots is introduced in the following section.

4.2.7 How to implement proactive Chatterbots

In order to resolve the non-determinism, thus allowing for an implementation of a proactive context-IR-chatterbot, another kind of extension is needed: the ongoing dialogue has to be monitored by an external module, based on an expert system or a decision maker tool. Its aim is to check at every interaction step if a set of constraints is satisfied and if the current dialogue plan can be interrupted for the initiation of a proactive subdialogue. The constraints may have the form of logical predicates or of variables to check within nested conditional statements. They may have been gathered from the user during the preceding dialogue or learned from specific fields of previously retrieved records.

For instance, Figure 4.13 shows a summary of collected contextual parameters which may be used for possible transitions to proactive states. The parameters are divided into three groups. "User" properties regard specific attributes of the user profile, such as his name, country of origin and preferred book genre. The group "transaction" contains properties describing the current status of business-related activities. The example assumes that no order has been placed up to now. Moreover, the general shipment fee applied to standard goods is contained in the property "transaction.shipmentFee". In this way an update due to price changes can be more conveniently applied, without having to modify individual rules of the chatterbot. Finally, the "Book" group, summarises all properties of the last retrieved record. Among them, the title, author, price, format and publication year can be found. Moreover, a specific parameter indicates the book’s eligibility to be considered in special shipment offers.

The predicates are used for building heuristics, i.e. rules which are based on the eval-
Figure 4.13: A summary of collected context parameters

4.2 A stepwise refinement of chatterbots towards proactivity

<table>
<thead>
<tr>
<th>User:</th>
</tr>
</thead>
<tbody>
<tr>
<td>userProperty.name=&quot;Peter&quot;</td>
</tr>
<tr>
<td>userProperty.country=&quot;England&quot;</td>
</tr>
<tr>
<td>userProperty.preferredGenre=&quot;horror&quot;</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Transaction:</th>
</tr>
</thead>
<tbody>
<tr>
<td>order.currentValue=0</td>
</tr>
<tr>
<td>transaction.shipmentFee=5</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Book:</th>
</tr>
</thead>
<tbody>
<tr>
<td>bookProperty.price=25</td>
</tr>
<tr>
<td>bookProperty.eligibleForShipmentOffer=&quot;true&quot;</td>
</tr>
</tbody>
</table>

| bookProperty.title="The Texas Chainsaw Massacre Companion" |
| bookProperty.author="Stefan Jaworzyn" |
| bookProperty.year="2002" |
| bookProperty.format="hardcover" |

**heuristic 1: "shipment to England is free on orders over 50 EUR, ask for order extension"**

[1] userProperty.country="England",
[3] → move to state \( z_{p1} \)

The heuristic can be read as follows: "If the user’s country of origin corresponds to England (line [1]) and if the sum of the current order value plus the price of the retrieved book is less than 50 (line [2]), then move to the proactive state \( z_{p1} \), in which the user is informed about the possibility to get free delivery by extending his order (line [3])."

State \( z_{p1} \) corresponds to the entry point of a new dialogue plan. It is first responsible for the output of a sentence, in which the goals of the new plan are clarified. According to the user’s reaction, three possibilities for continuing the dialogue are available. In case of agreement, the dialogue is continued on the basis of the new plan. In case of disagreement with the new goal, the option to exit the plan is given by either restoring the previous plan (see the transition from state \( z_{p1} \) to state \( z_1 \) in Figure 4.12) or moving to a new one on the basis of a proactive transition (i.e. the transition from state \( z_{p1} \) to
state $z_{p2}$). This transition is carried out if the following heuristic applies:

**heuristic 2:** "shipment within the UK for selected goods amounts to EUR 3"

\[ \begin{align*}
[1] & \text{bookProperties.eligibleForShipmentOffer,} \\
[2] & (\text{userProperty.country}=="Scotland") \text{ or} \\
[3] & (\text{userProperty.country}=="England") \text{ or} \\
[4] & (\text{userProperty.country}=="Wales") \text{ or} \\
[5] & (\text{userProperty.country}=="Northern Ireland") \\
[6] & \rightarrow \text{move to state } z_{p2}
\end{align*} \]

In other words, if the book in question takes part in the special shipment offer (for instance, because it can be shipped from a storehouse which has special agreements with a carrier) and if the user’s country of origin is within the UK (lines [2] to [5]), the shipment fees amount to EUR 3.

The decision to proactively substitute the current dialogue plan with a new one is therefore based on facts which can be specified at design time. Preferences and properties can be checked at run time once their values are available. The facts to check and the action to perform in case of compliance are considered as a whole as a heuristic. Indeed, until the user confirms her agreement with the goals of the new plan, there is no guarantee that the initiative taken led to the most appropriate alternative. Therefore, the system needs to keep track of the state in which the decision to be proactive was taken. A negative user reaction induces the system to either move to another proactive state or to restore the original plan. During the design of heuristics, particular care must be used to avoid the implementation of conflicting rules (i.e., heuristics which, based on the same premises, produce different transitions). Alternatively, a prioritisation mechanism can be introduced for unequivocally choosing one heuristic during conflict situations.

As opposed to the extensions considered in previous sections, proactive transitions cannot be implemented by introducing new action types for augmented transition networks. Indeed, actions are processed directly after a transition is chosen. The checking of heuristics has to be performed prior to moving to a new state. Thus, it becomes necessary to base the applicability of a chatterbot rule on additional constraints. In the frame of transition tables, this is realised by adding a new column, just after the column in which the pattern is specified. Figure 4.14 shows parts of a transition table of a proactive context-IR chatterbot implementing the example discussed above.

The first entry of the table is processed if the pattern matches the current user input and if the constraints defined in the "Heuristic" column apply. In this case, a transition to state $z_{p1}$ is carried out and the output corresponds to "For orders over EUR 50 we offer free shipment. Would you add something else to your cart?". Analogously, the transition to state $z_{p2}$ is performed and the text "Forwarding expenses within the UK for this book amount to EUR 3" used as output message. If none of the previous set of constraints can
be satisfied, the standard plan-based transition to state $z_1$ is performed and the output composed at run time by considering the value of $\text{transaction.shipmentFee}$. The order in which the transitions are reported in the table corresponds to the actual order in which they are processed, resulting in a prioritisation of heuristics.

<table>
<thead>
<tr>
<th>Pattern</th>
<th>Heuristic</th>
<th>State</th>
<th>Actions</th>
<th>Output</th>
</tr>
</thead>
<tbody>
<tr>
<td>&quot;Which are the shipment costs?&quot;</td>
<td>$\text{bookProperties.eligibleForShipmentOffer,}$ \small&lt;br&gt;$\text{(userProperty.country=&quot;England&quot;) or}$ \small&lt;br&gt;$\text{(userProperty.country=&quot;Scotland&quot;) or}$ \small&lt;br&gt;$\text{(userProperty.country=&quot;Wales&quot;) or}$ \small&lt;br&gt;$\text{(userProperty.country=&quot;Northern Ireland&quot;)}$</td>
<td>$z_1$</td>
<td>substitute($\text{bookId}$, ...$\text{valueOK(transaction.shipmentFee)}$);</td>
<td>&quot;Forwarding expenses amount to €1.1&quot;</td>
</tr>
</tbody>
</table>

Figure 4.14: A transition based on heuristics

The following chapter will discuss a possible architecture of a proactive context-IR chatterbot. It is designed on the basis of three partial architectures used for systems capable of solving problems which pertain to only one of the relevant problem classes. A real implementation is subsequently presented and evaluated, with the aim of assessing the efficiency of the chatterbot enhancements applied. But first, the achievements of this chapter are summarised in the context of a comprehensive model for proactive behaviour.

## 4.3 The Proactivity Model

With the aim of enhancing cooperativity in dialogues with pattern-matching-based conversational systems, we improved the problem solving capabilities of chatterbots by introducing a more proactive behaviour. Problems which extend over the whole dialogue can now be solved by structuring the rule base on the basis of goal-oriented dialogue management plans. Local dialogue problems which mainly arise due to limited language understanding of chatterbots are transformed into queries to external information sources. Finally, by taking the initiative and thus dynamically identifying new dialogue goals, the anticipation of possibly occurring problems and an overall flexibility in the adaptation of the dialogue strategy pursued can also be reached.
By considering the targeted enhancements as a whole, a generic model for proactivity can be sketched out. It describes and manages the execution of tasks on the operational level by trying to reproduce the behaviour adopted by humans in real dialogues (Figure 4.15).

![Diagram of a generic model of Proactivity]

Figure 4.15: A generic model of Proactivity

The dialogue management is based on predefined strategies, carried out by means of management plans. They consist of structured dialogue steps needed to reach a defined goal. The plans may involve external information sources whenever required. On the basis of a conversation snapshot, represented by the contextual properties collected so far of both dialogue and user, an expanded query may be used to interrogate external repositories for completing the output with possibly relevant data. A dialogue monitoring module supervises the ongoing dialogue (represented by the dashed arrow in the figure). Its aim is to interrupt the currently used dialogue plan in order to substitute it with a more appropriate one with new dialogue goals. Its decisions are based on a set of
predefined generic heuristics, which also involve available contextual data.

The model allows for a deterministic identification of a suitable point of time at which proactive actions have to be carried out. Indeed, every specific heuristic represents a unique transition to a proactive state, out of which a new dialogue management plan is initiated. A new plan always starts with a system output message, to which the user can immediately react with a plan-conformant input or with a message in disagreement with the goals pursued (mostly, this is recognized within a request for a new dialogue goal). In this case, an alternative plan has to be identified or the original one restored.

By means of the following lines of pseudo code, a more formal description of the proactivity model can be provided. The algorithm is processed whenever a new input is available. First, method main is executed.

```java
performTransition (user.input, state.current, context.current) {
    state = null;
    while (state == null) {
        findNextMatchingPattern (user.input, state.current);
        if (heuristic.exists) {
            checkHeuristic (context.current);
            if (heuristic.applies) state = matchingPattern.nextState;
        }
        else state = matchingPattern.nextState;
    }
    return state;
}

generateOutput (state.new) {
    performActions(state.new);
    return state.new.output;
}

main (user.input) {
    state.new = performTransition (user.input, state.current, context.current);
    chatterbot.output = generateOutput(state.new);
    state.current = state.new;
}
```

The user input is contained in user.input while the current state is set as a value of variable state.current. The method main consists of three steps. First, the target state for a transition is determined, by calling the method performTransition. On the basis of the new achieved state (contained in variable state.new), the chatterbot output is generated (method generateOutput). Finally, state.current is updated for being used in the next iteration, i.e. when a new user input is available.

Method performTransition uses the value of the auxiliary variable state as an exit
criterion for a while-loop. The iteration within the loop is achieved by the instruction findNextMatchingPattern, applied on the current state and the last user input. For every matching pattern the possibly related heuristic deciding on a proactive transition is checked for compliance. In positive cases the next state is determined. Otherwise, the next state is provided by the standard plan-based transition. Note that also the UNDEF-case is considered as a plan-based transition. It applies if no other matching pattern is found previously.

