BDI-based Generation of Robust Task-Oriented Dialogues

This work has been supported by SenterNovem,
Dutch Companion project grant nr: IS053013.

SIKS Dissertation Series No. 2011-08
The research reported in this thesis has been carried out under the auspices of SIKS, the Dutch Research School for Information and Knowledge Systems.

Printed by Wöhrmann Print Service, Zutphen
LaTeX template by Susan van den Braak
Cover photos by Hanny Breunese

ISBN 978-90-393-5514-5

BDI-based Generation of Robust Task-Oriented Dialogues

BDI-gebaseerde Generatie van Robuuste Taakgeoriënteerde Dialogen

(met een samenvatting in het Nederlands)

PROEFSCHRIFT

ter verkrijging van de graad van doctor aan de Universiteit Utrecht op gezag van de rector magnificus, prof. dr. J.C. Stoof, ingevolge het besluit van het college voor promoties in het openbaar te verdedigen op woensdag 9 maart 2011 des middags te 4.15 uur

door

Nieske Louise Vergunst

geboren op 12 september 1983 te Groningen

Promotor:	Prof. dr. J.-J.Ch. Meyer
Co-promotoren:	Dr. ir. R.J. Beun Dr. R.M. van Eijk

Contents

1 Introduction

If you're not failing every now and again, it's a sign you're not doing anything very innovative.

Woody Allen

It's ten o'clock in the morning on a Sunday. Lucy walks into the kitchen. She's in a cheerful mood and wants to make something special for breakfast. She turns on her new recipe robot, iCat.
Lucy: Good morning iCat, can you tell me... *[iCat interrupts]*
iCat: Good morning Lucy!
Lucy: Can you tell me how to make Eggs Benedict?
iCat: Yes, I can. *[pause]*
Lucy: *[confused]* Oh, I didn't mean that as a question, I meant that as a request.
iCat: I'm sorry, I don't know what you mean.
Lucy: Okay, sorry – please tell me how to make Eggs Benedict.
iCat: Of course. First, put half of an English muffin in the toaster.
Lucy: Okay. *[starts taking a bag of muffins out of the cupboard]*
iCat: Now, fry some bacon in a pan.
Lucy: Wait a second, I wasn't done with the muffin.
iCat: I'm sorry, I don't know what you mean.
Lucy: Just wait a second. *[cuts muffin in half, puts it in the toaster, takes bacon out of the fridge, puts it in a pan and places the pan on the stove]* Okay, now I'm done.
iCat: Please poach an egg.
Lucy: I don't know how to poach an egg.
iCat: I'm sorry, I don't know what you mean.
Lucy: *[sighs]* Please tell me how to poach an egg.
iCat: Would you like to abandon your current recipe and start another one?
Lucy: Oh. *[frowns]* It's a bit of a hassle, but alright.
iCat: First you have to boil some water.
Lucy: *[fills a pan of water, puts it on the stove and waits until it boils]* Okay.

iCat: Now, add some vinegar to the water.
Lucy: *[adds some vinegar]* Done.
iCat: Crack an egg into a soup ladle.
Lucy: Oh, but I don't have a soup ladle.
iCat: I'm sorry, I don't know what you mean.
Lucy: *[rolls eyes]* Can I use something else instead of a soup ladle?
iCat: I'm sorry, I don't know what you mean.
[smoke alarm starts beeping loudly]
Lucy: Oh no, the bacon!
iCat: I'm sorry, I don't know what you mean.
[Lucy removes the pan with charred bacon from the stove, chucks it into the sink and runs some cold water over the pan. The smoke alarm continues beeping]
iCat: I'm sorry, I don't know what you mean.
Lucy: Never mind, iCat. I suppose I'll just have a toasted muffin instead. Oh no! *[turns off toaster and removes charred muffin]* Can you at least order breakfast for me?
iCat: No. *[smiles]*
Lucy: Well, the good news about going out for breakfast is that I can at least put you in the garbage on the way.

This dialogue is a good example of how things can go terribly wrong if a robot has no flexibility, no way to detect and handle errors, no regard for whether the necessary ingredients and tools are available, etcetera. While it is a difficult task to solve all of these problems at once, this thesis presents work in some of these areas that can help researchers develop more robust task assistants. We propose a basic task-oriented system that is as simple as possible, and augment this system with an error handling module.

In this chapter, we first briefly introduce the context of this research. Then we will present a shorter and simpler example dialogue that we will use throughout the remainder of this thesis. Then, we present the problem statement, research aim, research questions, our methodology, the scope of this thesis, then we briefly discuss the contributions of this research, and finally we will finish this chapter with some practicalities about this thesis.

1.1 Context of this research

In this thesis, we present an agent-based dialogue system. In order to do this, this thesis must naturally cover aspects of agent research and dialogue research. Since most readers of this thesis will be familiar with, at most, one of these topics, we will briefly discuss both topics in this section. Then, we will present the context in which this research project originated: the Boon Companion project, later renamed to Dutch Companion. Finally, we will present a simpler example dialogue that we will use throughout this thesis to illustrate the framework that we present.

1.1.1 Dialogue systems

To build a task assistant that communicates with a human user, it is essential to have a dialogue component. The system must be able to give instructions to the user, and conversely, be able to understand what the user is trying to tell it.

Dialogue systems typically work in a three-step manner: processing of input, dialogue management, and generation of output. Naturally, these three processes are strongly intertwined: e.g., the kind of processing that is done on the input depends on the dialogue management rules, and vice versa. Generally, the choice for the kind of processing and dialogue management is made at design time and depends on the task that is to be performed by the dialogue system.

For some purposes, a minimal amount of processing and very simple rules are enough (e.g., ELIZA [131]), while other systems have very intricate and elaborate processing methods, taking into account all kinds of context and extracting all kinds of extra information such as user models, then using goals and intentions in the dialogue management process, in order to produce a more goal-directed response.

Between these two extremes, there is a whole spectrum of different types of dialogue systems. Here, we will briefly treat three types: keyword-based systems, state-based systems, and agent-based systems, and discuss some advantages and drawbacks of these three types. The three types are not mutually exclusive, and aspects of the three types can therefore be combined. However, they represent increasing levels of complexity.

Subsequently, we will motivate why we have chosen to develop an agent-based dialogue system.

Keyword-based dialogue systems

Keyword-based dialogue systems, or chatbots, are notoriously unflexible and lack robustness when the user is attempting to do something that falls outside of the previously defined purpose of the chatbot. The system only reacts to the current utterance from the user, meaning that no context is taken into account. Usually, chatbots are programmed to react to certain words, (parts of) sentences or sentence forms.

Chatterbots like ELIZA [131] and Verbots[1] work like this. Some are designed to help you (e.g., IKEA's Anna[2] helps visitors find their way around the website), some are designed just to make conversation (Jabberwacky[3]), some are designed to mimic particular behavior (PARRY [45] mimics a paranoid schizophrenic, ELIZA mimics a psychotherapist). A chatterbot cannot switch between these roles/tasks, because it uses only one set of rules to form reactions.

Besides a set of rules, chatterbots often also use a vocabulary/ontology (a classification of words into nouns, verbs etc, plus some synonyms) to minimize the number of rules; this allows the same rule to be activated on a whole class of words. For example, PARRY reacts the same to sentences containing any word that is classified

[1] http://www.verbots.com
[2] http://www.ikea.nl
[3] http://www.jabberwacky.com

as 'family member'[4]. Input can be annotated with information from the vocabulary to form structures such as:

```
My[POS-ADJ/I] brother[NOUN/family-member] is[VERB/be:simple-present]...
```

An example of how such a system works can be explained if we take as an example the input sentence "Are you listening to me?"

A system may typically contain a rule that looks like this:

```
   [verb] you [content]? →
Would you prefer it if I [verb*] not [content*]?
```

This rule is activated when the user inputs a sentence that starts with a verb, followed by the word 'you' and ending in a question mark. In the activation of this rule, some simple processing has to be done: `[verb*]` is `[verb]` in first person singular tense and `[content*]` is the content with all instances of 'I' and 'me' substituted by 'you', and the other way around[5]. If this rule is activated in response to the aforementioned input sentence ("Are you listening to me?"), the system's answer is: "Would you prefer it if I were not listening to you?"

Sometimes more than one rule is applicable. In this case, the most specific rule is generally chosen first. An example of multiple rules that may be applicable, from more specific to less specific:

```
my X[NOUN/family-member] Y[VERB] [Z] → Who else in your family
Y [Z*]?[6]
[*] [family-member] [*] → Please tell me more about your family.
[*] [NOUN] [*] → Can you tell me more about this [NOUN]?
[*]?  → Why do you ask?
[*] → Can you elaborate on that?
```

If no rule is applicable, the system usually gives a generic response along the lines of "Please go on", or "I'm sorry, I don't know what you mean."

Advantages of this type of system is that it is relatively easy to make and that it works quite well as long as the user does not step outside of the expected domain (casual chatting, answering simple questions, etc.). Disadvantages are that the system is not goal-directed or pro-active, and that errors are easily made but never detected or solved, which may lead to very frustrating, tedious and unproductive conversations.

[4]PARRY source code, `http://www.cs.cmu.edu/afs/cs/project/ai-repository/ai/areas/classics/parry/` (retrieved May 3rd, 2010)

[5]This may cause typical problems along the lines of:

> Nieske: You do not really believe you are a robot, do you?
> ELIZA: Does it please you to believe I am a robot do i?

[6]Apparently, ELIZA has an even simpler version of this rule, without the [VERB] check:

> Nieske: I know more about my brother than you do.
> ELIZA: Who else in your family than i do?

State-based dialogue systems

State-based dialogue systems have a set structure according to which the dialogue is conducted. This type of system remembers the current state that the dialogue is in, and moves from one state to another based on certain keywords in the input utterances. A general dialogue course is planned during the design time, together with (sets of) keywords that trigger dialogue state transitions. For example, when the system has asked the user a yes/no-question, a word from the set [yes, okay, affirmative] will move the dialogue to state S1, while a word from the set [no, sorry, negative] will move the dialogue to state S2. So, the utterance from the user is interpreted w.r.t. the current state of the dialogue. Examples of tools that can be used to construct this type of dialogue system are TRINDIkit [81] and the CSLU Toolkit [91].

Just as in keyword-based systems, words can be annotated with additional information. For example, consider a dialogue system that is currently in the state `start_dialogue` and the two options for next states are `recipe_dialogue`, which is activated by the word 'recipe', and `agenda_dialogue`, which is activated by the word 'agenda'. If the user says "I don't know what to cook tonight", the word 'cook' can be annotated with the category *recipe*, so with this utterance the state `recipe_dialogue` is also activated. If the user gives input that matches with none of the rules for this dialogue state, a general answer is given that prompts the user to give one of the expected instructions.

Advantages of this type of system are that the general course of the dialogue is coherent, but the dialogue system is still relatively easy to program. It is suitable for programming dialogues that always go according to the same pattern. Disadvantages are that the system is still not very flexible, and that scalability issues will quickly appear, since the flexibility of the system is proportional to the time taken in programming possible dialogue courses.

Agent-based dialogue systems

If the system has goals and intentions, it can conduct goal-directed dialogues. This means that not all initiative needs to come from the user, but the dialogue system itself also may have issues to address. This means that the system is not only reactive, but also proactive, and can also say something without input from the user. When the system forms a goal, it will take initiative to achieve this goal. Goals can be formed in cooperation with the user or autonomously.

Besides goals and intentions, an interaction history, context, domain knowledge and a user model can be used to reason about the user's intentions, to elicit these intentions from the user's utterances, and for attempts to infer the user's goals, beliefs, and plans. The system can then decide whether to adopt the user's goal and/or modify its own goals according to the user's goal. Whether the user's goal is adopted or not may depend on the roles in the dialogue, norms that the participants adhere to, whether the interaction is cooperative or competitive dialogue, etc. Besides joint activities, this type of system can also involve joint intentions [42], joint goals, and shared plans [70], as we will see in the next chapter.

For example:

Lucy: "I have some mushrooms and chicken, but I don't know what to cook."
(iCat's reasoning: Lucy is uttering this for a reason; she probably wants help with this. We are in a cooperative setting, so I want to help her.)
iCat: "Shall I find you a recipe with mushrooms and chicken?"

Advantages of this type of system are that the interaction can be pro-active and goal-directed, instead of just passive and reactive as in the systems described above. A disadvantage is that this type of system is difficult to model and program.

1.1.2 Agent systems

We have chosen to develop an agent-based dialogue system that can function as a task assistant, because it allows us to build a flexible, goal-directed system that can quite easily be extended to work with different types of tasks, by programming it with general task-oriented plans that can be used in multiple domains.

BDI-agents and multi-agent systems

An agent is an autonomous, social, reactive and goal-directed entity [135]. In our work we follow Wooldridge's [136] [135] definition of an agent: "a computer system that is *situated* in some *environment*, and that is capable of *autonomous action* in this environment in order to meet its design objectives."

To be able to work more easily with the concepts mentioned above, we model our agents according to the Belief-Desire-Intention (BDI) paradigm [41]. BDI is a theory pertaining to the internals of rational agents. It is mostly geared towards artificial agents, but approximates a theory of human action. A BDI agent has goals (*desires*) that it will try to fulfill. For each goal, the agent has one or more plans that will possibly lead to achieving the goal in question. To work with these concepts, the system also has *beliefs*, statements that it holds to be true; depending on whether certain beliefs hold, the agent can choose between different possible plans for its current goal. As soon as it commits to performing such a plan, we call it an *intention*, and from that point on the agent should be committed to completing it. The agent may only abandon its intentions if major problems arise.

A *multi-agent system* is a collection of agents in an environment [135]. These agents may communicate or perform activities together. In Chapter 2, we will elaborate on the concepts of coordination, communication, and cooperation.

1.1.3 The Dutch Companion project

The Dutch Companion project was originally initiated as a European project called 'Boon Companion' (an English expression meaning something along the lines of 'good friend'). In the Boon Companion project, the initial plan was that a number of researchers from different universities, companies and research institutions would develop a companion robot for children. As the project progressed, the goal shifted to a companion robot for the elderly, and then finally a recipe assistant. The project was eventually renamed to 'Dutch Companion'.

Figure 1.1: *The iCat, designed by Philips, showing some of its many possible facial expressions.*

The project included the iCat [23] (see Figure 1.1) as a research platform. The iCat is a stationary robot in the shape of a cartoon-like cat, developed by Philips. It is not mobile, but it has an abundance of input and output modalities, the most striking of which is its large expressive face that contains eyebrows, eyelids, eyes and lips, that are moved with 11 servo motors. The iCat also has DC motors to control head and neck movement, and multi-colored LEDs in its ears and paws, which can all be controlled through an animation editor. It has speakers in its paws that are controlled with a stereo soundcard. As input modalities, the iCat has a webcam in its nose, touch sensors in its ears and paws, proximity sensors in its paws, and a built-in microphone.

This thesis focuses on the dialogue part of the project, which was originally supposed to be only a small part of a more extensive system. The system was not fully implemented on the iCat robot collaboratively with the other project partners, and therefore we have not been able to test the system on the iCat. However, we have attempted to make the contribution of this work to the Dutch Companion project as clear as possible, by focusing on the architecture of the complete system in Chapter 3. Partial implementations were developed based on our framework [72] [80], which we will treat in Chapters 5, 6, and 7.

The main application of our system is in a kitchen setting, where the iCat will stand on a countertop and is able to observe the user and the domain, and instruct the user to perform various physical actions in the kitchen. In this thesis, we also use a secondary setting to illustrate the domain independence of the system: the iCat can instruct the user to set up a weblog. In this case, the iCat can be positioned next to the computer or, alternatively, be a virtual agent on the computer screen. During the

weblog setup task the iCat will be connected to the computer in a way that enables it to observe actions that the user performs on the computer.

1.1.4 A simpler example dialogue

In the example that we presented in the beginning of this chapter, a lot of things went wrong. Unfortunately, we cannot cover all of those problems in this thesis. In this subsection, we present a simpler example dialogue in which some errors occur, but which are repaired with the methods that we present in this thesis. In the remainder of this thesis, we will regularly refer to this example dialogue and in Chapter 7 we will present a more detailed look into the errors that occur in this dialogue and how they can be solved with the methods we have presented.

The example dialogue that we use in this thesis is as follows:

User: I would like to prepare French toast.
System: That's not possible, we don't have bread available.
User: Can you give me another recipe for breakfast?
System: Would you like to make a poached egg?
User: Alright, I would like to make a poached egg.
System: Okay, I'll teach you how to poach an egg. First, you have to boil some water.
User: Can you turn on the stove for me?
System: I'm not able to turn on the stove.
User: Okay, I will turn on the stove. (pause) I've boiled water.
System: Add some vinegar to the water.
(User opens a red vinegar bottle)
System: That's the wrong bottle. Please add white vinegar.
User: We don't have any white vinegar.
System: We do have white vinegar. I know this because I bought it.
User: Okay, you're right. I've added vinegar to the water.
System: Crack an egg into a soup ladle.
User: I've cracked the egg into a soup ladle.
System: Stir the boiling water.
User: How long should I stir the water?
System: I don't know.
User: Okay, I've stirred the water.
System: Drop the egg into the center of the water.
User: I've dropped the egg into the water.
System: Now wait 3-4 minutes.
(3 minutes later)
User: I've waited 3-4 minutes.
System: Remove the egg from the water.
User: I've removed the egg.
System: You have now poached an egg! Will you be able to do it by yourself next time?
User: Yes.
System: Okay, we're done!

Here, we use 'user' and 'system' instead of 'Lucy' and 'iCat' to denote that it is a dialogue that can be generated by the framework that we present in this thesis: the set of rules for the system that we present in Chapters 5 and 7, and, as we explain below, a simple set of rules to simulate a user.

1.2 Problem statement

The main problem that we address in this thesis is the fact that it is very difficult to plan a dialogue with a human dialogue partner. Many things can go wrong: the participants may disagree, misunderstandings may occur, actions may fail, previously believed information may be forgotten, the environment may change, etcetera. Robustness is recognized to be an important issue in dialogue systems research. Clark [40] states that if errors in dialogue are handled more adequately, this will benefit the development of natural language systems, and also, hopefully, the extent to which they are actually used.

Airenti et al. [3] acknowledge failures as an integral part of communication and state that they have to be resolved in "the standard model, without resorting to *ad hoc* strategies." We believe that two models are optimal: one for the standard (errorless) dialogue and one for error handling. This allows for the error handling module to be reused in different dialogue systems, which have a different model for the standard dialogue.

The research questions that we aim to answer in this thesis represent different tactics to work towards the developent of a robust dialogue system that is able to function in an unpredictable environment.

1.2.1 Research aim and research questions

In this thesis, we will address ways to make a task-oriented system as robust as possible. In order to do this, we will explore properties of cooperative dialogues, requirements and implementation of a basic system, a modular approach to language interpretation and generation, and an additional error handling component.

Research aim: *To present a BDI-based framework for a dialogue system that can instruct a user to perform tasks consisting of predefined sequences of actions, based on a joint goal that the user and the system have together, focusing on various tactics to make the system more robust.*

This research aim can be split up into more specific questions:

Question 1: *How does a cooperative dialogue (or interaction) result from a joint goal?* (Chapter 2)

Question 2: *What are the requirements for a robust cooperative task assistant?* (Chapter 3)

Question 3: *How can we implement a basic cooperative task assistant?* (Chapters 4 and 5)

Question 4: *What robust approach to language generation and interpretation can we use?* (Chapter 6)

Question 5: *How can we augment a basic cooperative task assistant with an error handling module?* (Chapter 7)

1.3 Methodology

As we have explained above, in this thesis we will present a model for a dialogue system that can conduct robust task-oriented dialogues. The model will be presented in BDI-rules in the style of the programming language 2APL. In the research that has lead to this thesis, we have looked mainly at human-human dialogues. However, the resulting model will not necessarily produce dialogues that are as humanlike as possible; instead, we aim to develop a system that behaves as cooperatively as possible and is robust in the sense that it will do its best to avoid communication breakdown.

In order to work towards this goal, we have taken the following steps. First, we have conducted a literature study about dialogues and dialogue systems. In this process, we have taken care to study both human-human dialogues and human-computer dialogues and the differences between them. Since this thesis focuses on dialogues that are cooperative and center around the performance of tasks, we have also investigated joint goals, especially in the context of BDI logic, and placed dialogue in the broader context of (linguistic and non-linguistic) interaction.

Taking this knowledge about cooperative dialogues and interaction into account, we have specified a requirements analysis for this specific type of system, which should behave cooperatively and be able to instruct a user to perform tasks. Based on these requirements, we have come to the conclusion that this specific system does not exist in the literature, and have therefore constructed a general conceptual model and architecture for such a system.

In order to design the system, we have constructed a set of rules according to the BDI paradigm that can work with (joint) goals and beliefs. We have also studied recipes and constructed a formalization that is based on nested lists. For parts of the design, 2APL implementations have been constructed that enable a user to perform cooking actions in a virtual environment, while being instructed by a Microsoft Agent on the screen.

Finally, we have evaluated and reflected on the results. We have constructed a set of beliefs and goals in addition to the rules that we have presented as our framework, and additionally a (simple) set of beliefs, goals and rules to emulate a user. Based on this complete 2APL-like set of beliefs, goals and rules, we have emulated a dialogue between the system and the user in which several errors occur and are handled by the system in a satisfactory way. This leads us to believe that our framework is suitable for conducting such dialogues. With different beliefs, recipes and joint goals, different but similar dialogues would result.

This evaluation has additionally produced a number of actions for future work, which we will present at the end of each chapter, and in a summary in Chapter 8. Most notably, a more extensive evaluation process with human users would produce a more thorough assessment of the quality of the system.

1.3.1 Scope of this work

This thesis aims to be a multidisciplinary work, centering mainly on agent research and dialogue research. Contrary to much agent research, we do not only deal with computational agents, but also with a human user. This introduces some interesting new problems, such as the unpredictability of human behavior, but also some advantages, such as the flexibility and improvisational skills of humans.

On the other hand, contrary to much dialogue research, we do not focus on language itself, but instead on dialogue acts, meaning that we will not delve deeply into aspects of dialogue such as speech recognition and parsing spontaneously spoken speech, and the problems that may arise in both of these processes. We do consider an approach to parsing and generating utterances based on speech acts in Chapter 6, but we will not discuss quantitative methods of computational linguistics, nor learning systems. This also means that in the handling of errors, we do not treat speech recognition errors and such, but we will only discuss the detection and handling of errors on the level of speech acts.

The dialogues that we treat in this thesis are cooperative and task-oriented. (We will explain these terms in more detail in Chapters 2 and 3.) This means that we will not deal with competitive situations, deceit, negotiation, and persuasion. On the other hand, the system that we present is also not solely a companion robot in the strictest sense of the word: being a social, friendly conversation partner is not its main goal. The main focus of the dialogues in this thesis is performing a certain task, cooperatively with the user.

The system that we present in this thesis is a *mixed-initiative* system: it is both proactive and reactive. If the system were purely reactive, the user would have to take into account the availability of kitchen appliances, tools, and ingredients himself. When the user asks for a certain recipe, the system would only be able to give the instructions as specified in the recipe, regardless of the user's level of proficiency. The system would not volunteer to assist the user, but would only be able to follow commands from the user. In short, omitting the proactive features of the system would result in a less effective interaction that is not tailored to the skills of the user and requires a lot more initiative and attention from the user. Also, a mixed-initiative interaction feels more natural for human users. [109]

The iCat is very much a multimodal robot, with its rich arsenal of sensors and actuators, but multimodal interaction is only one aspect of this research, not the central one. The system would benefit greatly from the addition of vision and sound processing to have more accurate beliefs about the current status of the domain and the actions that are being performed. We do treat various aspects of multimodal interaction in various chapters in this thesis, but have not implemented a real multimodal system at this time.

As we have mentioned earlier in this chapter, we will not give a detailed implementation of the system, but merely a framework in the form of pseudocode rules in the style of 2APL. We will, however, mention some work towards the implementation of functioning systems that are based on our framework.

Although we have not been able to implement a complete and working system, this thesis aims to give a complete, generic and principled framework for the development

of a robust multimodal interaction system. Parts of this are the generic task-oriented dialogue model that we present in Chapter 3, the generic companion robot architecture and the visualization of multimodal dialogues that we present in Chapter 4, and the generalized error detection and repair tactics that we present in Chapter 7. We hope that this thesis will be of assistance to dialogue and agent researchers in the process of developing a multimodal, task-oriented dialogue system.

1.4 Contributions of this research

As we have mentioned above, the main contributions of this research are in the presentation of generic models for a robust task-oriented dialogue system. In this thesis, we present a generalized architecture for a companion robot, which matches the companion robot architectures that we have encountered in the literature and makes it easier to compare such architectures to each other. We also discuss requirements for a cooperative task assistant that we believe any such system should adhere to, and present a visualization for multimodal dialogues that can be used in the development process of integrated multimodal dialogue systems. Furthermore, we present a BDI-based framework for a basic task assistant that can teach a user how to perform certain tasks, and a separate (also BDI-based) error handling module that avoids a communication breakdown by immediately addressing inconsistencies in the mutual beliefs of the participants and by addressing unexpected input. Although natural language is not the main focus of this work, we also present some first steps towards the development of a complete parsing algorithm using Functional Discourse Grammar.

There are several possible reasons why the implementation of an error handling strategy in a human-computer dialogue system is an interesting research topic. It can be used to study the way human agents handle errors in their dialogues. It can also be used to make a dialogue system more reliable and/or more natural. We have decided to build a very simple dialogue system (as presented in Chapter 5) and supplement it with an error handling module. We have attempted to make the error handling module as general as possible, so it can be used to augment any task-oriented instruction system with as few adjustments as possible.

1.5 Practicalities

1.5.1 Organization of this thesis

In Chapter 2, we will present theory about (joint) activities and cooperative dialogues and answer the first of our research questions. In Chapter 3, we will discuss the set of requirements that we believe any cooperative task assistant should have, and answer the second research question. The third research question will be addressed in Chapter 4, which deals with the architecture of our system, and Chapter 5, which presents the implementation of our basic system. In Chapter 6, we will present the theory of Functional Discourse Grammar, discuss a practical way to interpret indirect speech acts, and answer the fourth of our research questions. In Chapter 7, we will answer our fifth research question by presenting an error handling module that can

be used on top of the system that we present in Chapter 5. Finally, in Chapter 8 we will present conclusions, a summary of the answers to our research question, and some open questions that we believe would be interesting for future research. In the appendix, we present a glossary of linguistic concepts, which researchers in the field of computer science and/or agent technology may not be familiar with.

1.5.2 Some conventions

In this thesis, we use `U` to denote the user and `S` to denote the system. When we generalize over the participants in a joint activity, we use `A` and `B` to denote the dialogue partners. We use the term *agent* to generalize over human and computational participants, and will use *computational agent* and *human agent* when this needs to be specified.

Goals can be split up into *subgoals*, which must all be reached in order to reach the goal itself. *Atomic actions* are actions that cannot be split up into simpler actions. We use the term *task* to generalize over atomic actions, goals and subgoals. In this thesis, we use `a` to denote actions, `p` for propositions, `G` for goals, `S` for subgoals, `T` for tasks, and `P` for plans (lists of tasks). The abbreviation `I(taskname)` means that the speaker has the task with identifier `taskname` is the first action in its planbase.

Unless stated otherwise, the dialogue fragments in this chapter are taken from dialogues that were collected in the context of the Dutch Companion project or from previous experiments from the DenK project [1]. Some of the dialogue fragments are translated from Dutch.

We use male pronouns ('he', 'him', and 'his') when referring to a person of unspecified gender.

2 Cooperative Dialogues

> The real art of conversation is not only to say the right thing at the right place but to leave unsaid the wrong thing at the tempting moment.
>
> *Dorothy Nevill*

In this chapter, we explain the concept of dialogue in the broader context of joint activities, in order to enable us to treat linguistic and non-linguistic interaction in a unified way. We approach dialogue in a top-down fashion; like McTear [92], we believe that "dialogue structure emerges dynamically as a consequence of principles of rational cooperative interaction" (quoted from Morante [94]). In this chapter, we will present the definition and properties of cooperative dialogues, starting with a general definition of activities and refining this definition by exploring the concepts of coordination, language, and cooperation.

First, in Section 2.1, we will define *activities*, which are performed in order to achieve *goals* (2.1.1), and consist of *actions* (2.1.2).

In Section 2.2, we will see how activities can become *joint activities*. Joint activities rely partially on *mutual belief and common ground* (2.2.1) between the participants and are initiated in order to achieve *joint goals* (2.2.2). Joint activities are *coordinated* (2.2.3) by the agents that are involved. When coordinated by multiple agents, plans and projects become *joint plans and joint projects* (2.2.4). Actions in a joint activity are *participatory actions*, which might be individual or *joint actions* (2.2.5). In 2.2.6, we discuss which parts of this theory are used in our system.

A *dialogue* (Section 2.3) is a joint activity that involves the use of *language* (2.3.1). Language can be used as a coordination tool for a joint activity, but a dialogue is also a joint activity in itself. Participatory actions in a dialogue are called *utterances*, which correspond loosely to *dialogue turns* (2.3.2), but do not have a one-to-one correspondence: an utterance can span over more than one dialogue turn. When attending to an utterance of the dialogue partner, one constructs a *construal* (interpretation) of that utterance (2.3.3). These construals may be implicitly or explicitly *grounded* (checked) by the participants (2.3.4). Again, we discuss which parts of this theory

are used in our system in 2.3.5.

Section 2.4 pertains to *cooperative dialogues*. These are dialogues in which all participants adhere to norms of *cooperation* (2.4.1), meaning that they do not keep hidden agendas from each other, that they adhere to *Grice's maxims of conversation*, and that they are *sincere and helpful* (2.4.2) towards each other. To better define these concepts, we make the distinction between *private and public goals* (2.4.3). We finish this section with a discussion of which parts of this theory we use in our system in 2.4.4.

With all this information, we can define *successful dialogues* in Section 2.5. Since there are, as we have seen, different types of goals in a dialogue, dialogues can be more or less successful in different ways. Beside related work (2.5.1), we will look at *success at utterance level* (2.5.2) and *success at dialogue level* (2.5.3), and very briefly discuss the use of dialogue success in our system (2.5.4).

Finally, in Section 2.6, we will present a conclusion and future work.

2.1 Activities

An activity is a sequence of actions that are performed by an agent. An example of an activity is cooking a meal, which consists of cooking actions. An activity may be individual or joint; this can be either inherent to the activity (such as playing a duet, which is inherently a joint activity) or not (e.g., preparing a recipe, which can be done individually or jointly). Activities are usually executed with a particular purpose.

2.1.1 Goals

A goal is a state that an agent wants to achieve. An agent will typically initiate an activity to achieve a certain goal. The agent might have multiple possible plans (sequences of actions) that can be followed to achieve the goal. The agent can choose between these plans on the basis of some conditions that these plans have. For example, if Lucy has the goal 'eat dinner', she is able to choose between cooking a meal and going to a restaurant. Both plans have conditions: for cooking a meal, one has to have a kitchen, while a condition for eating at a restaurant is having enough money to pay for the meal. If Lucy does not have a kitchen (as is nowadays not uncommon in lofts in Manhattan), but if she has sufficient funds to eat at a restaurant, this is the plan that she will choose in order to achieve her goal of eating dinner.

2.1.2 Actions

Actions are one-shot (single) acts. We only treat intentional actions, which are executed purposefully by an agent, as opposed to unintentional (accidental) actions. Actions are part of *plans* that are used by the agent to work towards a *goal*. Just like activities, actions may be individual (single-agent) or joint (coordinated and performed by multiple agents). For example, when Lucy plans to eat at a restaurant, walking to the restaurant is an individual action that belongs to the plan that she chose. When Lucy arrives at the restaurant, the waiter hands her a menu. This is an

example of a joint action, since it is executed by Lucy and the waiter together, and cannot succeed if one of them withdraws from the joint action.

We distinguish between three different types of actions: domain actions, verbal communicative actions and nonverbal communicative actions. Domain actions are actions by an agent that influence the domain (e.g., picking up an object). Communicative actions are intended to make some information public; they can be verbal or nonverbal. An example of a nonverbal communicative action is Lucy raising her hand in order to signal to the waiter that she is ready to order. Uttering the order is, of course, a verbal communicative action.

Single-agent communicative activities such as giving a speech are, according to Clark [40], joint actions, even though they are *asynchronous*, because in the production of those activities, the actor does take into account possible listeners. However, at this point, we do not take coordination between speakers and hearers (or, more appropriately, broadcasters and receivers) into account and therefore postpone the notions of joint activities and coordination to the next section of this chapter; communicative actions may simply be broadcast (i.e. 'put out there', to nobody in particular) and heard only by *overhearers* (bystanders or eavesdroppers).

Some examples of communicative actions are *inform* (used when stating information that one considers to be true) and *request* (used when wanting information or action from another agent).

As we have mentioned in Chapter 1, we introduce the notion of *tasks* to generalize over goals and atomic actions: a task is either a goal (which can be split up into subtasks), or an atomic action.

2.2 Joint activities

Joint activities are projects that are executed by a number of participants together, as opposed to individual activities. For example, having a dialogue or cooking a meal together are joint activities. Cohen and Levesque [42] define a participant of a joint activity as an agent that is committed to the joint activity and to the other agents that participate in the joint activity.

Clark [40] makes some general claims about joint activities. Joint activities always have two or more participants. The participants have public roles and may have both public and private goals. Joint actions are built up hierarchically, meaning that a joint activity consists of a number of joint actions, which may consist of other joint actions themselves. The boundaries of joint activities are coordinated, which goes for both entry and exit. Joint activities may be simultaneous (like a game of tennis) or intermittent (like a game of correspondence chess) and the participants of a joint activity may change during the course of the activity. We will elaborate on these concepts later in this section.

All participants in the joint activity have an idea or expectation of the nature of the joint activity. Ideally, all participants should have the same beliefs about the nature of the joint activity, but this is not necessarily so: e.g., if Lucy engages in a dialogue with Jack, Lucy may want to find out some information, while Jack wants to make a nice chat in order to get to know Lucy better and might think that Lucy

has the same intention.

Some joint activities have to be accepted ('taken up') explicitly before they are valid joint activities; e.g., when a bet is proposed, the dialogue parter must first accept the bet before it is a joint activity. According to Clark [40], a proposal for a *joint purpose* may result in either full compliance (the most preferred result), alteration of the project, declination of the project, or withdrawal from project (the least preferred result). We will elaborate on the concept of uptake later in this chapter. The definition of commitment to a joint purpose is recursive, consisting of the following four conditions that must hold in order for two agents A and B to commit themselves to joint purpose G:

Identification: A and B must identify G;

Ability: it must be possible for A and B to do their parts in fulfilling G;

Willingness: A and B must be willing to do their parts in fulfilling G; and

Mutual belief: A and B must each believe that these four conditions are part of their common ground.

Joint activities vary on many dimensions; e.g., they may range from scripted to unscripted (a theater play vs. improvisation theater), formal to informal (a wedding ceremony vs. a party), verbal to nonverbal (a dialogue vs. a game of tennis), cooperative to competitive (solving a puzzle together vs. playing a game of chess), and egalitarian to autocratic (playing a duet vs. an oral exam). As we will explain later, we will not treat all types of joint activities.

2.2.1 Mutual belief and common ground

Airenti et al. [3] take mutual (or *shared*) belief as a primitive, rather than defining it solely in terms of individual (or private) belief. They justify this by mentioning "the human ability to deal easily with shared information". Shared belief (SH) does have individual belief as part of its definition, which is as follows:

$$\mathrm{SH}_{AB}\ \mathrm{p} \equiv \mathrm{BEL}_A\ (\mathrm{p} \wedge \mathrm{SH}_{BA}\ \mathrm{p})$$

That is: agents A and B hold proposition p to be mutually believed when A believes that p is true and that B and A hold p to be mutually believed.

In the definitions by Cohen and Levesque in [42], *mutual belief* is also not specified further, but is instead taken as a primitive. This is to avoid problems with mutual belief being infinitely recursive when defined w.r.t. individual beliefs; the formal definition of mutual belief is that A and B believe that p, A believes that B believes that p, B believes that A believes that p, A believes that B believes that A believes that p, B believes that A believes that B believes that p, ... (etc). In an unreliable setting, making certain information mutually believed would require an infinite dialogue of utterances like "p is true"; "Okay, now I believe that p is true"; "Okay, now I believe that you believe that..." etcetera.

Common ground is essential in coordinating joint actions and activities [40]. Unfortunately, common ground is not necessarily the same for both dialogue partners.

One can only *believe* that something is common ground, with a certain 'quality of evidence': a measure of how likely is it that something really is common ground. This can of course be misestimated. There are several things that can go wrong if one over- or underestimates common ground in an interaction. We will treat these types of errors later in this thesis.

Common ground accumulates during an interaction and consists of three different types. First of all, there is initial common ground, which is fixed at the beginning of the interaction, containing information about previous interactions and such. Secondly, there is the current state of the interaction, which changes with every dialogue action. The third type of common ground is the public events so far, including the registration of the dialogue history (textual and situational) and other salient events that occur during the interaction.

Communal common ground [40] is given by the social context in which we are situated (e.g., residence, occupation, religion, gender, ethnicity). In addition to communal common ground, agents also establish common ground with each other during an interaction. This is called *personal common ground*. This is based on what agents know about each other and joint history and activities. The more two agents interact with each other, the more common ground they have. Therefore, one has less common ground with strangers, more with acquaintances, friends, and the most with intimates. Communal common ground is more constant than personal common ground.

Jointly salient events are part of personal common ground. Agents rely strongly on saliency for anaphora resolution. The saliency of such referents (objects, events or concepts that are referred to) can depend on various conditions. Referents may be, for example, recent words in a sentence, objects or agents that are in the physical vicinity of the dialogue partners, or current events. Current events can be local (e.g., a ringing phone) or current on a grander scale (e.g., events that have been prominently featured in the media). Jointly salient events can get established by gestural indications (e.g., pointing to something in the vicinity), the dialogue partners' activities, or salient perceptual events.

2.2.2 Joint goals

To adequately model purposeful joint activities, we need a notion of joint goals. Castelfranchi [38] calls *social goals* the 'glue of joint action'. Without a notion of social goals, there could not be social commitment (i.e. 'the commitment of one agent to another'). He distinguishes the notion of social goals from joint goals: social goals may be joint or individual, but are adopted while taking into account other agents, while joint goals are adopted collaboratively by a number of agents.

Cohen and Levesque [42] define Joint Persistent Goals as in Figure 2.1. In this definition, box ($\Box$) stands for 'always', while diamond ($\Diamond$) stands for 'eventually'. BEL stands for (individual) belief: BEL x p means that agent x believes that p; GOAL stands for (individual) goals: GOAL x p means that agent x has p as a goal. MB stands for mutual belief, and MK stands for mutual knowledge. The other symbols and concepts will be explained below.

In order to explain the concept of joint persistent goals, we will first explain the underlying concepts of *mutual goals* (MG), *weak goals* (WG), and *weak mutual goals*

Mutual goal:
(MG x y p) $\stackrel{\text{def}}{=}$
(MB x y (GOAL x p) $\wedge$ (GOAL y p))

Weak goal:
(WG x y p) $\stackrel{\text{def}}{=}$
[$\neg$(BEL x p) $\wedge$ (GOAL x $\Diamond$p)] $\vee$
[(BEL x p) $\wedge$ (GOAL x $\Diamond$(MB x y p))] $\vee$
[(BEL x $\Box\neg$p) $\wedge$ (GOAL x $\Diamond$(MB x y $\Box\neg$p))] $\vee$
[(BEL x $\neg$q) $\wedge$ (GOAL x $\Diamond$(MB x y $\neg$q)))]

Weak mutual goal:
(WMG x y p q) $\stackrel{\text{def}}{=}$
(MB x y (WG x y p q) $\wedge$(WG y x p q))

Joint persistent goal:
(JPG x y p q) $\stackrel{\text{def}}{=}$
(MB x y $\neg$p) $\wedge$
(MG x y $\Diamond$p) $\wedge$
(MK x y (UNTIL [(MB x y p) $\vee$ (MB x y $\neg\Box$p)) $\vee$ (MB x y $\neg$q)] (WMG x y p))))

Figure 2.1: *Definitions from Cohen and Levesque [42]*

(WMG). The concept of mutual goals is straightforward: agents x and y have mutual goal p if x and y mutually believe that both x has goal p and y has goal p.

Agent x has weak goal p, relative to agent y if either of these four conditions hold:

- x does not believe that p is true and x has the goal that p will eventually become true; or
- x believes that p is (already) true and x has the goal that p will eventually become mutually believed by x and y; or
- x believes that p will never be true and x has the goal that this will eventually become mutually believed by x and y that p will definitely never be true; or
- x believes that q is not true and x has the goal that it will eventually become mutually believed by x and y that q is not true.

Note the sudden appearance of a fourth argument q in the fourth condition of this definition. The definitions in Figure 2.1 are quoted literally from Cohen and Levesque [42]. We assume that the fourth argument q has a function similar to that of the fourth argument of joint persistent goals: the condition q is introduced as a reason for the agents to keep goal p. Whenever q is no longer true, the goal can be dropped. This extra condition can be used in different ways, for example to express a supergoal of p (which, when it is canceled, will render p useless), or as a timeout condition after which p is no longer valid.

So, to summarize the definition of weak goals: when x does not (yet) believe that p is true, it has the goal that p will eventually become true, or when x believes that p is either true, will never be true, or that condition q is false, it has the goal that this will eventually become mutually believed by x and y.

The definition of weak mutual goals is also quite straightforward, extending the definition of weak goals in a similar way as mutual goals extend regular goals. Agents x and y have a weak mutual goal that p, relative to condition q, if they mutually believe that x has weak goal p relative to agent y and condition q, and that y has weak goal p relative to agent x and condition q.

Agents x and y have a *joint persistent goal* p if the following three conditions hold. First of all, x and y must mutually believe that p is not (yet) true. Secondly, x and y have a mutual goal that p will eventually become true. Thirdly, p will stay a weak mutual goal of x and y until both agents mutually believe that either p is true, p will never be true, or condition q is false.

When `(JPG x y p q)` is the active goal, it follows from the definition of joint persistent goals that until it is mutually believed that the goal is achieved, impossible, or there is no more reason to pursue it (e.g., time constraint q is violated), the participants should keep working towards it together as a weak mutual goal. This means that it is mutually believed between both participants that they both have the *weak goal* to achieve p (again, unless q holds). Here, the communication about the status of p emerges: when either of the participants finds out that p is either achieved, impossible, or not valid anymore (when q is no longer true), it becomes a goal of this participant to make this information mutually believed. They also both know that they have this commitment to making their beliefs about the achievability of p mutual.

According to this definition, an essential part of communication during a joint activity is informing each other of the status of plans, and therefore the communicative intentions that follow from this definition are all *inform*. Of course other types of speech acts also occur in the dialogue (asking for explanation, giving explanation, asking for clarification, ...) whenever the need arises.

2.2.3 Coordination

Clark [40] makes some claims about coordination in joint activities. In order to successfully execute joint actions and activities, the participants must coordinate the content and processes of the joint activity. In our earlier example, when the waiter tries to hand Lucy a menu, but Lucy fails to grab the menu when the waiter releases it, the joint action fails.

Partners in a joint activity can coordinate their actions in different ways: according to explicit agreement or according to precedent. Continuous coordination is periodic (i.e. synchronized by cadence/rhythm) or aperiodic. An example of a joint activity that is periodically coordinated is a musical duet, while dialogue is typically aperiodic. Coordination can be either balanced (no-one in the lead) or unbalanced. For example, dialogues are typically balanced, but not always: tutoring and instructing dialogues are unbalanced, since one of the participants (the tutor or instructor) takes the lead. Joint acts (one-shot) and joint actions (continuous) can both be relatively

easily coordinated because they divide into smaller-scale *phases* that can be used as coordination devices. A phase is a joint action that can be divided into an entry, a body and an exit. For example, saying a phrase, throwing a ball, and shaking hands are all phases.

According to Clark [40], there are three possible coordination strategies to synchronize joint actions. One possibility is cadence strategy (i.e. actions are coordinated by rhythm). This strategy is only applicable for periodic activities, such as dancing. Secondly, joint actions may be coordinated by entry strategy (i.e. only the entry point of the joint action is coordinated). In ongoing joint activities, the exit time of a phase is usually the same as the entry time of the next phase, such as in dialogue turn-taking; when one of the participants begins a dialogue turn, it is obvious that the previous turn has ended. Thirdly, coordination may be done by boundary strategy: coordination on entry and also coordination on exit. In this case, such as in shaking hands, the participants also try to coordinate the exit time; it would be impolite to withdraw one's hand too early from a handshake, but equally impolite to continue the handshake beyond the point where the activity partner starts to feel uncomfortable.

Castelfranchi [38] discusses coordination in the context of avoiding obstacles and exploiting opportunities. In order to coordinate with another agent's actions, one has to have beliefs about the other agent's mind. Because of this, coordination is always a social action. Castelfranchi also emphasises a number of possible variations in coordination. First of all, he distinguishes positive and negative coordination: the former is employed to profit from a favourable circumstance, while the latter is employed to avoid a negative circumstance. Besides that, he also separates reactive and proactive coordination, depending on when the coordinative action is planned (before or after the obstacle or opportunity is perceived). Finally, Castelfranchi also contrasts selfish and collaborative coordination: the first occurs when the agent merely 'uses' another agent and aims for the highest possible individual utility for itself, while the latter is about maximizing the total utility of all the agents involved. We will elaborate on this notion when we get to the topic of *cooperation*.

Besides coordinating on processes, the participants also coordinate their contributions on a syntactic [18] and semantic [59] level. This means that the participants adopt each other's vocabulary and terminology during the course of a dialogue or series of dialogues. This can be viewed as a form of personal common ground that is built up between the participants.

2.2.4 Joint plans and joint projects

A joint plan is a sequence of actions that make up a joint activity. The participants of the joint activity can plan their actions collaboratively in order to complete a joint goal together. Grosz and Kraus [71] explain collaborative plans in terms of full individual plans, partial individual plans, full shared plans, and partial shared plans. The possibility of partial plans means that agents are not required to have a complete idea of how they are going to fulfill their goals. The group of agents can together decide on their strategy to work towards a common goal. This involves identifying a plan and deciding which agent performs which actions in this plan. A disadvantage of partial plans is that the 'traditional' notion of commitment is compromised in two

ways: firstly, because agents can be committed to a goal without knowing exactly how to reach it, and secondly, agents can also form commitments towards others' activities.

A group of agents have a collaborative plan for a certain action if they have mutual belief of a (partial) plan, individual intentions that the action be done and that the collaborators succeed in doing the (identified) constituent subaction, and if they have individual or collaborative plans for the subactions. Grosz and Kraus' notion of collaborative plans is more complex than in previous work, because of the interaction of partiality and complex actions.

According to Clark [40], a joint project is a joint action projected by one of its participants and taken up by the others. We will elaborate on joint actions in the next subsection. Joint projects may be initiated with a proposal (in the form of, for example, a question or a request) and taken up with an utterance that displays construal of the proposal. Joint projects serve joint purposes, which are comparable to joint goals. Joint projects can be ordered in various ways, on which we will also elaborate in the next section. By minimal joint projects, people negotiate broader purposes. A joint project is a project that requires multiple participants that coordinate their actions.

2.2.5 Joint actions and participatory actions

Joint activities can be split up into smaller parts, called joint actions. Joint actions are the coordinations of individual actions by two or more people; e.g., attending to your dialogue partner's utterance, or answering your dialogue partner's question. For example, deceptive actions are not joint actions. An action that is part of a joint activity is called a *participatory action*; in contrast, *autonomous actions* are performed as part of an individual plan (and without coordination with any other actors). When performing a participatory action, all participants intend to perform their part and believe that they all have this intention, and additionally, believe that all other participants also have these intentions and beliefs.

Participatory actions are parts of joint activities, but may in themselves be either joint or individual actions. Participatory actions are comparable to 'social actions' from Castelfranchi [38], which, he stresses, are an important intermediate level between individual and collective actions. As we have seen before, the question whether an action is social depends on the intention of the agent that executes it; if this intention is social, the action is social, but it does not necessarily include actions of other agents. For example, an action that is performed by request of another agent is a social action. We will call participatory actions *moves* in the remainder of this chapter.

For example, a dialogue, which is a joint activity, consists of participatory actions that are performed by the participants. A dialogue is not built up of single utterances, but rather comprised of sequences of paired actions; utterances tend to come in pairs, rather than alone. Such action pairs in a dialogue are instantiations of *event-reaction pairs*, which, in its broadest sense, is the pairing of an event (verbal or non-verbal) and its instigating reaction (which is, too, verbal or non-verbal). By giving a reaction to a certain event, an agent can make his *construal* (interpretation) of the event

public; e.g., when accidentally stepping on someone's foot, apologizing will make it clear that you did not do it on purpose. We will elaborate on the topics of dialogues and construals later in this chapter.

Event-reaction pairs [40] consist of two ordered events: an instigating event and a reaction. The two events have different origins. The instigating event is any event mutually recognized by the participants A and B. The reaction is an action by B that is or includes a signal to A, and B's reaction is partially intended to display B's construal of the event.

We will explain event-reaction pairs by taking a slightly simplified sample of our example dialogue:

System: "First you have to boil some water."
User: "Can you turn on the stove for me?"
System: "I'm not able to turn on the stove."
User: "Okay." [turns on stove]

In this part of the example dialogue, there are a number of adjacent event-reaction pairs:

1. System says: "First you have to boil some water." – user says: "Can you turn on the stove for me?"
2. User says: "Can you turn on the stove for me?" – system says: "I'm not able to turn on the stove."
3. System says: "I'm not able to turn on the stove." – user says: "Okay."

However, there are additional event-reaction pairs, which are not adjacent to each other, and therefore less obvious:

4. System says: "First you have to boil some water." (turn 1) – user says: "Okay." (turn 4)
5. System says: "First you have to boil some water." (turn 1) – user turns on stove (turn 4)

Of the five event-reaction pairs, three are *system's utterance – user's utterance*, one is *user's utterance – system's utterance*, and one is *system's utterance – user's action*. In this short example, it is already clear that event-reaction pairs can be ordered in different ways. Some of the pairs overlap, while others are separate from each other. According to Clark [40], joint actions can be ordered in three different ways: embedding (e.g., for satisfying preparatory conditions), chaining (e.g., question – answer – evaluation), or pre-sequences (e.g., "There's something I wanted to ask you..."). In the above sample dialogue, one event-reaction pair is embedded in another one (request – [question – answer] – agreement), and there is also chaining of joint actions (request – agreement – compliance).

Some more examples of chained event-reaction pairs are:

- real question – answer – evaluation (e.g., "How long should I cook the spaghetti?" – "Ten minutes." – "Thanks.")

- test question – answer – verdict (e.g., "What's the capital of France?" – "Paris." – "Correct.")
- request – compliance – thanks (e.g., "Could you turn on the stove for me?" – [turns on stove] – "Thanks.")
- favor – thanks – acknowledgement (e.g., [turns on stove] – "Thanks" – "You're welcome")

An *adjacency pair* [108] is an event-reaction pair with the added property that the first part (the 'event') presents the uptake of a joint task, and the second part (the 'reaction') effects that uptake. Some typical instantiations of adjacency pairs are question–answer, greeting–greeting, offer–acceptance, and apology–minimization. To be more precise, adjacency pairs consist of two ordered utterances or actions. These two parts are uttered by different speakers and come in types that specify which part comes first and which comes second. Form and content of the second part depend on the type of the first part and are partially intended to display B's construal of the first part for A. (We will discuss construals of utterances in more detail in the next section.) Given a first pair part, the second pair part is conditionally relevant (i.e. relevant and expectable) as the next utterance.

2.2.6 In our system

Joint goals

As we have seen above, the following four conditions for joint activities must be fulfilled for the participants to have a joint goal together: *identification* (the participants both identify the joint goal in question), *ability* (the participants are capable of doing their part), *willingness* (the participants are both willing to work towards the joint goal) and *mutual belief* (the participants believe that these conditions are all fulfilled).

In our system, we do not explicitly implement the definition of Joint Persistent Goals according to Cohen and Levesque [42]; only the consequences that are expressed in the definition are implemented. This results in a number of principles. According to the definitions of joint persistent goals and weak goals, if the participants have a joint goal together, they should always inform each other of the (believed) status of the joint goal: when it is achieved, impossible, or when the (optional) condition q is false, the agent adopts the goal to make this information mutually believed. Additionally, the agent expects the other participants to behave in the same manner. The participants have their own private goals, and also beliefs about each other's goals.

In `(JPG x y p q)`, we take `p` to be the current (sub)goal and `q` to be the 'reason' why `p` has to be achieved and a condition on which any attempts to achieve `p` may be canceled. In our system, the recipe steps (subgoals) will be `p` and the completion of the whole recipe will be `q`, meaning that when the recipe preparation is canceled, the current recipe step does not have to be completed anymore either.

Joint plans

As we have seen above, Grosz and Kraus [71] explain collaborative plans in terms of full individual plans, partial individual plans, full shared plans, and partial shared plans. In order to implement collaborative plans in an agent-based system, we break down the definition of collaborative plans in terms of individual actions, and include rules that enable the execution of individual actions by any of the participants in the joint plan.

A difference between our work and that of Grosz and Kraus is that we only specify one of the agents in the joint activity, leaving the human user out of the formal definition. While this introduces some uncertainty, we do expect it to work in a practical setting, although of course the possibility of any formal proofs is greatly compromised by the lack of formal and computational definitions of the inner workings of a human user. The system's beliefs about the user's mental state are uncertain and probably inaccurate and the system can thus at most perform uncertain and tentative reasoning with this information. However, we do base the implementation rules of the system on Grosz and Kraus' work.

In our work, we do work with complex actions, but we assume that the system always has complete plans, since it is supposed to function as a recipe instructor. This makes reasoning about plans less complex than in Grosz and Kraus' work. This does not mean that the user cannot perform a complex action in a different way than the system's recipe for this complex action, as long as the user is supposed to do the complex action individually. Also, our system accounts for the possibility of different recipes for a certain goal, which Grosz and Kraus do not.

Mutual belief and common ground

In our *basic system* without error handling, as we present in Chapter 5, we assume a reliable communication channel. This means that there are two ways in which proposition p can become mutually believed: firstly, when one of the participants says that p and it is not denied by another participant; or secondly, when p occurs visibly and is observable by all participants. In both cases, information p is 'out there', and therefore public information for all of the participants. Of course it is still possible that one of the participants did miss this new information p because of some unexpected reason or other (e.g., noisy communication channels). If this happens, it should be detected and handled by our error handling mechanism. We will treat this in Chapter 7.

The system will have communal common ground with the user as a starting point, and everything that is built up from there is personal common ground (e.g., dialogue history, user preferences). This can become clear when the system makes mistakes in what it presumes to be common ground. If the error is in communal common ground, the system can react along the lines of: "I thought everyone knew that." However, when the error is in personal common ground, the reaction is rather: "I thought I told you this before!" In our system, we will represent common ground in the *contextual component* in Functional Discourse Grammar (see Chapter 6).

2.3 Dialogues

In this section, we will investigate what constitutes a dialogue. Dictionaries and encyclopedias[1] agree on some points, and combining their definitions yields the following definition in broad terms: a dialogue is a mostly linguistic interchange of information between two (or more) persons, usually initiated in order to reach one or more goals.

Clark [40] defines discourse as a type of joint activity in which conventional language plays a prominent role. Furthermore, he states that discourse may be spoken or written, and that conversation is a spoken form of discourse. Clark does not define conversation with respect to the concept of dialogue. We take the two to mean roughly the same, and for clarity we will mainly use the term *dialogue* in the remainder of this work.

Goffman [62] distinguishes three main types of listeners in a dialogue: overhearers (*unratified* or unapproved participants), ratified participants that are addressed by the speaker, and ratified participants that are not specifically addressed by the speaker (this type of listener only exists in dialogues with three or more participants).

Allwood [5] discusses some properties of *normal rational senders and receivers*. Communication between normal rational senders and receivers is intentional and purposeful, voluntary and free, and motivated (e.g., initiated in order to fulfill some goal). Also, communication is generally not avoided if it is pleasant, and is avoided if it is unpleasant. Communication is also adequate (efficient, succinct and relevant), and competent (only initiated when the sender "believes it possible to achieve his communicative purposes" [5]).

2.3.1 Language use

In the broadest sense, language can be viewed as the creation and usage of symbols, without specifying what those symbols are (they can be any form of signal, not necessarily verbal, as in sign language). However, for our current purposes we define language as a verbal form of interaction, adhering to the New Oxford American Dictonary definition: "the method of human communication, either spoken or written, consisting of the use of words in a structured and conventional way". Language is very useful for coordinating joint activities (though they can also be coordinated implicitly, as we have seen above).

Clark [40] proposes six working assumptions for language use. Firstly, language fundamentally is used for social purposes, by which he means that it is used primarily in joint activities. Secondly, besides a tool for coordinating joint action, language use is also in itself a species of joint action. Thirdly, language use always involves speaker's meaning and addressee's understanding (we will elaborate on this later in this section when we treat meaning and construal). Fourthly, the basic setting for language use is face-to-face conversation. Fifthly, language use often has more than one layer of activity (additional layers can be introduced by, e.g., story-telling or citing another person). Finally, the study of language use is both a cognitive and a

[1] Merriam-Webster online, Dictionary.com, American Heritage Dictionary, WordNet, Encyclopaedia Britannica, and Wikipedia

social science: we need to study both the individual minds of the participants, but also their social context in order to fully understand interaction.

Note that even though dialogue is a mostly linguistic form of communication, not all actions in a dialogue are linguistic actions. Consider again the short example dialogue that we have treated before. We view the action of the user turning on the stove as part of the dialogue, even though it is a nonverbal action:

System: "First you have to boil some water."
User: "Can you turn on the stove for me?"
System: "I'm not able to turn on the stove."
User: "Okay." [turns on stove]

Language use that is not a dialogue

Not all language use is a dialogue. In order to further specify the definition of dialogue, we must also explore what is *not* a dialogue. A monologue is an *exchange* of information, while a dialogue is an *interchange* of information. According to Clark [40], dialogues or conversations are different from monologues (such as essays and speeches) in a number of ways. When it comes to planning of either type of language use, the main differences are that essays and speeches are highly planned, under unilateral control, and have a clear topic, while dialogues are opportunistic, under joint control, and tend to have a much unclearer topic. The difference is also clear on a different level: essays and speeches are comprised mostly of assertions (with some exceptions, most notably rhetoric questions), while dialogues typically consist of many other speech acts besides assertions.

2.3.2 Utterances and dialogue turns

An utterance is "an uninterrupted chain of spoken or written language", according to the New Oxford American Dictionary. Clark [40] defines utterances as the actions of producing words, sentences, and other things on particular occasions by particular speakers for particular purposes. We take utterances to be deliberate linguistic acts that convey meaning. Contrary to the New Oxford definition, we do allow utterances to be interrupted; an utterance may take more than one dialogue turn when it is interrupted by the dialogue partner, by some other event outside the dialogue (e.g., the passing of a fire truck that carries a loud siren), or by the speaker himself, as in self-repair [112].

An utterance is a natural language instantiation of a *communicative act*, or *move*, in a joint activity. Just like utterances, moves in a joint activity do not have a one-to-one correspondence with dialogue turns; a turn may have multiple moves and a move may be distributed over multiple turns.

In turn-taking, the participants are simply trying to succeed in advancing their joint activities. According to Sacks et al. [105], turn-taking is done by means of *turn-allocation*, which is the (usually implicit) way in which dialogue partners switch turns. A turn consists of at least one *turn-constructional unit* (which may range

from a single word to long constructions of embedded clauses). Each of these units ends at a *transitional-relevance place*, at which a next speaker may take turn, or the same speaker may continue. Turn-taking is partially reactive; the end of a turn may be misprojected, in which case overlap or unintentional interruption may occur. A number of features may serve as evidence of end of turn: an elongated last syllable, a drop in the voice pitch, relaxation of the body, or the completion of gestures.

We can distinguish three different levels in which saying something is also doing something: the locutionary act, the illocutionary act and the perlocutionary act [88]. The *locutionary act* is "the utterance of a sentence with determinate sense and reference". The *illocutionary act* is "the making of a statement, offer, promise etc. in uttering a sentence, by virtue of the conventional force associated with it". Finally, the *perlocutionary act* is "the bringing about of effects on the audience by means of uttering the sentence, such effects being special to the circumstances of utterance". A locutionary act only involves the speaker, but perlocutionary acts often require acceptance (i.e. uptake) by the addressee and are consequently (parts of) joint actions.

In Dynamic Interpretation Theory, a distinction is made between two types of dialogue moves in an interaction: task-oriented acts and dialogue control acts [27]. Dialogue control acts (DCAs) are, for example, greeting, giving feedback, topic management, and turn management. DCAs may be linguistic or non-linguistic (e.g., nodding to signal understanding). Clark [40] uses the terms *track 1* (communicative acts) and *track 2* (metacommunicative acts) for the same purpose.

2.3.3 Meaning and construal

A communicative act is performed by the speaker in order to convey *meaning* and achieve a perlocutionary effect, as explained in the previous subsection. According to Grice [67], there are two different kinds of meaning: natural meaning and non-natural meaning. Symptoms have natural meaning (e.g., that smell means that dinner is burning), while signs and signals have non-natural meaning (e.g., Lucy's hand wave to the waiter means that she wants to order). Non-natural meaning is a social construct that has been agreed upon implicitly (social conventions) or explicitly ("When I call your number, you can pick up your order.").

Non-natural meaning can be divided into two types of meaning: signal meaning and speaker's meaning. Signal meaning is the way a signal should be interpreted, while speaker's meaning is what the speaker meant to communicate by performing the signal. The distinction is clearer in other languages; e.g., in Dutch: *betekenis* for signal meaning and *bedoeling* for speaker's meaning. Signals can be linguistic or non-linguistic; dialogues may contain both linguistic and non-linguistic signals.

These notions are important to us at this point, because the utterance of a word or sentence to an addressee means that the addressee will have to construe the (non-natural) meaning of this utterance. The formation of the *construal* (i.e. uptake, understanding) of an utterance is a non-trivial matter. For each signal, the speaker and addressee try to create a *joint construal* [40], meaning that they both agree on a construal and mutually believe that they do so.

One cannot always be certain that the construal that is reached by the addressee is the same as the speaker intended (the speaker's meaning), and since many signals

are open to differing construals [40], it is often useful for the addressee to signal his construal back to the speaker. This creates an opportunity for the speaker to check and, if necessary, repair the addressee's construal. The signaling of the addressee's construal can be done explicitly, but is more often done simply by reacting to the utterance in a way that implicitly signals the addressee's construal.

If the addressee signals a different construal than the speaker intended, the speaker may choose to repair the addressee's interpretation. However, this is not always necessary or even desirable; sometimes, the cost of repairing a misconstrual or narrowed construal is higher than just accepting the new construal. In Table 2.1, we show some possibilities for detecting and possibly repairing construals. The first column of the table shows the initial construal of the addressee. The second column shows some different possible actions that the speaker may perform to correct the addressee's construal. The third column shows the final (possibly repaired) construal of the addressee.

A's initial construal	S's intervening action	A's final construal
full construal	accept	verified construal
misconstrual	detect + correct	corrected misconstrual
misconstrual	detect yet accept	revised construal
misconstrual	not detect yet accept	undetected misconstrual
narrowed construal	accept	narrowed construal

Table 2.1: *Some examples of the handling of construals (from Clark [40], p. 195)*

For example, a *narrowed construal* occurs in the following example (from Clark [40], p. 195). Helen and Jack are sitting at a table together, and Kate joins them while holding two glasses of wine: one for herself and one for either Helen or Jack (since she could not hold three glasses at the same time). Jack replies "Thanks!" as if Kate brought the wine just for him. Even though Kate did not specifically intend the wine for Jack, she accepts Jack's construal, because it is consistent with her intentions to give it to either Helen or Jack.

We will elaborate on the notion of repairing misconstruals in Chapter 7 when we discuss error handling in dialogues.

2.3.4 Grounding

If something is spoken, it may be assumed to be common information. However, there is of course always the possibility of something going wrong: the addressee might not have heard the spoken information, or there might be a miscommunication or misunderstanding on a higher level. Therefore, one can never be completely certain that something is common information unless there is a confirmation from the addressee. In our basic system, we assume that something is common information if it is spoken; if this assumption turns out to be wrong, any consequences should be handled by the error handling module.

Grounding is assuring success on three different levels: attending to, hearing and understanding each other. In other words, grounding is establishing something as part of common ground well enough for current purposes. An utterance is only a

contribution to a dialogue if it is successfully understood, and to ensure that an utterance is successfully understood, we need the concept of grounding.

Grounding consists of two phases [40]. First of all, there is the presentation phase, in which participant A presents a signal and expects evidence of its construal. Secondly, there is the acceptance phase, in which B gives evidence of his construal. For the acceptance phase, signals from respondents divide into the following four classes: assertions of understanding (explicit confirmations), presuppositions of understanding (embedding the interpretation in a new utterance), displays of understanding (non-linguistic confirmations) and exemplifications of understanding (the speaker keeps talking while the hearer offers intermittent grounding).

As with mutual belief, grounding can theoretically be employed to an infinite degree, if the communication channel is not perfect. This can be solved by adopting more flexible models of mutual belief and grounding. For example, Traum [124] proposes a model for grounding in which it is not the goal to achieve perfect grounding every dialogue turn, but instead a sufficient level of grounding for the current purposes where any grounding mistakes can be resolved in a later stage if they are detected. Bunt et al. [31] introduce a level of mutual belief that is called *weak mutual belief*, which can be strenghened if evidence exists that the information in question is understood and accepted by the dialogue partner.

Grounding may be compared to Garrod's [60] notion of alignment, meaning that the participants align their *situation models* (context information). Garrod claims that dialogue is a relatively easy form of language use, because as opposed to speeches and essays, the mixed locus of control allows plenty of opportunity for alignment, and repair if needed.

2.3.5 In our system

For now, we will focus on dialogues between two agents, which are both ratified participants.

Dialogue properties

There are some properties of dialogues that can vary according to specific circumstances. According to Clark [40], the settings of language use can differ in a few ways. First of all, language can be spoken or written. We would like to deal primarily with spoken dialogues, although for ease of implementation, we will abstract these spoken dialogues to written speech acts (more on this in Chapters 3 and 6). Within spoken settings, the use of language can be personal (regular dialogue, "characterized by the free exchange of turns among the two or more participants") or nonpersonal (monologues); in our system, we only use the former kind. Also, we do not focus on language in institutional settings (limited by institutional rules), or fictional settings. Language can also be used in mediated settings, where "there are intermediaries between the person whose intentions are being expressed and the target of those intentions", such as telephone or email conversations. Since our system will speak directly with the user, we also do not deal with mediated settings. Finally, in our system, the setting of language use will only be public and not private (since the system will not talk to

itself).

To be more specific, in our system we are dealing with face-to-face conversation. Clark ([40] pp. 9-10) specifies ten features of face-to-face conversation. Some of these features are partially compromised in our dialogue setting. The features that fully apply to our system are *copresence* (the participants share the same physical environment), *audibility* (the participants can hear each other), *instantaneity* (the participants perceive each other's actions at no perceptible delay), *evanescence* (the medium is evanescent – it fades quickly), *recordlessness* (the participants' dialogue actions leave no record or artifact), *simultaneity* (the participants can produce and receive at once and simultaneously), and *extemporaneity* (the participants formulate and execute their dialogue actions in real time).

A feature that is compromised is *visibility* (the participants can see each other), which is not implemented in the current version of our system, although we do take it into account when we treat multimodal interaction in Chapters 3 and 4. Whether our system also has the feature of *self-determination* (the participants determine for themselves what actions to take when) is debatable and has more of a philosophical background: does the system actually determine for itself what actions to take or is it the programmer? The same goes for the feature of *self-expression* (the participants take actions as themselves): is the system a 'self' enough to be able to take actions as itself? We will not go deeper into these questions and leave this philosophical debate for another occasion.

Language use

While we have stated above that dialogues are unplanned (as opposed to planned language uses like speeches and essays), the dialogues conducted by our system are by necessity planned to a certain extent. This does not mean that our system cannot conduct natural dialogues, since the setting of the interaction (instruction dialogues) dictates that the dialogues usually follow predictable patterns. We will discuss these patterns to more detail in the next chapter when we discuss dialogue models. One of the important aspects of error handling that we treat in this thesis is the detection and repair of unexpected dialogue moves (see Chapter 7).

Grounding and construal

Grounding is a helpful concept in error handling. Proper grounding may help prevent errors by detecting them in an early stage, by checking whether the receiving agent has constructed the correct interpretation of the sending agent's message. If an error is detected in the receiving agent's construal, it can be addressed immediately, preventing any further errors that may result from the misinterpreted message.

2.4 Cooperative dialogues

Castelfranchi [38] stresses that social interaction, including communication, is not necessarily a cooperative activity. In this thesis, we will focus on cooperative dialogues, meaning that all participants in the dialogue will, in principle, attempt to achieve the

same goal(s), maximize the total utility of all participants, and do not have a hidden agenda or hidden beliefs. A great advantage of the assumption of cooperation is that one does not have to reason about dialogue partners' plan recognition [119].

Dialogues can be cooperative or uncooperative on two different levels. On the one hand, one (or both) of the participants might be uncooperative in a communicative sense: not answering questions, being impolite, or not attending properly to the interaction. On the other hand, independently from that, one might *interact* cooperatively but *act* uncooperatively. For example, answering all questions, but not giving any of the desired information, such as in the dreaded "no comment" from a politician when asked about a controversial issue, or promising one thing and doing another. We will treat both types of cooperation in this section.

2.4.1 Cooperation

Being in a cooperative interaction involves having the same joint goals, but it is not exactly the same. In non-cooperative dialogues, like buyer/seller-negotiations [107], both participants can still have the same goal (reaching an agreement about a transaction), but under the condition of maximizing their *own* utility rather than the *total* utility. Additionally, they may have a hidden agenda or hidden beliefs.

For example, in a car sales interaction, the seller wants to get a price for his car that is as high as possible, and he may have some beliefs about hidden rusty spots or other faults that he does not want to disclose to the potential buyer. On the other hand, the buyer typically wants to spend as little money as possible, and he might not want to seem too enthousiastic about the car, because this would compromise his position in the negotiations. For the same reason, the participants typically do not want to disclose complete information about their price boundaries, and the seller may be vague or uncommunicative about the price for which he acquired the car himself.

Also, in *deceit* situations ("the attempt to communicate a mental state which is not actually entertained" [3]), at least one of the dialogue partners is not cooperative, possibly even unbeknownst to the other dialogue partner(s). Since we only treat cooperative dialogues, we do not consider these situations for our current purposes.

2.4.2 Grice's maxims, sincerity and helpfulness

In construing the meaning of an utterance, people tend to use *implicatures*. For example, the utterance "I have two cats" can also be uttered in a logically sound manner by someone who has three or more cats, yet this is almost never the interpretation that the hearer will construe. This is more easy to see in a similar example that one could imagine in a liquor store in the USA:

> Lucy: "I would like a bottle of ruby port, please."
> Clerk: "Are you twenty-one?"
> Lucy: "Yes." (shows ID)
> Clerk: "Thanks. Alright then, that'll be eight ninety-nine."

This interaction is perfectly plausible even when Lucy is in fact *not* twenty-one years old, but older. Another example: when asking a friend how he likes his new job, the answer "Well, at least the coffee is nice", while not inconsistent with a positive attitude about the new job, implicates that the job itself is unfortunately not that great.

In order to work with these implicatures, Grice [68] introduces a main *maxim*: "Make your conversational contribution such as is required, at the stage at which it occurs, by the accepted purpose or direction of the talk exchange in which you are engaged." To be more precise about this general 'good practices'-rule, Grice splits it up into four separate rules, some with their own sub-rules.

Maxim of Quantity:
1. Make your contribution as informative as is required (for the current purposes of the exchange).
2. Do not make your contribution more informative than is required.

Maxim of Quality: Try to make your contribution one that is true.
1. Do not say what you believe to be false.
2. Do not say that for which you lack evidence.

Maxim of Relation: Be relevant.

Maxim of Manner:
1. Avoid obscurity of expression.
2. Avoid ambiguity.
3. Be brief (avoid unnecessary prolixity).
4. Be orderly.

Especially the maxim of relation requires further elaboration; after all, relevance can only be defined relative to a certain matter or circumstance. We define relevance as having a purpose with respect to the current joint activity (domain acts) or to the interaction itself (dialogue control acts [32]).

Grice's maxims can be used to draw inferences from utterances. When an agent hears his dialogue partner say p, he can reason in the following way:

- my dialogue partner said that p;
- he could not have done this unless he thought that q;
- he knows (and knows that I know that he knows) that I will realise that it is necessary to suppose that q;
- he has done nothing to stop me thinking that q;
- so he intends me to think, or is at least willing for me to think, that q. (example from Grice [69], p. 31)

For example, suppose that your dialogue partner utters the following sentence: *"John is in prison again"* (p). He can only say this if he thinks that *John has been in*

prison before (q). Since your dialogue partner knows that you will make this inference and does nothing to stop it, he is apparently willing to make you believe that John has been in prison before.

Grice's maxims are not restricted to linguistic contributions to a joint activity, but also pertain to actions; e.g., one would flout the Maxim of Quality by knowingly passing someone a can of tomatoes when he asked for the olive oil [88].

However, as Stone [117] points out, Grice's maxims are difficult to formalize. Frederking [58] states that Grice's maxims are useless for computational purposes and can generally be summarized as "do the right thing". Gazdar [61] makes some attempts in the direction, but does not attempt (or claim) to be comprehensive or exhaustive. Beun and Van Eijk [12] formalize Grice's maxim of quantity by using the distinction between private and mutual beliefs. The maxim of quantity can be taken as "do not say anything that is mutually believed". The maxim of quality can be generalized as "it should not be possible for A, from what B has said, say Q, to infer something which B believes to be false. If there is such a possibility then after saying Q, B should provide further information to prevent this."

Cohen and Levesque [43] coin comparable concepts with their descriptions of cooperative agents as sincere and helpful. When sincerity and helpfulness are embedded in a theory of rational (and linguistic) interaction, they are expected to yield similar behavior to the kinds of conversational behavior such as described by Grice's maxims. Sincerity pertains mostly to interacting cooperatively, while helpfulness is about acting cooperatively. Cohen and Levesque [43] (pp. 230–231) define sincerity as follows:

> "An agent x is SINCERE with respect to some other agent y and p, if whenever x has chosen to do something next in order to cause y to believe p, x has chosen to bring it about that y knows p."

Sincerity is not a property of an agent by itself, but is always defined with respect to another agent and a specific proposition. An important key to understanding the definition of sincerity is one of the conditions of *knowing*, which is that agents can only know something if it is true (otherwise it can only be *believed*, at most). Sincerity is most easily understood by its contradiction: "agent x would be insincere to y about p if x wants y to believe p, and x wants p to be false."