The generateOutput method is responsible for carrying out the necessary actions (if present) for determining the system response. They include the management of context as well as possible retrieval processes. Finally, the generated output is returned.
5 Architecture and technologies used

The aim of this chapter is to provide a flexible reference architecture for the implementation of proactive chatterbots, meeting the conceptual modelling requirements discussed so far. Instead of concentrating on technical properties and interface details, the architecture will be mainly viewed as a partitioning scheme, based on a logical arrangement of subsystems which map the user and system requirements identified in the preceding chapters. In this way, a higher degree of platform and domain independence can be achieved, hence requiring a further adaptation step in the case of a specific implementation. For some selected logical components, possible technological solutions will be introduced and used in the following chapters for an eventual implementation.

5.1 Logical Architecture of a proactive Conversational System

Conversational systems are mainly to be employed as a specific kind of intuitive user interface. Their means of interaction is natural language, employed in the context of coherent dialogues. As the object of interaction, both user and system knowledge have to be made available and accessed efficiently. This succinct description entails the requirement of a minimum system architecture based on three layers. The front end tier enables communication with the user, by providing input and output facilities through available channels. A middle tier is needed for controlling the conversational behaviour, which consists of interpreting the user input and generating the appropriate system message. The back end tier provides a link to various information sources containing appropriate representations of cognition and knowledge to be considered in the dialogue.

Various exchangeable modules can be used at every layer for customising the behaviour of a conversational system. They allow for specific properties and abilities of the system, fulfilling the requirements of the intended application. The modules employed also define the typology of information sources to be embedded and accessed. Some of the repositories used may require particular maintenance effort due to their dynamic nature. If they are to be used as storage media, i.e. for collecting and managing dialogue and user relevant data, overflow prevention and removal procedures for obsolete or erroneous data have to be taken into account and scheduled accordingly. On the other hand, repositories of current and time-dependent information need to be periodically updated and extended. Since offline external intervention may be required for these tasks, the
repositories are not directly embedded in the modules accessing them, but interfaced with the rest of the architecture on a separate tier. Moreover, some repositories may be accessed by more than one module, thus requiring them to be made more generically available.

Figure 5.1: The architecture of a proactive conversational system

Figure 5.1 shows the architecture described above, tailored to the requirements of proactive conversational systems. Modules have been added at each layer and related repositories included for unrestricted access. The input module located on the first layer is responsible for accepting the user’s message and transforming it into an internal representation. Specifically in the case of chatterbots deployed over the web, the input module provides server functionality, by receiving the input text stream and converting it to a string for the further processing steps. The input module can be also extended for allowing alternative input modalities (such as speech or haptics) and multimodal interaction mechanisms. Initial basic interpretation can also be carried out, thus providing an immediate feedback to erroneous or incomplete inputs without involving further processing
modules.

The reformulated user input is transferred to the second layer and processed with the aim of generating a system’s contribution to the dialogue. In the context of chatterbot systems, standard pattern matching procedures are performed, possibly supported by other modules such as the reasoning and proactivity module (see Figure 5.1). Since the dialogue management of proactive conversational systems is based on the utilization of management plans, an appropriate repository is added to the system. It is accessed at every dialogue step for the determination of the successive system reaction and updated offline whenever the designers recognize the need for new dialogue goals. Both system and user utterances are saved to a specific log file called dialogue history. Its entries are sometimes extended with meta-information describing the context in which they occurred.

Most applications aim at providing a real-time operation of the user data collected during the dialogue. Reasoning and computations are carried out by the reasoning module, which may be invoked by specific instructions defined in the currently used management plan. Eventual results are forwarded to the interface layer, whose output module either generates an integrated system response consisting only of a textual message or uses other visual means such as frames of web sites for displaying the results of the computation.

The proactive behaviour of a conversational system is achieved by means of the proactivity module. Based on a set of heuristics contained in an apposite repository (which can also be updated and improved offline), it is responsible for activating other modules or exchanging the current dialogue plan with a new one (thus also accessing the plans repository). The application of a heuristic relies on collected contextual information (for instance, contained in the dialogue history). Once it has been identified that a heuristic is applicable and its requirements are fulfilled, the proactivity module interrupts the normal flow of the dialogue, and prompts the system to replace the current dialogue management plan with a new one. As for the realisation of proactive recommendations, general search heuristics are employed with the aim of transparently collecting information to be used for recommending a potentially interesting item out of the system’s stock. For this aim, the specifications of the current dialogue plan are extended with additional messages and requests to be conveyed to the user.

Whenever a module of the first two layers of the architecture needs to access one of the information sources, an appropriate request is generated and forwarded to the information access layer. Its modules will process the request by either updating the repositories or delivering data back to the requesting module. For instance, the IR module implements an information retrieval engine which is responsible for submitting a query to one of the information sources and collecting a ranked set of results. Besides operating on the dialogue repositories for the handling of plans, heuristics and log files, the IR module is frequently invoked for retrieving application relevant data such as information about products and services offered. Another module of the information access layer, namely the personalisation module, is responsible for the management of user relevant data which
allows for an adaptation of the conversation to the user’s specific needs and properties. Every service-providing web site would expect its users to visit and use the services within the site as often as possible. But if the user has to start the dialogue from the beginning each time that he connects to the site, he will soon get bored and try to avoid communication. The personalisation module allows for skipping most of the preliminary small talk by generating and managing a repository of user profiles. A user visiting the site again in later sessions can thus be recognized and the dialogue planned accordingly. On the other hand, the information gained about former visitors within their profiles can be used for the generation of usage patterns, i.e. assumptions for new users when they start to converse for the first time. The personalisation module frequently also embraces machine learning processes, which aim at an automatic identification of user properties and preferences, based on known facts and performed activities with the application.

5.1.1 Properties entailing robustness

A software architecture is said to be robust if it exhibits a high degree of resilience and flexibility based on a set of observed properties. The following list can be assumed for the above introduced architecture describing a proactive conversational system. Particular care must be taken during an eventual implementation in verifying that the properties are still valid.

- **Fault-tolerance.** The dialogue management of a conversational system must provide for fallback solutions in the case of temporary failure or unattainability of modules located at the middle and back-end layers. For instance, if a rule of a management plan intends to complete a system output with information obtained from the user profile or application data repositories, standardized responses must be generated in the case of non-working IR or personalization modules. If the input or output modules experience momentary breakdowns, recovery processes need to be implemented, in order to roll-back to a previously working state. In this way, the faulty module is given an additional chance for performing its task (e.g. by asking the user to repeat her last input).

- **Portability.** Because of the variety of available platforms and computing frameworks (e.g. different CPUs, operating systems, or third party libraries), a main requirement for an efficient architecture is the ability to adapt it and make it usable in different environments. In the context of conversational systems, both the technologies adopted for individual implementations of modules and repositories and the integration methodology used for enabling communication between the components of the architecture need to be carefully chosen and designed. If the components are distributed over a network and on different computers, efficient and portable middleware has to be used, thus defining generic interfaces and protocols.
5.1 Logical Architecture of a proactive Conversational System

- **Extensibility.** The architecture must provide mechanisms for an easy extension and/or replacement of both modules and repositories without having to make major changes to the system infrastructure. For instance, the dialogue management component should be capable of handling more complex input representations, in the case of additional input means such as a pointing device or gesture recognition techniques. As an additional example, the IR engine used should be customisable enough to allow access to information sources added subsequently.

- **Maintainability.** As already mentioned in the previous section, modules and repositories may be subject to frequent maintenance. The modularity of the architecture makes it possible to perform independent direct intervention on components requiring updates. The system does not even need to be halted during maintenance measures of individual modules and repositories.

- **Serviceability.** The availability of log files, in terms of both generic recordings of the conversations (i.e. the dialogue history) and more module-related tracing of processes and operations, allows for an easy debugging or root cause analysis of technical problems. In the case of exceptional events, due to network failure or massive hardware crashes, an alerting system should be included, generating appropriate notifications for system administrators and technical support personnel.

- **Reliability.** Conversational systems are accessed by a large number of different users. The content and quality of the conversations must be guaranteed to remain consistent over sessions involving users with identical backgrounds and profiles. Therefore, dialogue plans and rules applied need to take into account different possible formulations of user sentences expressing the same intention and aims. Modules implementing computations, such as reasoning and proactivity, should be applied to abstractions or substitute representations rather than directly to the user input. In the proposed architecture, this is achieved by placing relevant modules on the middle tier, thus allowing for a preprocessing of the input strings during transition from the interface layer, before being used for further computations.

- **Scalability.** The dynamic nature of conversational systems requires the capability to increase total throughput and power of performed dialogues. Indeed, rules and plans are usually constantly updated after the analysis of log files. User input formulations containing unrecognized patterns are used as a source for extending the rule set, and past dialogues may reveal the occurrence of specific dialogue problems, requiring the design of new dialogue management plans. Thus, relevant components need to be built for managing different sizes of repositories accessed.

The overall efficiency of the architecture of a conversational system needs also to be supported by the adoption of effective and operative software technologies for the implementation of individual components. The following sections will provide suggestions for
suitable frameworks, with a focus on the dialogue management aspect, as it represents the principal and most fundamental component of the architecture.

5.2 Dialogue management with AIMA

AIML, which stands for Artificial Intelligence Markup Language, is an application of XML, the Extensible Markup Language. By definition, AIML objects are conforming XML documents, although AIML objects may also be contained within XML documents. As XML is itself an application profile or restricted form of SGML (the Standard Generalized Markup Language), AIML objects are also conforming SGML documents. AIML was developed by Richard Wallace and the Alicebot free software community during 1995-2000. It was originally adapted from a non-XML grammar also called AIML, and formed the basis for the first Alicebot, A.I.C.E., the Artificial Linguistic Internet Computer Entity. Since its inception, it has been adopted as a standard by the A.I.C.E. AI Foundation, which now holds its copyright, and whose Alicebot and AIML Architecture Committee is responsible for its maintenance and further elaboration. Nowadays, several commercial and academic projects around the AIML technology can be found. For a comprehensive summary, as well as extensive documentation see the website of the AI Foundation (http://www.alicebot.org).