Cohen and Levesque define helpfulness as follows:

> "An agent [is] HELPFUL to another agent if for any action, he adopts the other agent's goal that he eventually do that action, whenever such a goal would not conflict with his own."

In contrast to sincerity, helpfulness is only defined with respect to another agent, but not for any specific action. Instead, it means that an agent will do all and any necessary actions to achieve the other agent's goal. Note that this definition does not take into account the capabilities of the agent; an agent might as well adopt goals that require actions that he cannot perform. Cohen and Levesque stress that helpfulness does not always have an effect: "not taking on another's goals does not indicate unhelpfulness, since the agent may have reasons for not wanting the goal" (i.e. having a conflicting goal of its own).

We define a cooperative dialogue as a dialogue where all participants are sincere with respect to all other participants and the joint activities that they participate in (which include the dialogue and, possibly, a domain task), and helpful to all participants.

2.4.3 Private and public goals

Clark [40] groups dialogue goals into four categories: domain goals (e.g., preparing a recipe), procedural goals (doing this quickly and efficiently, making clear moves, attending to what is being done), interpersonal goals (maintaining contact with other participants, impressing them, being polite, maintaining self-respect), and private agendas (deceiving the others, getting rid of them, working the situation for personal advantage).

An important distinction can be made between public and private goals. Within the concept of public goals, some of those are explicit while others can be left implicit. Public goals make sure that the dialogue partners can coordinate their actions to reach these goals. Private goals are sometimes in conflict with public goals; making private goals public sometimes works directly against them; e.g., imagine that Jack is only openly donating money to a good cause, in order to impress Lucy. The goal to impress Lucy will probably fail when Lucy finds out that Jack does not actually care about the good cause in question but has only made a donation to make a good impression on her.

An important point that Grosz and Kraus [71] present is the agents' beliefs about their abilities; an agent cannot intend to perform an action without believing that it is able to perform it. In our work, one of the agents is a human agent that the system does not have control over, and we do not require the user to have any knowledge of the recipe. In order to account for this requirement, the system has to reason about the abilities of both itself and the user before committing to a certain recipe.

2.4.4 In our system

Cooperation

The system is always sincere and helpful w.r.t. the user and all of their joint goals. It also does not hold any private goals.

In our system, we explicitly program communication actions; communication is not emergent, as Grosz and Kraus specify as being desirable. However, from a practical point of view we do not stray from their principles: during the programming phase, communication is derived from a need to make certain information mutually believed or the need for belief conflicts or uncertainties to be resolved, as we have seen before when we treated the definition of Joint Persistant Goals.

Grice's maxims

As we have stated above, many researchers claim that Grice's maxims are too vague to be implemented in a computational natural language system. Rules have to be properly well-defined in order to use them for this purpose. Other paradigms such as

Relevance theory [133] (which is essentially the collapsing of Grice's maxims into one 'super-maxim' of relevance) may be more suitable for precise definition. We agree with the viewpoint that Grice's maxims should not be explicitly implemented in the system, although we use them to form some inferences in the implementation of our system; e.g., when agent A says something, agent B can assume that A holds this to be true (cf. Maxim of Quality). This is a similar policy as our approach to Cohen and Levesque's Joint Persistent Goals [42].

2.5 Successful dialogues

We would like to formalize dialogue success in a principled, generic way. Apart from the task success (completing a recipe), we can use principles from dialogue theory to look at the success rate of a dialogue. Also, we would like to formalize the success of a single dialogue step. For this, we can contrast the success of procedural or interpersonal goals to that of domain goals.

A successful dialogue is a dialogue that results in all dialogue goals being achieved. There are several gradations of success in dialogues, depending on how many goals are achieved or how much progress is made towards achieving the goals. A completely unsuccessful dialogue results in no progress in achieving any of the goals, for example because the addressee is completely uncooperative or doesn't understand the speaker. A dialogue can be partially successful if only some goals are achieved or if goals are partially achieved.

In this section, we will discuss what constitutes a successful dialogue. We will first treat some related work on dialogue success, and then discuss success at two different levels: the utterance level and the dialogue level.

2.5.1 Related work

In Steidl et al. [113], dialogues are defined as successful if a (seemingly arbitrary) part of the task has been carried out successfully: in telephone conversations about booking a flight, the task is successfully completed if at least the outbound flight has been booked. Dialogue success is predicted from the success of separate dialogue steps. Some indicators for a successful dialogue step are the word 'yes', the word 'no', and the absence of repetitions. Some indicators for unsuccessful dialogues are recognition errors, user corrections, the user withdraws from the conversation before the task is completed, a human operator has to take over, or a task fails completely. Also, the success of a dialogue step correlates strongly with the success of the previous dialogue steps. Cohen and Levesque [42] have also found confirmations and requests for confirmation to be indicators of successful dialogues, but do not specifically define a successful dialogue.

Carlson [35] formalizes dialogues as games, comparing successful dialogues with finding the 'solution' (rational game strategy). Contrary to what is often done in argumentation games, where maximizing one's own profit would be an optimal strategy, Carlson cites Grice's maxims of conversation [68] as criteria for such a solution. Dialogues are seen as exchanges of information that rule out possible worlds. In a dialogue, each participant ('player', in Carlson's terms) attempts to keep his set of

possible worlds consistent, as small as possible, and consistent with other participants' sets of possible worlds. These rules entail communication; after all, when an inconsistency is detected, it must be solved, which can be done by communicating about it with the dialogue partner.

Airenti et al. [3] also use game theory terminology: speech acts are always interpreted relative to a *behavior game*. The rules of the behavior game in question are known and (implicitly) agreed upon by both participants. There can be discussion about the behavior game that is played; it can also change during the interaction. According to Airenti et al., defining a *standard* communicative situation is difficult; it is also possible to have a successful communicative interaction if a non-standard strategy has been followed. This is reflected in the different types of dialogue goals: the success of interpersonal goals ('communicative goals') is independent from the success of domain goals ('behavior goals'). ("One can be non-cooperative from a behavioral point of view, while still willing to maintain a correct conversation.")

In Airenti et al. [3], successful communication is initially defined as the recognition of a particular set of mental states, among which are the intention to achieve an effect on the dialogue partner and the intention that the dialogue partner recognizes the intention to achieve this effect. Airenti et al. do not find this definition sufficient and call for a stronger condition: A has the communicative intention that p with respect to B, or A intends to communicate p to B, when A has the intention that the two following facts be shared by B and A: that p, and that A intends to communicate p to B.

Clark [40] gives some preconditions for successful dialogues. In a dialogue, all participants have the goal of establishing and maintaining the mutual belief that their utterances have been understood well enough for current purposes. This is done mainly by grounding. Also, similarly to Grice's maxims, Clark states that dialogue partners "try to manage the production and interpretation of communicative acts with the least collaborative effort, i.e., with the smallest cumulative effort of the speaker and hearer combined."

We distinguish success at utterance level and success at dialogue level. An utterance is *erroneous* when it contains incorrect information or when it is unexpected in the current context of the dialogue. We will further specify this in Chapter 3. A dialogue is *unsuccessful* when the dialogue ends before the dialogue goals have been achieved.

2.5.2 Success at utterance level

Success at utterance level depends on various factors. For example, Austin [9] explores *felicity conditions* for 'happy performatives'. Performatives are a special subset of linguistic acts that, when uttered by someone, do not have a truth value in the classic sense, but instead, constitute the performance of the action that they describe. Examples of performatives are thanking, congratulating, promising and betting, which can be done simply by uttering them. Performatives are not always successful; for example, the act of promising can only be successfully done if the agent that utters the promise is in the position to perform the promised action, or at least will be sometime in the future. Similarly, when someone has his fingers crossed while making

a promise, the promise is also void, even though not all attendees may be aware of that fact. These felicity conditions can be compared to the *necessary and sufficient conditions* from Searle [110]. If one or more of these conditions fail, the utterance is not successful.

Event-reaction pairs should always be completed in order to have a successful interaction. This does not always have to happen instantaneously, since there can be interjections of other event-reaction pairs, such as:

Lucy: Do you know where the post office is?
Passer-by: Are you by foot or by car?
Lucy: By foot.
Passer-by: Okay, just walk through this alley and then take a left turn.

In this dialogue, step 1 and 4 are an event-reaction pair, just like step 2 and 3. The second pair is embedded inbetween the first pair: the event-reaction pair that is initiated by A in step 1 is 'put aside' while B gathers necessary information to give a satisfactory answer. A still expects B to answer the question from step 1. In cases where either pair 1–4 or pair 2–3 are not completed (i.e., step 4 or 3 are absent), the dialogue would not be complete.

In a dialogue with steps s_1 ... s_n, dialogue step s_i is a valid dialogue action if it fulfills at least one of the following requirements: (cf. Bunt [33])

1. s_i is the second half of an event-reaction pair that has not been closed yet;
2. s_i is the first half of an event-reaction pair that makes a (sub)goal of the speaker explicit;
3. s_i is a dialogue control act

In the next example, the whole dialogue fails because of a non-understanding at the utterance level that is irrepairable [98].

Lucy: Excuse me, do you know where the post office is?
Passer-by: Scusa, non parlo Inglese.
Lucy: Okay, never mind.

This is an error at the utterance level, since the task level could be uncompromised; if Lucy would show an envelope to the passer-by and gesture in a questioning manner, the passer-by might be able to point Lucy to the nearest post office.

Lack of understanding can sometimes be repaired, in which case the success of the dialogue can still be saved:

Lucy: Pardon, weet u waar het postkantoor is?
Passer-by: I'm sorry, I don't speak Dutch.
Lucy: Oh, alright, do you know where the post office is?
Passer-by: Yes, it's just here around the corner.
Lucy: Okay, thank you.

2.5.3 Success at dialogue level

We will now define success at the dialogue level. A dialogue is completely successful if all goals are achieved, or less successful if one or more of the dialogue goals are not achieved. To recapitulate, there are different kinds of goals: domain goals (e.g., finding out what time it is, getting the other person to do something) and more general goals, like procedural goals (e.g., doing this quickly and efficiently) and interpersonal goals (e.g., maintaining mutual respect). As we have mentioned before, in our cooperative dialogue setting, we do not take private goals (e.g., hidden agendas) into account.

Successful dialogue: reaching all dialogue goals

A successful dialogue is a dialogue that results in all goals being reached. This is one of the smallest examples of successful dialogue:

> Lucy: Excuse me, do you know where the post office is?
> Passer-by: Yes, it's just here around the corner.
> Lucy: Okay, thank you.

Lucy presumably initiated the dialogue in order to reach her domain goal of finding out where the post office is. It should be noted that she could have initiated the dialogue in order to reach different goals, like getting her dialogue partner's attention as a starting point for further conversation, or distracting the passer-by while her associate Jack steals his wallet. However, if we assume a cooperative situation in which Lucy really wants to know where the post office is, she succeeded in reaching this goal. And because Lucy reached this goal in an efficient, polite way without asking too much of her dialogue partner's time or effort, all procedural and interpersonal goals were also reached or maintained.

Finding the post office could be a subgoal of Lucy's main goal: perhaps she wants to find the post office in order to post a letter or to buy some stamps. Her dialogue partner can only infer this if Lucy gives a hint in that direction, either willingly or accidentally. If the dialogue partner manages to identify and fulfill this greater goal for Lucy, he could have either a positive or negative effect with this. A negative effect could be because of different reasons: Lucy might get suspicious of the stranger's reasons for doing more than Lucy asked for, or Lucy's goal of finding out where the post office is might be an actual goal in and of itself.

> Lucy (while holding an envelope): Excuse me, do you know where the post office is?
> Passer-by: Yes, I'm actually just going there, do you want me to take your letter and post it?
> Lucy: That would be very nice of you! Thank you.
> (or: Lucy: Can I walk there with you? I want to know where the post office is for future reference.)

Not reaching domain goals

If the goals of both dialogue partners are not reached or only partially reached, the success of the dialogue is compromised. Because there are several goals in each dialogue,

dialogues are often somewhere inbetween completely successful and insuccessful. In a somewhat less successful manner, the dialogue could have also gone like this:

> Lucy: Excuse me, do you know where the post office is?
> Passer-by: I don't know, but there's a visitor's center around the corner, maybe they can help you.
> (or: Passer-by: I don't know where it is exactly, but I'm pretty sure it's around here. [points to general area on the map])
> Lucy: Okay, thank you.

This dialogue is not completely successful, because Lucy's goal is not reached, but since the passer-by gives some advice to help Lucy in the right direction, it is not completely unsuccessful either. The domain goal is not reached, but some progress is made, and the interpersonal and procedural goals are reached/maintained. This dialogue will probably give Lucy a new subgoal: going to the visitor's center to ask for the post office. In 2APL terms, this would be a *plan revision*. This does not make the dialogue unsuccessful, just like a plan revision does not mean that an agent's plan fails.

A less successful dialogue would be:

> Lucy: Excuse me, do you know where the post office is?
> Passer-by: I'm sorry, I don't live here.
> Lucy: Okay, thank you.

Here, Lucy makes no progress at all in reaching her domain goal, but still the other goals are not compromised.

Not reaching procedural or interpersonal goals

If the procedural and interpersonal goals are not maintained, a dialogue like this could result:

> Lucy: Excuse me, do you know where the post office is?
> Passer-by: Yes, stupid, it's just here around the corner. Now go away.

This dialogue was unsuccessful in maintaining interpersonal goals (being polite). However, since the domain goal was successful, the success of this dialogue can be disputed. Violation of procedural goals yields a dialogue that is neither quick nor efficient:

> Lucy: Excuse me, do you know where the post office is?
> Passer-by: The post office?
> Lucy: Yes.
> Passer-by: Ah, let's see. Which post office do you mean?
> Lucy: The nearest one, I guess.
> Passer-by: Well, there is a post office a few blocks from here, but they opened a new post office last year, you know.
> Lucy: Aha.

> Passer-by: They've done such terrible things to the city centre. There used to be a beautiful old building which they tore down to replace it with the new post office...
> (etcetera)

Here, the dialogue goal is not necessarily compromised, but in the time that this dialogue is taking, Lucy could have easily asked someone else for directions.

2.5.4 In our system

We use the success of dialogues and dialogue turns mainly as a starting point for error handling. In Chapter 7, we will explain how we use the concept of dialogue success in our error handling module.

2.6 Conclusions

In this chapter, we have explored what constitutes a successful cooperative dialogue. In order to define cooperative dialogues in a general and principled way, we have first explored the concept of activities, then refined it to joint activities, which involve concepts that have to do with coordination. Then, we refined joint activities to dialogues, adding language-related concepts. Finally, we have refined dialogues to cooperative dialogues, using the concept of cooperation.

We have stated that our system uses joint persistent goals according to the definition of Cohen and Levesque [42], although we do not implement this definition explicitly. Instead, we use the consequences of joint persistent goals that are expressed in the definition as a basis for communication, most notably the principle that the participants always inform each other of the (believed) status of their joint goal. In this way, communication emerges from the joint goals of the participants. We define joint plans are defined in terms of individual actions.

Furthermore, we have stated that our system engages in face-to-face conversation with the user. We have also expressed the assumption for our basic system that any spoken information is mutually believed. This is based on the assumption of a reliable communication channel. When belief errors occur (which we will treat in Chapter 7), they may be caused by errors in the communication channel. Other errors may be caused by the fact that dialogues are unplanned, which is discordant with the determinism of a computational agent. Any unplanned dialogue contributions by the user will be seen as errors.

The system always acts in a cooperative manner in a number of ways. First of all, it is sincere and helpful w.r.t. the user and any joint goals that they have together. This means that the system only wants the user to believe things that the system holds to be true, and that it always adopts the user's goals as long as they do not conflict with its own. Secondly, also its dialogue behavior should be cooperative and should adhere to Grice's maxims.

As future work, we can adopt a more advanced view of joint plans, which may also contain *joint actions* instead of only individual actions.

In the following chapters, we will use joint goals as a central concept and starting point for cooperative interaction.

3 Conceptual Model

Inside every small problem is a large problem struggling to get out.

The Schalkner Converse to Hoare's Law of Large Problems

In this chapter, we explore the context and setting of the problem and establish a set of requirements that we work with in the rest of this thesis. At the end of this chapter, we will have obtained a set of requirements and conditions that any solution to the problem at hand should satisfy. In order to come to this result, we will treat some related work and we will elaborate on various details pertaining to different aspects of the problem.

Together with the previous chapter, in which we explained the properties of joint activities and dialogue that are relevant to this research, and the next chapter, in which we present the choices that we have made for the architecture of the system, this makes up the basis for the framework that we will present in Chapter 5.

In this chapter, we use the cooking domain as our main focus. We also introduce a new task, setting up a weblog, to ensure that the requirements that we present in this chapter are domain-independent. An example of part of such a dialogue is:

iCat: Next, unzip WordPress.
Lucy: I've already done that.
iCat: Okay, now make a database and user on your domain.
Lucy: Done.
iCat: Configure wp-config.php.
Lucy: I don't know how to configure wp-config.php.
iCat: Alright, I will now teach you how to configure wp-config.php. Rename wp-config-sample.php to wp-config.php.
Lucy: Done.
iCat: Open wp-config.php in a text editor.
Lucy: Done.
iCat: Fill in your database name, user and password in the indicated positions in wp_config.php.

This chapter is split up in five main parts that focus on, respectively, the participants, the domain, the interaction, a separate section for multimodal interaction, and error handling. In Section 3.1, we will treat the participants of the interaction, specifically focusing on companion robots (3.1.1) and compile a list of requirements for reasoning (3.1.2). In Section 3.2, we discuss the domain, focusing on recipes (3.2.1) and also compiling a list of requirements (3.2.2). In Section 3.3 we will treat interaction, which we divide in three different types of dialogues: information-seeking dialogues (3.3.1), tutoring dialogues (3.3.2), and task-oriented dialogues (3.3.3). Then, we will discuss how these types of dialogues can be combined in one system (3.3.4) and also finish with a list of requirements for the interaction (3.3.5). In Section 3.4, we will discuss multimodal interaction, and present a list of requirements in 3.4.1. In Section 3.5, we treat error handling, first defining errors in 3.5.1, then elaborating on the handling of errors in our system in 3.5.2, and also concluding with a list of requirements (3.5.3). Finally, in Section 3.6 we will present conclusions and future work.

3.1 Participants

While the basic setting of a recipe instruction dialogue involves one instructor and one pupil, in principle an arbitrary number of (human or computational) agents may partake in the interaction. For example, an instructor may teach a number of pupils to all make the same recipe (imagine a cooking school, for example), or to prepare a complicated recipe collaboratively. Another possible scenario is one pupil and several different instructors that coach the pupil through different parts of the task; for example, one agent helps with the gathering of ingredients, after which another one takes over to instruct the pupil to perform the recipe.

It might even be possible for multiple instructor agents to act in parallel, when the pupil needs to prepare a four-course meal with interleaving tasks for each of the courses. In this case, the instructors need to be carefully attuned to each other and a sense of priority would be useful for cases where multiple tasks need to be done at the same time; e.g., if after a carefully timed period of cooking, the pasta needs to be drained and rinsed, this is in most cases more urgent than cutting some vegetables for the salad. This requires careful planning on the part of the instructors.

The participants have (partial) knowledge of the domain, including the availability of ingredients, kitchenware and such. They also have beliefs about their *capabilities*, which means that they have beliefs about which tasks they know how to perform, ranging from complete recipes to simpler and smaller subtasks such as boiling water. They may have beliefs about their own capabilities, but also about the capabilities of the other participants. While we assume that the beliefs about one's own capabilities are correct, there may be errors in the beliefs about the capabilities of other agents.

There are a number of reasons why the participants may not always be able to perform all tasks in the recipes: not all necessary materials (ingredients, appliances, cookware) may be available, or the agent may not know how to perform a task. Not all participants have to have complete knowledge of the recipe that is to be prepared; the pupil will of course get instructions from the instructor and thus does not need to know the main recipe, but additionally, if the instructor does not have a recipe for a

subgoal but the pupil does, the pupil can still prepare this recipe (cf. *distributed plan* [39] [71]: none of the participating agents know the complete plan).

3.1.1 Companion robots

Recently, research in companion robots and affective virtual characters has been increasing steadily. Companion robots are supposed to exhibit sociable behavior and perform several different kinds of tasks in cooperation with a human user. Typically, they should proactively assist users in everyday tasks and engage in intuitive, expressive, and affective interaction. Moreover, they usually have multiple sensors and actuators that allow for rich communication with the user. All of these factors contribute to the complexity of the task of designing and building a companion robot. We will further discuss issues in multimodal interaction later in this chapter.

A good example of this is the European COMPANIONS project [132], in which both an English companion and a Czech companion are researched. Particular topics of research are automatic speech recognition, natural language understanding, natural language generation, text-to-speech synthesis, and emotion analysis. The English companion is designed to exhibit social behavior that is not part of a task-oriented dialogue. The only task that the companion carries out is being social with the user, including "chatting to, advising, informing, entertaining, comforting, assisting with tasks and otherwise supporting her or him" [132], although in the future more information-based tasks may be carried out as the companion is the user's interface to the internet. The companions get to know the user as more conversations are held.

There are some general requirements for companion robots, but also some that depend on the specific situation in which it is used.

First of all, a companion robot should be able to perceive the world around it, including auditory, visual, and tactile information. The multimodality of the input creates the need for synchronization (e.g., visual input and simultaneously occurring auditory input are very likely to be related), and any inconsistencies between different modalities should be resolved.

Conversely, a companion robot should of course produce coherent and sensible output over all available modalities. Because different processes may produce output concurrently and because a companion robot typically has multiple output modalities, the system should be able to to synchronize, prioritize, and/or merge these output signals; e.g., speech should coincide with appropriate lip movements, which should overrule the current facial animation, but only the part that concerns the mouth of the robot. We will discuss these aspects in more detail later in this chapter when we talk about multimodal communication.

A companion robot should be able to communicate with the user in a reasonably social manner. This means not only producing sensible utterances, but also taking into account basic rules of communication (such as topic consistency and Grice's maxims). This will not only make the conversation more pleasant for the user, but it will also make it easier for the user to follow the conversation. In order to be a robust system, a companion robot must always be able to keep the conversation going until the user indicates that he is done with the conversation. This also involves real-time aspects; e.g., to avoid confusing or boring the user, long silences should be avoided.

Additionally, a companion robot is likely to be designed for certain specific tasks, besides communicating with its users. Depending on various factors, like the domain for which the companion robot is designed, the type of robot, and the types of tasks involved, this may call for capabilities involving planning, physical actions such as moving around and manipulating objects, or electronic actions (e.g., performing a search on the internet or turning on an electric appliance such as a microwave).

Proactiveness on the part of the robot is often desirable in tasks involving cooperation. A robot can be proactive on different levels: acting proactively in a conversation means not only answering questions from your dialogue partner, but also thinking ahead and taking advantage of the situation by, for example, switching topics and asking (return) questions. In a different way, one can act proactively in order to achieve one's own goals by taking initiative in performing tasks and instructing partners in cooperative situations to perform tasks, contrary to only complying with instructions. In Wooldridge [135], proactiveness is defined as *goal-directed behavior*. Wooldridge argues that programming a purely proactive system is easy, as is a purely reactive system. A purely proactive system only pursues its own goals and does not take into account other agents' goals and plans or changes in the environment. In complex, uncertain, multi-agent settings, this is not a very successful tactic.

On the other hand, even though purely reactive systems have their advantages, there are many problems with agents that only respond to their environment, such as the lack of information-gathering abilities and the purely 'short-term' views that reactive agents necessarily have, as explained by Wooldridge [135]. If the system were purely reactive, the user would have to take into account the availability of kitchen appliances, tools, and ingredients himself. When the user asks for a certain recipe, the system would only be able to give the instructions as specified in the recipe, regardless of the user's level of proficiency. The system would not volunteer to assist the user, but would only be able to follow commands from the user. In short, omitting the proactive features of the system would result in a less effective interaction that is not tailored to the skills of the user and requires a lot more initiative and attention from the user. Therefore, we need to achieve an optimal balance between proactiveness and reactiveness.

3.1.2 Requirements for reasoning

General Requirements

The participants should be able to have joint goals together (RR1). Some of the goals may be reachable in different ways, or not reachable at all, given the current circumstances and the capabilities of the participants. A goal should only be adopted when it is reachable under the current circumstances (RR2). After all, finding out that it is not possible to complete a goal when the execution has already been started, should be avoided whenever possible. (There is still a chance that this may go wrong; e.g., when the system erroneously believes that some essential ingredients are available or when the environment changes during the preparation of the recipe.) So when one of the participants proposes a joint goal that the other believes is not reachable, this goal should not be adopted.

Requirement	Code
Joint goals	RR1
Reasoning about feasibility of goals	RR2
Reasoning about plans	RR3
Plans consist of atomic actions and subgoals	RR4
Possibility to have different plans for a goal	RR5
Instructions should be given on an appropriate level	RR6
Beliefs about the domain	RR7
Beliefs about capabilities of the participants	RR8

Table 3.1: *Requirements for reasoning*

To work in this way, the system should have a notion of plans (recipes) that allows the system to reason about them (RR3). There might be multiple alternative recipes for a certain goal. The tasks in a plan are either subgoals (meaning that there is a plan to achieve them) or atomic actions (RR4). Also, in order to reason about the feasibility of goals, the system needs to have beliefs about (the current situation of) the domain (RR7).

The participants may not have the same plan for a certain (joint) goal (RR5). If there is a possible plan for the given goal, the participants can begin executing this plan. If one of the participants takes on the role of instructor, his instructions should be given on an appropriate level for the pupil: they should be understandable, but also not too easy (RR6). For this, the system should keep an administration of what it believes the user is capable of doing (RR8).

When giving instructions, the system can act as a tutor or an instructor. Although both terms are sometimes used interchangeably (e.g., see Goodman et al. [63]), there is a distinction that can be made. The main difference is that a tutor tries to teach its pupil something, while an instructor merely makes sure that the task is completed properly by the pupil. The practical difference is that in a tutor/pupil situation the user can to ask why something has to be done, because he tries to learn something from preparing the recipe. In an instructor/pupil situation we have not observed this in our corpus. In this thesis, we will use the instructor/pupil situation, since the tutoring aspect of the dialogues is only a secondary aspect of the dialogue, as we will explain later in this chapter.

Domain-Specific Requirements

In the example that we introduced in Chapter 1, the user proposes French toast as a joint goal in the first dialogue turn. In this case, not all actions in the recipe are possible in the current situation, because not all ingredients are available. The joint goal has to be abandoned and the system reports this to the user. The user then introduces a new joint goal, of making a poached egg, which fortunately is possible.

There may be different possible plans to prepare a dish (e.g., poaching an egg in a pan of boiling water vs. poaching it in the microwave). The tasks in the recipe might

also have recipes themselves: e.g., the recipe for poaching an egg contains the task 'boil water', for which the system also has a recipe.

The level of instructions should be appropriate. In the above example, it would be a suboptimal strategy for the system to instruct the user as follows: "Please poach an egg." This instruction is too high-level, since the user does not (yet) know how to poach an egg. Also, since the user knows how to boil water, giving instructions to fill a container with water, turn on a heat source, place the container on the heat source, and wait until the water starts bubbling, would be too low-level for the user and would make the interaction tedious and frustrating for him.

In a different domain, such as setting up a weblog, we can use the same notion of 'recipes' that consist of tasks and subtasks, although of course this term should be intepreted in a less literal way than when we are actually preparing (culinary) recipes. For the current purposes, however, we will also use the same term for different domains; a recipe for setting up a blog consists, among other tasks, of registering a domain and installing a blogging service such as Wordpress. We will elaborate on the specific aspects of such recipes in the next section. Just like in the cooking domain, the participants have beliefs about their own and each other's capabilities and about available tools and 'ingredients' such as an internet connection or an FTP client.

3.2 Domain

The system should be able to reason about the environment. One complicating factor in our application is that the environment is dynamic, which means that the system's beliefs about the environment are not always accurate, since the environment may have changed.

3.2.1 Recipes

A recipe can be viewed as a tree of actions, or, similarly, a nested list of actions. Goals consist of actions and subgoals, which in turn consist of their own subgoals or actions. Take for example the recipe for poached egg:

1. Pour at least four inches of water into a large pot. Bring water to a boil.
2. Season water with salt and pepper
3. Add approximately one tablespoon of vinegar to the water
4. Reduce heat to simmer
5. Swirl the water around quickly to make a whirlpool, helps to keep the eggs in shape.
6. Crack one egg into a ladle and lower egg into water, gently releasing egg into the water
7. Gently move egg around in water to prevent egg from spreading in water
8. Remove egg after two to three minutes[1]

[1] `http://www.wikihow.com/Make-Poached-Eggs`, retrieved on November 10, 2009 at 3:15 p.m.

To illustrate the principle of a recipe being like a tree of actions, or a nested list, we can also find a recipe for breaking an egg:

1. Grasp the egg in your dominant hand.
2. Hold it between the thumb and first two fingers.
3. Now tap the egg firmly onto a hard surface to crack the shell. You should always use a flat surface to crack an egg rather than the edge of a counter or bowl because there is less likelihood that bits of the shell will go into the egg itself.
4. If the egg's side has not completely ruptured place your two thumbs in the cracked dent and pull apart.
5. Empty egg into a bowl or similar container before it hits the table.[2]

Similarly, for boiling water:

1. Use a large enough pot for the water you are heating and don't fill it all the way.
2. Place the pot on the stove and turn the heat under it to high.
3. Turn down the heat once the water starts boiling[3]

To illustrate the possibility of different recipes for a certain goal, an alternative recipe for boiling water:

1. Put the water in a microwave-safe cup or bowl.
2. Place a non-metallic object such as a wooden spoon, chopstick, or popsicle stick in the water.
3. Put the water in the microwave. Heat in short intervals, stirring regularly, until the water is steaming.[4]

Similar 'recipes' exist for setting up a weblog. For example, to set up Wordpress:

1. Download and unzip the WordPress package, if you haven't already.
2. Create a database for WordPress on your web server, as well as a MySQL user who has all privileges for accessing and modifying it.
3. Rename the wp-config-sample.php file to wp-config.php.
4. Open wp-config.php in a text editor and fill in your database details as explained in *Editing wp-config.php* to generate and use your secret key password.
5. Place the WordPress files in the desired location on your web server: [...]
6. Run the WordPress installation script by accessing wp-admin/install.php in a web browser. [...][5]

[2] `http://www.wikihow.com/Break-an-Egg`, retrieved on November 10, 2009 at 3:14 p.m.
[3] `http://www.wikihow.com/Boil-Water`, retrieved on November 11, 2009 at 3:21 p.m.
[4] `http://www.wikihow.com/Boil-Water`, retrieved on November 11, 2009 at 3:21 p.m.
[5] `http://codex.wordpress.org/Installing_WordPress#Famous_5-Minute_Install`, retrieved on October 4, 2010 at 4:36 p.m.

Requirement	Code
Reasoning about actions (pre- and postconditions)	RD1
Action identifiers are associated with physical actions	RD2
Action identifiers are associated with communication	RD3
Actions can be delegated	RD4
Possibility of open delegation	RD5
Plans are sequential	RD6
Success of performed action should be checked	RD7

Table 3.2: *Requirements for the domain*

3.2.2 Requirements for the domain

As we have seen in the previous chapter, there are different types of atomic actions: domain actions, verbal communicative actions and nonverbal communicative actions. These actions can be single-agent (individual actions) or multi-agent (joint actions). As we have mentioned in Chapter 1, we use the notion of *tasks* to generalize over goals and atomic actions: a task is either a goal (which can be split up into subtasks), or an atomic action.

General Requirements

The instructor should be able to use the atomic actions for three different purposes. First of all, the instructor can perform an atomic action itself (RD2). Performing an action does not only involve satisfying the preconditions and then adding the postconditions to its beliefs; usually, it also includes performing a physical action or a communicative act. We will elaborate on communicative acts in the next section.

Secondly, the instructor can instruct the pupil to perform an atomic action (RD4). This requires having a linguistic representation of the action in order to form an utterance (RD3). Besides atomic actions, the system can also instruct the user to perform higher-level tasks or achieve (sub)goals in a manner that is not specified by the instructor. This means that the pupil has some freedom as to how the task will be performed (RD5). This is similar to Castelfranchi's concept of *open delegation* [39], as opposed to *pure executive (close) delegation*, when a completely specified task is delegated. In the case of open delegation, which Castelfranchi states is an important concept in collaboration theory, the delegating agent believes to delegate either a complex or abstract action (what we call a higher-level task) or a goal state.

Thirdly, the representation of an atomic action should allow the system to reason about whether a plan that contains the action in question is possible, given the circumstances (RD1). This requires the system to have beliefs about the pre- and postconditions of an action in order to reason about them; an action is possible at a certain given time if all preconditions are fulfilled. The sequentiality of a plan (the fact that the tasks in a plan have a fixed order) allows the system to reason about the possibility of a certain action in a plan, it can also use the postconditions of all previous actions to check whether the preconditions of the action in question will be

fulfilled at the time of execution in the plan (RD6).

Domain-Specific Requirements

External actions that the system can perform in a kitchen setting include turning on electric appliances and performing non-physical actions like keeping the time. External actions may fail, when for example an appliance is broken, so it is useful to check the success of the action (RD7).

In the blog setting, some examples of actions are writing a blogpost and various actions on files, such as uploading, downloading, unzipping or editing. Unlike in the kitchen setting, these actions are all non-physical actions, since they all happen on a computer. However, we still have to take into account the possibility that some of these actions may fail. For example, the action of uploading a file will fail when the internet connection breaks down during the process.

Communicative actions are similar in both domains. We will elaborate on the required communicative actions in the next section when we discuss the requirements for the interaction.

3.3 Interaction

In this section, we will present a formalization of the specific type of dialogues that will typically be conducted by the participants in this particular domain, and examples of errors that can occur in these dialogues.

A dialogue is in itself a joint activity, but can also be used to coordinate a joint activity such as preparing a recipe. This dichotomy reflects different types of goals that dialogue partners have.

Since we have defined dialogues to be purposeful joint activities, we assume that the initiator of a dialogue has at least one goal in initiating a dialogue. For current purposes, we assume that he has one (primary) domain goal. Dialogues generally follow a dialogue pattern that depends on the type of this domain goal. In this section, we treat three different types of domain goals and the types of cooperative dialogues that result from those goals: information-seeking dialogues, tutoring dialogues, and task-oriented dialogues.

Since we would like to formalize these types of dialogues, we also treat some research on dialogue systems and models of these types. First of all, we will briefly introduce the three different types, and then elaborate on each type in the rest of this section.

In Figure 3.1, we show the triangle metaphor from Ahn et al. [1]. Here, we have two participants, A and B, and a domain. The arrows between the participants represent information that is exchanged between them, usually in the form of linguistic utterances in a dialogue. The arrows from the participants to the domain denote the fact that the participants can manipulate the domain (i.e. perform actions in the domain). The arrows from the domain to the participants denote the fact that the participants can observe the domain.

Information-seeking dialogues are dialogues where only participant A in the triangle metaphor can get information from the domain, so, in order to get this information,

participant B (the information seeker) must ask A (the information provider) for this information. *Tutoring dialogues* are dialogues where participant A is an expert on the domain and may quiz B (the pupil) on domain knowledge. Also, similarly to information-seeking dialogues, B can get domain information from A when needed. In some *task-oriented dialogues*, either B is an expert on the domain and/or task and therefore gives A instructions on how to perform a certain task, or B wants to have a certain task performed on the domain and therefore gives A instructions. Another type of dialogue that we could consider, but which we leave for future work at this time, *social dialogues*, do not pertain to the domain at all, but merely to the interaction between the two participants.

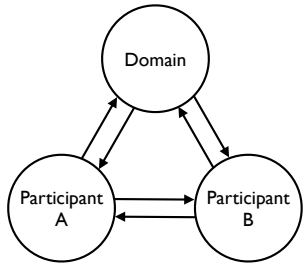

Figure 3.1: *The triangle metaphor [1]*

The types of dialogues are fundamentally different in the way in which the two participants interact with each other and the domain (see Figure 3.1). When looking at the triangle metaphor above, tutoring and task-oriented dialogues have the 'full' range of interactions as represented in Figure 3.1. Both participants can interact with the domain (represented by the arrows from the participants to the domain), perceive the domain (the arrows from the domain to the participants), and interact with each other (the arrows between the participants). Information-seeking dialogues miss a few arrows, as represented in Figure 3.2: the arrows between participant B and the domain are omitted, meaning that B can only access the domain via A.

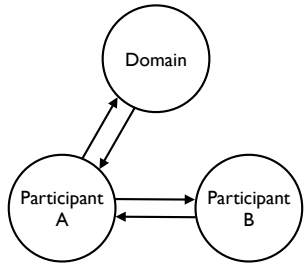

Figure 3.2: *A variant on the triangle metaphor*

We make the distinction between these different types of dialogues because they

tend to follow different patterns, which we will elaborate on later in this section. Also, we use the different types of dialogues as the basis for our system. The dialogues generated by our system will have some of the key properties of all these dialogues. At the end of this section, we will explain how the different dialogue types are combined in our framework.

We will also explore some typical errors that can occur in cooperative dialogues. Generally, any dialogue act that does not fit into (a combination of) the dialogue models that we will present below, is an erroneous dialogue act. This does not mean that the models will be unsuccessful, but the move in question should be repaired, in order to avoid a total communication breakdown.

3.3.1 Information-seeking dialogues

In an information-seeking dialogue, one of the participants is the information seeker, the other is the information provider. The information provider is collaborative and may get a reward for providing the information to the information seeker. Typically, the information seeker initiates the dialogue by providing a query to the information provider, after which the information provider gathers the information and gives it back to the information seeker. If the information provider does not have the requested information, he informs the information seeker of this. Morik [95] describes information-seeking dialogues as dialogues between a user (the information seeker) and a system (the information provider) as interactions where "the system brings about changes only on the user's mind by communicative actions".

The initiator of an information-seeking dialogue is typically the information seeker. His primary goal is obtaining information. This is caused by an *information gap* in the information seeker's beliefs. However, information gaps are not always relevant enough to be addressed; sometimes the cost of finding the required information is higher than the cost of accounting for the multiple possible worlds that stem from the information gap. The information seeker expects his dialogue partner to have this information, or at least does not know that he does not have the required information [94]. Alternatively, the information provider can initiate the dialogue to offer information. He may do this with or without the prospect of getting a reward from the information seeker.

Bunt [32] states that information dialogues are the most practical type of dialogue to study, because of practical reasons (it is a useful application to develop) and methodological reasons (the premises are relatively simple and straightforward, but not trivial).

Example of an information-seeking dialogue

This is an example of a human-human information-seeking dialogue from our corpus:

> A: I eh would like to eat something.
> B: That sounds like a good plan.
> A: I'll check the fridge. There's bacon, spinach, mushrooms, carrots, ehm cream cheese, regular cheese... Can you find something?

B: Well, maybe we could make a spinach salad?
A: Spinach salad.

Previous work on information-seeking dialogues

Stein and Maier [114] present a formalisation of information-seeking dialogue sequences in the so-called 'Conversational Roles' (COR) model, developed from analyses of a corpus of information retrieval dialogues. There are two different ideal courses of action: either initiated by the information seeker (A requests information from B), or initiated by the information provider (B offers information to A). Alternative courses of action include situations where either participant withdraws from the interaction, or where the information seeker is not happy with the information that B offers.

Ideal course of action (complying with role expectations)
Dialogue(A,B) → request(A,B); promise(B,A); inform(B,A); be-contented(A,B)
Dialogue(A,B) → offer(B,A); accept(A,B); inform(B,A); be-contented(A,B)

Some examples of alternative courses of action
Dialogue(A,B) → offer(B,A); withdraw(B,A)
Dialogue(A,B) → offer(B,A); reject(A,B)
Dialogue(A,B) → offer(B,A); accept(A,B); withdraw(B,A)
Dialogue(A,B) → offer(B,A); accept(A,B); inform(B,A); be-discontented(A,B)
Dialogue(A,B) → request(A,B); reject(B,A)
Dialogue(A,B) → request(A,B); promise(B,A); inform(B,A); continue(A,B); Dialogue(A,B)
Dialogue(A,B) → request(A,B); promise(B,A); withdraw(B,A); Dialogue(A,B)
...

As shown in the model above, there are many different possibilities for information-seeking dialogues. For ease of presentation, it is possible to represent the possibilities in an information state visualization [123], as we will do later in this chapter with a number of different dialogue types.

Hintikka and Saarinen [75] treat information-seeking dialogues as two-player games with a payoff that depends on the information-content of the final thesis of the player: more informative theses earn higher payoffs. There are four possible kinds of moves that can be performed by the players: initial moves, deductive moves (where one or more steps of *tableau* construction are applied), interrogative moves (where the player asks a question) and assertive moves (where the player puts forward a new thesis). It should be noted that an initial move is simply an assertive move that is posed at the start of the interaction. When one of the players poses a question (an interrogative move), his dialogue partner may either give a direct full answer or deny the presupposition of the question.

Clearly, the goal of both dialogue partners is to gain as much information as possible in order to make their final theses more informative and thereby earn higher payoffs. This is a difference with our system, where only one of the participants is trying to gain information (the information seeker) and the other is simply enabling

him to do so. Also, the information seeker does not necessarily attempt to gain as much information as possible, but only wants to obtain a specific piece of information.

A model of information-seeking dialogues

We have recorded and analyzed several information-seeking dialogues and combined them into an information state visualization (cf. information states [123]), that represents typical information-seeking dialogues that we would like our system to conduct, in Figure 3.3. It should be noted, as with the other models that we will present later in this chapter, that the states in the model are only *believed* states, not necessarily *factual* states; the system can mistakenly believe that a certain state is the case (e.g., by mishearing the user's reply). The reasoning and dialogue rules that generate the dialogues should represent this model.

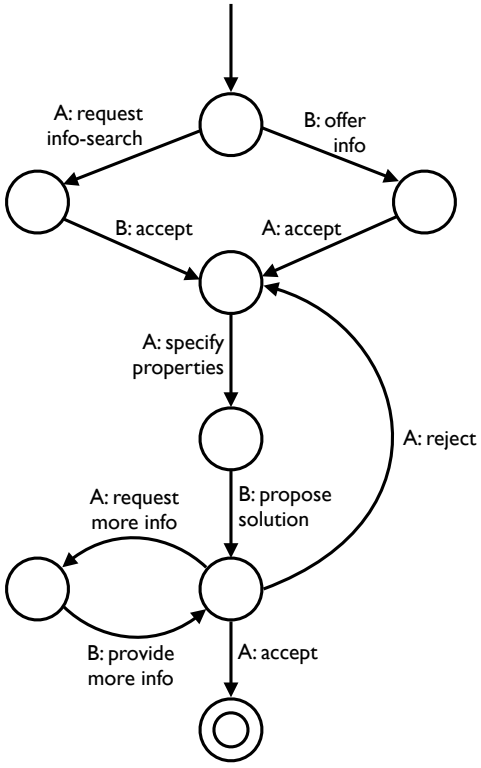

Figure 3.3: *An information-seeking dialogue where A is the information seeker and B is the information provider*

Although our model is similar to the COR model that we presented above, there are some differences. Most notably, Stein and Maier introduce opportunities for withdrawing from the conversation in every single step; while this is something that can happen in a dialogue, we would like our model to show only the successful dialogues,

since we will treat all other dialogue moves as errors (see Chapter 7). Also, what we miss in the COR model is an extra type of move in which the information seeker specifies some properties of his desired result, separately from his request for an information-search, since almost all of our recorded dialogues show this move.

A dialogue turn by one of the participants may consist of more than one dialogue act as we show them in our model in Figure 3.3. For example, participant B can initiate the dialogue with an offer to find a recipe, to which A replies with two subsequent dialogue acts: the acceptance of the offer, and the specification of properties.

In short, the dialogue is conducted as follows. First, the participants establish that B will find certain information for A, either because A requests information and B accepts, or because B offers to find information and A accepts. Then, A specifies properties that the information should adhere to, and B proposes a solution that matches these criteria. Then, A has three different options: he may either reject the proposal (after which he may specify additional properties), request more information about the proposal (after which B should give him this information), or accept the information given by B.

We can map the example dialogue above on this model. Participant A requests an information search (specifically, a recipe query) by stating: "I would like to eat something." Then, B accepts by saying: "That sounds like a good plan." A states his desired properties for a recipe: "There's bacon, spinach, mushrooms, carrots, cream cheese, regular cheese..." B proposes a solution: "Maybe we could make a spinach salad?" To this, A answers positively: "Spinach salad." This concludes the information-seeking part of the dialogue.

Errors specific to information-seeking dialogues

Most typical errors that occur in the information-seeking dialogues in our corpus involve misunderstanding of queries, caused by incorrect presuppositions like beliefs about the dialogue partner's preferences or discrepancies between the ontologies of the participants. For example:

A: Something with vegetables, and meat.
B: Lasagna with chicken and tomato?
A: That's not meat.
B: Chicken is meat, right?
A: No, that's chicken.

In the above dialogue, A does not think chicken is meat, presumably contrasting (red) meat with poultry, as is sometimes done in categorizing recipes. In contrast, B thinks chicken is meat, which is true according to the standard definition of meat[6]. Similar errors are likely to occur when definitions are not clear, like in the case of tomatoes and cucumbers, which are botanically classified as being fruit, but in colloquial use typically grouped under vegetables.

Let's compare this dialogue excerpt to the information state visualization for information-seeking dialogues that we presented in Figure 3.3. Participant A specifies

[6] *the flesh of animals as used for food*, Dictionary.com, retrieved on May 27th, 2010

his desired properties ("Something with vegetables, and meat"), to which B replies with a proposed solution ("Lasagna with chicken and tomato?"), but then A rejects this solution on the grounds of a misinterpreted property specification ("That's not meat"). Apparently, something had already gone wrong in the first step of this interaction: the properties that A meant to specify were not the properties that B used to search for a recipe. In order to repair this, A and B have to reach an agreement on A's desired properties. They do not necessarily have to agree on whether chicken is meat or not, but B merely has to understand that A wants a recipe with any meat *other than chicken*.

If the information provider does not have expert knowledge on the domain, he might not be able to answer some questions from the information-seeker about the query results:

A: And eh, a packet of tutti frutti.
B: Tutti frutti? What's that?
A: Well ehm, fruit I guess.
B: Fruit, what kind of fruit?
A: Well, I don't know.
B: You don't know.
A: It doesn't say [in the recipe].

Additionally, it is possible that the information provider does not have the desired information that the information seeker is attempting to obtain:

A: Do you have access to the list of passengers?
B: No, we don't get the passenger lists. They stay at the departure airport.
A: So you don't know who landed?
B: No, I don't.

Here, the dialogue is aborted unsuccessfully when B cannot accept A's request for an information search.

3.3.2 Tutoring dialogues

When initiating a tutoring dialogue, the primary goal of both participants is to increase and/or test the pupil's knowledge, usually on a specific subject. In order to achieve this, the tutor asks the pupil test questions (as opposed to real questions, such as in information-seeking dialogues) and/or gives the pupil information, usually in the form of hints, explanations, and corrections. The pupil's answers and/or actions are evaluated.

Example of a tutoring dialogue

The example of a tutoring dialogue that we quote here is not an exerpt from a real dialogue. The teaching/learning process in all of our recorded dialogues is left implicit by the participants, so there are no 'visible' signs of the tutoring aspect in the dialogues. To make the dialogue more perspicuous for the user, we do explicitly include

tutoring in our model. This example of a tutoring dialogue is taken from the example dialogue in Chapter 1:

> B: I will now teach you to poach an egg. First you have to boil some water.
> A: Okay.
> [rest of recipe preparation]
> B: Remove the egg from the water.
> A: I've removed the egg from the water.
> B: You have now poached an egg. Do you think you can do it by yourself next time?
> A: Yes.

Previous work on tutoring dialogues

According to Beun et al. [10], at least two different kinds of tutoring dialogues can be distinguished. First of all, in the 'traditional' tutoring interaction, the basic move is the teacher giving information to the student; then, in the diagnostic phase, the teacher asks the student a question, the student answers, and then the teacher gives feedback. In the second type of tutoring dialogue, the student and the teacher can engage in a question-answer game where the teacher tries to elicit certain responses in order to get the student to reflect on *his own* beliefs; take for example teaching tactics such as the Socratic dialogue (debates about questions like "What is justice?"). Beun later introduces a third type of tutoring dialogue where the teacher gives the student a example case with a certain underlying problem. It is then up to the student to figure out the questions that have to be answered, and then of course to answer them. In this situation, it is important that the teacher gives feedback on both phases. In this thesis, we only treat tutoring dialogues of the first type, since this is the type that is most useful in our particular domain.

AutoTutor [65] is a tutoring system that engages in a mixed-initiative dialogue with the pupil, meaning that both dialogue partners can initiate new topics of discussion or ask questions. AutoTutor has a "curriculum script" that contains questions, problems, expectations, misconceptions, and most relevant subject matter content. Dialogue moves in tutoring dialogues include test questions, real questions, feedback from the tutor (positive, neutral and negative), and several types of declarative moves: answers, hints, corrections, assertions and summarizations.

One obvious difference between 'pure' tutoring dialogues like in AutoTutor and the recipe dialogues in our system is that in tutoring dialogues, the pupil rarely asks questions [65], while we have found that in the recipe dialogues in our corpus, questions are quite common. Of course this is because the tutoring in the recipe dialogues is only a relatively minor aspect of the dialogues, whereas the execution of the task is the most important one. We can explain this by emphasizing the difference between information tutoring and task tutoring; in the former, the pupil is trying to learn information, while in the latter, the pupil is trying to learn how to perform a task. This may also include information (learning the steps in the task), but the main focus of the dialogue is the task itself.

Bunt [33] emphasises that transparency and naturalness are very important in tutoring dialogues. Naturalness is important for the student's focus on the task at hand (not having to put extra effort in conducting the dialogue), while transparency will improve the student's understanding of his own performance. In order to achieve this, the tutor should have a model of the student's level of expertise on the topic. Also, this calls for a distinction between task-oriented dialogue acts and dialogue control acts. Using dialogue control acts makes the dialogue more transparent; e.g., after the student's answer to a question, the tutor should say something along the lines of: "Correct. Next question: ..." before moving on to the next question.

A model of tutoring dialogues

In Figure 3.4, we illustrate how a tutoring dialogue in our system is typically conducted.

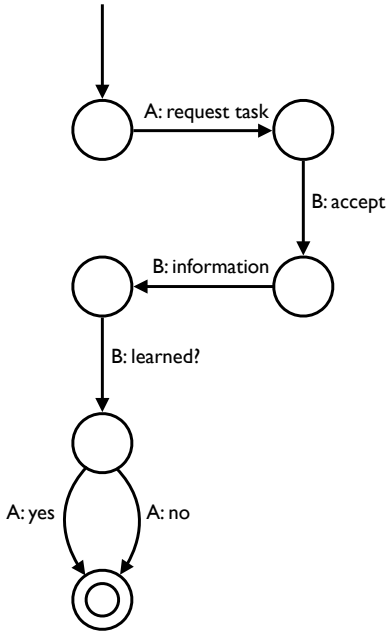

Figure 3.4: *A generic tutoring dialogue where participant A acts as pupil and participant B acts as tutor*

The dialogue is conducted as follows. Participant A requests a task to be performed, which B accepts. Then, B gives A the requested information (the instructions in the task), which may take several dialogue turns. After the task is finished, B asks A if he has learned the task. Then, A can reply positively or negatively. In either case, the dialogue is then finished.

Errors specific to tutoring dialogues

Some typical errors that occur in tutoring dialogues are wrong responses and misunderstandings of questions or answers. An interesting factor of tutoring dialogues is that many errors of the pupil may be expected/anticipated by the tutor. This means that we do not call them dialogue errors, since they still fit in the dialogue model, but instead they are treated as an integral part of the tutoring dialogue.

Expectations can be used in tutoring systems; e.g., the relatively simple expectation- and misconception-tailored (EMT) dialogue [64]. This system asks the user test questions, expects the user to answer correctly, and compares the actual answers with its expectations. Misconceptions (misunderstandings) are also anticipated, which allows for a relatively easy way of addressing them. According to Graesser [65], this is also how human tutors deal with their pupils.

3.3.3 Task-oriented dialogues

When beginning a task-oriented dialogue, the initiator's primary goal is completing a task or getting his dialogue partner to complete a task. In our case, the system acts as an expert that guides the user through the preparation of a recipe.

Example of a task-oriented dialogue

In our system, a task-oriented (sub)dialogue would typically be something like this exerpt from a dialogue from our corpus:

> A: And now you can arrange the tomatoes on the baking dish in a single layer.
> B: Okay.
> A: And the goat cheese on top of it.
> B: Okay [whistling] okay, almost done, okay that's done.

Previous work on task-oriented dialogues

Other examples of task-oriented dialogue systems are TRAINS-95 [4] [53] and its successor TRIPS [54]. In TRAINS-95, the user and the system are given a set of cities, routes and trains on certain locations, and need to figure out the most efficient set of train routes possible. The route planner in the system is deliberately weak, so the system needs the user to figure out the most optimal solution. Research with TRIPS focuses on a similar planning task, but is more advanced and more complex with respect to the domain (the task is more complicated), the reasoning (plans can be more complex), and the interaction (the collaborative problem-solving model is more advanced). In both systems, the user can ask the system questions ("How long will it take to move from A to B?") and give it commands ("Pick up the people at location A."). The system will then execute the tasks in the (virtual) domain. In this way, is actually doing exactly the opposite of our system, where the system is giving instructions to the user, who then executes the tasks.

Winograd's SHRDLU [134] works in the same way: the user can give the system instructions on grasping and moving objects in a 'block world' and ask questions about the objects (e.g., "How many blocks are not in the box?").

A model of task-oriented dialogues

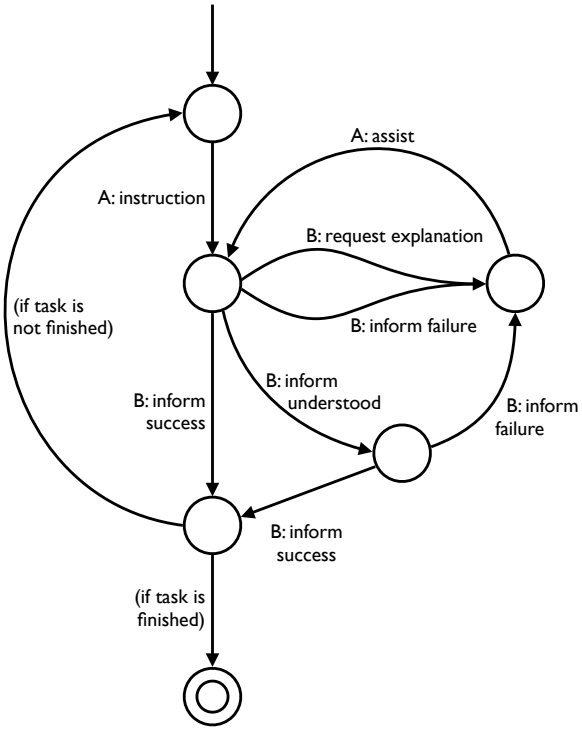

Figure 3.5: *A generic task-oriented dialogue where participant A instructs participant B to perform a certain task*

Figure 3.5 represents the typical task-oriented (instruction) dialogue. In this dialogue, participant A has the 'manual' or recipe of a certain task, and participant B has access to the domain where he can perform the required actions in order to complete the task. A recipe instruction dialogue is an instantiation of this generic task-oriented dialogue.

For each instruction in the recipe, A starts by giving the instruction. Then, B can perform any of four different dialogue actions. Firstly, B may inform A that he has successfully completed the task, in which case the dialogue is either concluded successfully (if there are no more instructions left) or the next instruction is given. Secondly, B can inform A that he has understood the instruction, after which he will (attempt to) perform the task and inform A of his success in doing so (in which case, the dialogue ends or continues as specified before) or of his failure in doing so, after

which A provides assistance. Thirdly, B can inform A that the action has failed, after which A provides assistance. Fourthly, B can ask A for an explanation, after which A also provides assistance. In the cases where A provides assistance, B can subsequently again choose between these four options.

We can map the example dialogue on the model as follows. A gives an instruction ("And now you can arrange the tomatoes on the baking dish in a single layer"), to which B replies that he has performed the task ("Okay"). Then, A gives a new instruction ("And the goat cheese on top of it"). Then, B informs A that he has understood the instruction ("Okay [whistling] okay, almost done") and then gives a confirmation that he has succeeded in performing the task ("Okay that's done").

Errors specific to task-oriented dialogues

Some typical errors that occur in task-oriented dialogues are misunderstandings of instructions, and misunderstandings in the meaning of 'okay' ('understood' vs. 'done') [66]. Furthermore, from our corpus of dialogues, slips of the tongue and turn-taking errors appear to be more common in task-oriented dialogues than in information-seeking dialogues.

Similarly to in information-seeking dialogues, a situation may occur where the instructor gives an instruction that he himself does not completely understand. This only happens if the instructor in the task-oriented dialogue does not have expert knowledge of the domain but merely has a list of instructions for the task.

A: Then the sauce has to simmer for circa fifteen minutes.
B: With the lid on?
A: Eh, doesn't say, but simmering is probably with the lid on.

Also specific for task-oriented dialogues are errors in the following of instructions. These errors might be detected some dialogue turns after they occur, like in the next example, where participant B is trying to rebuild a LEGO® construction from an example that only A has. A and B can both observe the domain and point at blocks, but only B is allowed to manipulate the domain. Halfway through this segment of the dialogue, A discovers that he made a mistake earlier in the dialogue, which is discovered when B needs a brick that is not available.

A: Yeah, ehm... A red square one here.
B: The long one?
A: No, a square, sm– small.
B: That's not possible, I don't have it. A red one, I have to remove that somewhere then... Is this red one...?
A: Yeah, that one is correct.
B: That one is correct?
A: Yes. Oh wait, I did make a small mistake.
B: Should...
A: Uh take this one off... Yes, and then this brick off of here
B: Yeah.
A: And then remove the square and put the long one in. Yeah, like that.

3.3.4 Mixtures between different types of dialogues

An example of a task-oriented tutoring system is Steve [102]. Steve teaches his pupils how to perform certain tasks. For this process, he chooses one of three different roles: a tutoring move (e.g., giving feedback to the student), an instruction move, or a dialogue control move (a turn-taking or grounding act). Steve's highest priority is always to respond to the student (tutoring moves). If no such move is needed, Steve will perform a dialogue control move, if required. If neither a tutoring or dialogue control act is needed, Steve will continue with the next instruction of the task.

The virtual character REA [15], a digital real estate agent that is capable of interacting multimodally, is specifically designed to make small talk with the user in order to build up trust. Since REA functions as a (virtual) real estate agent, it is very important to build trust with the user, which is indeed one of the effects of social dialogue that Bickmore and Cassell found [15]. REA assesses her dialogue moves on a scale of face threat, combines this with the current level of solidarity and familiarity that she holds with the user, and picks a relevant topic for the user. Based on these factors, she chooses whether or not the next dialogue move will be a social move (small talk) or a domain-related move.

Our recipe dialogues are also a mix of some different types of dialogues. They are information-seeking dialogues in the sense that the user seeks information about certain recipes, they are tutoring dialogues in the sense that the system teaches the user to prepare a certain recipe, and they are task-oriented dialogues because they focus on a certain task (preparing the recipe). A system that combines these categories will, typically, for each turn choose a role. This means that not all three categories are used simultaneously; rather, the system switches between the different types of dialogues.

For our system, we have chosen a set pattern according to which the dialogue is conducted. At the start of the dialogue, the user will usually initiate a recipe choosing dialogue, a typical example of an information-seeking dialogue. The recipe preparation dialogue is mostly a task-oriented instruction dialogue, but has properties of a tutoring dialogue when the system presents the task at hand to the user ("I will teach you how to poach an egg") and closes off the task and asks the user whether he knows the task now ("You have poached an egg now. Do you think you can poach an egg by yourself the next time?").

In Figure 3.6, we describe possible recipe searching and preparation dialogues that we would like our system to be able to produce. In this model, U stands for user and S for system. The user always initiates the dialogue, either by directly requesting a joint goal or by requesting a recipe. In the latter case, the system will propose a recipe, which the user can then accept or reject. If the user rejects the recipe, the system will propose a different one. When the user has accepted the recipe or when system accepts the joint goal that the user requested, the system will acknowledge this.

Then, the recipe preparation will start. In this part of the dialogue, the main process is as follows. Either the system will instruct the user to perform a certain task, after which the user may or may not acknowledge that he is going to perform the task, then the user performs this task and may or may not confirm that he has

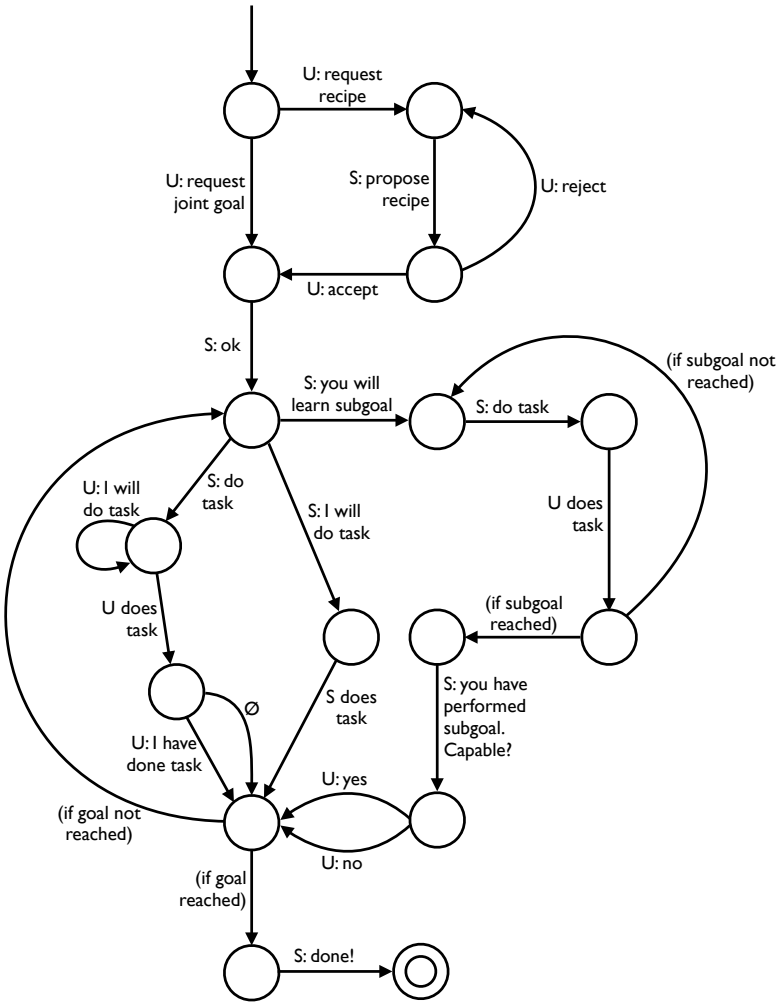

Figure 3.6: *A recipe searching and preparation dialogue*

performed the task. Alternatively, the system will inform the user that it will perform a certain task, after which it performs the task. As long as the goal is not reached, this will be repeated; if the goal is reached, the system informs the user that he is done.

Additionally, there is the tutoring process, where the system informs the user that it will teach him to prepare a certain subgoal. In this case, during the tutoring process, the system will not offer to perform tasks, but will only instruct the user to perform tasks (after which the user is, of course, expected to perform a task). This will be repeated as long as the subgoal in question is not reached. When the subgoal is reached, the system informs the user of this, and asks him whether he is capable

Requirement	Code
One primary domain goal	RI1
At least speech acts *inform* and *request*	RI2
Participants inform each other about status of goals and plans	RI3
Modeling of mutual belief	RI4
Implementation of dialogues according to the dialogue model in Figure 3.6	RI5
Ability to answer questions	RI6
Ability to respond to requests from other participants	RI7

Table 3.3: *Requirements for the interaction*

of performing the task in question on his own the next time. The user can answer positively or negatively to this; either way, the dialogue will return to the regular instruction process.

3.3.5 Requirements for interaction

General Requirements

In order to maintain correct and sufficient mutual beliefs, we need to model the concept of mutual belief (RI4). Also, the system should always communicate about the status of the tasks and the goals in question (RI3). The same goes for joint persistent goals, as we have seen in the previous chapter. To communicate this necessary information, the system needs a few different kinds of speech acts: at least *inform* and *request* are required (RI2). It should be noted that these speech acts are at the pragmatic level and could take different syntactic forms. For example, a request can take the form of an *inform* speech act on the syntactic level: e.g., "You need to boil some water." We will discuss the analysis of such indirect speech acts in Chapter 6.

Inform is used when the user states that he is done with a task, and when the system informs the user that it will perform a task. When one of the participants has performed a task or is planning to perform a task, the other participant is not necessarily aware of this. In order to make the status of the task in question mutually believed, the performer of the task must inform the other participant of this. Also, when the current goal is completed or dropped, the system will inform the user of this in order to make the status of the goal mutually believed.

Request is used when one of the participants requests the other to perform a certain task. Morante [94] distinguishes between *request* and *instruct*; the former is a weaker version of the latter. When *instruct* is used, the addressee's attitude towards the instructed action is not taken into account by the speaker, while for *request*, the willingness of the addressee is taken as a precondition. In our research, this distinction is irrelevant, both for requests from the system and from the user. In case of requests from the system, the user is always the initiator of the joint plan and we assume that when one adopts a joint goal, one is always willing to perform the necessary actions for this. Conversely, in case of requests from the user, because of the system's role as

a cooperative assistant, it is always willing to perform actions that the user requests from it.

The different types of dialogues that we have treated in this section should all be accounted for in our system. The dialogues should follow the dialogue model that we have presented in this chapter (RI5). Additionally, in order to behave cooperatively, the system should be able to answer questions from the user (RI6) and follow requests from the user (RI7).

Domain-Specific Requirements

In a recipe dialogue, different kinds of questions from the user should be expected. The different kinds of questions can range from not knowing how to do something ("How do I boil water?") to needing more detailed information ("How thinly should I slice them?"). Handling these questions involves starting a new (parallel) 'side' process for answering these questions while the recipe instruction dialogue is temporarily suspended until the issue is resolved.

Another type of question is a request to expand the current recipe step to its subtasks ("How do I poach an egg?"). Questions about the future steps in the recipe include requests for the next instruction ("What do I have to do next?") and more general questions about the recipe ("Should I cut them now or can I leave it until later?" or "What do I need the wine for?").

Additionally, the user can request the system to perform an action ("Can you keep the time for me?"). If the system is capable of performing this action, it should do so; otherwise, it should inform the user that it is not capable of performing the action.

The same principles hold for the blog instruction system.

3.4 Multimodal interaction

As we have seen in the previous chapter, interaction is usually conducted over multiple modalities. This multimodal interaction consists of (joint) activities and dialogue. In multimodal interaction in general, users interact with a computer through several different input modes. Besides the usual mouse and keyboard (or touch screen) interface, users may also be able to talk to the computer or point at physical objects or at things on a screen. This multimodal interaction can also apply to output modalities of the system: the computer might be able to respond to the user not only on the computer screen, but also in other ways, like speaking to the user. In the same way, multimodal dialogues are conversations that use different modalities, as humans usually do when talking face-to-face. The most notable other modalities besides speaking and listening are visual modalities, such as pictures on a computer screen, or gestures or facial expressions, either on an animated face on a computer screen or on a physical robot.

The large amount of information that is involved in in a multimodal interaction is both an advantage and a disadvantage: a multimodal interaction is usually more robust than an interaction that is only conducted on a single modality, but on the other hand, misunderstandings may emerge if information in different modalities is inconsistent. Also, the fact that there may be input over multiple modalities that is

Requirement	Code
Synchronization of input	RM1
Synchronization of output	RM2
Intuitive representation of multimodal interaction	RM3

Table 3.4: *Requirements for multimodal interaction*

connected but may or may not occur exactly simultaneously, means that multimodal interaction has a relatively large tendency to become complicated and confusing. This creates the need for synchronization of input and output.

Van Dam [47] presents a model for human-computer interaction in which actions in graphical user interfaces (GUIs) are interpreted as dialogue turns. This allows for an integrated approach of multimodal human-computer interaction where information is exchanged through linguistic (speech or text) and graphical (most notably on-screen) means. We present a similar model in which visual information (in the form of events taking place in the domain) is also taken into account as input, besides the linguistic input that forms the basis of the dialogue.

3.4.1 Requirements for multimodal interaction

Input over different modalities needs to be synchronized (RM1). For example, causally dependent actions in different modalities rarely happen exactly at the same time [97], but should be interpreted together. For example, a deictic gesture (some form of pointing) is often not performed at exactly the same time as the uttering of a sentence with a deictic component, e.g., "Could you give me more information about *this* restaurant?" Humans usually interpret these events as happening simultaneously. This illustrates the need for synchronization of input to different modalities.

In our system, if the user gives an answer to the system's question and the system sees the user performing some action a very short timespan later, these events should be interpreted as happening at the same time instead of one after the other. The time span can be defined differently for different users; e.g., elderly and disabled people may need a longer time span to perform concurrent actions than, for example, younger people with computer experience.

Besides multimodal input, also multimodal output needs to be synchronized (RM2). For example, a speech synthesizer may need a relatively long time to prepare a sentence and turn it into a sound file, while screen output may be ready almost instantaneously. If both of these output modalities are so strongly interconnected that they need to be conveyed to the user at the same time, this means that the screen output needs to be put 'on hold' until the speech synthesizer is also ready.

Because multimodal interaction is much more complex than text-only conversation systems, we believe that it is important to have an intuitive way of representing and modeling multimodal interaction (RM3).

3.5 Error handling

There are several different reasons to implement error handling in a dialogue system. Because is it unlikely that a conversation goes exactly as planned, it is desirable to implement error handling in order to ensure that the dialogue partners will be able to get out of an error situation. In this way, error handling will "contribute to a more efficient and satisfactory human-computer interaction" [12]. In this section, we present our definition of errors, explore some related work, and give a list of requirements that we believe an error handling module for a task-oriented system should comply with.

3.5.1 Our definition of errors

An error is usually defined as a deviation from accuracy or correctness; a mistake, as in action or speech. Errors have to do with discrepancies between the intended consequences of an action (including communicative actions) and the actual consequences of that action. In order to use this definition, we need to define what this 'correctness' is. We will take the dialogue models from this chapter as the norm for successful dialogues. Basically, this means that dialogue moves that adhere to the dialogue models are allowed, and any other moves are errors. These moves that fall outside the dialogue models must be handled in some way that will bring the conversation back to the dialogue as it was planned.

As we have seen in Chapter 2, the notion of success is twofold: an utterance can be successful or unsuccessful, and the dialogue itself can also be successful or unsuccessful. To take a pragmatic viewpoint on the matter, we will only look at the success of utterances. However, the aim to adhere to the dialogue models as much as possible in order to prevent a dialogue breakdown is of course strongly connected to the aim to avoid unsuccessful dialogues.

Beside moves that fall outside of the dialogue models, we must make sure that the mutual beliefs of the dialogue participants are both internally consistent and consistent with the dialogue partners' private beliefs. Since we assume that dialogue contributions are always added to the dialogue partners' mutual beliefs unless specified otherwise, we need to repair such inconsistencies immediately. This means that all inconsistencies between utterances and the hearer's beliefs should be detected and addressed as quickly as possible.

According to Airenti et al. [3], there are two types of errors: the 'inference chain' (cf. dialogue model) is interrupted, or it"follows a complete path different from the one intended by the actor". In the first case, the dialogue is aborted (before all goals have been achieved) and therefore unsuccessful. The second case is caused by the incorrect application of rules, caused by incorrect beliefs that the agent has. "In fact, it is precisely the presence or absence of certain representations that may lead the partner to apply a rule that the actor did not want him to apply [or vice versa]." In this case, the dialogue does continue, but in an undesirable way. Errors in mutual beliefs are also mentioned: one of the possible causes for the application of incorrect rules is if "representations supposedly shared by the actor and the partner, in fact are not."

Here, we have our two types of errors: the application of a rule that is unexpected

by the addressee, or incorrect beliefs about the mutual beliefs of the dialogue partners. In conclusion, we define an erroneous dialogue move as a move that satisfies at least one of the following conditions:

1. It is an unexpected contribution; that is, it is inconsistent with the addressee's intentions.

2. It signals or introduces an inconsistency in the mutual beliefs of the dialogue partners.

If an error is detected, it should be repaired immediately so that the participants can return to the planned course of the dialogue. Goffman [62] sees repair dialogues simply as 'side sequences' from the main dialogue. This view is supported by Taylor [120], who proposes a "general protocol grammar" that also describes handling of ambiguity and errors. Taylor proposes a flow chart that looks similar to our dialogue models as presented in Chapter 3, though on a different level than our dialogue models, for each primary message in the dialogue. Every *Primary* dialogue contribution by an *Originator* can either be accepted (with or without feedback) by the *Recipient*, or a subdialogue is activated in case an error or ambiguity is observed by the Recipient or in case a correction is provided by the Originator after the Recipient's feedback.

Related work on error handling

The digital real estate agent REA [15] can correct errors whenever the agent misunderstands the user's utterance. This error correction, however, seems to consist mostly of answering requests for clarification from the user [36], while our error handling system handles a broader scope of errors.

In Turunen and Hakulinen [125], an error handling system with seven phases is programmed in Jaspis. First of all, an error is detected. Second, a diagnosis of the cause of the error is made. Third, the system plans a correction. Fourth, the correction is executed. Fifth, the system informs the user about the error. Sixth, the error handling phase is closed and the system returns to the primary dialogue. The seventh step that Turunen and Hakulinen mention is preventing errors, which is rather a separate process than a step in the main error handling process. In our system, the detection of errors is comparable to step one and two, and the repair of errors corresponds with step three and four. We do not explicitly inform the user of the fact that an error is being repaired (step five) or explicitly close the error handling phase (step six). Error prevention (step seven) is something that we leave for future work.

Beun and Van Eijk [12] state that the goal of repairing an error is establishing *alignment* between the dialogue partners. This can be achieved by employing two tactics: first of all, resolving detected discrepancies, and secondly, avoiding potential discrepancies as a result of quantity implicatures. Resolving detected discrepancies is done by generating a feedback message, while avoiding potential discrepancies is done by analyzing the possible discrepancies that can result from a message and then generating the message that will avoid possible discrepancies as much as possible. The addressed discrepancies are purely of the form of conceptual mismatches (on the

ontological level). While this is an important type of discrepancy, we will address a broader scope of errors in this thesis. The repair depends on the formal pragmatic model (which contains information such as private and common beliefs and a user model), the role of the system, and the disparities between the user's and the system's beliefs. A difference with our work is that Beun and Van Eijk have a closed world assumption because the system is an expert (if the system does not believe that p, then it believes that not p). We do not have a closed world assumption; in our system, ¬Bp is not the same as B¬p.