5.2.1 Categories

The goal of AIML is to enable pattern-based, stimulus-response knowledge content to be served, received and processed on the Web and offline. AIML objects are made up of units called topics and categories, which contain either parsed or unparsed data.Parsed data is made up of characters, some of which form character data, and some of which form AIML elements. AIML elements encapsulate the stimulus-response knowledge contained in the document. An AIML knowledge base (also called "brain") consists of a set of categories (like the one shown in Figure 5.2).

```
<category>
    <pattern>HOW ARE YOU</pattern>
    <template>
        I am fine, thanks. And you?
    </template>
</category>
```

Figure 5.2: An AIML category

Every category contains a "pattern" section, defining a possible user input and a "template" section, containing a system action or output. A pattern may consist of a single word or a whole sentence and may also contain one or more wildcards. For instance the pattern "What * of language do you *" would match both the input variants "What
kind of language do you speak" and "What type of language do you usually speak". Patterns without the star-wildcard are handled first (e.g. the pattern "My name is Peter" will be checked before the pattern "My name is "). Once that a category with a pattern matching the last user input is determined, the template part of that category is executed. A template mainly contains a combination of system instructions processed at runtime, which lead to the generation of a system response. Words or sentences are output without modification (see for instance Figure 5.2). If the output of a chatterbot is displayed within a web browser window, standard HTML-tags can also be used in the template to produce different formatting or to include other HTML elements such as pictures or links.

AIML allows for the management of global and local string variables, called predicates. The values of global predicates are set at design time within specific configuration files. They are used for storing chatterbot specific data such as the chatterbot name or the name of its designer (usually called the "botmaster"). Local predicates are managed in templates. They can be used for a more dynamic response generation providing facilities for the implementation of a primitive form of chatterbot memory. For instance, consider the template displayed in Figure 5.3. The user input represented by the wildcard * in the

begin{figure}[h]
\centering
\begin{verbatim}
<category>
  <pattern>MY NAME IS *</pattern>
  <template>
    Pleased to meet you, <set name="user_name">*</set>
  </template>
</category>
\end{verbatim}
\caption{Managing local predicates in AIML}
\end{figure}

pattern is set as a value for the predicate "user_name" and is simultaneously output as a part of the system response. If this behaviour is not desired, the "think" element can be used to cause its content to be handled only internally, thus omitting the results of a "set" instruction or of every other kind of predicate processing from the system response (see Figure 5.4). A predicate can be reused in a different category (its value can be exploited

begin{figure}[h]
\centering
\begin{verbatim}
<category>
  <pattern>MY NAME IS *</pattern>
  <template>
    Thank you! I will remember your name. <think>
      <set name="user_name">*</set>
    </think>
  </template>
</category>
\end{verbatim}
\caption{Using "think" for omitting output}
\end{figure}
by the instruction "<get name = "user_name"/>". Predicates can also be part of conditional elements. Like the If-then-else statement of a programming language the condition element causes specified content to be returned, depending upon the results of matching a predicate against a pattern. In Figure 5.5 the output to generate depends on the value of the predicate "user_name".

```xml
<category>
  <pattern>DO YOU KNOW ME</pattern>
  <template>
    Well, I know that your name is <condition name="user_name">
      <li value="Peter">Your surname would be helpful.</li>
    </condition>
    <li value="Peter Parker">Are you Spiderman?</li>
    <li>But I still don't recognise you!</li>
  </template>
</category>
```

Figure 5.5: Conditional Elements in AIML

Sometimes it may be the case that the same template can be applied to a set of different patterns. Since AIML does not allow more than one pattern per category, the <srai> operator can be used for implementing simple recursion. Like shown in Figure 5.6, different formulations for a question can be used and only one template is coded for its response.

```xml
<category>
  <pattern>HOW CAN I CALL YOU</pattern>
  <template>
    <srai>WHAT IS YOUR NAME</srai>
  </template>
</category>

<category>
  <pattern>I DO NOT KNOW YOUR NAME</pattern>
  <template>
    <srai>WHAT IS YOUR NAME</srai>
  </template>
</category>

<category>
  <pattern>WHAT IS YOUR NAME</pattern>
  <template>
    You can call me <get name="bot_name"/>
  </template>
</category>
```

Figure 5.6: Simple Recursion with the "srai" operator
5.2 Dialogue management with AIML

The srai element allows also the implementation of other language interpretation techniques such as symbolic reduction (complex grammatical forms are reduced to simpler ones), "divide and conquer" (which consists of splitting an input into two or more subparts, and combining the responses to each), keyword detection and spelling correction.

Besides managing the system output, the template part of a category may also contain functions or subroutines written in a scripting language (such as Perl or Javascript). In this way external processes can be controlled and more complex data structures managed.

5.2.2 Dialogue Plans and Topics

The dialogue management of most existing AIML chatterbots is not usually based on management plans, like the ones introduced in section 3.3. The structuring of their brains is limited to the distribution of the categories to different files, in order to ease the editing task of a botmaster. Every file may contain categories dealing with a particular thematic area, such as politics, movies or computer systems. Once the chatterbot has started, all files are loaded into the system’s internal memory thus losing the defined structure. The user input is checked against the patterns of all loaded categories and if a matching pattern is found, the related template is processed. As a consequence, the system reaction to a user input is rather context-independent, since the contents of the utterances exchanged in preceding steps are not taken into consideration. By means of the "that" element a relation between two categories can be established. As shown in the example of Figure 5.7, the template of a category containing a "that" element will be processed only if the preceding system response corresponds to the contents of "that".

```
<category>
  <pattern>I LIKE ANIMALS</pattern>
  <template>
    Do you like dogs?
  </template>
</category>

<category>
  <pattern>YES</pattern>
  <template>
    <that>DO YOU LIKE DOGS</that>
    I like them too!
  </template>
</category>

<category>
  <pattern>NO</pattern>
  <template>
    <that>DO YOU LIKE DOGS</that>
    Do you prefer cats?
  </template>
</category>
```

Figure 5.7: Linking AIML categories with "that"
For the implementation of dialogue management plans consisting of more than two
dialogue steps the usage of the referencing mechanism based on the "that" element
cannot be applied, since the category to apply at step $n+2$ cannot be made dependent
on the category applied at step $n$. For instance, consider the example plan depicted in
Figure 5.8. It describes a dialogue consisting of a sequence of 2 questions (Q1 and Q2)

![Diagram of a dialogue plan with questions Q1 and Q2 and answers A1, A2, A3, A4]

Figure 5.8: An example Dialogue Plan

which can be answered only with "yes" or "no". Any other input produces a repetition of
the current question. The ending states of the plan contain different outputs, according
to every different sequence of answers that can be provided during the dialogue (for
instance the output of A1 is given only if Q1 and Q2 are both answered affirmatively).

For the implementation of this plan another AIML construct needs to be employed.
Categories can be grouped into so called topics (see Figure 5.9). The current topic is
set by assigning its value to the global variable "topic", forcing the system to consider
categories pertaining to the current topic with a higher priority than others. Once a user
initiates a conversation with a chatterbot the system automatically sets the pattern to
Figure 5.9: Using Topics in AIML
the value "connect". Therefore, in the example, the first template to be output is the question Q1. At the same time a new topic is set, namely the one called "Q1ANSWERS". It contains three categories, handling all possible user inputs. If the user says "yes" the question Q2 is output and the current topic set to "ANSWERYESQ1", which causes the system to "remember" that Q1 has been answered affirmatively. Analogously, the input "no" after the question Q1 causes to set "ANSWERNOTQ1" as the current topic. The third category, reacting to the pattern "*" (i.e. all other types of user input), repeats the output of Q1 and leaves the value of the predicate topic unchanged.

5.2.3 Parsing with Program D

A software module called an AIMA interpreter is used to read AIMA objects and provide application-level functionality based on their structure. Currently, several different AIMA interpreters are available, implemented in various programming languages and working on different platforms and operating systems. Some of them are open source, allowing developers to introduce modifications to the standard AIMA element set. For the implementation of VIRMA we used the Java version of the interpreter Program D (available at http://aitools.org/downloads/#programD). Besides offering parsing functionality, the distribution of Program D also includes a Web Server module, called Jetty, allowing us to employ chatterbots in a client-server like architecture over the World Wide Web. We extended the syntax of AIMA by allowing for system calls to web services. For this aim, a new tag was introduced (called "SOAPAccess") and the parser instructed for its interpretation (see Figure 5.10 for the signature).

```xml
<SOAPAccess>
  <serviceURL>
    http://<get name="server"/>
    :8080/soap/servlet/rpcrouter
  </serviceURL>
  <problem>
    urn:aURN
  </problem>
  <command>
    aCommand
  </command>
  <aParameter>
    aValue
  </aParameter>
  <anotherParameter>
    anotherValue
  </anotherParameter>
  ...
</SOAPAccess>
```

Figure 5.10: The signature of the tag "SOAPAccess"
Within the "SOAPaccess"-tag the URL and the URN of the service to be called are specified and possible parameters provided (as a list of attribute-value pairs). Web Services returning a string as a result can be directly included in the output generation part of a "template" or as values of predicates. Moreover, a web service may also return an additional set of categories used for the initiation and management of a subdialogue. As we will explain in the following, our implementation makes extensive use of specifically implemented Web Services, both for the control of an underlying web application and for the improvement of the dialogue behaviour with proactivity.

5.3 Other relevant technologies

Data repositories used may be based on different alternative solutions, depending on the required content type and access modalities. If structures and relationships are relevant, databases, XML repositories and ontologies are appropriate. Otherwise, even text documents locally organised in file systems can be adopted. Other modules of the architecture of a conversational system can be built by using various technologies. For instance, the IR module used in our implementation discussed in the following chapter is based on a commercial inference engine called HYSPRIT [RLK01]. HySpirit is an implementation of probabilistic Datalog based on Horn clauses, used for modelling uncertain facts and rules (similar to Prolog). As for the reasoning module, more complex frameworks for the implementation of expert systems such as CLIPS (http://www.grc.net/clips/CLIPS.html) can be employed. Our approach consisted of using computational facilities of a risk management web application, which takes the user through a Q&A session with predefined questions before producing a risk analysis matrix. It will be described in more detail in the following section.