3.5.2 Error handling in our system

The goal of the error handling framework that we present in this thesis is to augment a basic instruction system with an error handling module that makes the system robust [16] [101]; in other words, the system should not crash, stop before reaching its goals, or resort to generic responses such as "I'm sorry, I don't know what you mean." While the latter option is less bad than simply withdrawing from the interaction, it is not an optimal response for three reasons. First of all, a response that is more tailored to the specific type of error will give the user a better suggestion of how to continue the conversation. Secondly, giving specific responses results in a more transparent error handling subdialogue, since it gives the user more information about what the system believes went wrong. Thirdly, especially in cases where errors tend to occur frequently, a generic response may cause more displeasure for the user, as we have seen in the first example dialogue in Chapter 1.

We have investigated human-human dialogues for the handling of errors in a natural way, but we do not necessarily handle all errors in a similar way to humans. In human-human dialogues, a generic reaction like 'what?' or 'sorry?' may indicate, amongst others, a mishearing, a misunderstanding, or insecurity about the meaning or purpose of the utterance. The answer to such an utterance may differ between or even combine different approaches: e.g., repeating the information literally or partially, offering a more detailed explanation of (part of) the information, or indicating the reason for the utterance. However, we wish to be as clear as possible in the handling of errors and should thus avoid such generic responses.

One of the most important problems that should be avoided is the presence of inconsistencies between the mutual beliefs of the participants. In any interaction between two or more agents that are not specifically tailored to each other's goals, values, ontology, and vocabulary, and that do not by design have complete and accurate beliefs about each other's mental states, etcetera, there are always uncertain factors at play. Also, in real-world situations (as opposed to deterministic computer models), there are uncertainties in the environment, such as unexpected events or unexpected consequences of actions.

Since it is not always possible to find out the exact status of these uncertain factors, or computationally feasible to take all possibilities into account, agents can have assumptions that might in practice sometimes be false (cf. default reasoning). If an error occurs, it might be caused by one of these assumptions being false, and in that case we must initiate the corresponding repair strategy.

Examples of assumptions that may be false:

Requirement	Code
No generic responses	RE1
Always respond	RE2
Generic error handling tactic for task-oriented dialogues	RE3
Address errors immediately when they occur	RE4
(Mutual) belief inconsistencies should be avoided/resolved	RE5
Be as cooperative as possible, even when an error occurs	RE6
Take multimodal input into account when handling errors	RE7
Hold dialogue according to the dialogue model as much as possible	RE8

Table 3.5: *Requirements for error handling*

- all participants have accurate beliefs about their own and each other's capabilities
- all participants have accurate perceptions of and beliefs about the domain
- if one of the participants says something, all others hear and understand it correctly and it becomes mutually believed
- all participants only say things they believe to be true
- all participants are cooperative
- all participants are communicative about the status of the current (sub)goal (i.e. when it is finished, they inform the other participant(s) of this)

This is not a complete list of such assumptions. Also, these assumptions are of different types and different levels. Some are about communicative conventions that are based on the cooperative nature of the participants (e.g., adherence to Grice's maxims, communication about the status of subgoals), while others are basic assumptions about the circumstances of the interaction (e.g., reliable communication channel, accurate perceptions of the domain). The fact that we explicitly take these to be assumptions (and not facts) gives us an entry point for error handling: when something out of the ordinary occurs, it is probably caused by one of those assumptions that is, in fact, false. For example, if the user disagrees with the system about whether a certain food item is in stock, the assumption that the system has accurate beliefs about the domain is in this case possibly false.

We do not take into account possible errors in all of these assumptions, but instead focus on a selected subset of them.

3.5.3 Requirements for error handling

In Table 3.5 we show the list of requirements for error handling.

Since we present our dialogue system framework on the level of speech acts, we will not treat speech recognition errors such as mishearings. In Chapter 7, we will explain in more detail which errors we will and will not treat. However, as we have stated

above, one of the main errors that we focus on is belief inconsistencies, which should be avoided as much as possible and resolved whenever they are detected (RE5).

As we have stated in the previous subsection, the system should not resort to giving generic responses such as "I don't know what you mean" (RE1). Additionally, the system should always respond to input from the user (RE2).

We wish to construct our error handling framework in a way that enables us to reuse it in other task-oriented dialogue systems (RE3). This means that our error handling framework is a generic set of rules for error detection and handling, that forms a separate module from the main dialogue system, and that can be used with other task-oriented dialogue systems that follow the principles that we present in this chapter and in Chapter 4.

When our system detects a problem serious enough to warrant a repair, it should try to initiate and repair the problem at the first opportunity after detecting it (Clark's *Principle of repair* [40]; RE4). It is important to detect errors as soon as possible and address them immediately when they are detected, in order to prevent an error spiral caused by an increasingly frustrated user [76]. For these reasons, the responses that our framework produces should be constructive and should reflect the error that has been detected.

In handling errors, the system should be as cooperative as possible (RE6); this means that even when a question is not expected, the system should still answer it. For the correct interpretation of input it may be necessary to consider input over multiple modalities. To account for this, the system needs to have a unified error handling strategy to deal with multimodal input (RE7). Because the system can only conduct dialogues according to its dialogue rules, the system should always attempt to have the dialogue go according to its dialogue models. This means that any unexpected dialogue moves should be addressed by informing the user of the dialogue move that was expected, to motivate a return to the planned course of the dialogue (RE8).

3.6 Conclusions and future work

In this chapter, we have presented a number of requirements for a cooperative system that can produce dialogues that have information-seeking, task-oriented, and tutoring aspects. Tables 3.1, 3.2, 3.3, 3.4, and 3.5 summarize the requirements that we have presented, sorted into requirements that pertain to reasoning, the domain, the interaction, multimodal interaction, and error handling, respectively. In the next chapters, we will show how our framework fulfills these requirements. We have also presented a dialogue model for recipe instruction dialogues that our system should be able to produce (Figure 3.6).

Our system is mainly designed for goal-oriented purposes: instructing a pupil, making sure a task is performed or a goal is reached, and conveying information in a matter-of-fact manner. We take these task-oriented aspects of the interaction as the main constituent of the system, but some form of social behavior and politeness may also be taken into account to ensure a pleasant interaction. When beginning a social dialogue, the initiator's primary goal is building up a positive relationship and an emotional bond between the dialogue partners. Another goal of such a dialogue

can be to teach the dialogue partner "more sophisticated communication skills" [22] by conditioning with positive responses.

But then, also errors in social dialogues must be taken into account. A typical error that can occur in social dialogues is impoliteness or *rudeness* [24]. *Face-threatening acts* (ibid.) are acts that might lower the face (i.e. value, honor, social status) of the dialogue partner. These acts should be avoided as much as possible. Besides deliberate face-threatening acts, participants in a joint activity might also make errors in assessing the face-threat of their acts (i.e. the extent to which their actions are face-threatening for other participants). This can especially happen when agents from different (sub)cultures interact, since both the content of face and the acts that are face-threatening differ in different cultures and subcultures. For example, when offered a small gift or favor, it is customary for a member of Western society to reply with a simple "Thanks", while in India, a more appropriate reaction is "I am humiliated, so awful is my debt." [24] Either reply would be considered impolite in the other culture. This reflects the difference in the extent to which the action is considered a face-threatening act in both cultures; in Western cultures, an offer is not very face-threatening, but in Asian cultures, an offer as small as a glass of ice-water implies a large debt. We leave the addition of such factors to our dialogue model as a possible topic of future work.

In future work, it could be interesting to explore also the requirements that we believe are not essential; e.g., a system that can work with multiple users or a system consisting of multiple instructors.

4 Architecture

> The question of whether computers can think is just like the question of whether submarines can swim.
>
> *Edsger W. Dijkstra*

In this chapter, we will take the requirements from the previous chapter and construct an architecture for an agent that can conduct task-oriented dialogues with a user. The architecture we present in this chapter is based on our view on a generic companion robot architecture [116] and contains the components that are needed to satisfy the aforementioned requirements. Furthermore, we will present a visualization that can assist designers in creating a multimodal interaction system. In the next chapter, we will discuss the specific implementation of our basic system, and in Chapter 7, we will augment the basic system with error handling that follows the requirements that we have presented in Section 3.5.

The chapter is divided in four parts, which mirror the first four parts from Chapter 3; error handling (the fifth part) will be treated in Chapter 7. For clarity, we have marked parts of our proposed implementation with the requirement codes that we have listed in the tables in the previous chapter. In Section 4.1, we discuss the participants in the interaction, treating the user and the system (4.1.1), the reasoning (4.1.2), specifically focusing on the concept of capabilities (4.1.3) and the user model (4.1.4), and finishing with a companion robot architecture in 4.1.5. Section 4.2 pertains to the domain, specifically to the formalisation of the domain (4.2.1), the formalisation of recipes (4.2.2) and actions (4.2.3) and introducing the concept of opportunities (4.2.4). In Section 4.3, we discuss interaction, focusing on dialogue goals (4.3.1), dialogue planning that results from those goals (4.3.2), the information that can occur as communicated content of speech acts (4.3.3), and briefly, on natural language (4.3.4). In Section 4.4, we discuss multimodal interaction in more detail, and in Section 4.5 we finish the chapter with conclusions and future work.

4.1 Participants

4.1.1 User and system

Despite the possibility of having multiple participants in various roles, we have developed our framework for a setting where a single user acts as the pupil who will execute most of the actions and a companion robot takes on the role of instructor. The specific companion robot for which we have developed our framework is the iCat, as we have explained in Chapter 1.

As we have seen in the previous chapter, our system should allow for mixed-initiative action and interaction in dynamic environments. Important aspects of this are the presence of joint goals that all participants want to achieve, and the fact that it is not a priori clear which of the participants are able to perform certain actions that are needed to achieve the joint goal. The latter aspect implies that it is also not a priori clear which plan should be selected in order to achieve a given joint goal, since the appropriateness of a plan depends on the abilities of the participants and on the environment.

4.1.2 Reasoning

As we have seen in Chapter 2, mutual beliefs, joint goals and joint plans are key concepts in any cooperative interaction. Therefore, we augment the BDI model [19] with some extra rules that enable us to work with the mutual/joint versions of its basic concepts of beliefs, goals, and plans (requirements RR1 and RI4).

For each joint goal `G` that the system and the user have together, they will attempt to achieve this goal together in a cooperative manner. The system does not necessarily have exactly one plan for a certain joint goal (requirement RR5); it may have no plan, or more than one plan. Before a joint goal is adopted, the system checks whether it has a (joint) plan to achieve the current joint goal, and then checks whether the actions in this plan are possible, given the circumstances (requirement RR2). We will elaborate on this process later in this chapter when we discuss actions. If there is no such plan, the system informs the user of this. If there is more than one possible plan available, the system randomly chooses one.

If the recipe passes these checks, the preparation of the recipe can start by calling the procedure `prepare(G, P)`, for goal `G` and plan `P`. This procedure involves jointly preparing the recipe with the user by giving the user instructions about the preparation of the recipe and/or performing certain actions, depending on the system's beliefs about which tasks can be performed by which participant.

`checked-done(T)` is a property of task `T`. If the system believes that `checked-done(T)`, this means that the system has checked that task `T` is done, regardless of how and by whom, which is not important at this point. This belief can be reached in a number of ways: whenever the system has performed the task, when it has observed that the task is done, or when the user informs the system that he has performed the task (requirement RD7).

4.1.3 Capabilities

Having the *capability* to perform a certain task means having the skill to perform it and knowing how it is done. Capabilities are used to determine the level on which the system gives instructions and the choice of the participant that will perform the task. The system always gives instructions it believes the user can work with (not too easy, not too difficult); i.e. tasks that the system believes the user is capable of performing (requirement RR6).

`capable(A, T)` is a relation between agent `A` and task `T`. If the system believes `capable(A, T)`, this means that the system believes that agent `A` can perform task `T` autonomously and single-handedly. It should be noted that the concept of capabilities does not take into account the current circumstances in the performance of the task (such as the availability of ingredients), but merely states tasks that can be performed by the agent in question whenever the necessary preconditions are fulfilled.

Grosz and Kraus' [71] concept of CBA ('can bring about') is similar to our concept of capabilities. Something that we do not include in our definition of capabilities, but is included in the definition of CBA, is the notion of "various situational constraints on the performance of the action" that the particular plan needs to satisfy (e.g., being done by a certain time).

Capabilities are explicitly registered in the beliefbase of the system and are of the form `capable(U, Boil_water)`. They are added to the beliefbase when the user informs the system that he is capable of performing a certain task. Also, when the user has performed a certain task and confirms that he is able to do it autonomously the next time, the system adds the belief that the user is capable of performing this task to the beliefbase.

If the system believes that the user is not capable of performing a certain task, the system will not instruct him to perform this task, but instead split the task into subtasks that it believes the user is capable of performing, and instruct the user to perform these (requirement RR8).

Recall Castelfranchi's open delegation [39], where is it not necessary for an agent to have a complete plan of a goal state or higher-level task that it delegates to another agent. To enable our system to openly delegate tasks to the user, it is possible that the system believes that the user is capable of performing a certain higher-level task T without believing that he is also capable of performing all atomic actions that are in the system's recipe for T (requirements RD4 and RD5). This can have different causes: either the system has incomplete beliefs about the user's capabilities, or the user has a different recipe to achieve T. For example, the capability of making Eggs Benedict (half of a breakfast muffin, topped with ham, a poached egg, and hollandaise sauce) does not necessarily imply the capability of making breakfast muffins; after all, one could simply buy them instead of making them. Making the breakfast muffins from scratch is only one of the possible plans that can be formed in order to achieve the subgoal in question.

4.1.4 User model

In order to have an effective communication, it is necessary to have beliefs about the dialogue partner's mental states [51] [106]. We use such a *user model* also for

error detection and repair, but also on a more basic level, it can be seen as one of the key principles of communication. Scassellati [106] states that a *theory of mind* ("the ability to correctly attribute beliefs, goals, and percepts to other people") is one of the defining concepts of human interactions. Scassellati uses a theory of mind in a humanoid robot and integrates it with vision and visual processing, which allows the robot to determine the direction of gaze and deduce the point of attention of the interaction partner. We do not use such physical cues and therefore are not able to deduce information about attention of the user, but we can use other input (most prominently, the user's communicative acts) to deduce as much information as possible.

Scassellati bases his system partially on work by Leslie [85], who uses three different modules that process three corresponding classes of events: events that can be explained by mechanical rules (*mechanical agency*), events that can be explained by intentions and goals of agents (*actional agency*), and events that can be explained in terms of attitudes and beliefs of agents (*attitudinal agency*). Of these three, we only use the latter two. The concept of attitudinal agency can furthermore be used for "understanding that others hold beliefs that differ from our own knowledge or from the observable world" [106], which we may in turn use for error handling (see Chapter 7).

The user model is also tightly connected to pre- and (expected/desired) postconditions of communicative actions. For example, the action of informing is "generally defined to be an action whose main effect on hearer's mental state is that the hearer believes that the speaker believes the propositional content [of the communicative act]." [51]

4.1.5 Architecture

In Steunebrink et al. [116], we have presented a generic architecture for companion robots (see Figure 4.1), which abstracts from specific robot details, making it useful for different types of companion robots. The architecture contains functional components that each contain several modules that are functionally related. Modules drawn as straight boxes represent data storages, the rounded boxes represent processes, and the ovals represent sensors and actuators. Each process is allowed to run in a separate thread, or even on a different, dedicated machine.

This architecture takes into account the possible existence of multiple *input modalities*, multiple *input preprocessing* modules for each input modality, *databases* for filtering, storing, and querying relevant information, *action selection engines* for complex, goal-directed, long-term processes such as conversing, planning, and movement, an *emotion synthesizer* producing emotions that influence action selection and facial expressions, multiple (reactive) *low-level behaviors* that can compete for output control, multiple *output preprocessing* modules including a conflict manager, and finally, multiple *output modalities*. The interfaces (arrows) between different modules indicate flow of data or control. Note that only the 'ultimate' companion robot would fully implement all depicted modules; a typical companion robot implementation will probably leave out some modules or implement them empty, awaiting future work.

For a more detailed description of the complete architecture, we refer to Steune-

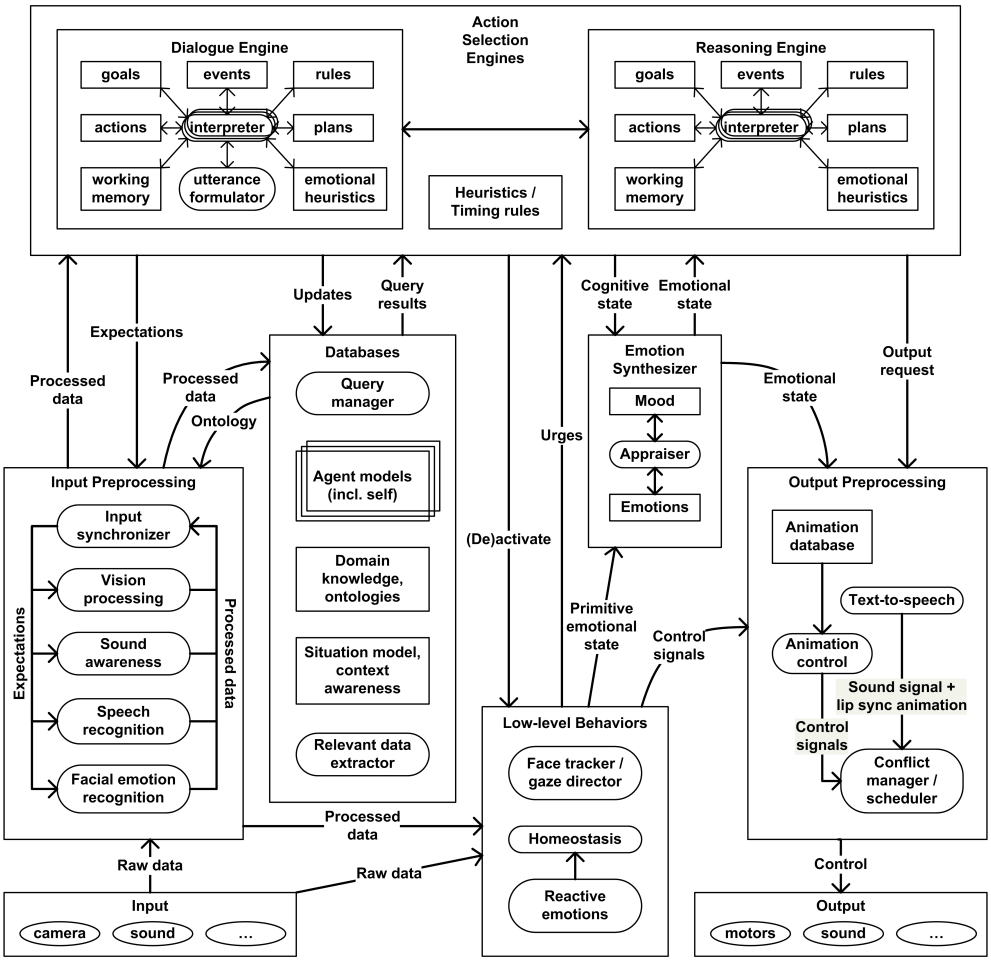

Figure 4.1: *A generic architecture for a companion robot [116]*

brink et al. [116]

Our instantiation of the generic architecture

From this proposed architecture, we will only describe the parts that we use in this thesis. In Figure 4.2, we show only the parts that we use in our framework. In this section, we will first describe the functional components (represented as blocks in the figure) and then the interfaces (represented by arrows) between the components.

The *input modalities* in our system are, in essence, a camera and a microphone. The *input preprocessing* that we use consists of speech recognition followed by natural language processing. For this, we use Functional Discourse Grammar (see Chapter 6). The input from the camera should be processed to recognize certain objects. As we have seen in the previous chapter, there is a need for an input synchronizer that

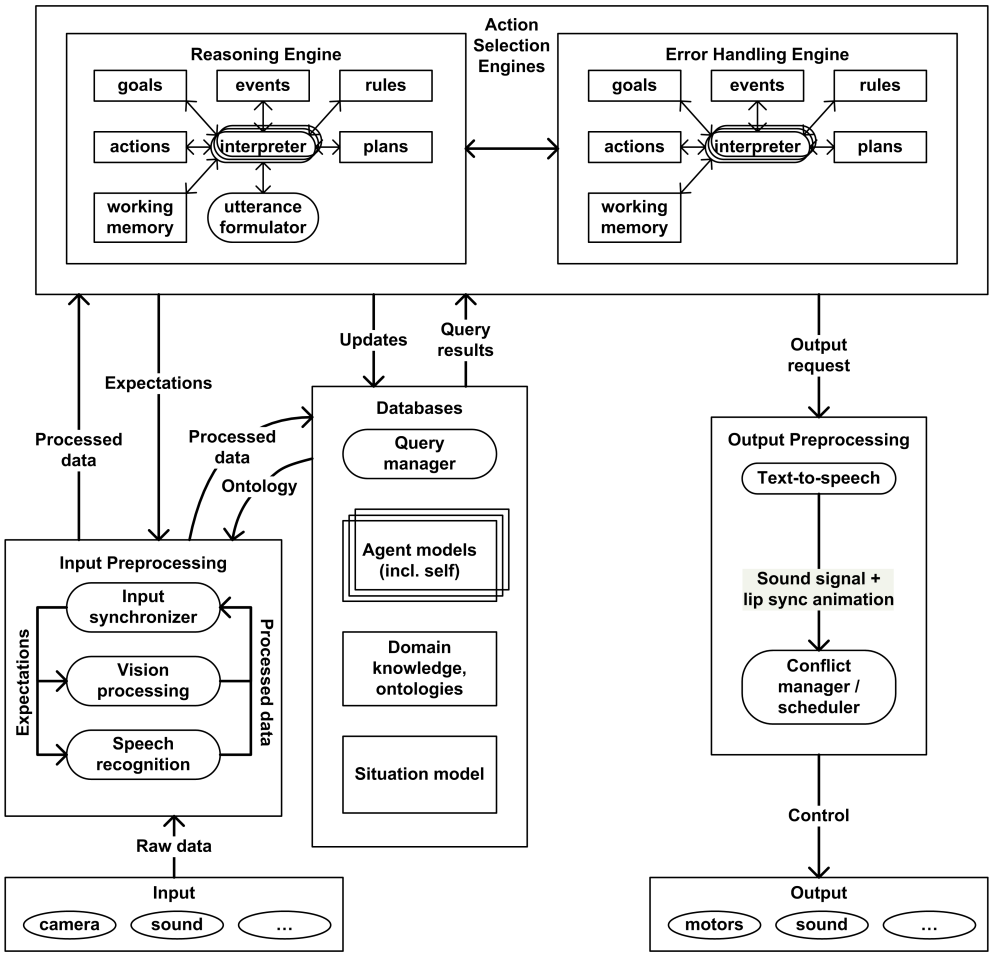

Figure 4.2: *Our system architecture, based on the generic architecture from Steunebrink et al. [116]*

can link processed data from different modalities, in order to pass it as a single event to another module.

We do not use any *low-level behaviors* or such as face tracking and gaze directing, blinking, breathing, and other 'idle' animations at this time. We have also not taken into account any homeostasis processes such as the need for interaction, sleep, and 'hunger' (e.g., low battery power), nor reactive emotions such as startle and disgust. Similarly, we also do not use the *emotion synthesizer*.

The 'heart' of the architecture is formed by the *action selection engines*. These are cognitive-level processes that select actions based on collections of data, goals, plans, events, and rules. The outputs that they produce can generally not be directly executed by the actuators, but will have to be preprocessed first to appropriate control signals. As we have mentioned above, the reasoning engine is based on the BDI

paradigm [19]. It decides which actions to take based on events, rules, goals, beliefs, and plans. There may be multiple action selection engines; for example, we could separate the reasoning about recipes and ingredients, dialogue management, and error handling. In this thesis, we take one action selection engine for the basic system and one for error handling. We do not have a separate dialogue engine, like the generic architecture in Figure 4.1, because in our framework, the dialogue emerges from joint goals and is therefore tightly coupled to the main reasoning process that handles the joint goals.

In our system, the *databases* in the architecture are different parts of the beliefs of the BDI agent. For example, the domain knowledge (recipes) and user model are all part of the agent's beliefs. The agent also has a mental model of its own goals and plans in order to be able to reason about them. It should be noted that in terms of the BDI theory, the databases component plus the working memories of the action selection engines constitute the system's beliefs.

The actions selected by the action selection engines may be sent to an output preprocessing module that prepares them for the output modules. The reasoning engine contains an extra process called the utterance formulator; the task of this module is to convert an illocutionary act to fully annotated text, i.e. the exact text to utter together with information about speed, emphasis, tone, etc. This text can then be converted to audio output by the text-to-speech module (in the output preprocessing component).

The *output preprocessing* module makes sure that the different processes that all try to control the robot's actuators at the same time are not conflicting. The conflict manager takes care of this. The output preprocessing module also converts output from the utterance formulator to motor and audio signals. This functionality is provided by the text-to-speech module, which is also assumed to produce corresponding lip sync animations. All *output modalities* or actuators are grouped in the lower right corner of Figure 4.2. When implemented on the iCat, our system would be able to produce sounds and facial expressions, and perform non-physical external actions such as turning on and off electrical appliances.

For clarity, the arrows in Figure 4.2 appear to lead from one functional component to another, while they actually connect one or more specific modules *inside* a functional component to other modules inside another functional component. The *raw data* that is obtained by the input sensors is sent to the input preprocessing component for processing. Data from each sensor is sent to the appropriate processing module; e.g., input from the camera is sent to the vision processing module, while input from the microphone is sent to the speech recognition module.

After the processed data is synchronized by the input synchronizer, it is sent to the action selection engines, where it is placed in the *events* modules inside the engines. The processed data is also sent to the databases. Furthermore, the action selection engines can form *expectations* about future events. These expectations are sent from the action selection engines back to the input processing modules. They can then use these expectations to facilitate processing of input. For example, the speech recognizer will function better if it has a number of possible input options to match the input with. In Chapter 6, we will discuss how these expectations can facilitate natural language processing.

All processing modules in the input preprocessing component have access to *ontology* information, which they might need to process raw data properly; e.g., the vision processing module might need ontological information about a perceived object in order to classify it as a particular item. This also ensures the use of consistent data formats. *Updates* to the databases can be performed by the action selection engines. *Query results* can be requested by the action selection engines from the databases.

Output requests are sent from the interpreters inside the action selection engines to the output preprocessing component. The different types of output requests are sent to different modules inside the output preprocessing; e.g., (annotated) utterances from the dialogue engine's utterance formulator are sent to the text-to-speech module. The synchronization of all output signals is taken care of by the conflict manager. Finally, *Control* signals are gathered and synchronized by the conflict manager inside the output preprocessing component and sent to the appropriate output modality.

Other instantiations of the generic architecture

The division between layers in horizontally layered architectures (e.g., Lemon et al. [84], Taylor [120] [121], Thórisson [122]) is similar to the division between the action selection engines and the low-level behaviors in the generic architecture. The concept of layers is useful for deliberative systems: the system can give the user quick feedback, e.g., a tentative reply (e.g., "Just a second" or "Let me think about that") from the reactive layer (usually visualized as the bottom layer), while the deliberative layer (usually the top layer) processes the input further and more thoroughly, performing operations such as deep parsing and database queries that may take more time than is acceptable for a dialogue turn. This dichotomy allows for a system that is both quick and thorough. In our system, we have not at this point employed a lower level in the sense of the low-level behaviors from the generic architecture.

Lemon et al. [84] use two layers: the Content Layer and the Interaction Layer, which are modeled according to Clark's [40] distinction between the communicative track and the meta-communicative track. Lemon's dialogue architecture contains several modules that are grouped into the two layers: the Interaction Layer handles processes like turn management, immediate grounding (e.g., "ok" directly after input from the user), anaphora resolution and generation, and shallow parsing. The Content Layer takes care of utterance planning, communicative intentions, content management, and communicates with "the rest of the agent architecture" (presumably, the reasoning engine).

Thórisson [122] even sees the need for three layers: a Content Layer, a Process Control Layer and a Reactive Layer, plus an Action Scheduler Module that coordinates them. The Reactive Layer is usually able to respond within one second, and controls processes such as blinking and determining the next fixation point. The Process Control Layer regulates aspects of an interaction such as dialogue turns and gaze direction. The Content Layer contains knowledge (separated into dialogue knowledge and topic knowledge) and reasoning. Thórisson's Process Control Layer is an intermediate level between our action selection engines and our low-level behaviors, although its functionality may be compared to that of the input synchronizer and the output synchronizer.

Breazeal [21] uses competing behaviors for the robot Kismet in order to achieve an emerging overall behavior that behaves like a small child. Kismet has a number of different response types with activation levels that change according to its interaction with a user. In Breazeal's architecture for Kismet, the Behavior System and Motivation System can be mapped on the low-level behaviors of the generic architecture; however, it lacks cognitive reasoning (which was not necessary for this application), which is in our architecture provided by the action selection engines. Other components in the architecture pertain to input and output processing, which map to corresponding preprocessing modules in our architecture. The difference between the architecture for Kismet and our instantiation of the generic architecture is that Kismet has low-level behaviors and does not use any action selection engines, while our system only has action selection engines and no low-level behaviors.

The architecture of Max, the "Multimodal Assembly eXpert" developed at the University of Bielefeld [77], can also be mapped to the generic architecture. It uses a reasoning engine that provides feedback to the input module to focus attention to certain input signals, which is similar to our concept of *expectations* as we will explain in Chapter 6, when we discuss natural language processing. It also has a lower-level Reactive Behavior layer that produces direct output without having to enter the reasoning process, and a Mediator that performs the same task as our conflict manager (i.e. synchronizing output). Max's architecture is quite similar to our architecture, although Max only has one (BDI) reasoning engine, where we have provided for two or more action selection engines.

4.2 Domain

4.2.1 Formalizing the environment

In order to reason about actions, the system has beliefs about the environment, or more precisely, the availability of ingredients, appliances, tool and such. This can be represented as, for example, `available(egg)` and `available(saucepan)` (requirement RR7).

It may be useful to represent kitchen tools in an ontological model in order to reason more generally about categories of tools, since in most kitchens there is more than one type of pan, and more than one pan of a specific type, for example. In this sense, specific pans may be denoted as `frying-pan1`, `frying-pan2`, `saucepan1`, `saucepan2`, etc., whereas in pre- and postconditions of actions it is more useful to say that a saucepan is needed, instead of specifying a specific saucepan. Then, `saucepan1` and `saucepan2` will have to be classified as saucepans, so that the availability of either saucepan is acceptable as a precondition for the action in question.

4.2.2 Recipes

The recipes that we have presented in the previous chapter are not formal enough to be used by our system. We formalize these 'colloquial' recipes by adding steps that are left implicit (such as "fill the pot with water") and removing vague instructions ("use a large enough pot") to obtain more formal representations of the recipes. For

example, the two recipes from the previous chapter can be formalized as we have done in Figure 4.3.

A recipe is a relation between a goal (i.e. a recipe) and a plan. A recipe looks like

```
Recipe_name = [Task1, ..., Taskn]
```

The names of the recipe and the tasks are only identifiers (strings) that cannot be semantically analyzed. These identifiers point to the actual actions (requirement RD2). For ease of reading and understanding, we use the description of an action with underscores as the identifier of the action.

A different term for a list of tasks is a *plan* (requirement RR4). Because the tasks in the list can also represent recipes themselves (e.g., `boil_water`, which also has a recipe, is a subtask of the recipe `poach_egg`), a recipe is really a nested list, as we show in Figure 4.4. Such a nested list can also be viewed as a tree, as we show in Figure 4.5. Every task in a recipe either has a recipe or is an atomic action; this means that the leaves of the tree are atomic actions. Note that this is only true in theory; in reality, participants may have incomplete beliefs, e.g., missing certain recipes for subgoals. As we have stated before, this does not mean that such a recipe cannot be taken on, it simply means that the subgoal in question has to be openly delegated to one of the other participants in order to complete the recipe.

Plans are sequential, meaning that `Taskn` has to be completed before `Taskn+1` can be initiated (requirement RD6). This is a simplified view on recipes; usually, there is some flexibility in the order of the instructions, especially when multiple tasks are being done at the same time, when the order of steps is not important, or when timing factors are in play; e.g., when preparing a salad, the tomatoes may be cut before or after the cucumbers. However, in our framework, we take recipes to be strictly sequential, for ease of reasoning. We leave a more complex representation of recipes for future work.

To illustrate that a recipe for setting up a weblog has the same properties, we show two examples in Figure 4.6.

```
Beliefbase:

boil_water = [
fill_container_with_water,
turn_on_heat_source,
place_container_on_heat_source,
wait_until_water_starts_bubbling]

poach_egg = [
boil_water,
add_white_vinegar,
crack_egg,
stir_boiling_water,
drop_egg_into_center_of_water,
keep_time_3-4_minutes,
remove_egg]
```

Figure 4.3: *Two examples of recipes*

```
poach_egg = [
[fill_container_with_water,
turn_on_heat_source,
place_container_on_heat_source,
wait_until_water_starts_bubbling],
add_white_vinegar,
crack_egg,
stir_boiling_water,
drop_egg_into_center_of_water,
keep_time_3-4_minutes,
remove_egg]
```

Figure 4.4: *A recipe as a nested list*

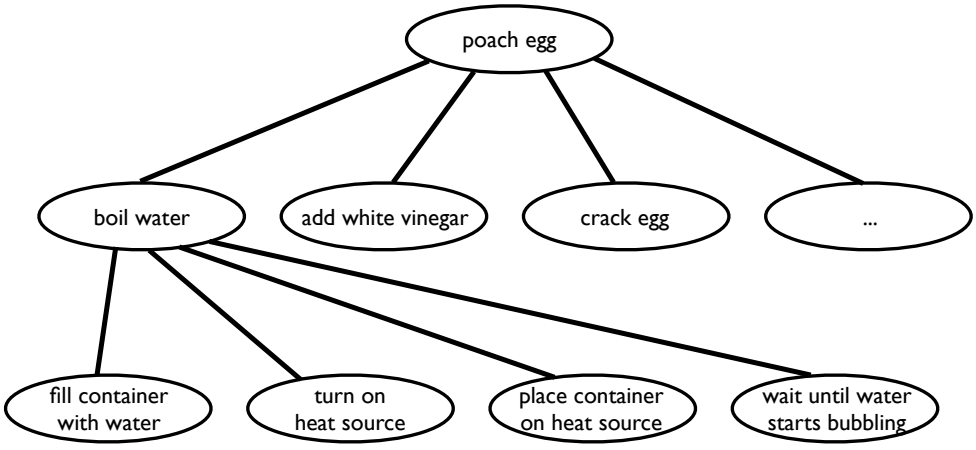

Figure 4.5: *A recipe as a tree*

4.2.3 Actions

As we have seen in the previous chapter, atomic actions have preconditions and postconditions (requirement RD1). An action can only be performed if all of its preconditions are (believed to be) true. The postconditions of an action become true as soon as the action is completed. The action is represented as follows:

```
{ Preconditions } Action_identifier { Postconditions }
```

An example of an action in a recipe is:

```
 { available(C) and container(C) and available(water) }
fill_container_with_water { filled(C, water) and not available(C) }
```

```
Beliefbase:

configure_wp-config = [
rename_wp-config,
open_wp-config,
fill_in_wp-config,
save_wp-config ]
```

```
install_wordpress = [
download_wordpress,
unzip_wordpress,
make_db_and_user,
configure_wp-config,
upload_to(wordpress, httpdocs),
run_install_script(wordpress) ]
```

Figure 4.6: *Two examples of recipes that can be used in the process of setting up a weblog*

Thus, the action of which `Fill_container_with_water` is the identifier, is only possible when an object that is a container is available and if water is available. The action identifier itself (`Fill_container_with_water`) does not specify which container should be filled with water; the variable `C` is introduced in the preconditions and will be bound to a suitable object if such an object is found. After the action is performed, the object is believed to be filled with water and no longer available.

As we have mentioned in the previous subsection to apply to tasks in general, the names of the actions are actually irrelevant, though of course a suitable name helps for the readability of the recipes. The identifier of the action does not have a semantic correlation with the action itself. The above action could just as well be represented as follows:

```
 { available(C) and container(C) and available(water) }
actionId_523 { filled(C, water) and not available(C) }
```

Besides the beliefs that are updated by performing an action, there are also connections between the action identifier and the external action and between the action identifyer and its linguistic representation (requirement RD3; see Chapter 6 where we elaborate on the linguistic representation of actions).

4.2.4 Opportunities

For reasoning about whether goals are feasible, we introduce the concept of *opportunities* (requirement RR3). There is a clear difference between opportunities and capabilities, even though both pertain to whether an action can be performed by an agent; opportunities depend on the environment and are used to reason about whether a goal should be adopted, whereas capabilities depend on the skills and knowledge of the participants and are used in the process of working through a plan for a goal that has already been adopted, to determine what instruction the instructor will give. This difference is comparable to the difference between opportunities and abilities in Van der Hoek et al. [126].

Having the *opportunity* to perform a certain task means having the physical and practical possibility to do it (depending on circumstances and available resources). For example, if some actions in a certain recipe require a toaster, but there is no toaster available, this means that none of the participants have the opportunity to perform

these actions and the recipe in question will not be pursued. Checking opportunities is especially useful in a dynamic environment [50].