Risk management refers to the process of measuring or assessing risks and then developing strategies to manage the risks. In an ideal risk management scenario, a prioritisation process is followed whereby the risks with the greatest loss and the greatest probability of occurring are handled first, and risks with lower probability of occurrence and lower loss are handled later. In practice the process can be very difficult, and balancing between risks with a high probability of occurrence but lower loss vs. a risk with high loss but lower probability of occurrence can often be mishandled. A first step in the process of managing risk is to identify potential risks. The risks must then be assessed as to their potential severity of loss and to the probability of occurrence. Afterwards, several techniques and their combinations can be applied to manage the identified risks. They comprise avoidance, reduction, retention and transfer, which means causing another party to accept the risk (as in the case of drawing up insurance policies). For more information see [Car98].
5.3.1 Risk Manager Online

The Risk Manager Online (RMO - available at https://entry2.credit-suisse.ch/cs/business/p/d/irm/index.jsp) is a static expert system in German designed by Winterthur Group (http://www.winterthur.com). The aim of the tool is to provide a self-service application accessible via the internet, allowing Winterthur customers to build an initial risk analysis. It is to be used as a starting point for further face-to-face meetings with professional advisors from the company. The RMO is structured into five subsequent phases, providing also intermediate results at the end of each phase, which can be locally stored and exploited in further processing steps. Once connected to the RMO web site, the user has the options of starting a new session or of watching a short introductory movie, called the "Quick Tour", giving an overview of the aim and usage of the RMO and illustrating the processes and actions to carry out on the basis of a case study (see Figure 5.11). For the management of multiple users accessing the application at the same time, a cookie is also set locally in order to grant the delivery of pertinent data throughout the whole session. The first phase of a new session, called "Introduction", targets the generation of a business profile. First, the business sector in which the user’s company is operating has to be determined. For this aim, a list of 15 entries is displayed.
5.3 Other relevant technologies

Figure 5.12: Selecting the business sector

Figure 5.13: Selecting business activities in the transportation sector
Figure 5.14: The checklist

(see Figure 5.12) and the most appropriate one selected by the user by clicking on the related entry with the pointing device. Examples for items on this list are the Retail Trade or the Transportation sector. Upon confirmation of the selected sector a new list is displayed for the user's attention. In this second step the main activities carried out within the business have to be chosen (see Figure 5.13). For every different business sector a different set of activities is generated. For example, the trade sector leads to a list containing among others, the activities "purchasing", "selling" and "wholesale", whereas the Transportation sector leads to a list including the activities "transportation of people" and "transportation of goods". Several choices can be made from the list, and the confirmation of the selection completes the first phase of the RMO. The second phase, called "Risk Detection", aims at determining an initial set of possible risk factors that may apply to the user. For this task, on the basis of the profile produced, the system generates a list of generic assertions (called "the individual checklist"), which are evaluated according to different levels of appropriateness (see Figure 5.14). For instance, the user may have to provide the level of dependency on its customers or suppliers (in the case of a business pertaining to the sector "retail trade" and exercising the activity "purchasing" or "selling"). For each sentence (typically the list contains about 20 entries, according to the number of activities selected in the previous step), the user may choose one of the following options: "it applies fully", "it applies partly", "it doesn't apply at all" and "I don't know". Some sentences are formulated positively (such as "There is
5.3 Other relevant technologies

Figure 5.15: First results of the Risk Analysis

no unique dependency on one supplier or customer") and some others negatively (such as "Processes which involve the risk of combustion or flammable materials are used in the enterprise") Once again, the choices are made via the pointing device, and a "submit"-button at the end of the page induces the system to move on to the next step (if the evaluation of some sentences is omitted, a pop-up window appears with a short notification). As previously mentioned, a risk portfolio is generated according to the evaluated checklist. The identified hazards are grouped into five categories, namely business, financial, property, liability and personal risks. Their weighting is shown by means of a diagram, in which the categories containing risks with a higher impact on the user’s business (and therefore requiring greater consideration) are placed higher (see Figure 5.15). Just below the diagram the risks appearing in each category are listed. By clicking on a risk a pop-up window appears containing a short definition and possible consequences in case of occurrence. The risk analysis generated can be printed out or saved in the user’s local file system. In the following phase of the RMO (which is called "Risk Assessment"), the list of identified risks can be refined, by highlighting the ones that definitively apply (see Figure 5.16). Based on his/her own opinion, the user can exclude risks from the list and also add other risks which need to be considered in the rest of the risk management process. The fourth phase, called "Risk Coverage", allows the user to
Figure 5.16: Assessing the identified risks

Figure 5.17: Selecting measures against risks
select appropriate measures to cover the risks identified (see Figure 5.17). For this aim, the list produced in phase three is presented again and for each entry a set of actions for reducing or avoiding the risk is suggested. The user can select the most appropriate measures by directly clicking on it or by entering his/her own measure via the keyboard. For every identified risk also a list of possible insurance types is offered. The entries of this list are links to specific pages of the Winterthur web site offering more details on the proposed risk transfer option. In the next step a comprehensive planning of the selected measures can be achieved (see Figure 5.18). Target dates for their accomplishment as well as the names of people responsible are entered in a predefined form which can be saved or printed out. The generated plan represents the basis for the final phase of the RMO, called "Risk Control". The aim of this phase is to draw the attention of the user to the monitoring of the defined schedule. No further interaction possibilities are given.
6 Implementation

The following chapter describes a prototypical implementation of a proactive chatterbot meeting the requirements defined in chapter three and based on the model and architecture discussed in chapters four and five. Within the context of the RMO, the implemented chatterbot assumes the role of an advisor providing assistance and helpful suggestions during the interaction. The general aim is to improve usability by substituting the strict form-based interaction style of the original application with the usage of natural language for both input and output. In this way, the user can request help in unclear or problematic situations and is more naturally guided during the elicitation of the required data. Moreover, on the basis of the user properties collected during the dialogue, the system can proactively change the predefined sequence of interaction steps by adapting it to specific needs and problems arising locally.

In the following sections we will first analyse in more detail the implementation aspects of the prototype. We subsequently provide an example walkthrough based on a real scenario.

6.1 RMO with a virtual advisor

Even though the RMO provides a simple interaction methodology based on the intuitive utilization of a pointing device (typically the computer mouse) for making choices and selections, its usability is rather questionable. At the beginning of each phase a short introductory text is displayed and on every screen requesting user intervention the main tasks are summarised within a few sentences. These help texts are static, in the sense that they do not adapt to the properties of the user and to the choices made so far. Moreover, the instructions do not actually refer to the contents of the selections or to the object of the current processing task. As a consequence, whether or not a choice is correct is often a matter of the user’s own skills of interpretation. During the selection of the user’s business sector, for instance, in some cases it may be hard to recognize the most appropriate one to choose. Problems with the ambiguity of and differentiation between the terms used to represent business sectors may frequently appear (consider, for instance, the sectors "motor vehicle industry" and "transportation"). As a fallback solution the designers of the system have included a sector called "General Enterprise (other sectors)" in the displayed list. It can be chosen by users having difficulties in identifying the most suitable sector. On the other hand, the idea of being treated as a
"generic" enterprise, may induce the user to think at a more abstract level and thus lead to a less tailored risk analysis during the forthcoming phases. Therefore, the interest and motivation may be drastically reduced at the very beginning of the interaction. Similar interpretation problems may also apply while processing the list of business activities. Some entries of the list may appear to be superordinate concepts of some others (as in the case of the activity "wholesale" which might include also other activities offered for selection, such as "purchasing" and "distribution") causing confusion and indecisiveness for most users. Moreover, there is no hint about the number of activities to choose: it is not clear whether a limited selection may lead to a more concise analysis or a wider selection may improve the level of detail of the subsequent processing steps. The considerations made so far can be extended to almost all phases of the RMO. These problems were recognized soon by the designers at Winterthur and the first reaction consisted in setting up a dedicated call centre, offering restricted telephone support to users with problems. Since the application of call centres is often related to a number of drawbacks (for instance, the high costs for both the company and the user, or the consequences of long waiting times), in our approach we concentrated on a different kind of solution for improving the usability of the RMO. The required support should be directly provided by the application in a clear and user-friendly way. Moreover, the user should be able to use a common and intuitive means for expressing his/her problems. Therefore, the utilisation of a chatterbot acting as a virtual advisor appeared to be the most suitable enhancement. Especially the proactive features, leading to more personalised dialogues, can result in a great advantage for the user during the risk analysis. Before starting the implementation we had to carefully specify and design the virtual advisor’s dialogue behaviour and strategy. For this aim, we analysed the way in which professional human risk managers of Winterthur perform risk analysis with their customers. We learned that most of them usually carry out a first risk assessment by using the so called Risk Management CD, which is an offline version of the RMO based on the same sequence of phases. During a face-to-face meeting the RMCD is processed together with the customer. The application is controlled by the advisor, who asks the customer for the required data to input. Examples and more detailed explanations are frequently used to solve problematic requests, and the advisor’s considerations and appraisals can often support the gathering of the needed information. The results achieved with the RMCD, mainly consisting of a risk portfolio and an initial measure plan, are processed and checked in further meetings and discussions with the customer. Similarly, also the virtual advisor accompanying the RMO should have the main aim of supporting a web user in determining the most suitable answers to the application’s requests. But obviously, as discussed in chapter 2, simulated reasoning of computer systems cannot be compared to the reasoning capability of humans. Therefore, only selected properties and behavioural characteristics can be implemented. For this aim, we systematically set up a model of advisory activities based on the outcomes of interviews carried out with 4 professional risk managers of Winterthur, who usually employ the RMCD during their work [ELT04]. The interviews
contained both generic questions about the risk management strategy adopted and more specific questions about the way of introducing and processing the RMCD phases. We also investigated the advisors’ experience for determining common problems encountered by their customers, such as frequently asked questions or difficulties in fulfilling specific tasks of the dialogue. As a result of this preliminary work, we implemented an initial prototype of a virtual advisor guiding the user during the first two phases of the RMO. Its main aim was the substitution of the standard interaction mechanisms, consisting of the utilisation of a pointing device for making selections within web forms, with natural language understanding and output capabilities. By means of the keyboard, the user is able to input answers to the system’s questions in the form of both elaborated sentences or concise utterances consisting of only a few words. Moreover, besides reacting to the system’s requests, the user can always take over the initiative by asking for clarifications or more detailed instructions at any time during a risk management session. At a later stage of the implementation process the Winterthur advisors interviewed were asked to perform functionality tests on the prototype. The tests aimed at recognizing the system’s ability to reach the same quality of results as the standard RMO. Moreover, based on their experience with real users, the professional advisors highlighted the interaction steps in which a more proactive behaviour, consisting of unsolicited advice and support, can be particularly useful for novice users. Most of the outcomes of the functionality tests were used to improve the prototype, whose implementation is described in the following sections.

6.2 VIRMA

The implemented prototype was named VIRMA (Virtual Insurance Risk Management Advisor). It reflects the architecture of a proactive chatterbot described in chapter five, providing a user interface layer on the basis of a web application, components for dialogue management and access to external information sources and a module for proactivity, which monitors the dialogue and interrupts the conversation whenever a proactive intervention is required. The dialogue management component and the basic set of natural language interpretation rules are implemented in AIML (see section 5.2). The primary target of the prototype VIRMA was the implementation of a suitable testbed for proactivity. For this aim, a chatterbot leading to dialogues with a potential for the problem classes identified in section 3.1 had to be chosen. The RMO enhanced with a conversational interface revealed itself as an appropriate basis. First of all, an entire risk management session can be regarded as a persistent problem, to be solved within a conversation. Thorough dialogue planning is essential for guaranteeing the achievement of a minimal set of goals and related user satisfaction. Moreover, advisory conversations are characterised by frequent requests for assistance and clarification by the advisee. Even if the subject area is restricted during the conversation by the selection of profile related
properties (such as the business sector or the business activities), the risk management domain still remains complex, thus leading to the occurrence of local dialogue problems often requiring context-awareness for their solution. Finally, an advisor's efficient behaviour cannot refrain from providing anticipated support to potential user problems. A risk analysis can turn into a long process, thus requiring the advisor to avoid further complications and inconveniences, which would prolong the duration of the advisory process.