Resources that are taken into account are, most notably, tools, appliances, and ingredients. Counting appliances and tools as resources avoids situations where a recipe step requires an appliance or tool that is already occupied by another recipe step. For example, this excludes combinations of a main course and a dessert that both have to be prepared in the oven. In actual situations, sometimes these can be combined, when both dishes have to be in the oven at the same temperature and the oven is sufficiently large for both containers, but for ease of reasoning, we rule out these combinations at this time, and leave them for future work instead. Having ingredients as resources ensures that there are always enough ingredients in stock before a recipe is selected.

Opportunities are used in the planning phase to rule out any recipes that are not possible at the given time. Stating that there is an opportunity to perform task T means that at this specific time it is practically possible to perform T given the current circumstances as represented in the system's beliefs about the domain. Opportunities are not explicitly registered in the beliefbase of the system. Instead, the system uses the pre- and postconditions of the atomic actions in the recipe to reason about them, together with its beliefs about the availability of ingredients, appliances and tools; e.g., if a certain pan is needed for a task but the system believes the pan is dirty, there is no opportunity to perform the task in question (unless one of the participants cleans the pan, making it available again, of course). Similarly, if a microwave is needed for a task but the system believes that the user does not own a microwave, there is no opportunity to perform the task in question.

For this, we use `checked-opportunity(P)`, which is a property of plan `P`. If the system believes that `checked-opportunity(P)` is true (i.e. if it can be derived from the beliefbase), this means that plan `P` is practically possible for the participants to perform given the circumstances at the present moment.

Grosz and Kraus' [71] concept of CBAG ('can bring about group') is similar to our concept of `checked-opportunity`, meaning that it is possible for the participants to perform the task in question together, given the current circumstances.

4.3 Interaction

4.3.1 Dialogue goals

The system has one primary domain goal (requirement RI1); we take the primary application for our system to be the preparation of recipes. Therefore, the default dialogue type is the task-oriented dialogue. Some aspects of the other dialogue types may be present; e.g., the user may ask the system for more information about recipe steps. In this case, the system searches for this information in the beliefbase and inform the user of the answer or of its lack of knowledge on the subject (requirement RI6).

4.3.2 Dialogue planning

The planning of the dialogue follows from the content of the recipes, the theory of joint activities as we presented in Chapter 2, and the dialogue models that we presented in Chapter 3 (requirement RI5).

The two most notable speech acts in the dialogue are `inform`, which is used whenever a task is completed, and `request`, which is used whenever the system delegates a task to the user (requirements RI2 and RI3). Conversely, when the user requests the system to perform a task, it should comply whenever possible (i.e. if it believes that it is capable of performing the task in question) (requirement RI7). Unless proven otherwise, the system will assume that when it has said something, it is mutually believed by the user and the system (requirement RI4).

4.3.3 Communicated content

The communicated content of the communicative acts can be one of the following:

`jointgoal(goalname)` means that the goal with the identifier `goalname` is a joint goal

`no_plan(goalname)` means that the system does not have a plan for the goal with the identifier `goalname`

`accept(goalname)` means that the system accepts the goal with the identifier `goalname`

`possible(taskname)` means that the task with the identifier `taskname` is possible at this moment

`available(ingredient)` means that the ingredient with the identifier `ingredient` is available

`available(tool)` means that the tool with the identifier `tool` is available

`recipe([requirements])` represents a list of requirements for a recipe

`performed(participant, taskname)` means that the task with the identifier `taskname` is performed by `participant`

`next-action(participant, taskname)` means that the next action in the current recipe is the task with the identifier `taskname` and that it should be performed by `participant`

`expected(event)` means that the speaker expects `event` to happen next

`start-learning(participant, taskname)` means that `participant` will now start learning the task with the identifier `taskname`

`finished(taskname)` means that the task with the identifier `taskname` is now finished

`reason(proposition, explanation)` means that the reason why `proposition` is `explanation`

`no-information(proposition)` means that the system does not have the requested information about `proposition`

`abort(goalname)` means that the goal with the identifier `goalname` is aborted

These options may all be preceded with `not` to denote their negation.

Besides the optioned mentioned above in the list, all atomic statements in the beliefbase can also be the communicated content of `inform` and `query-if`.

4.3.4 Natural language

Though our example dialogues are in natural language, we use FIPA Standard Communicative Acts [55] as the interaction language of our system. In Chapter 6, we will elaborate on the generation and parsing of natural language using Functional Discourse Grammar. In the architecture as pictured in Figure 4.2, this is represented in the Input Preprocessor (parsing) and the Utterance Generator in the Action Selection Engine (language generation).

4.4 Representation of multimodal interaction

As we have shown in the previous chapter, a situated dialogue system should be able to plan multimodal interaction. Although we have not completely implemented this functionality at this point, we have developed a way to represent multimodal interaction that we believe is intuitive and facilitates the development of such systems [129] (requirement RM3).

In our example dialogue that we have presented in Chapter 1, an error occurs when the user takes a bottle of red vinegar instead of white vinegar (as the system intended him to). In order to treat our visualization of multimodal interaction, we will first discuss a slightly modified version of this part of the example dialogue, where the user correctly follows the system's instruction:

System: "Please add some vinegar to the water."
User adds vinegar to the water.
User: "I have added some vinegar."
System: [next instruction]

To illustrate in a principled way how dialogue partners coordinate their actions, both internally and externally, we present a representation scheme for multimodal dialogues in the style of a music score in Figure 4.7 [129]. We visualize this small dialogue in a way that resembles sheet music in several ways.

Both dialogue partners have six *tracks* that represent different modalities of input, output, and reasoning, comparable to staves in musical notation. The tracks resemble the idea of different instruments that play together in an orchestra. The set of relevant modalities may be different in other dialogue systems or situations, in which case the tracks of the dialogue score can be adapted, just like different pieces of music that may require different instrument groups in an orchestra. Each dialogue partner can be seen as a section of instruments inside an orchestra, like strings, wood instruments, or a rhythm section. The instruments inside each section are closely connected, like

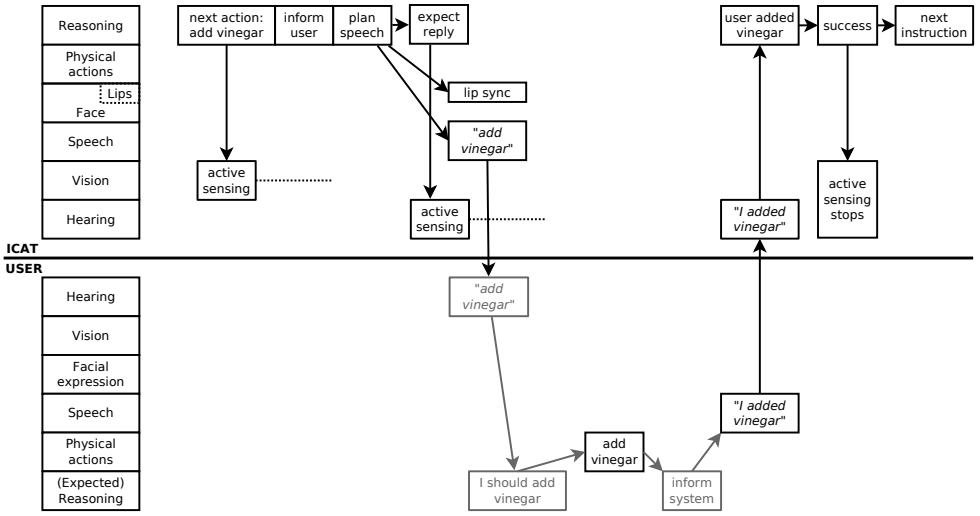

Figure 4.7: *Part of the example dialogue, in a dialogue score visualization [129]*

the different modalities of a dialogue participant are, but they do interact with the other instrument groups like dialogue partners interact with each other.

The system's part of the dialogue is represented by six tracks: two input tracks (hearing and vision), three output tracks (speech, facial expressions, and physical actions), and a reasoning track. The reasoning track is the only internal track. In order to facilitate the correct interpretation of input, the hearing and vision tracks are driven by expectations about the user's actions, which are in turn formed by the dialogue models as we have presented in the previous chapter and on which we will elaborate in Chapter 6. For example, if the system expects the user to grab the white vinegar bottle in order to add vinegar to the water, the vision module will actively watch out for the white vinegar bottle. The output tracks are quite straightforward in their behavior. The 'lips' track is a subtrack of 'face', which means that the lips follow the facial expression, unless this is overruled by speech output, in which case the lips have to perform lip sync actions. The six tracks of a human user are the same as those of the system, except we do not distinguish the subtrack 'lips' inside the 'face' track of the user.

Everything that happens in the dialogue is represented in the order of occurrence, from left to right. Events that happen (practically) simultaneously are situated directly above/below each other, showing how input on different modalities may be connected (requirement RM1). The arrows show which events are related and depend on each other, in a way that notes in a melody depend on each other: they have to be played in a certain order.

Not all of the arrows in Figure 4.7 have exactly the same meaning: arrows can represent internal or external events. For example, an arrow from a block in the reasoning track to a block in the face, speech, or physical action track means a signal to execute a plan (internal event), while an arrow from the vision or hearing track to

a block in the reasoning track is an external event (input). External arrows (that lead from one participant to another) represent perceptions, i.e. information given from one dialogue partner to the other. A typical example of this is speech by dialogue partner A that is heard by dialogue partner B. Internal arrows point from one channel to another within the private score of one of the dialogue partners. They represent internal information moving from one module to another, either because the sending module gives the information to the receiving module or because the receiving module actively retrieves the information from the sending module.

Information may also be given to several modules at the same time, for example when speech and lip movement are synchronously driven by information from the reasoning channel. Then, these actions are synchronized by the output synchronizer (requirement RM2). There are four different types of internal arrows. First, arrows from an input channel to a reasoning channel represent sensory data that is being offered to a module that can process the information. Secondly, arrows from a reasoning channel to an input channel represent execution of an active sensing plan, which enables the input channel to actively watch out for certain input. Thirdly, arrows from a reasoning channel to an output channel represent plan execution. Finally, arrows within the reasoning channels represent goal/plan revision.

Only the hypothetical 'orchestra conductor' (the designer of the system) would have the complete 'sheet music' of the dialogue; both dialogue partners only have access to their own half of the dialogue score and can only form hypotheses and expectations about each other's internal actions based on the input they get from their dialogue partner and the output they produce themselves. To illustrate this, we have made a distinction between grey blocks and arrows and black blocks and arrows in Figure 4.7: the grey blocks and arrows are events that the system cannot know or control. The system can only presume that certain processes are going on inside the user, and the only way to check this is to synchronize on the black blocks in the bottom half of the dialogue score (e.g., 'takes white vinegar'): the output by the user. If these *expected events* happen, the dialogue proceeds as planned.

The dialogue score in Figure 4.7 is a typical example of an extract from a dialogue that goes exactly as expected, since it fits in the dialogue model that we have presented in the previous chapter: the system instructs the user to perform a certain task, and then the user performs the task and informs the system of this. We will treat a version of the dialogue score where an unexpected event occurs in Chapter 7, and elaborate on the error handling tactics that can be employed by the system in order to return to the planned course of the dialogue.

With respect to converting a dialogue score to a program, the different blocks in the score can essentially be viewed as (parts of) plans. If different tracks contain blocks at overlapping time points in the dialogue score, then there are plans that have to be executed in parallel. Moreover, in most cases these parallel plans have to be synchronized (e.g., facial animation should be timed with the speech synthesis for lip synchronization and word emphasizing animations). This poses a challenge for programming the behavior of the robot. The dialogue score visualization helps the designer of a multimodal system by identifying which tracks and blocks there are in a particular scenario, and how the flow of information and control between blocks is organized.

4.5 Conclusions and future work

In this chapter, we have presented an architecture for our system and proposed some general properties of the participants, the domain and the interaction. The properties that we proposed are based on the requirements that we put forward in the previous chapter. We have proposed the use of a BDI agent as the central reasoning engine of the system. We use the notion of capabilities to be able to reason about the level of instructions that the system gives to the user and the notion of opportunities to reason about whether a proposed goal is feasible under the current circumstances.

In Table 4.8 we show that we have treated all requirements from Chapter 3, except for the requirements that pertain to error handling, which we will treat in Chapter 7. The page numbers in the table refer to the page in this chapter where the requirement in question is treated.

Req.	Page	Req.	Page	Req.	Page	Req.	Page
RR1	78	RD1	87	RI1	89	RM1	92
RR2	78	RD2	86	RI2	90	RM2	93
RR3	88	RD3	88	RI3	90	RM3	91
RR4	86	RD4	79	RI4	78		
RR5	78	RD5	79	RI5	90		
RR6	79	RD6	86	RI6	89		
RR7	85	RD7	78	RI7	90		
RR8	79						

Figure 4.8: *The page numbers in this chapter where the requirements from the previous chapter are treated.*

Implementing the multimodal aspects is something that we have left for future work; however, we believe that the dialogue score visualization will be helpful in this process. More information can be added to the dialogue score visualization, like visual recognition of facial expressions of the user by the system. As future research, a more advanced way to handle multimodal interaction to our system is also something to be explored; e.g., a multimodal output generator in the style of Max [78]. The BEAT system by Cassell et al. [37] appears to offer a good starting point for the generation of gestures based on linguistic and contextual information.

It would also be interesting to construct a parser for recipes that can be used to easily add recipes to the database. With the development of recipe markup standards such as RecipeML [56] and Google Recipes[1], it is becoming easier to do so, although these formats leave the recipe instructions as they are, in natural language. They only provide for the formalization of ingredients and required tools and such. In our system, the instructions are the most important part of the recipe that is to be formalized, in order to allow the system to reason about capabilities and opportunities, so a recipe parser should include a way to elicit a formal representation of recipe instructions, including their pre- and postconditions, from recipe instructions that are written in

[1] `http://www.google.com/support/webmasters/bin/answer.py?hl=en&answer=173379`, retrieved June 14th, 2010

natural language.

When reasoning about opportunities, it would be useful to also count agents as resources. This enables the system to work out parallel plans; e.g., recipe steps like 'wait for n minutes' can be performed in parallel with other tasks. For example:

```
 { available(C) and container(C) and available(water) and available(user) }
Fill_container_with_water { filled(C, water) and not available(C) }
```

This means that the system can more easily reason about the execution of parallel actions; for example, leaving a dish in the oven for a certain timespan does not require the user to be available, which means that he can work on another part of the recipe in the meantime. In contrast, two tasks that need to be performed by the same actor can generally not be performed simultaneously. Also, counting agents as resources makes the system more suited for multiple users.

5 Implementation of a Recipe Assistant

> Every program has (at least) two purposes: the one for which it was written, and another for which it wasn't.
>
> *Alan J. Perlis*

In this chapter, we specify the implementation of an agent that is capable of having task-oriented dialogues with a user, based on the requirements and architecture that we have presented in the previous chapters. We present a framework for a dialogue system that engages in mixed-initiative interaction by not only reacting to queries and instructions from the user, but also taking initiative in a proactive way in order to reach a shared goal. In order to do this, the framework is based on the BDI model [41], augmented with notions of joint goals, abilities, capabilities, and different speech acts, which we present in 2APL-like pseudocode.

In Table 5.1, we show part of the example dialogue in the form of FIPA ACL communicative acts and (examples of) their corresponding natural language utterances. In this chapter, we will present a framework that can produce such dialogues.

We will first present some details of 2APL and the interaction language that we use in Section 5.1, treating the beliefbase and the goalbase (5.1.1), the different types of basic actions (5.1.2), the types of reasoning rules (5.1.3), the deliberation cycle that uses those rules (5.1.4), and the subset of FIPA-ACL communicative acts that we use in our framework (5.1.5). Then, in Section 5.2, we present the program, which consists of four distinct part: the selection of joint goals (5.2.1), the planning phase, which regulates the adoption of plans (5.2.2), the instruction phase, which pertains to the task itself (5.2.3), and a set of rules for answering questions and following requests from the user, which may occur at any time in the interaction (5.2.4). In Section 5.3, we will present a simpler version of the example dialogue, without any errors, that can be constructed with the rules that we present in this chapter and a smaller set of rules to represent the user (5.3.1). In 5.3.2, we show how those rules can indeed construct the simper example dialogue. In Section 5.4, we discuss some aspects of the concrete implementation (5.4.1) of the framework and the problems that were encountered in this process (5.4.2). In Section 5.5, we will present conclusions and

FIPA ACL Communicative act	Natural language utterance
`send(S, request, jointgoal(french_toast))`	User: I would like to prepare French toast
`send(U, inform, not possible(french_toast))`	System: That's not possible
`send(U, inform, not available(bread))`	System: We don't have any bread
`send(S, request, recipe(breakfast))`	User: Do you have another breakfast recipe for me?
`send(U, propose, poach_egg)`	System: How about poached egg?
`send(S, inform, accept(poach_egg))`	User: Alright, I would like to make a poached egg
`send(U, inform, possible(poach_egg))`	System: Okay
`send(U, inform, start-learning(U, poach_egg))`	System: I'll teach you how to poach an egg
`send(U, request, next-action(U, boil_water))`	System: Please boil some water
`send(S, request, next-action(S, turn_on_heat_source)`	User: Can you turn on the stove for me?

Table 5.1: *Part of the example dialogue in FIPA ACL Communicative acts*

future work.

In the (pseudo)code that we present in this chapter and in Chapter 7, `B(p)` means the belief that p (or: p is present in the Beliefbase of the agent), and `I(p)` means the intention that p (or: p is present in the current Planbase of the agent).

5.1 The 2APL programming language

We present the algorithms in pseudocode in the style of the agent-oriented programming language 2APL [48]. 2APL allows us to specify the beliefbase, goalbase and planbase according to the principles of the BDI model. In this section, we briefly introduce the concepts from 2APL that we work with in the remainder of this thesis.

5.1.1 The beliefbase, the goalbase, and the planbase

The *beliefbase* is a Prolog file containing statements the agent holds to be true. These statements can be either simple (atomic) beliefs, e.g., `capable(U, Boil_water).`, or complex rules, which define concepts that make it easier to write and read plans. One example of such a complex rule that we use in our program, on which we will elaborate later in this chapter, is the concept *checked-done*, which is used to 'summarize' three different conditions under which the system believes that a recipe step is done. To avoid having to write down all three of these options each time the concept is used, we use a Prolog rule for abbreviation. These Prolog rules are situated in the beliefbase

of the agent and take the form `head :- body.`, meaning that whenever `body` is true (i.e. derivable from the beliefbase), `head` is true.

A goal denotes a situation that the agent wants to realize. Goals reside in the agent's *goalbase*. In order to achieve a goal, an agent can adopt a *plan*. In this thesis we focus on joint goals, which are of the form `jointgoal(recipename)` (for example, `jointgoal(poach_egg)`). Joint goals are not a standard concept in 2APL, but we introduce them as a subset of regular goals that will be treated in a unified way, as we will present in Section 5.3. In contrast to regular goals, which specify a goal state, we specify our joint goals with recipe names. This is only a naming convention that we use for ease of reading (the alternative, which specifies a goal state, would look something like `jointgoal(done(poach_egg))`) or `jointgoal(prepared(poach_egg))`).

In the planbase, the intentions of the agent are represented. When a plan is adopted, it is added to the planbase.

5.1.2 Types of basic actions

In 2APL, there are six different types of actions an agent can perform: belief updates, goal dynamics actions, belief and goal test actions, communication actions, external actions, and abstract actions. In this section, we briefly introduce these 2APL action types.

When a *belief update action* is executed, the beliefbase of the agent is updated. Beliefs can be added, altered, or removed. Belief updates are internal actions and therefore not visible to other agents (except of course if the program specifically states that (some) updates should be communicated to other agents). A belief update action has preconditions, which specify conditions under which the belief update can be executed, and postconditions, which specify the changes in beliefs that are made when the belief update action is executed.

Goal dynamics actions are also internal actions. They regulate the adopting and dropping of goals. For adopting goals, there are two options: adding a goal at the beginning of the goal base (`adopta`), meaning that it will be addressed first, or at the end (`adoptz`). Abandoning goals is done with the action `dropgoal(G)`. It is also possible to drop all goals that are a subgoal of G (`dropsubgoals(G)`), or drop all goals of which G is a subgoal (`dropsupergoals(G)`).

The system can also perform *belief and goal test actions*, which are basically (Prolog) queries. They are expressions of the predicate argument form that check whether a belief or goal is derivable from the agent's beliefbase or goalbase, respectively. These actions can be used to instantiate a variable with a value, or to block the execution of (the rest of) a plan (if the test fails).

A *communication action* is an action that passes a message to another agent. In our system, the other agent can be either another computational agent or a human agent (the user). A communication action is of the form `send(receiver, speech act, content)`. The speech acts are the FIPA Standard Communicative Acts [55], on which we will elaborate in Section 5.1.5, and include `inform` and `request`, which fulfills our communication requirements as stated in the previous chapters. If the receiver is a human agent, the communication action is sent to the language generation module, which turns it into *discourse acts* in natural language. We will elaborate

further on discourse acts and the language generation module in Chapter 6.

External actions are actions that have an effect on the physical domain, such as turning on the microwave. In our application domains, recipes typically consist of external actions, like cutting vegetables, turning on electric appliances and (in the weblog setting) uploading files. It should be noted that not all of these actions end up being done by the system, but most of them are instead redirected to the user by way of an instruction, which is a communication action that has the external action in question as (part of) its *content*.

Finally, *abstract actions* allow the encapsulation of a plan by a single action, making plans easier to write and to read.

5.1.3 Reasoning rules

Most importantly, the system contains Planning Goal Rules (PG-rules) and Procedural Rules (PC-rules). PG-rules translate a goal into a plan and take the following form: `Goal <- Guard | Plan`. This rule will get activated when `Goal` is the first goal in the goalbase and `Guard` (a Prolog query) is derivable from the beliefbase. Then, `Plan` (a list of actions) will be put in the planbase for subsequent execution. PC-rules look the same as PG-rules, but their head is an atom instead of a goal; they are activated when an external event is detected, when an abstract action is executed, or when the system receives a message from another agent. Messages from other agents take the form `message(sender, speech act, content)`. In Section 5.1.5, we will elaborate on such messages.

To distinguish between these different kinds of rules and information in the system, we specify for each of the pseudocode rules in this chapter in which part of the system it resides (PC-rules, PG-rules, Beliefbase or Goalbase).

5.1.4 The deliberation cycle

In 2APL, rules are applied in the following order: first the system tries to apply each PG-rule once, then it executes the first actions of all selected plans. Then, it processes external events, then internal events and finally any incoming messages.

5.1.5 FIPA-ACL communicative acts

The FIPA-ACL Communicative Acts [55] that we use are:

Inform The sender informs the receiver that a given proposition is true. Content: A proposition.

Propose The action of submitting a proposal to perform a certain action, given certain preconditions. Content: A tuple containing an action description, representing the action that the sender is proposing to perform, and a proposition representing the preconditions on the performance of the action.

Query If The action of asking another agent whether or not a given proposition is true. Content: The action of asking another agent whether or not a given proposition is true.

Query Ref The action of asking another agent for the object referred to by a referential expression. Content: A descriptor (a referential expression).

Request The sender requests the receiver to perform some action. (One important class of uses of the request act is to request the receiver to perform another communicative act.) Content: An action expression.

Of course, all natural language utterances from the user need to be parsed into 2APL messages. Also, conversely, all outgoing communicative acts need to be processed into natural language utterances. For both of these processes, we use Functional Discourse Grammar [74] [128]. We will elaborate on this in Chapter 6. We have described the possibilities for the communicative content of these speech acts in Section 4.3.3.

5.2 The program

We repeat the dialogue model from Chapter 3 in Figure 5.1.

In our formal framework, the dialogue consists of three main phases. First of all, the user and the system select a joint goal together. This may be an information-seeking dialogue, where the user specifies a number of desirable properties and the system proposes an appropriate joint goal, which the user can then accept or decline. Alternatively, the user can simply state a joint goal. The second phase of the dialogue is the planning phase. In this phase, the system checks whether there is a viable plan for the selected joint goal, given the current domain circumstances (e.g., availability of tools and ingredients). For this, we use the concept of opportunities that we introduced in Section 4.2.4, and on which we will elaborate later in this section. If there is no such plan, the system informs the user of this, and the user can either accept the system's assessment (and the joint goal is abandoned) or overrule it, for example in cases where the user knows a different plan for the (sub)goal in question which does not require tools or ingredients that are not available. The third and final phase of the dialogue is the instruction phase, which includes the actual preparation of the recipe and some tutoring aspects.

5.2.1 Joint goal selection

The joint goal can be selected in two different ways: either the user simply requests a joint goal, or he requests a recipe that satisfies certain criteria, in which case a recipe selection subdialogue is iniated.

In Figure 5.2, we show how a joint goal is adopted when the user requests it. If the system receives a message from the user, requesting of the system that they adopt joint goal `G` together, the system adopts this as a joint goal at the beginning of its beliefbase. In situations where the system is used by more than one user, the joint goal should also specify the set of participants of the joint goal. However, as we have stated in the previous chapters, at this time we assume that there is only a single user and that the participants of the joint goal are always the user and the system.

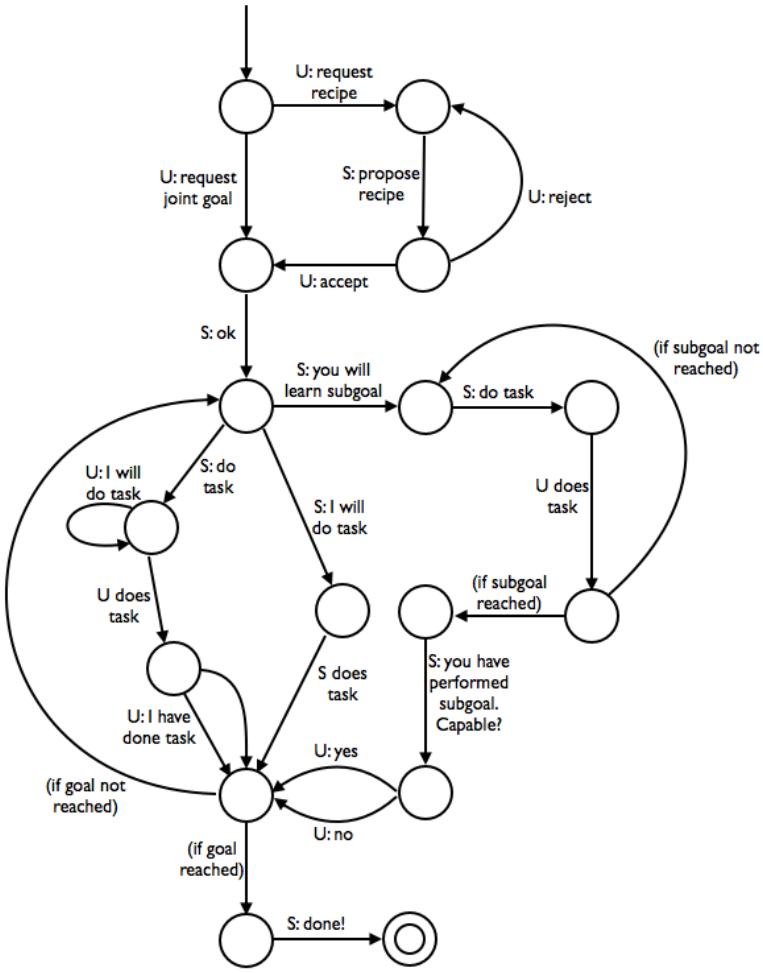

Figure 5.1: *The recipe searching and preparation dialogue*

```
(1)  PC-rules:
     message(U, request, jointgoal(G)) <- true |
     adopta(jointgoal(G))
```

Figure 5.2: *Adopting a goal*

Therefore, we leave out the specification of the set of participants from the joint goal and only specify the goal state itself.

Alternatively, the user can initiate the recipe selection phase, as illustrated in

```
(2) PC-rules:
    message(U, request, recipe(C)) <- true |
    recipesearch(C)
(3) PG-rules:
    recipesearch(C) <- suitablerecipe(C, R) |
    send(U, propose, R);
    if message(U, inform, accept(R))
    then adopta(jointgoal(R))
    else recipesearch(C)
(4) Beliefbase:
    suitablerecipe(C, R) :-
    fitcriteria(C, Rlist),
    pickrandom(Rlist, R).
```

Figure 5.3: *The recipe selection phase*

Figure 5.3, by requesting a recipe that fits a list of criteria `C`. The criteria for the dish can be of various types; e.g., ingredients that should or should not be in the recipe, cuisine (country of origin, etc.), cooking method, dietary restrictions, etc. Then, the system uses this list of criteria to search the database for a suitable recipe `R`. If there is more than one recipe that fits the specified criteria, the system randomly chooses one of these dishes, and proposes this to the user. The user can then either accept recipe `R` (which means that it will be adopted as a joint goal) or reject the dish, in which case the system performs a new recipe search.

5.2.2 Rules for the planning phase

As illustrated in Figure 5.4, joint goals are represented as `jointgoal(G)`, where `G` is the goal and the participants are the system and the user. The first condition that the system checks after adopting a joint goal is whether there is a recipe for this goal in the beliefbase (i.e. whether `G = P` is derivable from the beliefbase, for the specified goal `G` and any plan `P`). If there is no recipe for `G` in the beliefbase, the user is informed of this and the goal is dropped (rule 5a).

If there is a recipe for the goal in question, the system checks whether for all actions in the recipe, at least one of the participants has the opportunity to perform it, given the current circumstances in the domain (e.g., available tools, ingredients, appliances). This is done by checking whether `checked-opportunity(P)` is true for plan `P` that has just been instantiated as a plan for the current joint goal `G`. We will explain `checked-opportunity` in more detail below.

Depending on whether `checked-opportunity` is true, two courses of action may be taken. First of all, in case it is true (rule 5b), the user is informed that the system accepts the joint goal, and the instruction phase is initiated by calling rule `prepare(P, G)`. We will explain this in the next subsection. However, if `checked-opportunity`

```
(5)  PG-rules:
     a. jointgoal(G) <- not B(G = P) |
        send(U, inform, no_plan(G));
        dropgoal(jointgoal(G))
     b. jointgoal(G) <- B(G = P) and
        B(checked-opportunity(P)) |
        send(U, inform, possible(G));
        prepare(G, P)
     c. jointgoal(G) <- B(G = P) and reason(not possible(G, X))
        not B(checked-opportunity(P)) |
        send(U, inform, not possible(G));
        send(U, inform, reason(not possible(G, X));
        dropgoal(jointgoal(G))
```

Figure 5.4: *Criteria for pursuing joint goals*

is not true (rule 5c), the preparation of the recipe cannot be initiated, because (at least) one of the actions in the plan cannot be performed due to the fact that its preconditions will not be satisfied when it has to be performed, given the current state of the domain. In our example, this happens when the user proposes French toast: one of the actions in the recipe has `available(bread)` as a precondition, and since the system does not believe this to be true, the plan cannot be initiated and the goal must be dropped. In this case, the system informs the user of this fact and also gives the reason why the goal is not possible (the failed precondition). It is then up to the user whether he wants to choose another joint goal or to take actions to satisfy the failed precondition (by buying bread, for example).

In Figure 5.5, we show how `checked-opportunity` is computed by a number of Prolog rules in the beliefbase of the system. In order to compute `checked-opportunity` of a plan `P`, the system will reason about the pre- and postconditions of the atomic actions in `P` (including `P`'s subtasks). This requires 'unfolding' the recipe tree to a list of atomic actions (rule 6): all subgoals in the task tree of `P` are fully expanded, resulting in a nested list of atomic actions (`makenestedlist`), and this nested list is converted to a flat list (`flatten`).

Subsequently, for each action in the list, the system checks if its preconditions (`Prec`) are derivable from the relevant part of the beliefbase (`DomainBeliefs`, the system's beliefs about the domain, e.g., availability of ingredients) and the postconditions of all previous actions in the recipe (`Postcondlist` in Figure 5.6, which we will explain below). In other words, if the preconditions of task T_n in the list are derivable from (the list of) the system's beliefs about the domain plus all postconditions of the preceding tasks T_1 ... T_{n-1}, `check-opportunity` is true for task T_n. If each task T_n in the expanded, flattened list for task `P` satisfies this condition, `checked-opportunity(P)` is true (rule 6).

The list of postconditions of a list of actions is computed by `postcondlist`. The

```
(6) Beliefbase:
    checked-opportunity(P) :-
    makenestedlist(P, Pcomplete),
    flatten(Pcomplete, Pflat),
    check-opportunity([], [], Pflat).

(7) Beliefbase:
    a. check-opportunity([], [], [X|Tail]) :-
       check-opportunity([], X, Tail).
    b. check-opportunity(Prevlist, X, [Y|Tail]) :-
       { Prec } X { Postc } ,
       postcondlist(Prevlist, Postcondlist),
       append(Postcondlist, DomainBeliefs, BeliefList),
       derivable(Prec, BeliefList),
       append(Prevlist, [X], NewPrevlist),
       check-opportunity(NewPrevlist, Y, Tail).
    c. check-opportunity(Prevlist, X, []) :-
       { Prec } X { Postc } ,
       postcondlist(Prevlist, Postcondlist),
       append(Postcondlist, DomainBeliefs, BeliefList),
       derivable(Prec, BeliefList).
```

Figure 5.5: *Checking whether the participants are able to perform the recipe under the current circumstances*

```
(8) Beliefbase:
    a. postcondlist([], _).
    b. postcondlist([X|Tail], Y) :-
       { Prec } X { Postc } ,
       append(Y, Postc, Y1),
       removeifnegated(Y1, Y2),
       postcondlist(Tail, Y2).
```

Figure 5.6: *Checking whether the participants are able to perform the recipe under the current circumstances*

first argument of `postcondlist` is the list of actions for which the postconditions have yet to be derived, and the second argument is the list of postconditions that have already been derived by this recursive function. When the first argument is an empty list, all postconditions have been gathered (rule 8a). For now, we assume that the pre- and postconditions are conjunctions of literals. Still, postconditions

may contain negations, in cases where the postconditions of an action specify the removal of certain information from the beliefbase. In order to account for this, we have introduced `removeifnegated`, which for every negated literal in a list, removes all instances of this literal (positive and negative) from the list. What remains is list `Y2`, which contains only positive literals that have not been negated by other postconditions.

5.2.3 Rules for the instruction phase

In Figure 5.7, we show how the instruction phase works. Briefly, the main process of the instruction phase consists of giving an instruction to the user, waiting until the user has completed it and then repeating this process with the next instruction, until

```
(9) PC-rules:
    a. prepare(G, []) <- true |
       send(U, inform, finished(G)); dropgoal(jointgoal(G))
    b. prepare(G, Tlist) <- not B(capable(U, G)), B(capablelist(U,
       Tlist)) |
       send(U, inform, start-learning(U, G));
       teachprepare(G, Tlist);
       send(U, query-if, capable(U, G));
       if message(U, inform, capable(U, G)) then
       AddBelief(capable(U, G)) else skip;
       send(U, inform, finished(G)); dropgoal(G)
    c. prepare(G, [T|Tail]) <- B(capable(U, T)) |
       send(U, request, next-action(U, T));
       checked-done(T)?; prepare(G, Tail)
    d. prepare(G, [T|Tail]) <- B(capable(S, T)) |
       send(U, inform, next-action(S, T));
       @domain(T, performed(S, T), Time-out);
       checked-done(T)?; prepare(G, Tail)
    e. prepare(G, [T|Tail]) <- not B(capable(U, T)), B(T = Tlist),
       B(capablelist(U, Tlist)) |
       send(U, inform, start-learning(U, T));
       teachprepare(T, Tlist);
       send(U, query-if, capable(U, T));
       if message(U, inform, capable(U, T)) then
       AddBelief(capable(U, T)) else skip;
       prepare(G, Tail)
    f. prepare(G, [T|Tail]) <- not B(capable(U, T)), B(T = Tlist) |
       prepare(G, Tlist); prepare(G, Tail)
```

Figure 5.7: *Rules for preparing the recipe*

```
(10) PC-rules:
     a. teachprepare(T, []) <- true |
        send(U, inform, performed(T))
     b. teachprepare(T, [Tfirst|Tail]) <- true |
        send(U, request, next-action(U, Tfirst));
        checked-done(Tfirst)?; teachprepare(T, Tail)
```

Figure 5.8: *Rules for teaching the user to perform a task*

the recipe is finished. If the user is not capable of performing a task but the system is, the system will perform the task and inform the user about this. As we have stated in previous chapters, the system should avoid giving too easy or too difficult instructions, in order to make the interaction pleasant and useful for the user. This means that the system always gives instructions that it believes the user is capable of understanding and performing.

For clarity, we will first explain some of the most basic concepts that are used in this complex rule: `capablelist` and `checked-done`. In Figure 5.9 we show how `capablelist` works. It computes whether a certain participant `X` is capable of performing all tasks in list `Y`. Note that while `capablelist` is recursive, `capable` itself is not: if the system believes that the user is capable of performing a certain task `T`, he is not automatically capable of performing all of `T`'s subtasks. After all, the user may have a different recipe for `T` that involves different subtasks. For example, imagine a situation where the system's only recipe for making spaghetti bolognese involves making fresh pasta, and the system believes that the user is capable of making spaghetti bolognese. This does not mean that the user is capable of making fresh pasta, if the user has a different recipe for the subtask 'preparing pasta', which involves cooking store-bought pasta.

```
(11) Beliefbase:
     a. capablelist(_, []).
     b. capablelist(X, [Y|Tail]) :- capable(X, Y), capablelist(X,
        Tail).
```

Figure 5.9: *Checking whether a participant is capable of performing all actions in a list*

As illustrated in Figure 5.10, the system knows that a task is done when either of the following three conditions holds: when the user has informed the system that he has performed task `T` (rule 12a), when the system itself has successfully performed `T` (rule 12b), or if the system has observed that task `T` is done (rule 12c).