VIRMA's dialogue plan was designed on the basis of the interaction steps defined by the RMO. It is restricted to the first two phases, leading to the determination and visualisation of a set of possibly occurring risk factors. In the following, we will discuss the implementation details of the main interaction steps, by pointing out the dialogue contributions entailing to a proactive behaviour of the chatterbot. First, however, we will present the more general aspects of dialogue management and the interfaces to external information sources.

### 6.2.1 The user interface

Figure 6.1 shows a screenshot of VIRMA. The system response is displayed as formatted text, using highlighting mechanisms (bold text or different colors) for sentences or words which are particularly relevant for the user. The user can type his input within a predefined text box (which is a standard HTML element used in forms) and confirm his message by pressing the enter button on his keyboard or by clicking on the appropriate button with the pointing device. Like the standard interaction mechanism of chatterbots, the system response is updated only after the confirmation of the user input. The system output is sometimes extended to the lower part of the browser window. In particular, specific content from the RMO, such as the Quick Tour and the risk category diagram are displayed here.

Once a dialogue with VIRMA is initiated, an interaction session with the RMO is started and kept running in the background. The solicited user input is used to update the RMO process and possible responses are interpreted and incorporated within the dialogue management of VIRMA. This is realised by using deep links, completed with the values of parameters gained from the user input.

Since multiple users may access VIRMA at the same time, for each user a specific session identifier (called the "sessionID") is generated. Through the invocation of a webservice, the sessionID is related to the value of the cookie set by the RMO. In this way it is granted that the current user always receives data from the RMO relevant to his current session (see Figure 6.2 for a visual representation of the relation between a user, VIRMA and the RMO). Indeed, every time that a deep link is generated and sent to the RMO, the web service is called and the cookie related to the user's sessionID is determined.
Figure 6.1: The VIRMA user interface

Figure 6.2: Multiple users, VIRMA and the RMO
6.2.2 Dialogue Management

As introduced above, the dialogue management of VIRMA is based on the processing of a dialogue plan, whose abstraction is represented in Figure 6.3. The dialogue starts with an introduction and an offer to play the quick tour, which is a short flash animation about the process of risk management. In the next steps the user's business sector and business activities are determined. Afterwards, phase two begins with the evaluation of the generated risk checklist. The sentences to evaluate are not presented all at once as happens within the RMO but one by one, giving users the chance to ask for clarification and advice for each assertion. Finally, the risk category diagram is visualised and the list of risk factors shown for the user’s consideration. The individual steps of the plan are described in more detail in section 6.2.5.

Figure 6.3: The Dialogue Plan used by VIRMA
For every state of the plan a dedicated AIML topic has been created. It groups together the categories required for processing the user input at the related state. Moreover, categories concerning the identification of questions and confirmations (which may be positive, negative or neutral, i.e. used to communicate the wish to go on to the next state) are also grouped into individual topics. Every user input is therefore processed in a fixed order. First, the situation specific categories of the current state are checked. If no applicable pattern can be found, the categories of the "CONFIRMATION" topic are browsed in order to determine the user’s intention to move on. If this is not the case, the "QUESTIONIDENTIFIER" topic is invoked and an answer to a possible question provided. Finally, if no response could be generated, the input is classified as "UNIDENTIFIED", thus requesting the user to reformulate his last message. This behaviour, as well as the transition between the states of the dialogue plan, is implemented by the management of a set of predicates, containing possible values to assign to the predicate topic (which determines the set of categories to be considered for pattern-matching). Each topic contains a default category reacting to the pattern "*". This category is only processed if the other categories of the current topic could not be applied. The default category is responsible for assigning the values to the predicates (see Figure 6.4 for an overview). For instance, consider the first step of the dialogue plan, in which

<table>
<thead>
<tr>
<th>Predicate</th>
<th>Links to a topic which is processed after...</th>
</tr>
</thead>
<tbody>
<tr>
<td>next_topic_yes</td>
<td>...a positive user confirmation</td>
</tr>
<tr>
<td>next_topic_no</td>
<td>...a negative user confirmation</td>
</tr>
<tr>
<td>next_topic_done</td>
<td>...a confirmation entailing to move on to the next step</td>
</tr>
<tr>
<td>topic_after_question</td>
<td>...a user question</td>
</tr>
<tr>
<td>topic_after_unident</td>
<td>...an unidentified user input</td>
</tr>
</tbody>
</table>

Figure 6.4: Predicates used for the management of topics

the user is asked about his willingness to see the quick tour. No situation specific categories are contained in this topic (called "CONNECT"), therefore the default category, reacting to the pattern "*", is directly applied to the user input. As can be seen in Figure 6.5, the predicate "topic" is set to "CONFIRMATION" and the last user input forwarded by means of the instruction "</srai> * </srai>". If a confirmation pattern is detected, the next topic will be set to the value of the related predicate (i.e. "next_topic_yes", "next_topic_no" or "next_topic_done"). If not, the topic will be
<aiml version="1.0">
  <topic name="CONNECT">
    <category>
      <pattern>*</pattern>
      <template>
        Introduction and question about quicktour
        <think>
          <set name="topic">CONFIRMATION</set>
          <set name="topic_after_question">CONNECT</set>
          <set name="topic_after_unident">CONNECT</set>
          <set name="next_topic_yes">QUICKTOUR YES</set>
          <set name="next_topic_no">QUICKTOUR NO</set>
          <set name="next_topic_done">QUICKTOUR YES</set>
          <srai>*</srai>
        </think>
      </template>
    </category>
    <category>
      <pattern>BACKAFTERQUESTION</pattern>
      <template>
        Repeat question about quicktour
        <think>
          <set name="topic">CONFIRMATION</set>
          <srai>*</srai>
        </think>
      </template>
    </category>
    <category>
      <pattern>BACKAFTERUNIDENTIFIED</pattern>
      <template>
        Request for reformulation and Repeat question about quicktour
        <think>
          <set name="topic">CONFIRMATION</set>
          <srai>*</srai>
        </think>
      </template>
    </category>
  </topic>
</aiml>

Figure 6.5: The contents of topic CONNECT (shortened for the reader’s convenience)
set to "QUESTIONIDENTIFIER" (not shown in the figure, since the assignment is carried out in topic "CONFIRMATION") and the user input forwarded again. In case of a user question, an answer will be generated and displayed (see section 6.2.4 for details about the answer determination process) and the topic set to the value of the predicate "topic_after_question" (which is, in this example, the topic "CONNECT"). By using the "srai"-tag, the correct entry point in "topic_after_question" is set. Indeed, if a question was recognized, the category reacting to the pattern "BACKAFTERQUESTION" will be processed. In this example its template contains the repetition of the question about the quick tour. If no question was detected, the next topic will be set to the value of "topic_after_unidentif" and the entry point will be a different category, addressed by the pattern "BACKAFTERUNIDENTIFIED". Its template will display a request for reformulation (typically a variant of the question "I am sorry but I did not understand you...") and also a repetition of the question about the quick tour. As we will see below, the entry points after questions or unidentified input may also contain some proactive actions, trying to provide better support to the user. Indeed, if he has posted a question or an unrecognized message, one can assume the existence of a local problem.

In the following, the topics "CONFIRMATION" and "QUESTIONIDENTIFIER" are described in more detail, afterwards the main characteristics of the individual phases of the dialogue with VIRMA will be explained.

6.2.3 Confirmation

The transition between the states of the VIRMA dialogue plan often require an explicit user confirmation message, informing the system of the intention to move on. In order to avoid the implementation and thus repetition in every state of categories handling those messages, a specific topic was created. It is mainly responsible for the control of the system behaviour in the following cases:

- **affirmative confirmations**, i.e. positive answers such as "yes" or "ok" required for instance as answers to questions like "do you want to watch the quick tour?" or "Are you ready to move on?"

- **negative confirmations**, i.e. negative answers such as "no" or "not yet"

- **neutral confirmations**, i.e. messages such as "I am finished" or "I want to go on" uttered for instance for the interruption of the quick tour or after having concluded the choice of business activities

For every confirmation variant, categories processing synonyms of the intended message are included and managed by means of the < srai > recursive operator. As we have seen in the example in the previous chapter, once a confirmation has been recognized, the topic to use next is determined by the values of specific predicates ("next_topic_yes", "next_topic_no" or "next_topic_done"), which are set before the invocation of the
"CONFIRMATION" topic. The default category, applied when no confirmation is input, is responsible for forwarding the current user input to the "QUESTIONIDENTIFIER" topic, described in the next section.

6.2.4 Question Identifier

The efficient handling of user questions is one of the most crucial enhancements of VIRMA, since the RMO provides very limited support to users' enquiries. Questions can be posted at any time during the dialogue, and often require different answers depending on the currently valid context in which they are made. User properties, the dialogue history and the currently discussed topic may all have relevant influence on the determination of the most suitable answer to provide. Adapted answers to user requests form the basis for the proactive solution of local problems, defined within the input used to express the question. Therefore, an implementation of a query expansion model had to be included in VIRMA. First of all, the question needs to be identified from the user input. For this aim, about 300 categories recognizing patterns of questions (such as sentences starting with "What", "Which", "Can", "Is" and so on) have been included in the topic "QUESTIONIDENTIFIER". The identified question leads to the determination of a problem definition, consisting of a characterisation of the problem by means of four predicates, namely "type", "matter", "subject" and "content". The parameter "type" is used for determining the kind of information requested. Usual values are "data", "explanation" or "comparison". The parameter "matter" specifies what exactly is being requested. For instance, common values are "definition", "process", "feature" or "lastAction". The last two parameters provide a more detailed characterisation of the problem: "subject" is used for specifying the topic of conversation and "content" refers to the object of the request. The assignment of values to its pertinent parameters is carried out by both analysing the grammatical structure of the sentence and by considering contextual information. For instance, during the business activities selection phase, the user may post the question "What is meant by manufacturing?". The situation specific input for that stage of the interaction only expects the selection of a business activity (it includes patterns such as "please select *\text{" or "I choose *\text{"), therefore no valid match can be found, and the "QUESTIONIDENTIFIER" topic is set. A specific category (see Figure 6.6) ascertains that the sentence is indeed a request for an "explanation" (typically expressed by the pattern "What is meant by *\text{"). Specifically, a "definition" is being sought, referring to a business activity (the value "activity" is assigned to the predicate "subject" before entering the "QUESTIONIDENTIFIER" topic), namely "manufacturing" (which is the value of the wildcard in the pattern). As you can see in Figure 6.6, the problem definition generated as well as the last user input are used as a list of attribute-value pairs in a call to a web service. Other contextual data collected thus far is included in the parameter "contextList". It consists of a comma-separated list in which each item has the form "contextAttributeName, contextAttributeValue.". The list is updated
Figure 6.6: The SOAP Call in the case of a user question (adapted for the reader's convenience)

during the dialogue, whenever a new user property becomes known to the system. For instance, the "contextList" is extended with the string "selected_Business_Sector, Motor Trade" after the business sector selection phase for users having chosen the "Motor Trade" sector. The web service called consists of a case-based reasoning based problem solver. An answer to the user's enquiry is retrieved from a case base, consisting of a repository of problems and related solutions which occurred previously. Problems from the base are characterised in the same way as problems occurring in a dialogue. They are described by the four parameters type, matter, subject and content as well as by a natural language sentence. Moreover, specific attributes are used to characterise the circumstances in which the problem solution is particularly applicable (such as the level of expertise or the business sector). A problem solution contained in a case was useful (and used) in the past; hence, the more the current problem is similar to the old one, the more likely it is that the old solution applies. The similarity of a case to a current problem is determined in three subsequent steps. In the first step, the four parameters of the problem definition are compared. If a perfect match is found, step 2 is skipped. If less than 4 parameters apply also the last user input is considered in step 2. By means of a probabilistic information retrieval based approach the probability that the user text matches the text describing a case is computed and used as a retrieval weight. In case of an acceptable result (i.e. a weight which is above a predefined threshold), in step 3 also the contextual parameters are compared. The more parameters are found in common, the more similar the case will be to the user problem. The solution contained in the most similar case is delivered back as a system response. For more details about the process described so far see [LFTN04]. The initial case base contained about 500 different prob-
lem definitions and related solutions. For the collection of the cases, several knowledge sources were used, including the FAQs and glossaries of various insurance and financial web portals as well as the interviews carried out with professional advisors. The cases are divided into different categories, according to the topic they refer to. For instance, cases about pertinent laws, risk factors, and insurance types are included, but also FAQs about business sectors and activities, general financial and insurance concepts and explanations of specific processes and functions of the RMO can be found. Currently, the case base is regularly updated by examining the system log files: problems which could not be solved are considered as a new case after the identification of an appropriate solution. A more detailed description of the case base used is given in [LKT04].

6.2.5 The Dialogue Plan

In the following, the implementation of the dialogue plan used by VIRMA is analysed. For every state, the situation specific input is characterised and the main circumstances leading to proactivity are discussed.

The Quick Tour

VIRMA starts the conversation with a short introduction about the aim of the forthcoming dialogue and an invitation to post questions in case of support needed. The interaction modalities are also briefly explained and a short sentence asking if the user wants to see the quick tour concludes the utterance. The meaning of quick tour has deliberately not been introduced before. The aim here was to stimulate a novice user into posting a question (something like "What is a quick tour?"). Indeed, even if clearly stated at the beginning of the conversation, beginners often refrain from taking over the initiative and using the opportunity to ask for support. This is partly due to the difficulty of predicting the system behaviour in case of a question and to the fear of interrupting the flow of the interaction conducted by the system. Nevertheless, the user's behaviour in these early steps of the conversation is used for making assumptions about his profile and attitudes. For instance, if he utters questions and if he agrees to see the tour, a novice profile can be assumed and used for decisions in later steps of the dialogue. Otherwise, if he declines to see the quick tour a higher level of expertise can be expected. Also the contents of possible questions can be relevant for inferring specific user properties. For instance, if he first asks about the length of the movie and then refuses to see it, he probably hasn't got too much time to spend on the risk analysis session. Obviously, these assumptions need to be confirmed in the remainder of the conversation. They are treated as heuristics rather than definite rules. The quick tour is shown in the lower part of the window (provided that the user has the Flash plug in installed on his system) and can be interrupted at any time by inputting a neutral confirmation message. If the user refuses to see the tour, VIRMA offers again its availability to respond to questions or to
provide explanations about the ongoing risk management process. If the user still avoids asking questions and directly wants to start the analysis, then the user’s level of expertise is set to high. This will influence the length of the texts used to introduce forthcoming dialogue steps and inhibit specific proactive support (see below).

The Business Sector

Compared to the RMO, VIRMA offers an additional means for the determination of the economic sector. If the user has been initially categorised as novice, the list of sectors is not presented for his attention but instead he is invited to communicate the profession carried out by his business. A dedicated web service is responsible for mapping the profession entered to one of the 15 main business sectors. For instance, if the user states that he is the owner of a restaurant (e.g. by inputting "I run a restaurant" or "I am a restaurateur") the catering sector will be proposed as selection. The mapping is based on a table containing about 200 different professions. For each entry possible synonyms are also considered (such as "discotheque" or "dance bar" for "night club"). If the profession entered is not known to VIRMA, then, instead of repeating the same request, the list of sectors is shown. Finally, in the case of unidentified input, the system output is extended with suggestions and examples.

The Business Activities

The dialogue step handling the selection of business activities also provides some proactive enhancements with respect to the RMO. Indeed, as reported by the human advisors from Winterthur who were interviewed, almost every user requires support at this stage of the analysis. Questions like "Which activities should I select?" or "Am I supposed to select the activity ...?" are very frequent. Therefore, we included in the case base several different answers to those questions, taking into account specific properties of the user such as the chosen sector or other selected activities (for instance, in the trade sector, users which have already chosen the activity "Retailers/Kiosk" should also select "Purchasing", since the goods offered in their shop have to be somehow acquired). Moreover, if the user’s profession is known, a specific web service is invoked. It computes a list of activities that have been chosen in previous sessions by users with the same profession. As a result, the list of activities presented to the user contains an initial selection. It may be simply confirmed or slightly updated with more appropriate choices before going on to the next interaction step. The selection and deselection of the activities can be flexibly carried out by entering complete sentences (e.g. "Please select the activity wholesale") or by just inputting the name of the activity. Moreover, multiple activities can also be included in one sentence (e.g. "I choose production, administration and consulting").
The Check List

As previously mentioned, the risk check list is not presented all at once as done by the RMO. Instead, every assertion is output individually by VIRMA, giving users the chance to request support before deciding on the most appropriate ranking. For this aim, another web service had to be implemented. It is responsible for parsing the HTML web site containing the table with assertions of the risk check list and extracting the sentence to include in VIRMA’s output. The text containing the sentence is sometimes extended with relevant information. Based on the profile collected, suggestions or examples are offered to novice users with the aim of easing their task. Consider for instance the following sentence: "The principal is personally concerned with safety issues and organises a risk assessment at least once a year". In case of medium sized enterprises, VIRMA adds a comment to this sentence, remarking that besides the principal, also other employees can be considered as responsible for safety issues. The size of a business can be for instance inferred from the profession (a road construction business is seldom run by a small enterprise) or from the combination of sector and chosen activities (within the "construction trade" sector, performing the activity "building construction", requires at least a medium sized enterprise). Moreover, some sentences can be directly evaluated by VIRMA, thus reducing and optimising the time spent on the analysis. In the transport sector, a sentence appearing frequently in the risk check list says that "For the execution of the business activities the means of transport are appropriate and safe". The evaluation of this assertion obviously intends to verify the occurrence of a transportation risk (e.g. the use of inappropriate transportation means which risks the safety of freight/persons, causes loss of goods and damages upon loading and unloading). A cash transport company, handling the safe transportation of money, would certainly not be able to carry out its business with inappropriate means of transport. Therefore, the evaluation of this sentence appears superfluous and can be skipped. In the walkthrough provided below, other examples of enhancements to the risk check list are given.

Phase Five: The Diagram and Risk Factor List

The final step of VIRMA consists of the visualisation of the risk assessment diagram. According to the Winterthur advisors, novice users occasionally have problems in correctly understanding the aims and contents of the diagram. Therefore, we implemented another web service, which, similar to the parsing of the checklist, analyses the HTML page in order to identify the risk category and some pertinent risk factors with the highest impact on the user’s business. VIRMA proactively outputs a recommendation, in which the outcomes of the web service call are pointed out. Besides asking for the meaning and possible consequences of the identified risks, the user can also enquire about applicable insurance types and arrange a personal meeting with a human advisor.
6.2.6 Conclusions

The development of the prototype took about two years of work. Even if an acceptable status could be reached, several improvements are still feasible. This applies especially if you consider that the variety of possible natural language expressions can lead to almost unlimited alternative dialogues and advisory sessions. Moreover, the high diversity of user stereotypes and related profiles can extend the set of applicable proactive system actions, leading to an improvement of the dialogue behaviour and of the results of the application. To achieve this aim, the session log files will be continuously monitored and analysed for as long as the prototype is still available on the internet, thus determining possible enhancements and future improvements. VIRMA has been evaluated in the context of usability tests. The outcomes of the tests performed mainly showed that dialogues featuring proactive system actions led to more satisfied users and more appropriate results. The prototype evaluation will be discussed in more detail within the next chapter.

6.3 Example Walkthrough

This section contains a step by step description of a real usage session with VIRMA. Since the application is completely in German, the user input and the most relevant messages of the system output have been translated into English and reported in the following for the reader's convenience. The original system responses and images are shown by means of relevant screenshots. Moreover, also the most relevant internal actions and processes carried out transparently to the user, are described.

The user considered for this example is a gas station owner with moderate knowledge about risk management and limited experience with computer systems and the internet. Once connected, VIRMA introduces the aims and goals of the forthcoming dialogue and gives some explanations about the natural language based interaction methodology. For this aim, examples for possible inputs are given and instructions about where to input and how to submit the messages are included. Finally, VIRMA offers to play the quick tour. The question is displayed in bold face, in order to induce the user to consider it with a higher priority than the rest of the output (see Figure 6.7).

Since the user is chatting with VIRMA for the first time, he replies with "What is a quick tour?". The question is categorised as a request for a definition, issued during the "introduction" phase and referring to the content "quick tour". The problem definition generated is searched for within the case base and a matching problem solution is found. The solution (consisting of the sentences "The quick tour is a short movie giving an overview of the aim and usage of the application. The processes and actions to carry out within the four phases are illustrated on the basis of a case study.") is output and the last question ("Do you want to see the quick tour?") repeated (Figure 6.8).

At the same time, the predicate "questionDuringIntroduction" is set to the value
Figure 6.7: The Welcome Screen of VIRMA

Figure 6.8: Defining the Quick Tour
"true". It will be used for an assessment of the user’s level of expertise. By the input "yes, please", the user agrees to watch the tour, which is promptly displayed in the lower part of the window. The system output is updated with the message "The quick tour is played below. Once finished, or if you want to interrupt the movie earlier please tell me that you want to go on" (Figure 6.9). At the same time, another predicate called "quickTourWatched" is set to "true". Since the predicate "questionDuringIntroduction" is also set to "true", the user’s level of expertise can be assumed to be "low". This assumption will contribute to the generation of more appropriate system responses at later steps of the dialogue.