Now, we will explain the `prepare` procedure in more detail. Starting with the highest node in the recipe tree, if the system believes that either participant is capable of performing this task, it should be performed by this participant; in this case, the

```
(12) Beliefbase:
     a. checked-done(T) :- message(U, inform, performed(U, T)).
     b. checked-done(T) :- performed(S, T).
     c. checked-done(T) :- observe(done(T)).
```

Figure 5.10: *Checking whether a step is done*

system either informs the user that he should perform the task (if the user is capable of doing it) or informs the user that the system itself will perform the task (if the system is capable of doing it), and in the latter case of course it also performs the task. If the system believes that neither participant is capable of performing the task, the same process is repeated for each subtask of the task in question. After the user is instructed to perform a task, the system will check whether the task is done before continuing with the next instruction.

In more detail, the procedure `prepare(G, P)` has two arguments: `G` is the joint goal that the system and user are working towards, and `P` is the list of tasks that still need to be done. The code in Figure 5.7 has five different cases: first of all, if the list of tasks is already empty (rule 9a), this means that there are no more tasks to perform and the recipe is done. The user is informed of this and goal `jointgoal(G)` is dropped.

Secondly (rule 9b), there is the possibility where the user is not capable of preparing the complete recipe `G`, but is capable of preparing all subtasks of `G`: `capablelist(U, Tlist)`. In this case, the system will attempt to teach the user how to perform `G`. We have chosen to use the precondition `capablelist` in order to keep the learning process perspicuous (to avoid nested learning processes, which may cause unclear situations). The system will first tell the user that it will try to teach him how to perform `G`. Then, a `teachprepare` procedure is called for `G`, which is similar to the regular `prepare` procedure and on which we will elaborate below. When `G` is finished, the system will inform the user of this and ask him whether he thinks he will be able to perform `G` by himself the next time. Note that for computational agents there is no reason why they would not be capable of `G` at this point in the interaction, but in order to take into account the unpredictability of human users, we have built in this check. Only when the user gives a positive reply, the system adds `capable(U, G)` to its beliefbase.

The third case (rule 9c) is the case where the user is capable of performing the first task from the list of tasks, task `T`, to be done. If this is true, the system requests the user to perform task `T` and then waits until the task is done (`checked-done`) to continue. Then, the procedure `prepare` is called anew, but of course task `T` is removed from the head of the list of tasks that still need to be done.

The fourth case (rule 9d) is activated only when the user is not capable of performing task `T`, but the system is. (Note that the order of the rules is important here: if the user would have been capable of performing `T`, rule 9c would have been activated first, and rule 9d would not have been reached.) The system informs the

```
(13)  BeliefUpdates:
      a. { p } RemoveBelief(p) { not p }
      b. { not p } AddBelief(p) { p }
```

Figure 5.11: *Adding and removing beliefs*

user that it will perform task `T` itself, and subsequently of course also performs the task.

Fifthly (rule 9e), we have a procedure that is similar to the way the learning process is initiated for the goal itself, but here it is for one of the subtasks of the goal.

Finally (rule 9f), if none of the participants are capable of performing `T` and when it does not qualify for teaching (i.e. if the user is not capable of performing all of its subtasks), `T` will be expanded into the list of its subtasks, `Tlist`, and stuck to the front of the queue of next tasks. This means that the `prepare` procedure will be repeated for each of the subtasks of `T`, followed by the rest of the task list that is still in the queue after `T` is done.

The procedure `teachprepare`, which we show in Figure 5.8, is a simplified version of `prepare`. Because the system has already checked if the user is capable of performing all of the (direct) subtasks of the learning task `T`, the only thing that `teachprepare` does is instruct the user to perform the subtasks of `T` in the order in which they are listed (rule 10b), and when all subtasks of `T` are done, inform the user of this (rule 10a). Then, the remainder of the teaching procedure in `prepare` is continued as we have specified above.

Finally, in Figure 5.11 we show how beliefs are added to and removed from the beliefbase.

5.2.4 Rules for answering questions and following requests

We make one addition to the dialogue models that we have presented in Chapter 3: because the system should always be cooperative w.r.t. the user, we always allow the user to ask questions to the system, and request actions from the system, provided that the actions are relevant to the joint goal. (If they are not, the error handling module will address this, as we explain in Chapter 7.)

During the instruction phase, the user can ask the system questions about the recipe, the ingredients, and the preparations. When a `message` from the user enters the eventbase, the system puts the current plan (the instructions in the recipe) on hold and first answers the question. Another type of message from the user is a request for action (e.g., keeping the time, printing out a document). In this case, the system will not only have to perform the action itself, but it also has to inform the user of this in order to keep the mutual beliefs of system and user correct and complete.

In the example dialogue in Chapter 1, the user posed the following question:

User: "How long should I stir the water?"
`message(U, query-ref, info(stir_water, duration)`
System: "Stir the water briefly."
`send(U, inform, duration(poach_egg, stir_water, briefly))`

The rules for answering questions from the user are shown in Figure 5.12. These rules are quite straightforward: if the user asks about the truth of a certain proposition `p` (by uttering `message(U, query-if, p)`), there are two main options: if `p` is in the beliefbase of the system, it will answer positively (rule 14a); if `not p` is in the beliefbase of the system, it will answer negatively (rule 14b). Note that we do not adhere to a closed-world assumption in our system; if an agent does not believe that `p` is true, this does not automatically imply that `not p` is true. If the system believes neither `p` nor `not p`, this means that it does not have any information about `p` and can therefore not answer the question. In this case, an error handling subdialogue is initiated (which we will explain in Chapter 7).

The only exception to this is belief about capabilities; the lack of belief that participant A is capable of performing p is equal to the belief that A is not capable of performing `p`. For this, we have added an extra rule that is activated if the propositional content of the message is `capable(A, p)` (rule 14c). Then, if the system does not believe `capable(A, p)`, this means that A is believed not to be capable of p, and therefore the system should also reply negatively. For all other types of propositional content, if neither `p` nor `not p` are in the beliefbase and the user asks whether `p` is true, this is an error and should therefore be repaired, as we will explain in Chapter 7.

The user can ask the system extra information about actions by asking `info(A, Property, G)` for an action `A` and a `Property` that can be various types of information, but is always related to the current joint goal `G` (rule 14d). In the example dialogue, the user asks for the duration of the action `stir_water` in this recipe. The system has `duration(poach_egg, stir_water, briefly)` in its beliefbase and will therefore answer `send(U, inform, duration(poach_egg, stir_water, briefly)` ("Stir the water briefly").

If the user requests the system to perform a task, the system will perform this task if it is capable of doing so (`capable(S, T)`) and if it believes that the task is possible at this time (`checked-opportunity(T)` (rule 15). The opportunity check is necessary for two reasons: first of all, the environment is dynamic and therefore it is possible that some of the circumstances have changed; secondly, since the system openly delegates tasks to the user, it is possible that this particular task was not in the system's original plan for the joint goal. If the task is not possible at this time and/or if the system is not capable of performing the task, an error subdialogue is initiated, as we will explain in Chapter 7.

5.3 Generation of a simpler example dialogue

In this section, we show that part of the example dialogue can be generated with the rules presented in this chapter. Since we have not yet treated the rules for error

```
(14) PC-rules:
     a. message(U, query-if, p) <- B(p) |
        send(S, inform, p)
     b. message(U, query-if, p) <- B(not p) |
        send(S, inform, not p)
     c. message(U, query-if, capable(X, p)) <- not B(capable(X, p)) |
        send(S, inform, not capable(X, p))
     d. message(U, query-ref, info(A, Property, G)) <-
        Property(G, A, X) |
        send(U, inform, Property(G, A, X))

(15) PC-rules:
     message(U, request, next-action(S, T)) <- capable(S, T) and
     checked-opportunity(T) |
     send(U, inform, next-action(S, T));
     @domain(T, performed(S, T), Time-out);
     checked-done(T)?;
     send(U, inform, performed(S, T))
```

Figure 5.12: *Answering questions and following requests from the user*

handling yet, we will now show how an alternative version of the example dialogue, in which no errors occur, is generated by the rules in this chapter.

In order to create a complete dialogue from the rules that we have presented in this chapter, we would also have to model the user in the same way, as an agent. Below, we present a simple version of a user agent, which complies with the system's instructions when it believes that it is capable of performing them, and can also learn new recipes.

5.3.1 The user agent

When the user has a goal `G`, but no recipe (plan `P`) to achieve this goal, it will request the system to adopt `G` as a joint goal, as illustrated in Figure 5.13 (rule 16). If the system confirms the joint goal (i.e. if the recipe successfully passes the system's opportunity check in rule 5), `G` is added as a joint goal at the beginning of the goalbase. Note that the user will still keep `G` as a private goal until the joint goal is completed.

In Figure 5.14, we show how the user handles the initiation of the learning process. When the system informs the user that he is to start learning task `T`, the belief `tentativeRecipe` is added with an empty list as the tentative recipe for `T` (rule 17). During the course of the instructions for `T`'s subtasks, this empty list will be filled with tasks, as we will explain below.

In Figure 5.15, we show how the user handles receiving instructions. If the user believes that he himself is capable of performing the instructed task `T` (rule 18a), he will perform task `T`, make sure that it is done, and then inform the system that he

```
(16)  PG-rules:
      goal(G) <- not B(G = P) |
      send(S, request, jointgoal(G));
      if message(U, inform, possible(G)) then adopta(jointgoal(G));
      else skip
```

Figure 5.13: *User code: rules for introducing a joint goal*

has indeed performed the task.

If the user has a tentative recipe in his beliefbase (rule 18b), he will append the new instruction `T` to the existing tentative recipe `P`, creating a new tentative recipe `NewP`. He will then substitute `P` with `NewP` in the beliefbase. Finally, since the user is not only learning the recipe but also preparing it at the same time, he will of course also perform `T` in the same way as usual.

When the system informs the user that the current joint goal `G` is finished (Figure 5.16), the user will simply drop goal `G` as a regular goal and a joint goal if he was not in the process of learning a recipe for `G` (rule 19a). If the user was learning a recipe for `G` (rule 19b) and the system informs the user that he has performed the task in question, the user will substitute the tentative recipe with a finished recipe, which is simply of the form `G = P`, in which `P` is the list of instructions that has been built up step by step during the instruction phase. The user also adds the belief that he is capable of performing the goal in question to his beliefbase (`capable(U, G)`).

The system may ask the user questions about his beliefs, for example when it has just taught the user a recipe and is about to add the new capability to its beliefbase, as we have seen before in this chapter. In Figure 5.17, we show how the user handles this in a very straightforward way: simply, if the system asks whether `X` is true, the user answers positively if he believes that `X` is true (rule 20a) and negatively if he believes that `X` is not true (rule 20b). If the question is about a capability that is not present in the beliefbase, the answer is also negative (rule 20c). These rules are comparable to the system's rules for answering questions that we have presented earlier in this chapter.

5.3.2 An alternative version of the example dialogue

In the three Tables 5.2, 5.3, and 5.4 at the end of this chapter, we present a version of the example dialogue that we can construct with the rules that we have presented

```
(17)  PC-rules:
      message(S, inform, start-learning(U, T)) <- true |
      AddBelief(tentativeRecipe(T = [])
```

Figure 5.14: *User code: starting to learn a recipe*

```
(18) PC-rules:
     a. message(S, request, next-action(U, T)) <−
        capable(U, T) |
        @domain(T, performed(U, T), Time-out);
        checked-done(T)?;
        send(S, inform, performed(U, T))
     b. message(S, request, next-action(U, T)) <−
        tentativeRecipe(G = P) |
        append(P, T, NewP);
        RemoveBelief(tentativeRecipe(G = P));
        AddBelief(tentativeRecipe(G = NewP));
        @domain(T, performed(U, T), Time-out);
        checked-done(T)?;
        send(S, inform, performed(U, T))
```

Figure 5.15: *User code: following instructions from the system*


```
(19) PC-rules:
     a. message(S, inform, finished(G)) <−
        dropgoal(G); dropgoal(jointgoal(G))
     b. message(S, inform, performed(G)) <−
        tentativeRecipe(G = P) |
        RemoveBelief(tentativeRecipe(G = P));
        AddBelief(G = P);
        AddBelief(capable(U, G))
```

Figure 5.16: *User code: finishing a recipe*

in this chapter. As we have mentioned above, this example dialogue contains none of the errors that the dialogue in Chapter 1 does, since we have only presented our basic task-oriented dialogue system in this chapter; in Chapter 7, we will show that we can construct the complete dialogue from Chapter 1 if we add an error handling module to the basic system.

The Tables 5.2, 5.3, and 5.4 should be read in order and from top to bottom, where each line represents an internal or external event. When multiple events result from the application of one rule, the rule number is augmented with the comment '(contd.)' for each subsequent event after the first one.

The left column of the table represents the actor of the current rule and the number of the rule that is applied. The middle column represents the specific instantiation of the rule, communicative act or belief update that is being executed. The third column represents any observable events that result from the application of the rule

```
(20) PC-rules:
     a. message(S, query-if, p) <- B(p) |
        send(S, inform, p)
     b. message(S, query-if, p) <- B(not p) |
        send(S, inform, not p)
     c. message(S, query-if, capable(X, p)) <- not B(capable(X, p)) |
        send(S, inform, not capable(X, p))
```

Figure 5.17: *User code: answering yes/no-questions*

in question, such as utterances or actions that are being performed by the actor in question.

In brief, the resulting dialogue goes as follows:

1. User: I would like to prepare a poached egg.
2. System: Alright, I will teach you to poach an egg. Please boil some water.
3. (User boils water)
4. User: I have boiled water.
5. System: Please add some vinegar to the water.
6. (User adds vinegar to the water)
7. User: I have added some vinegar.
8. System: Please crack an egg into a soup ladle
9. (User cracks egg into soup ladle)
10. User: I have cracked an egg into a soup ladle
11. System: Please stir the boiling water
12. (User stirs boiling water)
13. User: I have stirred the boiling water

(etc., until the recipe is done)

14. System: You have now poached an egg. Will you be able to do it by yourself the next time?
15. User: Yes.
16. System: Okay, we're done!

Now, we can clearly see that this dialogue matches with the dialogue model that we have presented in Figure 5.1:

```
1 U: request joint goal
2 S: ok; S: you will learn subgoal; S: do task
3 U does task
4 U: I have done task
5 S: do task
6 U does task
7 U: I have done task
```

```
8 S: do task
9 U does task
10 U: I have done task
11 S: do task
12 U does task
13 U: I have done task
(etc., until subgoal is reached)
14 S: you have performed subgoal.  Capable?
15 U: yes
16 S: done!
```

With this, we have shown that the dialogue model from Chapter 3, the rules from this chapter, and the simpler example dialogue are all consistent with each other.

5.4 Implementation and issues with 2APL

5.4.1 Implementation

A functional demo of a recipe assistant has been constructed [72] with an implementation of our framework in 2APL. The environment is a virtual 'kitchen' on a computer screen, with the workspace on the left hand side and a virtual 'kitchen cabinet' with icons that represent ingredients and tools on the right hand side, as is shown in Figures 5.18, 5.19, and 5.20.

Besides performing actions on the environment, the user can send messages to the system by selecting one of a number of given options in a drop-down menu at the bottom of the screen. The dialogues produced by the system follow the dialogue model from Chapter 3. The dialogue model was also the basis for the selection of options that are represented in the drop-down menu at the bottom of the screen, from which the user can choose his dialogue contribution.

The instructor agent is represented as a Microsoft Agent that gives instructions through speech and text (represented in a text balloon). There are a number of different Microsoft Agents that can be selected, including Peedy (Figure 5.18), Genie (Figure 5.19), and Merlin (Figure 5.20). In order to create a varied interaction, each of the speech acts has a number of different linguistic instantiations; e.g., for the communicative action `send(U, inform, p)`, some of the options are "Yes", "Of course", "Sure" and "Yup".

The user can select tools and ingredients on the right hand side of the screen and drag them onto the workspace. They can either be placed on an empty spot in the workspace, or onto other objects in order to perform actions such as filling a pan with water (by dragging the 'water' icon onto the 'pan' icon in the workspace), turning on the heat source under a pan (by dragging the 'fire' icon onto the 'pan' icon in the workspace), or cutting an ingredient (by dragging the 'knife' icon onto the ingredient icon in the workspace and selecting the action 'cut').

While we abstract from the meaning of cooking actions in our framework and only represent them by identifiers such as `boil_water`, the cooking actions in the implementation have to be specified semantically, representing tools and ingredients

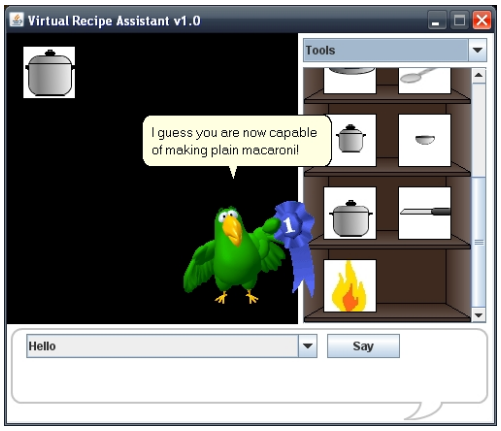

Figure 5.18: *The system informs the user that he has just prepared macaroni.*

in such a way that allows the user to manipulate them by performing cooking actions. Preconditions are also listed: an action such as `fill_container_with_water` requires a suitable container and a specified amount of water, while `stir_water` requires a tool that is suitable for stirring.

In the implementation, the tools are ordered in two categories: *container-tools*, such as pans and bowls, which can contain ingredients, and *executive-tools*, such as knives and spoons, which can be used to perform operations on ingredients. Four parameters can be defined on tools: besides a *name* (identifier), *title* (the name that is shown in the interface) and an *image*, container-tools also have a certain maximum *capacity*, while for executive-tools it is necessary to define which *actions* can be performed with it (e.g., a spoon can be used for stirring, but not for cutting).

Ingredients have six parameters: besides a *name*, *title* and *image*, the parameter *capacity* specifies the minimum capacity that a container must have to contain the ingredient, the parameter *unit* indicates the measuring units that are used for the ingredient (e.g., grams, pinches, cups, tablespoons), and the parameter *amounts* indicates the steps in which the ingredient can be measured in the interface (e.g., water can be added in amounts of 50 ml: 50, 100, 150, 200, and so on).

The recipes are defined as lists that look like $R[a_0, a_1, a_2, a_3]$, where the actions a_0 ... a_3 are defined separately; for example, $a_0 = \text{select(pan)}$. The tools and ingredients are defined separately from the recipe, to allow for reuse of tools and ingredients in different recipes. As in our framework, recipes can contain sub-recipes with the tag *include* and a specification of the identifier of the recipe that is included at this point. The system's beliefs about the capabilities of the user are implemented and allow the system to give subgoals of the recipe as instructions to the user (e.g., 'boil water' instead of its subtasks).

The implementation also includes the functionality of teaching the user how to prepare (parts of) recipes. Corresponding with the rules that we have presented in this chapter, the system announces to the user that it is going to teach him how to perform a certain task, then gives the instructions, and then concludes by informing

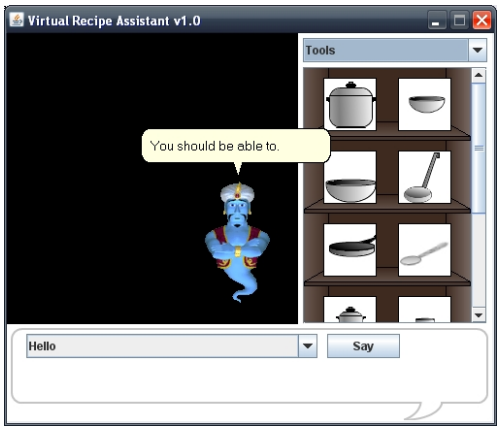

Figure 5.19: *The system has just instructed the user to prepare plain macaroni as a subtask of the recipe for macaroni bolognese.*

the user that the task is completed and asking him whether he can perform the recipe in question by himself the next time (see Figure 5.18).

When the system instructs the user to prepare (a subtask of) a recipe that he believes the user is capable of preparing, he informs the user of this. In Figure 5.19, we show the situation where the system and the user are making macaroni bolognese together, the system believes the user is capable of preparing plain macaroni, and has just instructed the user to do this.

The user can ask the system for explanation at any point in the instruction dialogue. For example, when the system has instructed the user to stir, the user can ask "How?" and the system will answer with an object that can be used for stirring (see Figure 5.20).

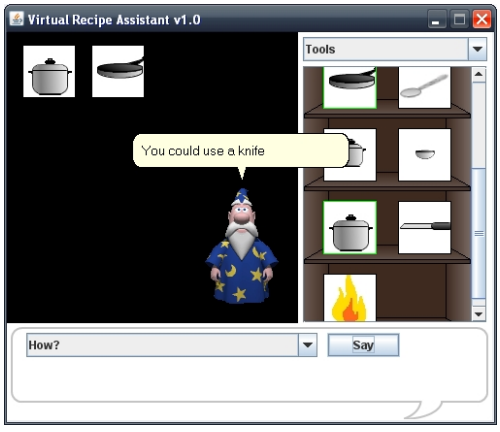

Figure 5.20: *The user asks the system how to stir.*

The implemented system does not cover our complete framework. For example,

the concept of opportunities is not implemented at this time, and some additional extensions would have to be programmed before opportunities can be implemented. First of all, the available quantity of the ingredients is not registered in the implemented system; instead, all ingredients are taken to be infinitely available. Also, the state of the environment cannot be saved at the end of the interaction and loaded for the next interaction. Both of these aspects are needed in order to reason with opportunities.

Another aspect that is not implemented at this time is the possibility for the user to request actions from the system. This also requires an extra functionality that is not implemented: in this implementation, the instructor is not able to manipulate the domain. This means that the system cannot volunteer to perform tasks. In a future (extension of this) implementation, we would like to integrate this functionality.

5.4.2 Issues with 2APL

Although we have represented our framework in a 2APL-like pseudocode, there are some aspects of our system that cannot be easily or optimally implemented in 2APL. Our algorithms require a substantial amount of introspection: reasoning about one's own goals, plans, and actions. Unfortunately, 2APL does not support the ability to reason about anything outside of the beliefbase, which means that most of the rules in our framework that work with recipes are not possible in 2APL. To work around this shortcoming, a recipe has to be represented in two locations: first of all as a plan in the system's planbase, and secondly in the beliefbase, so the system can use it to reason about opportunities before the plan in question is adopted.

In the current implementation, the recipes, tools, actions, and ingredients are all defined in three different programming languages: Java, Prolog and XML. While XML is the main representation language of the recipes, the system also required a Prolog representation for use as part of the beliefbase, and a Java representation is needed for constructing the objects in the interface. It may be worth investigating whether it is possible to construct a parser that can convert the XML representations of the recipes, tools, actions, and ingredients to their counterparts in Prolog and Java, so that they only have to be represented in XML.

Another issue that was encountered in the implementation is the synchronisation of the Microsoft Agents with the 2APL agents. The Microsoft Agents, which are represented by the virtual characters on the computer screen, have to pronounce utterances and make gestures, which takes much longer than the computational operations that the 2APL agents have to perform. In order to prevent the Microsoft Agent and the 2APL agents to run out of sync, the interface has to be blocked, or frozen, when the Microsoft Agent performs an action. This means that the user cannot manipulate the objects on the screen if the Microsoft Agent is performing an action or saying something.

5.5 Conclusions and future work

In this chapter, we have presented a set of rules in the style of a 2APL program that can generate a recipe preparation dialogue. The resulting dialogue that is gen-

erated depends on the agent's beliefs, in particular beliefs about the capabilities of the participants and beliefs about the current state of the task domain, and a set of recipes. The resulting system engages in mixed-initiative dialogues, with reactive aspects (e.g., being able to answer questions from the user) and proactive aspects (e.g., rejecting goals that cannot be achieved under the current circumstances). The resulting dialogues adhere to the dialogue model that we presented in Figure 5.1.

We have specified the rules of the system in a 2APL-like pseudocode. The successful implementation of part of our framework has shown that it is indeed suitable for the development of a task-oriented dialogue system. Unfortunately, not all algorithms can be easily implemented in 2APL and it should therefore be investigated whether other agent programming languages may be more suitable for the implementation of our framework. Alternatively, workarounds can be found for most of the issues.

For future work, a possible extension for the system is more flexibility in the handling of recipes that fail the opportunity check. If the system concludes that none of the participants are able to perform one of the actions, the recipe might still be an option, if the user knows a way to achieve one of the action's supergoals in a different way that is not known to the system. For example, imagine there is no water boiler present in the kitchen and the only recipe for boiled water that the system knows is to boil it in a water boiler. In this case, the precondition for one of the actions in the recipe will be the availability of a water boiler, which is not derivable from the system's beliefbase. However, the user might be able to boil water in a pan on the stove or in the microwave. To account for this, the system would need to ask the user if he is capable of performing one of the action's supergoals; for example by saying: "We have to boil water for this recipe, but there is no water boiler available. If you know how to boil water without a water boiler, we can still prepare this recipe." If the user responds positively, the system should treat the task 'Boil water' as sort of a black box, not unfolding it to its subgoals.

We have decided that if both participants are capable of performing a certain task, the system will always ask the user to perform the task in question. Because the system also fuctions as a tutor for the user, we have decided to keep the system's involvement in the recipe preparation to a minimum. Alternatively, we could specify a laziness (or eagerness) property for the system, meaning that if this parameter is set to 'eager', the system will perform all tasks in the recipe that it is capable of performing, regardless of the user's capabilities. (The system's behavior as it is specified in this chapter would fall under 'lazy'.)

In the future we would also like to account for more complex recipe trees that involve parallel (interleaving) tasks. This will involve a different algorithm for computing opportunities, where different possible configurations w.r.t. the order of subtasks of parallel tasks should be checked. An option for this is GPGP (generalized partial global planning) [86], a framework that is developed for use in teams of agents to facilitate the coordination between the agents.

Furthermore, we would like to eventually extend the system with a more advanced recipe selection phase.

Actor: Rule no.	**Communicative act, PC-rule or BeliefUpdate**	**Action or utterance**
U	Starting state: `goal(poach_egg)`	
U: 16	`send(S, request, jointgoal(poach_egg))`	User: I would like to prepare a poached egg
S: 1	`adopta(jointgoal(poach_egg))`	
S: 5b	`send(U, inform, possible(poach_egg))`	System: Alright
U: 16	`adopta(jointgoal(poach_egg))`	
S: 9b	`send(U, inform, start-learning(U, poach_egg))`	System: I will teach you to poach an egg
S: 9b (contd.)	`teachprepare(poach_egg, [boil_water, add_white_vinegar, crack_egg, stir_boiling_water, drop_egg, wait_3-4_minutes, remove_egg] )`	
U: 17	`tentativeRecipe(poach_egg = [])`	
S: 10b	`send(U, request, next-action(U, boil_water))`	System: Please boil some water
U: 18b	`tentativeRecipe(poach_egg = [boil_water])`	
U: 18b (contd.)	`@kitchen(boil_water, performed(U, boil_water), Time-out)`	(User boils water)
U: 18b (contd.)	`send(S, inform, performed(U, boil_water))`	User: I have boiled water
S: 10b (contd.)	`checked-done(boil_water)`	
S: 10b (contd.)	`teachprepare(poach_egg, [add_white_vinegar, crack_egg, stir_boiling_water, drop_egg, wait_3-4_minutes, remove_egg] )`	
S: 10b	`send(U, request, next-action(U, add_white_vinegar))`	System: Please add some vinegar to the water
U: 18b	`tentativeRecipe(poach_egg = [boil_water, add_white_vinegar])`	
U: 18b (contd.)	`@kitchen(add_white_vinegar, performed(U, add_white_vinegar), Time-out)`	(User adds vinegar to the water)
U: 18b (contd.)	`send(S, inform, performed(U, add_white_vinegar))`	User: I have added some vinegar

Table 5.2: *An alternative version of the example dialogue in which no errors occur (part 1)*

Actor: Rule no.	Communicative act, PC-rule or BeliefUpdate	Action or utterance
S: 10b (contd.)	`checked-done(add_white_vinegar)`	
S: 10b (contd.)	`teachprepare(poach_egg, [crack_egg, stir_boiling_water, drop_egg, wait_3-4_minutes, remove_egg] )`	
S: 10b	`send(U, request, next-action(U, crack_egg))`	System: Please crack an egg into a soup ladle
U: 18b	`tentativeRecipe(poach_egg = [boil_water, add_white_vinegar, crack_egg])`	
U: 18b (contd.)	`@kitchen(crack_egg, performed(U, crack_eggr), Time-out)`	(User cracks egg into soup ladle)
U: 18b (contd.)	`send(S, inform, performed(U, crack_egg))`	User: I have cracked an egg into a soup ladle
S: 10b (contd.)	`checked-done(crack_egg)`	
S: 10b (contd.)	`teachprepare(poach_egg, [stir_boiling_water, drop_egg, wait_3-4_minutes, remove_egg] )`	
S: 10b	`send(U, request, next-action(U, stir_boiling_water))`	System: Please stir the boiling water
U: 18b	`tentativeRecipe(poach_egg = [boil_water, add_white_vinegar, crack_egg, stir_boiling_water])`	
U: 18b (contd.)	`@kitchen(stir_boiling_water, performed(U, stir_boiling_water), Time-out)`	(User stirs boiling water)
U: 18b (contd.)	`send(S, inform, performed(U, stir_boiling_water))`	User: I have stirred the boiling water
	(etc., until teachprepare-list is empty)	
S: 10a	`send(U, inform, performed(poach_egg))`	System: You have now poached an egg
U: 19b	`RemoveBelief(tentativeRecipe( poach_egg = [boil_water, add_white_vinegar, crack_egg, stir_boiling_water, drop_egg, wait_3-4_minutes, remove_egg])`	

Table 5.3: *An alternative version of the example dialogue in which no errors occur (part 2)*

Actor: Rule no.	**Communicative act, PC-rule or BeliefUpdate**	**Action or utterance**
U: 19b (contd.)	`AddBelief(poach_egg = [boil_water, add_white_vinegar, crack_egg, stir_boiling_water, drop_egg, wait_3-4_minutes, remove_egg])`	
U: 19b (contd.)	`AddBelief(capable(U, poach_egg))`	
S: 9b (contd.)	`send(U, query-if, capable(U, poach_egg))`	System: Will you be able to do it by yourself the next time?
U: 20a	`send(S, inform, capable(U, poach_egg))`	User: Yes
S: 9b (contd.)	`AddBelief(capable(U, poach_egg))`	
S: 9a	`send(U, inform, finished(poach_egg))`	System: Okay, we're done!
U: 19a	`dropgoal(poach_egg); dropgoal(jointgoal(poach_egg))`	
S: 9a (contd.)	`dropgoal(poach_egg)`	

Table 5.4: *An alternative version of the example dialogue in which no errors occur (part 3)*

6 Parsing and Generating Natural Language

> The great thing about human language is that it prevents us from sticking to the matter at hand.
>
> *Lewis Thomas*

Although we only treat errors on the speech act level in this thesis, and abstract from any communication errors such as mishearings, we will still treat some insights in the field of natural language processing and generation in this chapter. We have chosen Functional Discourse Grammar (FDG) [74] as our preferred way of natural language processing and generation. There are of course many other possibilities for language interpretation and generation, but FDG matches well with our BDI system, for reasons which we will explain below.

FDG describes how a communicative intention is transformed into an utterance via four levels of representation, and consequently, FDG literature mostly describes the process that translates a communicative intention into a natural language utterance. We have taken up the challenge of focusing our research on the reverse process, and an algorithm has been developed for parsing natural language utterances to elicit the communicative intentions that lie behind them [128] [80]. Briefly, this algorithm takes the representation of an utterance on one of the four levels of representation of FDG, takes it apart into smaller pieces called 'chunks', translates these chunks to another FDG level, and then puzzles these chunks back together according to a set of rules. Although most discourse acts have more than one function [28], we will focus on eliciting the main domain-related function of the discourse act.

Because the communicative acts that are given as output by the system are quite simple and straightforward in our current application, we will not delve too deeply into the generation of natural language. For this purpose, we can simply program a straightforward utterance generator, that adds some variables (e.g., the linguistic representation of tasks) to some standard frames (e.g., a standard form in which instructions are given). We only have to define sentence structures for the different communicative acts and have a database of natural language representations for the propositional content, such as "poach an egg" for `poach_egg`.

Note that there can be different natural language instantiations of the com-

municative acts in the example dialogue. For example, `send(U, inform, not possible(french_toast))` can be translated into many different natural language utterances; beside the version in the example dialogue, "That's not possible", other options are "We cannot make French toast", "It is not possible for us to prepare French toast at this time", or "Sorry, that's not an option". Some communicative acts can even be instantiated as different sentence types; for example, `send(U, request, next-action(U, boil_water))` can be instantiated as an imperative, as in the example dialogue, "Please boil some water", but also as a question: "Can you please boil some water?" or as a declarative sentence: "You should boil some water." In the implementation of our model, a number of natural language utterances were implemented for each possible dialogue move by the system; e.g., as we mentioned in the previous chapter, "Yes", "Of course", "Sure" and "Yup" are some of the options for the dialogue move `send(U, inform, p)`.

Following most literature on Functional Discourse Grammar [74], we will describe the theory 'top-down' in Section 6.1, starting with a communicative intention that is formed in the conceptual component (6.1.1), the contextual component (6.1.2), the grammatical component with its four levels: interpersonal (6.1.3), representational (6.1.4), morphosyntactic (6.1.5) and the phonological level, which we treat together with the output component (6.1.6). In Section 6.2, we will explain how we can make the parsing of a natural language utterance in FDG easier with our knowledge of the dialogue models that we presented in Chapter 3 of this thesis. In Section 6.3, we will then describe how a natural language utterance can be parsed to a communicative intention using FDG, first treating the process of parsing to the representational level (6.3.1) and then to the interpersonal level (6.3.2). This generates some specific problems in translating from one FDG level to another, and in Section 6.4 we will focus on one specific example of such a problem, namely the interpretation of indirect speech acts, first explaining about the *verb-construction* (6.4.1) and then the speech act assignment (6.4.2). Finally, in Section 6.5 we will finish this chapter with conclusions and future work.

6.1 A brief introduction to Functional Discourse Grammar

Functional Discourse Grammar [74] is a grammar theory that describes how communicative intentions are encoded into linguistic utterances. It does not strive to be a theory of discourse; the word 'discourse' in the name only indicates its focus on discourse acts and on linguistic utterances in the context of a discourse. FDG is a 'typology-based theory', meaning that it attempts to encompass *all* languages. On the other hand, if FDG is applied to a language where certain phenomena do not apply, they can simply be left out of the FDG representations, as we will see later in this chapter.

FDG is based on Functional Grammar [49]. It is, as the name suggests, a functional theory, as opposed to Chomskyan transformational grammars. As all functional theories, FDG relates the function of an utterance to its form (a 'function-to-form' approach), focusing on the process of translating a communicative intention (the 'func-

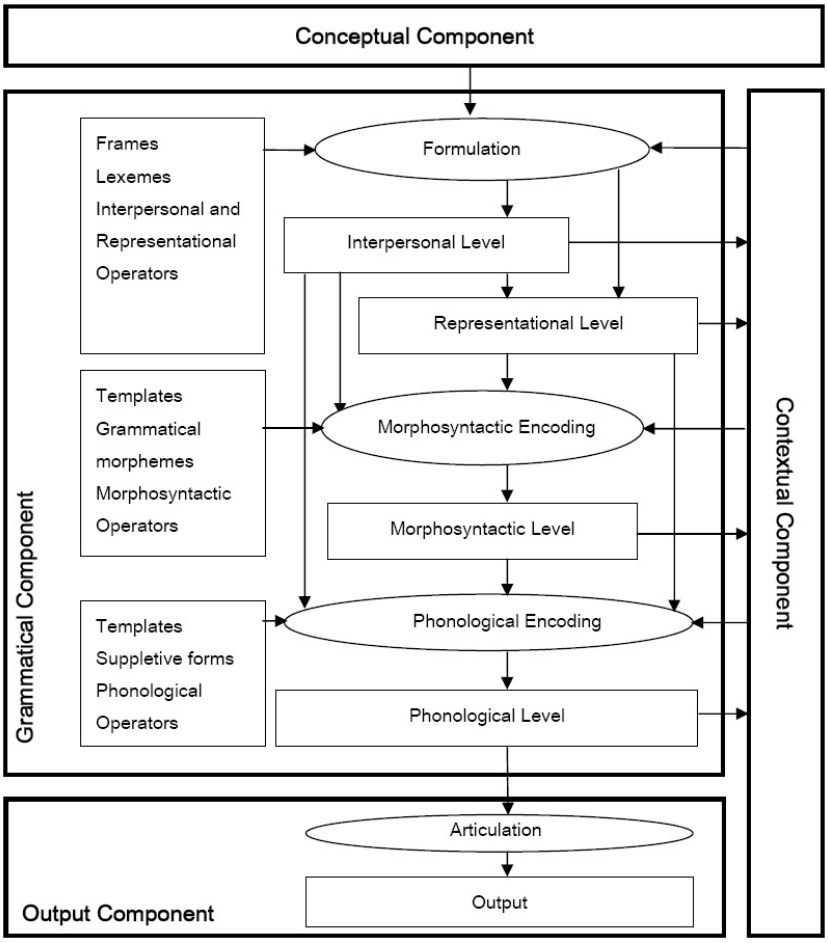

Figure 6.1: *The components of Functional Discourse Grammar [74]*

tion') into a natural language utterance (the 'form') by way of a *formulation* process. In contrast, its predecessor, Functional Grammar [49], takes a 'form-to-function' approach, focusing rather on the explanation of formal properties of syntactic units in terms of their functions in communication.