![Image](image_url)

Figure 6.9: Watching the Quick Tour in the lower frame of the browser window

After having watched the quick tour, the user communicates his readiness to go on with the risk analysis by inputting the sentence "I’m done". The frame showing the quick tour is cleared and a new system output displayed (see Figure 6.10). VIRMA informs the user that this stage corresponds to the beginning of the risk analysis process. The system needs to know more details about the user’s enterprise in order to build a more appropriate risk assessment. For this aim, the user is asked to communicate his business sector. He has the options to either name it directly or to input his profession, in order to let the system automatically identify the pertinent sector (this option is made available
Figure 6.10: After the Quick Tour

since the user’s level of expertise is categorised as "low".

The user’s input at this time corresponds to "I own a gas station". Based on an internal mapping between profession types and business sectors, the system infers the most appropriate mapping (in this case, the sector "motor trade") and offers it for confirmation in the successive step (Figure 6.11).

Figure 6.11: A request for confirmation of the business sector motor trade

The user agrees with the suggested sector and the dialogue moves on to the business activities selection phase. For this aim, a list of 13 activities is displayed for the user’s attention (Figure 6.12).

Since the profession is known, a preselection can be offered. The names of the activities which other gas station owners usually select are displayed in bold green and marked with a "+" icon. The user can now modify the selection by adding or removing entries or ask questions about the listed activities.
6.3 Example Walkthrough

![Image of a computer screen showing the VIRMA application]

Figure 6.12: The selection of business activities

One of the selected activities is "training". Since the user isn’t sure about its actual meaning, he posts the following question: "What is exactly meant by training?". The retrieval capabilities of VIRMA determine a first rather vague answer, namely "Training embraces all activities which deal with knowledge transfer". Dissatisfied with this answer, the user tries asking again, by slightly reformulating the question: "Which fields are covered by the activity Training?". VIRMA replies with "Training concerns the education of your employees as well as instructions for your apprentices". Finally, the user asks directly for a recommendation: "Should I really select training?". The answer confirms that training is an activity selected often, especially in cases in which the business has employees to be instructed and guided.

Moreover, the user recognizes that the activity "Shop" has not already been selected, but should be added, since he also sells food and beverages in his gas station shop. On the other hand, car body shop maintenance is not performed, and the related activity is selected. Thus, he updates the list by inputting the sentence "please select kiosk and remove body shop". The list can now be confirmed and the dialogue moved on to the next phase (see Figure 6.13, in which the user inputs the sentence "I'm done").

At this point, all relevant information has been collected for the generation of a checklist. This is communicated to the user by means of the system message shown in Figure 6.14. The user is informed about the aims and duration (about ten minutes) of the
Ihre Eingabe: bitte wähle Kook und entnehme einkasse

VIRMA: Die Liste wurde nach ihrer letzten Auswahl aktualisiert.
Sie können jetzt weiterhin diese Liste nach belieben verändern, indem Sie mir mitteilen, wo ich Aktivitäten erfreue oder auswählen soll.

Wenn Sie bei Ihrer Auswahl Hilfe benötigen oder Aktivitäten genauer erklären möchten, dann zögern Sie bitte nicht, mich zu fragen.
Teilen Sie mir bitte mit, wenn Sie mit der Auswahl fertig sind.

- Einkauf
- Vertrieb, Verkauf (Marketing)
- Verkauf
- Schulung
- Service, Unterhut, Wartung
- Vorwärter
- Verkaufstradition/Shop, Kiosk
- Nutzermietung (Leasing)
- Tankstelle
- Werkstatt
- Einkasse mit Sortenwerk
- Reinigung (Wäscherei)

Ich bin fertig
übermitteln

Figure 6.13: A modified business activities selection

Ihre Eingabe: Ich bin fertig

VIRMA: Gut gemacht!

Nun habe ich genugend Wissen, um einige Aussagen zu treffen. Ihre Aufgabe wird es sein, diese Aussagen zu bewerten und mir mitzuteilen, ob meine Annahmen korrekt waren.

Dieser Vorgang wird in etwa zehn Minuten in Anspruch nehmen.

Falls Unklarheiten auftreten, kann ich gerne bereit auf Ihre Anfragen eingehen.

Sind Sie bereit?

übermitteln

Figure 6.14: Before entering the checklist phase
forthcoming phase. Once again, full support is offered in case of emerging doubts or ambiguities. A confirmation is requested for entering the checklist phase.

The first sentence to evaluate deals about the net operating profit of the last three years (Figure 6.15). The user has to assess if the predefined targets have been reached and if there is no room for further improvements. He has three choices, by stating that the sentence fully applies, that it partly applies or that it doesn’t apply at all. Moreover, the user can also skip this sentence, leaving it not evaluated. The user assesses this sentence as partially applicable.

![Figure 6.15: Assessing the net operating profit of the last three years](image)

The following sentence to assess refers to security issues. The assertion states that the
principal is personally involved with the security of the business by performing tours of
the company premises at least once a year. Since the user decided to delegate security-
related concerns to his son, he wonders if the sentence really refers to the principal only.
The system answers promptly by specifying that the term "boss" in the sentence is meant
in a broader sense. It refers more to the availability of a safety representative within the
business (Figure 6.16). The sentence to evaluate is reported again and the user asked for
input.

The interaction goes on similarly for the following 24 sentences. The actual checklist
foresees 28 sentences to evaluate. Two of them are automatically assessed by the system
and not shown to the user in order to shorten the dialogue. The skipped sentences refer
to the handling of combustibles within the business and to the eventuality of products
or services exported abroad. The decision to automatically evaluate and skip these
sentences is taken in accordance with the selected business activities (cf. Figure 6.13).
Indeed, it is quite obvious that combustible material is employed in a business in which
the activity "petrol station" is performed. Similarly, since the activity "Leasing" is not
chosen, it is unlikely that a petrol station exports its products abroad. The evaluation
of the remaining sentences is omitted in this walkthrough.

Finally, after the evaluation of the last sentence of the checklist, the results of the risk
analysis are shown. By means of a diagram (see Figure 6.17) the main risk categories
are visualised and rated on the basis of their impact on the business. The category
"property risks" is pointed out in the system output, as it exhibits the highest exposure.
Probably, this result may sound quite predictable if one considers the nature of the
business run by a petrol station owner. But probably, much more interesting appears
to be the risk category "financial risks", as it has an exposure almost as high as the
previous one. The user scrolls down the frame displaying the risk diagram as suggested
by VIRMA's response. He discovers a list of identified risk factors, sorted on the basis
of the five categories. Among the financial risks, one factor is particularly interesting for
the user, as he cannot figure out its actual impact on his business. The user asks about
the meaning of the factor "diminishing cash-flow". The system replies with the following
output: "Diminishing cash-flow applies if the net profit plus depreciation (the cash flow)
falls under 5% of turnover. Possible Consequences are reduction or cancellation of bank
credits, insufficient liquidity, lack of investment and reduction in demand for products"

The final request of the user is meant rather generically. In response to the question
"How can I avoid a risk?", VIRMA suggests making a preliminary distinction. For risk
factors which do not require particular preventive measures, it suffices to make himself
and the employees aware that the risk may exist and that appropriate precautions can
be taken to avoid it. All other risks should be handled with measures which comprise
taking out insurance contracts.

The following steps of the risk manager online deal with the issues raised by the user's
last question. For evaluation purposes, VIRMA's risk analysis ends at this point.
Figure 6.17: The results of a risk analysis with VIRMA
Chapter 6. Implementation
7 Evaluation

A common method for testing the efficiency of natural language based systems is the Turing test [Tur50, Rap00]. Its aim was to determine whether or not a computer can be said to think like a human brain. In an attempt to cut through the philosophical debate about how to define "thinking", Turing devised a subjective test to answer the question, "Can machines think?" and reasoned that if a computer acts, reacts and interacts like a sentient being, then it can be indeed called sentient. The test consisted of giving an isolated human interrogator the task of distinguishing between a human and a computer based on their replies to questions that the interrogator poses. After a series of performed tests, the interrogator attempts to determine which subject is human and which is an artificial intelligence. The computer’s success at thinking can be quantified by its probability of being misidentified as the human subject.

Despite the Turing test’s popularity, many criticisms influenced its acceptance [Shi04, OD05]. Even if no computer system ever passed the test, many chatterbot-based attempts came very close. At least, these are the assumptions made for the winners of the "Loebner Prize Contest", which is a well-known Turing-like test offering awards "for the first computer [system] whose responses were indistinguishable from a human’s" (http://www.loebner.net/Prizef/loebner-prize.html). Since mainly addressing participants whose systems merely employ pattern-matching technologies, the contest was also subject to numerous critical discussions [Shi94]. On the other hand, no clear evidence of Turing tests performed on intelligent conversational agents can be found in contemporary research.

Instead of pursuing a proof of intelligence, in our approach, we aimed at evaluating VIRMA as a possible improvement for the usability of online applications which offer a natural language based interaction. Therefore, we had to design a different evaluation strategy, based on two subsequent steps. The first one consisted of the analysis of more functionality related tests. For this aim, a restricted group of professional risk advisors was asked to solve problems with VIRMA on the basis of a predefined scenario. Based on the outcomes of these functionality tests, the system was improved and prepared for a more usability-related evaluation, in which users were asked to test the system on the basis of their own real background and business environment. The outcomes of both steps will be described in the following sections.
7.1 Functionality Tests with Domain Experts

After an initial implementation phase, it was necessary to examine VIRMA’s ability to achieve the same quality of results as the standard RMO. For this aim, a group of six professional risk advisors, who had partly contributed to the design and implementation of the Risk Manager Online, were invited to perform test sessions with VIRMA. Each advisor was provided with a predefined scenario, consisting of a detailed description of a running enterprise. During a first session, the tester had to take on the role of the managing director of the business described, and perform a risk analysis with VIRMA. The results achieved were compared to a baseline session performed with the Risk Manager Online. As expected, no decisive differences in terms of recognized risk factors and their weighting were ascertained, providing hints that VIRMA’s control of the reasoning process was successfully implemented. Moreover, in the context of a second test session, each advisor had to devise his own application scenario, by freely choosing a combination of business sector, activities and evaluation of the check list. The aim of this step was to let the tester concentrate more on the aspects of the interaction rather than on the contents of the performed analysis.

After the test sessions the advisors were interviewed with the aim of learning about their impressions and judgements on the alternative interaction mode offered by the application. Only two of them were not satisfied at all with the natural language-based interface. Their comments were characterised by the assumption that users would basically not profit from the conversational facilities offered by VIRMA, since the form-based control of the RMO allows interaction to be faster and more concise with regard to the intended results. Their considerations could be partly justified by analysing the log files of their test sessions. Indeed, their interaction style was characterised by shorter inputs, consisting of only commands and single words. They rarely employed full natural language sentences, thus missing the opportunity to exploit the extended interaction possibilities offered by a conversational assistant. The more positively impressed advisors confirmed our assumptions. Indeed, they realised that most of the problems encountered during face-to-face meetings with their customers are based on misunderstandings and need for clarification, which the standard RMO can not fulfill. Their only concern regarded the question if users would be willing to establish a sufficiently trusting relationship with VIRMA compared to a human advisor. The exposure of personal and delicate problems, frequently occurring while talking about details and faults of a business, is only forthrightly carried out in confidential situations. The credibility and reliability of a human advisor is barely possible to simulate in the context of a synthetic conversational partner.

The interviews after the tests also provided invaluable hints on the dialogue phases which could particularly benefit from a proactive behaviour. Based on their experience with real users, the professional advisors highlighted the interaction steps in which unsolicited advice and support can be particularly useful for novice users, and a more concise
formulation of the advisor’s contribution can foster the analysis in case of more expert users. These considerations led us to incorporate facilities for the identification or estimation of the user’s level of expertise in VIRMA’s user modelling processes. Heuristics were added with the aim of extending support or decreasing the interaction effort required depending on the user’s profile. After the implementation of the identified extensions, usability tests were performed, addressing owners of running enterprises.

7.2 Usability Tests with Real Users

Through a newsletter issued by a regional chamber of commerce and industry several hundreds of small and medium enterprises were invited to perform online risk management sessions on the basis of their own scenarios (i.e. combination of business sector and activities). The newsletter provided a link to a website containing an introduction to VIRMA’s risk analysis process and an explanation about the aims and targets of the evaluation scheme. After the performed interaction, the users had the option of filling in an online questionnaire with multiple choice questions, providing feedback about the performed session. The first part of the questionnaire contained generic questions aiming at determining the user’s attitude and experience with internet applications and conversational systems. The second part, consisting of 5 questions, was more specifically intended to assess the analysis carried out in terms of different criteria such as speed, precision of results, helpful advice and so on.

Two months after the publication of the newsletter, the evaluation web site was removed from the internet. The log files revealed a participation in the evaluation process of about 100 different users, carrying out interactions with VIRMA. Almost 40% of the users performed a thorough conversation, reaching the results of the risk analysis and thus the diagram displaying the weighted risk categories. Only 19 of them provided a completely filled in questionnaire, which were considered for a statistical assessment.

The 19 sessions were divided into two sets, one containing dialogues featuring proactive problem solving (8 in total) and one without (11). In this way, a comparison could be performed, clarifying the advantages or the disadvantages of proactivity. The users were also categorised according to their level of expertise. 10 users were considered as "novices" and 9 as "experts". The classification was performed on the basis of the answers provided within the first part of the questionnaire (specifically by the evaluation of questions such as "How often do you access the internet?" or "Have you ever interacted with a virtual conversational assistant on the web?") and by VIRMA’s user profiling processes.

The following sections will describe the outcomes of the evaluation of the second part of the questionnaire. For each question, two diagrams will be presented, the first one depicting the distribution of the answers according to the four user classes (i.e. novices, experts, reactive and proactive dialogues) and the second one visualising the relative sum of users providing a specific answer for the classes reactive and proactive dialogues.
7.2.1 Usefulness of the system output

The first question of the second part of the questionnaire aimed at providing an assessment of the user’s impact on VIRMA’s messages introducing a new interaction step (and not the answers to specific user questions). Figure 7.1 shows how many users out of the 19 sessions assessed decided to select each individual answer. 10 of them expressed their satisfaction by choosing the third answer. Moreover, Figure 7.2 clarifies that users having experienced proactive behaviour were relatively more satisfied than users who had not.

![Graph 1](image1.png)

Figure 7.1: The assessment of VIRMA’s dialogue contributions

![Graph 2](image2.png)

Figure 7.2: The relative sums for question 1
7.2.2 Quality of system responses to user questions

The quality of Virma’s answers to user questions was evaluated by means of the second question. The majority of users were only partly satisfied with its answering capabilities (see Figure 7.3), thus prompting us to head for an improvement of this task. As shown in Figure 7.4, proactivity influenced the quality of answers in the case of users being completely satisfied (i.e. those who chose answer 3).

![Figure 7.3: The quality of VIRMA’s answers to user questions](image1)

![Figure 7.4: The relative sums for question 2](image2)
7.2.3 VIRMA’s own initiative

By means of the third question we were able to assess the users’ impact on VIRMA’s initiative. Figure 7.5 clearly shows that users were able to perceive situations in which initiative alters the expected dialogue flow, especially in the case of novices. As further confirmed by the results displayed in Figure 7.6, proactivity led to improved satisfaction. Almost 90% of users experiencing it selected the second answer, while 70% of users having strictly reactive dialogues desired further support.

![Graph showing the assessment of VIRMA's initiative](image)

Figure 7.5: The usefulness of VIRMA’s dialogue initiative

![Graph showing the relative sums](image)

Figure 7.6: The relative sums for question 3
7.2.4 Quality of results

An interesting factor is also the extent to which proactivity influenced the overall quality of the risk analysis results. This is measured by evaluating the fourth question (Figure 7.7). The majority of users were only partly satisfied. In the case of novices, no difference between proactive and reactive dialogues can be found. Figure 7.8 confirms that proactivity, even if improving the quality of the dialogue, is not a determinant for the quality of results. They obviously depend on the reasoning performed by the RMO, and not on the way in which information is elicited.

![Graph showing the quality and usefulness of risk analysis performed with VIRMA](image1)

Figure 7.7: The quality of VIRMA’s risk management

![Bar chart showing relative sums for question 4](image2)

Figure 7.8: The relative sums for question 4
7.2.5 Session length

Finally, question 5 aimed at assessing how content the users were with the duration of the risk analysis performed with VIRMA. First of all, as shown in Figure 7.9, the number of completely dissatisfied users was irrelevant. More interesting are the results for answers 2 and 3. The number of users judging the speed as appropriate is twice as much in the case of merely reactive dialogues, meaning that proactivity led to a slowdown of the overall process.

![Figure 7.9: The length of the interaction with VIRMA](image)

On the other hand, Figure 7.10 clarifies that 50% of users experiencing proactivity were very satisfied with the duration as opposed to only 20% of users whose dialogues were strictly reactive. This apparent contradiction can be explained by having a closer look within the dialogue log files at the kind of proactivity taking place. Users choosing answer 2, had been supported in the business activities selection phase, but had to intensively modify the list of preselected entries, thus taking up some additional time. On the other hand, users opting for answer 3 were proactively supported during the checklist, i.e. by automatic assessments. Therefore, the proactive preselection of business activities still needs to be improved.
7.3 Lessons learned

The evaluation of the VIRMA prototype showed that the proactive expansion mechanisms introduced led to an increased overall effectiveness of the system. Even if specific tasks still leave room for improvements, the impression is that higher user satisfaction could be achieved. Beyond the questionnaire, relevant clues could also be gained from the analysis of the dialogue log files. As in the example of the question about the appropriateness of VIRMA’s speed, the assessment could be better clarified by observing the actual circumstances which led to specific choices of the answers. The quite unsatisfactory answering behaviour of VIRMA can also be partly justified by analysing the log files. Indeed, in the majority of cases, inappropriate or missing answers were due to a defective language understanding and problem definition identification rather than to ineffective information retrieval.

Apart from integrating the results of the questionnaire evaluation, the log files deliver also interesting clues about the dialogue behaviour of users. As noticed in the case of the two dissatisfied professional advisors discussed in section 7.1, also several users involved in the usability tests adopted relatively short input sentences or single words during their interactions. This phenomenon can be partly explained by the lack of anthropomorphisation occurring for these users. Indeed, the missing visualisation and personification of VIRMA, as well as the quite unusual means for performing natural language conversations (i.e. the keyboard), may prejudice the user against inputting full natural language sentences. Moreover, less typing allows the user to save time, which is always insufficient in the context of spontaneous test procedures.

Even if the assessment of questionnaires can be considered as a completed process, the evaluation and analysis of log files is still an ongoing process. Indeed, new problems, requirements and behaviours can be constantly observed within additional usage sessions, which therefore represent a continuous source for improvements and enhancements.
8 Conclusions and future work

With the aim of enhancing cooperativity of natural language based interfaces, in this work we devised a model of proactive behaviour addressing the solution of specific dialogue problems possibly occurring during a conversation. After having briefly browsed through the history of artificial intelligence, discovering the roots and first developments of conversational systems, we observed that two basically opposed approaches for the implementation of natural language based systems can be found nowadays. The first one, based on techniques derived from computational linguistics, is characterised by the use of a semantic grammar mapping the constituents of the user input to semantic entities, which encompass the meaning and thus the intention of the user utterances. The price to pay for efficient interactions, based on intelligent reasoning and mixed-initiative dialogues, consists of a complex implementation and a rather difficult portability of the built systems to additional domains and scenarios. For these reasons, we preferred to follow the directions defined by the second class of approaches to conversational systems (i.e. chatterbots), based on the employment of string pattern-matching for the selection of a rule, which produces the most suitable system reaction to the user’s input. By further analysing case studies of chatterbots featuring specific problem solving behaviours, we derived a taxonomy of problem classes, whose solution as a whole fit the requirements for a generic proactivity model to be included in chatterbot systems. The resemblance of their dialogue management strategy to the sequential processing of finite state machines, prompted us to adopt automata theory for the achievement of a formal definition of chatterbots enhanced with proactivity. By means of a stepwise approach, each enhancement addressed the solution of one particular problem class, leading to a final automaton, based on non-determinism and on the properties of augmented transition networks. The description of a suitable architecture for proactive chatterbot systems, based on a three-tier model and on exchangeable modules, allowed us to tackle the implementation of a real system, featuring dialogues about risk management, and providing a suitable environment for the proactive solution of dialogue problems. Its analysis and evaluation revealed the strengths and weaknesses of the achieved model, and gave us valuable suggestions for appropriate improvements and further developments.

A remaining open question is the effectiveness and suitability of the proactivity model achieved in the context of different domains and environments. Indeed, the taxonomy of problems devised has not yet been proven to be either exhaustive or expandable. Moreover, its applicability needs also to be verified in the setting of conversations in which humans and computer systems exchange their roles. Indeed, our assumptions were
those of a user seeking advice and services, particularly based on a deeper subjective knowledge of her virtual dialogue partner. A deeper investigation would be advisable in order to assess if the same problems apply in the case of dialogues in which the user assumes predominantly the role of an information and knowledge provider. The computer partner would exploit the dialogue for learning and improving its wisdom by taking the initiative for information gathering purposes rather than for problem solving reasons.

Once additional technologies enhancing and augmenting the interaction modalities for natural language based interfaces are mature enough and fully applicable, the model of proactive behaviour in dialogues will have to be extended and adapted. Indeed, as in the case of human-to-human conversations, the solution of specific problems often requires the inclusion of gestures (for instance for the intuitive visualisation of shapes or for the support of metaphorical allusions), sounds or melodies and facial expressions.
Bibliography