The framework of FDG consists of four main components: the *conceptual component*, in which communicative intentions are formed; the *contextual component*, which contains contextual information; the *output component*; and, most importantly, the *grammatical component*, which translates a communicative intention via four levels to a phonological structure that is then given to the output component. The four levels in the grammatical component are called the *interpersonal level*, the *representational level*, the *morphosyntactic level*, and the *phonological level*. These levels are roughly comparable to the four levels of Van Dam [47], which were inspired by the

Layered Protocols from Taylor [120] to which we referred earlier in this thesis. The components of FDG are illustrated in Figure 6.1.

The architecture of FDG mirrors the three fundamental modules in a language system from Levelt [87]: the conceptualizer, the formulator, and the articulator. In FDG, these correspond to the conceptual component, the grammatical component and the output component, respectively. (The contextual component from FDG is not present in Levelt's theory.) In our architecture as we have presented in Chapter 4, the modules from Levelt roughly correspond to the reasoning engine, the utterance formulator and the output component.

The conceptual and contextual components are the only parts of the FDG structure that are non-linguistic; all layers in the grammatical component, and naturally the output component, pertain specifically to linguistic information. As we have mentioned above, all components in FDG are language-specific. For the non-linguistic components, this is reflected in the types of information that are present there; for example, information that does not have any direct effect on the communicative intention is not represented in the conceptual component, as we will explain below. What information this is, differs between languages.

The translation from a communicative intention to representations on the interpersonal and representational levels is done through the process of *formulation*, while the translation from these pragmatic and semantic representations into morphosyntactic and phonological structures is called *encoding* [74].

One of the aims of FDG is to be a single theory that can explain all grammatical aspects of an utterance, encompassing everything from phonological to pragmatic information. This allows for a unified treatment of linguistic phenomena that can occur on different levels, such as anaphora; e.g., the demonstrative pronoun 'that' can refer to a referent on different levels, as is illustrated by these examples:

A: Get out of here!
B: Don't talk to me like *that*!

A: I had *chuletas de cordero* last night.
B: Is *that* how you say 'lamb chops' in Spanish?[1]

In the first example, B's *that* refers to the communicative strategy employed by A, which is a pragmatic aspect of A's utterance. In the second example, *that* refers to the specific term *chuletas de cordero*, not to the referent, making it an anaphor on the morphosyntactic level. For similar examples for the representational (semantic) and phonological levels, see Hengeveld and Mackenzie [74].

On the other hand, not all levels in the grammatical component are used every time (the *maximal depth principle* [74]). For example, an utterance such as "Okay" does not contain any direct references to the world outside the conversation and therefore does not need to pass through the representational level, which deals with these aspects of linguistic representation.

In FDG, the *discourse act* is taken to be the basic unit of analysis [74]. A discourse act is defined as "the smallest identifiable unit of communicative behavior" (cf. Kroon [79]). There is not necessarily a one-to-one mapping between discourse acts and

[1] Examples from Hengeveld and Mackenzie [74]

sentences or clauses, which other theories of language tend to take as basic units. A discourse act may be part of a *move*, which is defined as "the minimal free unit of discourse that is able to enter into an exchange structure", meaning that it has some function in moving towards the conversational goal (in contrast with discourse acts, which do not have to have such a function). A move may contain more than one discourse act, but always has one central (main) discourse act.

FDG is a suitable choice for us to combine with a BDI-based reasoning engine, since FDG specifically connects the form of a natural language utterance to the goals and beliefs of the agent that utters it, via the conceptual component. Since FDG does not specify the conceptual component in much detail and only states that it develops the communicative intentions, we can use a BDI reasoning engine as our conceptual component.

The representations of discourse acts on all four levels in the grammatical component are in the form of hierarchical structures. For example, the representation of the sentence "I can't find the red pan" on the interpersonal level is as follows:

```
(M_i : [
    (A_i : [
        (F_i : DECL (F_i))
        (P_i)_S
        (P_j)_A
        (C_i : [
            (+id +s R_i : I (R_i))
            (T_i)_FOC
            (+id +s R_j : [(T_j) (T_k)](R_j))_TOP
        ] (C_i))
    ] (A_i))
] (M_i))
```

Roughly, this means that there is one move M_i (the complete utterance), which contains one discourse act A_i (also encompassing the complete utterance), which contains one delarative sentence type (F_i : DECL), one participant who is the speaker ($(P_i)_S$), one who is the addressee ($(P_j)_A$), and the communicated content of the discourse act, C_i. This communicated content consists of a referent R_i (referring to the speaker again, here represented by the personal pronoun *I*), a subact of ascription that is the focus of the utterance $(T_i)_{FOC}$ (denoting the relation 'can't find' between the speaker and the red pan), and another referent, which is the topic of the utterance $(R_j)_{TOP}$ ('the red pan') and which contains two subacts of ascription T_j and T_k (referring to 'red' and 'pan'). Both referents have two *operators*, +id and +s, denoting respectively the fact that they are identifiable and specific. A referent can be identifiable (+id) or unidentifiable (-id), denoting whether or not the addressee is familiar with the referent; a referent can be specific (+s) or unspecific (-s), denoting whether or not there is one distinct referent that the speaker refers to. An example of an unidentifiable, non-specific referent is "I am looking for *someone to help me.*" [74]

We do not aim to give a complete FDG tutorial here. For more detailed informa-

tion and for examples of FDG structures on the other levels, we refer to Hengeveld and Mackenzie [74]. However, we hope that the above example gives the reader a general idea of how FDG structures look and what types of information they contain.

6.1.1 The conceptual component

The conceptual component in the general layout of FDG is the part where communicative intentions and the corresponding mental representations are formed. The conceptual component roughly corresponds to the BDI reasoning engine in our system; we can use the communicative acts that are formed by the BDI reasoning engine as the communicative intentions in the conceptual component of FDG. A communicative intention can be encoded into different types of sentence types. For example, a request can get encoded into an interrogative ("Can you give me the salt?"), a declarative ("I need the salt"), or an imperative ("Pass me the salt!"). The choice between these options may depend on various factors.

Interestingly, the FDG literature uses the term 'illocution' not in the same way as Searle [110] and Austin [9] do. In FDG literature, the term 'illocution' is used for sentence types such as declaratives and interrogatives, while in contrast, Austin and Searle use the term to denote the act that is performed in saying something, such as promising or offering. This sense seems to correspond rather to the communicative intention in FDG. In the remainder of this chapter, to avoid confusion, we will wherever possible use 'sentence type' for the concept 'illocution' as it is used in FDG-literature, and 'speech act' to denote the concept that Searle and Austin call 'illocution'.

In the FDG literature, not much focus is put on the conceptual component. The exact form of a communicative intention is not specified. Hengeveld and Mackenzie [74] stress Butler's [34] proposal that 'conceptual content proper' and 'affective/interactional content' should be distinguished, which corresponds with the distinction between the interpersonal and representational level.

As we have mentioned above, our BDI reasoning engine can be seen as the conceptual component of FDG. It forms communicative intentions based on the beliefs, goals and plans of the agent.

6.1.2 The contextual component

Dialogue context is a highly important source of information in an interaction. The context of a dialogue is all information that influences the formulation and encoding of a communicative intention in a significant way. In other words, it is the combination of all (background) information in a dialogue situation: dialogue history, discourse topic, salient words or topics, a world model, user model, etcetera. According to Dynamic Interpretation Theory [26] [27], context corresponds to the complete set of beliefs in the BDI paradigm [94]. Without context, it is impossible to interpret input correctly; usually, not all information is present in the utterance itself. For example, the meaning of an utterance that only consists of the word 'yes' depends highly on the question or utterance that it is a reply to.

The contextual component in FDG is an essential component. After all, the 'Discourse' in the term Functional Discourse Grammar emphasizes the fact that utterances should always be interpreted in the context of the discourse. The contextual component contains descriptions of the content and form of the preceding discourse, the physical context, and social relationships between the participants in the discourse. Reflexives, anaphora, and narrative chaining are all based on information that the grammatical component receives from the contextual component. The contextual component contains two types of information: it contains information that it receives from the grammatical component which is relevant to the form of subsequent utterances (comparable to a dialogue history or our dialogue models), and longer-term information, including sociocultural context [46].

This dichotomy between dialogue context and longer-term information is similar to the two subcategories of context, local and global, in which Bunt [25] splits all of his five dimensions of context. The five dimensions of context are as follows: linguistic context (in a broad sense, including prosodics), semantic context (underlying task and task domain), physical context (place, time, communication channels), social context (roles, institutional setting, rights and obligations) and cognitive context (beliefs, intentions, plans, attitude, discourse topic). These five dimensions of context are all split into two subcategories: local and global aspects (global aspects stay the same throughout the dialogue, local aspects change per utterance).

As in the case of the conceptual component, Hengeveld and Mackenzie [74] do not elaborate on the internal constitution of the contextual component. However, they do state some more details on the contents of the contextual component. For example, it only contains information that influences the processes of formulation and encoding in a significant way. Information that is present in the contextual component includes information that influences phenomena such as anaphora and reflexives. Hengeveld and Mackenzie state that only information that one would need to correctly interpret an utterance is included in the contextual component.

On the other hand, a lot of real world knowledge is needed in the higher-level interpretation of some utterances. Take, for example, the combination of these two sentences:

> "I read a book yesterday. The author lives in Amsterdam."

In order to correctly interpret 'the author' in the second sentence as the author of the book mentioned in the first sentence, the system needs to have knowledge about the concepts 'book' and 'author' that connects the two. So, to be able to deal with this type of phenomenon, the contextual component would need to contain a lexical database containing semantic and ontological information, such as WordNet [52] or FrameNet [104].

6.1.3 The interpersonal level

The interpersonal level of FDG pertains to the relation between the speaker and the addressee, and to the effects that the speaker intends his utterance to have on this relation. It is mostly comparable to the pragmatic content of the utterance,

although it only contains pragmatic elements that pertain to the linguistic form of the utterance.

The translation from the communicative intention in the conceptual component to a structure on the interpersonal level includes the addition of specific language-bound information, such as the distinction between informal and formal forms of certain words (e.g., *vous* vs. *tu* in French, a distinction that is not present in English). Information that influences such decisions is taken from the contextual component.

The structure on the interpersonal level contains sentence types, such as DECL (declarative), INTER (interrogative), and IMPER (imperative). The exact set of sentence types that is used depends on the type of natural language that the utterance is to be expressed in; only sentence types that are "justified by the grammatical distinctions present in the language" will be used on the interpersonal level. English only contains declaratives, interrogatives, imperatives, optatives ("Let him..." or "May he..."), hortatives ("Let's...") and miratives ("How beautifully...!") [74]. Some languages have specific grammatical structures for other sentence types, such as admonitives (warnings). For example, in Mandarin Chinese the word *ou* is a specific admonitive marker. However, in English, warnings are usually expressed through the use of imperatives ("Watch out!"), but do not have their own specific syntactic markers. Therefore, the concept of admonitives is not used in FDG structures for the English language.

Some decisions pertaining to the concrete form of the utterance are already made on this level; for example, the distinction between implicit and explicit performatives: "I will do the washing up" vs. "I promise I will do the washing up". Also, some modifiers may be introduced on this level, e.g., 'sincerely', 'to be honest', 'frankly'. The interpersonal level also includes operators such as irony.

The participants are represented on this level as Speaker and Addressee. Beside the participants and the sentence type, there is of course the *communicated content*, which contains "the totality of what the Speaker wishes to evoke in his/her communication with the Addressee" [74]. Part of the communicated content may be distinguished as Focus or Background, which are marked by so-called *Focus markers*. The Focus is the most important/salient part of the utterance; the Background is the rest of the utterance. A second dimension that may be distinguished in the communicated content is Topic/Comment: the Topic is what the utterance is about; the Comment is what is being said about the topic. The two dimensions Focus/Background and Topic/Comment are notably different, although they may get instantiated on lower levels in similar ways with various devices such as word order (on the morphosyntactic level) and/or stress (on the phonological level).

6.1.4 The representational level

The semantic properties of a discourse act are represented on the representational level. Similarly to the interpersonal level, this level also only represents information that pertains to the linguistic form of the resulting utterance, though on this level the information is semantic, while on the interpersonal level it is pragmatic. The representational and interpersonal levels are strongly related and have a considerable degree of parallellism; whereas the interpersonal level pertains to the relation

between the participants, the representational level deals with the relation between the utterance and the world outside the discourse. The way the structure on the representational level is built up depends partially on the structure at the interpersonal level and partially on the communicative intention from the conceptual component.

In FDG, four basic semantic categories are distinguished in which entities can be classified on the representational level: as an *individual* (e.g., chair), a *state of affairs* (e.g., meeting), a *propositional content* (a mental construct; e.g., idea), or a *property* (e.g., color). Additionally, when FDG is applied to a certain language, entity categories that have specific linguistic properties in that language are also distinguished, following the principle that FDG is tailored to specific languages. In English, there are additional entity categories for *location*, *time*, *episode*, *manner*, *reason* and *quantity*.

6.1.5 The morphosyntactic level

As with the interpersonal and representational levels, there is also a certain amount of parallellism between the morphosyntactic level and the phonological level. As is shown in Figure 6.1, both the morphosyntactic and the phonological level take input from both the interpersonal and representational levels.

The morphosyntactic level represents all lexical information in the utterance. It is at this point in the translation from communicative intention to natural language utterance that the clauses, phrases and words are chosen and put into place. The structure at the morphosyntactic level looks very similar to other types of syntactic structures, using concepts such as subject, object, adjective and noun phrase to denote parts of the structure.

In the translation from the interpersonal and representational levels to the morphosyntactic level, there are two important processes that take place: *positioning* and *alignment*. The former process determines at which position in the utterance hierarchically related units will be positioned; e.g., adverbs with a scope over the whole sentence will be positioned differently than adverbs with a scope over only a verb. Alignment takes care of the positioning of non-hierarchically related units, such as location and time (as in: "I read a book *in a pub yesterday*).

6.1.6 The phonological level and the output component

We will not deal with the phonological level and the output component in much detail here, but only shortly describe them. More extensive information on these topics can be found in Hengeveld and Mackenzie [74].

The phonological level contains phonological representations (phonemes) of the information that is present on the morphosyntactic level. Additional information such as stress and tonal patterns are also present on the phonological level. Even though the phonological level may contain important information, we have chosen to abstract from phonological information in this thesis.

The output component deals with 'analogue' matters such as intensity, duration, and characteristics that reflect individual voice quality and such, whereas the phono-

logical level (and all other levels in the grammatical component) encodes digital information.

6.2 Expectations in FDG

There are various assumptions that can assist us in eliciting the speaker's communicative intention from his utterance. In a cooperative dialogue situation, both dialogue partners usually act with a considerable amount of predictability. As we have seen in the previous chapters, in order to use these types of information, it is useful to keep a (BDI) user model in the form of a formalization of the conceptual component of the user. With this model, some expectations about the FDG structures can be formed by the system. When receiving input from the user, the system will attempt to match the input from the user with the (partial) structures that can already be formed at the FDG levels.

These expectations are based on the dialogue model that we presented in Chapter 3, and the current state of the dialogue. Forming expectations can help with speech recognition and input processing. From the set of possible communicative acts that the system expects the user to perform, some possibilities for parts of the FDG representations can be listed. The events in the dialogue can be linguistic or non-linguistic, but in this chapter we focus on linguistic events. For example, when the system has given the user an instruction, the system expects the user to comply with his instruction and inform the system of this. Alternatively, the system may expect the user to request explanation about the current instruction (e.g., "How long should I stir the water?").

Using the dialogue model is also helpful in interpreting an utterance: the vocabulary of the speech recognizer can be adapted to the expected events, which will yield greater confidence and a smaller word error rate in the recognized utterances. Also, some beliefs about the context will help with the parsing of the input to the higher FDG levels. For example, a question starting with the words "Can you...?" can be interpreted either as a query-if ("Can you understand Dutch?") or as a request ("Can you tell me what time it is?"). The interpretation of such *indirect speech acts* may benefit from a framework where the system expects one of these speech acts, but not the other, depending on factors such as the current (joint) goal or whether certain information is present in the mutual beliefs of the participants. We will present an algorithm for the interpretation of such utterances in Section 6.4.

There is no one-to-one mapping from a communcative intention to a natural language utterance; there are many different FDG structures that can result from one particular communicative intention. One aspect that can facilitate the process is that the system only has to distinguish between the different expectations (in the example mentioned above: `message(U, inform, performed(U, A))` and `message(U, query-ref, info(A, Property, G))`). This means that the system only has to find out which part(s) of the FDG structure are different for the possible expectations. Based on those differences, it is easier to find the relevant information in the parsing process when receiving input from the user.

An advantage of using FDG is that it allows us to encode various pieces of infor-

mation on different levels, that can all be used to distinguish between the different expectations. Parsing to FDG structures on the morphosyntactic and representational level is quite feasible, as we will see in the next sections, but parsing to the interpersonal level and eliciting the communicative intention is much more difficult. On the other hand, expectations for the upcoming input are more certain on the higher levels; while it is difficult to predict the morphosyntactic structure of a certain communicative intention, we have noticed that predicting partial structures on the interpersonal and representational levels is easier. We can use the expectations that we base on our dialogue models together with the results of parsing on all four levels, which allows us to match input with expectations with a higher degree of certainty and robustness than if we would only have one level on which to match input with expectations.

6.2.1 Translating the dialogue model to FDG expectations

We can construct a number of different options for the upcoming dialogue move that the user will perform, based on the dialogue model that we have presented in Chapter 3. Then, these options are translated to partial FDG structures on the different levels, to represent the expectations for the different options. When a dialogue move from the user is detected, it will be parsed to (partial) FDG structures and compared to the expectations.

In this subsection we will treat an example of how expectations can be formed for partial FDG structures after the system gives an instruction to the user. To keep the examples clear and simple, we will treat only two options between which the system has to distinguish: the user informs the system of his successful completion of the task (`message(U, inform, performed(U, A))`) and the user asks for certain information about the current task (`message(U, query-ref, info(A, Property, G))`).

Apart from specific expectations that will distinguish these two options, we can also define some general expectations about the FDG structures that we expect to be present in *both* options. For example, on the interpersonal level, there will probably be one move, consisting of one discourse act (which specifies whether the user informs the system that he has performed the task in question, or asks for explanation about the task). An additional discourse act with more information might be present (a request for a new instruction, for example), but in this section we will solely discuss the main discourse act.

Translating these general expectations to lower levels, we can predict that the discourse act corresponds with one state-of-affairs (denoted by `e`) on the representational level, and on the morphosyntactic level with one clause. On the representational level, the user is usually the Actor, the object that he performed (or should perform) the instruction on is expected to be the Undergoer, and the action he performed is expected to be a function (denoted by `f`).

Expectations for `message(U, inform, performed(U, A))`

When the user informs the system that he has performed the requested action, the structure on the interpersonal level will probably look like this:

(F_i : DECL (F_i)) (P_i)$_S$ (P_j)$_A$
(C_i : [
    (- expression of success -)$_{FOC}$
    (- (part of) instruction -)$_{TOP}$
] (C_i))

The sentence type F_i is a declarative, and the discourse act has two participants: (P_i) as the speaker and (P_j) as the addressee (note that the S and the A on the interpersonal level stand for Speaker and Addressee and are not the same as A and U for Actor and Undergoer on the representational level, or S and U for System and User that we use throughout this thesis). The communicated content C_i contains two parts: an expression of success that is the focus of the utterance, and the instruction (or part of it) that is the topic of the utterance.

On the **representational level**, one discourse act corresponds with one state-of-affairs (denoted by `e`). Again, the structure will most probably contain referring expressions to (part of) the instruction, the user is expected to be the Actor, the object that he performed the instruction on is expected to be the Undergoer, and the action he performed is expected to be a function (denoted by `f`). Most likely, a negative modifier will not be present in the structure.

We cannot construct very accurate expectations at the **morphosyntactic level**, beside the fact that the utterance will most likely be declarative, as we have already seen on the interpersonal level. Most likely, there will be one clause (corresponding to the one discourse act on the representational level). Some words that are associated with a successful action might be present, such as 'done' or 'succeeded'. However, attention should be paid to the possible presence of a negative term in combination with one of these words; e.g., "I didn't succeed" is obviously not an expression of success.

Expectations for `message(U, query-ref, info(A, Property, G))`

There are a few features that are likely to be present in an utterance from the user asking the system for more information. The user will most likely state the specific aspect of the instruction that he needs additional information about, and the sentence type of the utterance will most likely be a question. On the **interpersonal level**, the structure will probably contain:

(F_i : INTER (F_i)) (P_i)$_S$ (P_j)$_A$
(C_i : [
    (- property -)$_{FOC}$
    (- (part of) instruction -)$_{TOP}$
] (C_i))

As we have said, the sentence type (F_i) is interrogative. There are two participants of the discourse act: P_i as the speaker and P_j as the addressee. The communicated content C_i now consists of a property that is the focus of the sentence (e.g., duration,

temperature, size) and the instruction (or a part of it) that is the topic of the sentence. For example, when the user asks: "How long should I stir the water?", the instruction 'stir the water' is the topic of the sentence, and the property 'how long' is the focus.

The structure on the **representational level** is expected to contain a state-of-affairs (denoted by `e`), a reference to the instructed action (denoted by `f`), and a reference to a property of the instructed action.

Again, we cannot make an accurate expectation of the representation of a communicative intention on the **morphosyntactic level**. The utterance is most likely a question and may therefore contain wh-words such as 'what', 'how' or 'why'. The utterance most likely contains words that indicate parameters of the instructed action, such as, in this case, 'duration', 'time' or '(how) long'. Additionally, words that refer to (parts of) the instruction and/or objects may be present, although they may be absent if ellipsis is used (e.g., when the user simply asks "How long?").

6.3 Natural language parsing with FDG

The translation from a communicative intention in the conceptual component to a representation on the interpersonal and representational levels is done through the process of formulation. For interpreting input and inferring the speaker's intentions from it, we need to focus on reversing this process: eliciting the communicative intention from the speaker's utterance. This is a non-trivial process, as there is no one-to-one mapping from a natural language utterance to a communicative intention and vice versa.

In order to automatically transform a transcribed utterance to a communicative intention, we need to go 'backwards' through the four levels of representation. A representation that dissects the text or speech from the input module into morphosyntactic constituents can be reasonably well achieved by any off-the-shelf speech recognition and/or parsing technology, with small modifications to generate FDG structures on the morphosyntactic level instead of another hierarchical representation of the syntax. The difficulties will be in deriving the representational and interpersonal level from the morphosyntactic level and the input utterance, and transforming the structures on the representational and interpersonal levels to a communicative intention.

FDG is a relatively new grammar theory, and our parser which can translate morphosyntactic structures into corresponding structures on the interpersonal and representational level [128] [80] is the first to do so. The algorithm involves a way to parse utterances to a morphosyntactic representation, an algorithm to compute the structure at the representational level given the structure at the morphosyntactic level, and an algorithm to compute the structure at the interpersonal level given the structures at the representational and morphosyntactic levels. From the interpersonal level, the utterance still needs to be matched to a certain communicative intention. We leave this for future work.

A structure on the morphosyntactic level can be relatively easily obtained from a fragment of natural language with a conventional parser. We will not elaborate on this process in this thesis.

6.3.1 Parsing to the representational level

A Prolog parser has been developed [80] that can translate FDG structures on the morphosyntactic level to structures on the representational level. The algorithm has been tested on a small fragment of English; the following sentences can successfully be parsed by the system:

- "I'm hungry."
- "What would you like to cook?"
- "Can you do something with those ingredients?"
- "I'll look for a recipe."

Given a structure on the morphosyntactic level, the algorithm works in three stages, which we will explain below. To understand the algorithm, it is convenient to view an FDG structure as a tree, where the hierarchical structure is represented as branches of a tree. In the example of an FDG structure that we have seen above in Section 6.1, the Move is the top node of the tree, with the Act as its only subnode. The Act has four subnodes: the sentence type (F), the speaker (P_i), the addressee (P_j), and the communicated content (C).

The first stage of the algorithm is the *treewalk stage*, in which the nodes of the morphosyntactic tree are, part for part, translated into partial representational structures. The result of this stage is a sequence of such 'chunks'. This translation process uses a lexicon that contains entries for nouns, verbs, adjectives and adverbs. *Grammatical words*, such as articles and prepositions, are represented by the structure itself and therefore do not occur as nodes.

Each of the chunks that result from the treewalk stage has a number of *combination criteria nodes*, which specify how the chunks can be combined to form a tree on the representational level. These are used in the second stage, the *composition stage*. Additionally, the algorithm may use extra chunks that represent *representational frames*, which can be used as 'glue' to put the content chunks together. The combination criteria nodes that have not been combined yet are represented as a list, the *open spots queue*. The goal of the composition stage is to combine the chunks in such a way that the open spots queue is empty. The algorithm takes the first open spot from the queue, tries to find another node that fits it, and when such a node is found, combines the two nodes and removes the appropriate open spots from the queue. Prolog's backtracking mechanism can be used whenever the algorithm is stuck (when the open spots queue is not empty, but no matching nodes can be found).

The final phase of the algorithm is the *coreference resolution phase*. In this phase, any existing coreferents are filled in in the representational structure. Besides obvious coreference phenomena such as anaphora, this phase also resolves *implicit subjects*. For example, in the phrase "I like to play chess", the agent 'I' is both the subject of 'like' and also, implicitly, of 'play chess'. While this double representation is not represented on the morphosyntactic level, it is represented as such on the representational level. The parser is only built to process such verb phrases when they are fronted by the word 'to'. In this case, a special *coreference node* is generated by the

treewalk stage. This node is ignored by the composition stage, but resolved in the coreference resolution stage, when it is replaced with a node that corresponds with the morphosyntactic subject of the clause.

6.3.2 Parsing to the interpersonal level

Though we have not developed an algorithm to translate structures from the morphosyntactic and representational levels to the interpersonal level, we envision it as a similar algorithm to the process that we have explained above. However, there are some aspects of this process that make it a bit more complicated. For example, information from the morphosyntactic and representational levels has to be combined. This makes the process a two-dimensional algorithm, so to speak.

Also, parsing to the interpersonal level introduces some extra problems such as non-literal interpretation. This may require some extra 'non-literal interpretation' algorithms. We will elaborate on one such algorithm, which deals with the interpretation of indirect speech acts, in the next section.

There are, however, some partial structures that we can already derive from lower levels. For example, we can use syntactic information such as left dislocation (e.g., "*It is the red pan* that I can't find"), which points to a topic/comment or focus/background distinction. Also, information from the phonological level can be used for this distinction (stress and tonal patterns). For the distinction between topic and comment, the contextual component plays an important role, since it contains the dialogue history so far; the part of the input utterance that has already been mentioned before in the dialogue is then likely to be the topic.

Additionally, some standard translations may be present in a lexicon, similarly to the lexicon of tree chunks that the algorithm uses. For example, the participants (speaker and addressee) can be extracted from the structures on the lower levels. Operators such as +id and +s can be extracted from indicators such as the presence of a definite article in the morphosyntactic structure.

6.4 Parsing indirect speech acts

In this section, we will present an example of one of the problems that may arise in natural language processing when we try to translate an FDG structure from one level to a higher level, and finally, to a communicative intention. As with many of these processes, we need extra information than is present in the utterance itself (or in the FDG structure) in order to make the correct translation. When using FDG for the generation of utterances, communicative intentions are already instantiated as sentence types on the interpersonal level. This means that in the process of parsing an utterance, information from the morphosyntactic level can be directly used to construct (part of) the communicative intention of the utterance.

As we have pointed out above, there is no one-to-one correspondence between communicative intentions and natural language utterances. A communicative intention can be instantiated into different sentence types. For example, a request can be instantiated as an interrogative ("Can you please stir the water?"), an imperative ("Stir the water!"), or a declarative ("You have to stir the water."). The reverse is of

course also true: a specific sentence type can originate from any number of communicative intention types, meaning very different things. For example, an interrogative can be a real question ("What time is it?"), a test question ("How much is 2 + 2?"), a request ("Can you pass the salt?"), an expression of surprise ("Are you serious?"), or a compliment ("Did you know that you look really good in that dress?").

This poses an issue in natural language parsing that can only be resolved by taking into account the context of the utterance. For example, the question: "Can you turn on the microwave?" can be, depending on the circumstances, interpreted in a direct sense as a question about someone's physical abilities or in an indirect sense as a request to turn on the microwave. Indirect speech acts are speech acts in which the communicative intention does not match with the communicative act; e.g., an interrogative that is an instantiation of a request for action, and not, as it may seem at first sight, a question: "Can you tell me what time it is?" The difference is clear to see in the expected responses. If the question "Can you turn on the microwave?" is meant as a request for action, the appropriate response is to turn on the microwave. However, when it is meant as a real question, the expected answer is either "Yes" or "No".

The relation between the linguistic form of an utterance and its underlying communicative intention is a complex matter (see e.g., Beun and Piwek [14], Levinson [88], Searle [111]). Although surface features such as wh-words (what, why, etc.), prosodics, and the grammatical form of the sentence may considerably contribute to the determination of the speech act, the relation is also determined by contextual cues, such as the participants' physical abilities. So, when a user is exploring the functionalities and capabilities of his new kitchen assistant, the question may well be interpreted as a real question, while in the context of a recipe preparation dialogue, the question should more likely be interpreted as a request for action.

When some of these contextual factors are known, such as the goal of the dialogue, we can use surface features of an utterance to elicit its communicative intention and yield a correct result in most cases. In the DenK-project [29], a computational framework was developed for the detection of the communicative intention of utterances that contain modal verbs, such as 'can', 'would' and 'must' [13]. This framework is part of a dialogue system that can conduct dialogues about an electron microscope domain. The architecture of the system reflects a natural dialogue situation where a user has direct access to the domain of discourse (by pointing or direct manipulation) or indirectly by symbolic means (communication). Here, we present an adaptation of this framework for use in a task-oriented (instruction) dialogue where the system assists the user with the preparation of a recipe.

The basic functionality of the algorithm is as follows. If an utterance by the user contains a modal verb or the verb 'want', the system uses a special parsing algorithm to parse the utterance into a so-called *verb-construction*. The verb-construction contains particular characteristics of the utterance that contribute to the interpretation of the speech act, such as the modal verb in question, the sentence type (declarative or interrogative), and the actor (system or user). Based on a combination of these characteristics, the system uses a collection of speech act assignment tables to look up the desired interpretation.

So, for instance, the example that we mentioned above, "Can you turn on the mi-

crowave?", contains the modal verb 'can', has an interrogative form, and the system is the actor of the verb phrase that follows the modal verb ('turn on the microwave') (see Beun et al. [14]). Since we assume the task-oriented context of a recipe preparation dialogue, we abandon the literal meaning of the utterance, and the system will look up the meaning of the utterance in the table for the modal verb 'can'. The utterance will be interpreted as a request, in this case a request for switching on the microwave. Below, we will treat in more detail how the system reaches this conclusion.

Although the linguistic expressions uttered by the user may take an enormous variety of surface forms, the DenK-system distinguishes only a limited number of speech acts: `RI` (request for information), `RA` (request for action) and `RI_POS` (request for information about a possibility). We have adapted the algorithm for our system and use `query-if`, `query-ref`, `request`, and `inform` as the results of the algorithm.

6.4.1 Special verbs and the verb-construction

In order to determine the communicative intention of the utterance, the system uses 'special verbs' that pertain to relations between the dialogue partners and the domain information. We distinguish between *modal verbs* (can, will, etc.), and three types of other verbs: 'tell', 'ask', and 'know'. These verbs all stand for categories of verbs that have a similar meaning; for example, 'inquire' falls under the category 'ask', and 'believe' falls under the category 'know'. Other verbs, which pertain to the domain, are called *domain verbs* (cook, cut, take, stir, turn on, etc.).

Utterances by the user are transformed into so-called verb-constructions, represented in BNF-notation: [13]

```
VERB_CONS(partner, verb1, polarity, partner, verb2, sentence_type,
domain_info)

partner ::  = S | U | nil
verb1 ::  = modal | want | nil
modal ::  = can | could | may | must | might | shall | should
| will | would
polarity ::  = pos | neg
verb2 ::  = tell | ask | know | nil
sentence_type ::  = dec | int
domain_info ::  = method | prop-if | prop-ref
```

The *partner* (*S* for System, *U* for User) occurs twice in this framework: the first time as the subject of *verb1*, the second time as the (usually implicit) subject of *verb2*. *Verb1* is the modal auxiliary or the verb *want*; if the main verb of the sentence is another verb, the value of this field will be *nil*. Furthermore, it should be noted that only when *verb1* is *want* there can be a difference between the two occurrences of *partner* in the verb-construction; compare 'I want to know...', where the subject of 'want' is the same as the (implicit) subject of 'know' with 'I want you to know...', where the subject of 'want' (which is 'I') is not the same as the subject of 'know'

(which is 'you'). The values *pos* and *neg* in the field *polarity* mean that the sentence is, respectively, positive or negative.

Verb2 is the second verb in the sentence; we use *tell*, *ask* and *know* as prototypical verbs to represent the verb categories mentioned above, but they can be replaced with others, as we have mentioned above. If no such verb occurs in the sentence (i.e. when *verb2* is either not present in the utterance or it is a domain verb such as 'cook' or 'turn on'), it is represented as *nil* in the verb-construction.

The field *sentence_type* represents the sentence type: declarative (*dec*) or interrogative (*int*). Imperative sentences are not included in the system, because they do not occur in combination with modal verbs (sentences such as "Can turn the microwave on!"). Finally, *domain_info* represents the information about the kitchen domain (the communicated content). We make a distinction between *methods*, which do not have a value and represent actions that can be performed in the domain (e.g., '(to) turn on the microwave'), and content that has a value, which we divide into propositions (*prop-if*) on the one hand, which have a truth value (e.g., 'whether we have white vinegar') and properties (*prop-ref*) on the other hand, which have a string as a value (e.g., 'how long I should stir').

Because the verb-construction was developed specifically for use in an instruction system, we assume that it only parses utterances that are directed to the system, in a single-user situation. Therefore, the word 'you' in the utterance to be parsed is always the system (S) and 'I' is always the user (U). The example that we presented earlier in this section ("Can you turn on the microwave?") is represented in a verb-construction as `VERB_CONS(S, can, pos, nil, nil, int, method)`.

Other examples of sentences with their representation in the verb-construction are:

"May I ask you whether we have white vinegar?" →
`VERB_CONS(U, may, pos, U, ask, int, prop-if)`

"I want you to turn on the microwave." →
`VERB_CONS(U, want, pos, S, nil, dec, method)`

"I want you to tell me how long I should stir." →
`VERB_CONS(U, want, pos, S, tell, dec, prop-ref)`

6.4.2 Speech act assignment for utterances with 'can'

In this subsection, we present our adapted version of the table for the modal verb 'can'. The DenK-tables for the other modal verbs [13] can be adapted in a similar manner. The resulting values in the tables are, as mentioned above, `inform`, `request`, `query-if`, and `query-ref`. We use `query` to generalize over `query-if` and `query-ref`, which we have found to correspond with, respectively, `prop-if` and `prop-ref`, as we will explain below. Additionally, `*` indicates that the utterance is inadequate, whether pragmatically, semantically or syntactically. Utterances of this type are not expected, but if they do occur, they are clearly a case for an error handling module, albeit on a

CAN		tell		ask		know		nil	
		method	prop	method	prop	method	prop	method	prop
U	dec	*	*	*	*	*	*	inform	*
U	int	request	*	request	query	*	query	query	*
S	dec	*	request	*	*	*	*	request	*
S	int	*	request	*	*	*	*	request	*

Figure 6.2: *Speech act interpretation of* `VERB_CONS(partner1, can, pos, partner2, verb2, sentence_type, domain_info)`*, where partner1 equals partner2.*

different level than the error handling that we present in Chapter 7. Since we do not treat problems in input parsing in Chapter 7, we leave this for future work.

All possible instantiations of sentences of the form '(partner) can (verb1) (domain_info)' (e.g., "You can turn on the microwave.") and 'Can (partner) (verb1) (domain_info)?' (e.g., "Can you tell me whether we have white vinegar?") are summarized in Table 6.3, which can be found at the end of this chapter. For these sentences, we use '(to) turn on the microwave' as `method`, and 'how long I should stir' as `prop-ref`. To keep the size of the table somewhat manageable, we have only specified sentences with `prop-ref` in this table; sentences with `prop-if` yield the same results, except that utterances featuring `prop-if` yield `query-if`, where utterances featuring `prop-ref` yield `query-ref`. E.g., the result of "Can I ask you how long I should stir?" (where `domain_info` is `prop-ref`) is `query-ref`, while the result of "Can I ask you whether we have vinegar?" (where `domain_info` is `prop-if`) is `query-if`.

In this table, we have indicated, for instance, that the sentence "I can know how long I should stir" yields a pragmatically inadequate interpretation in the current context. While this sentences is grammatically correct, it does not make sense in the recipe preparation context and is therefore indicated in the table as pragmatically inadequate (`*`). As an example of a pragmatically felicitous utterance, the sentence "Can you turn on the microwave?" is classified as a `request`.

The interpretations in this table were decided based on our intuitions as natural language users. The question of pragmatic correctness (and interpretation) of some sentences is more difficult than for others; e.g., the meaning of utterances such as "I can tell you to turn on the microwave" is the subject of more debate than utterances such as "Can you tell me how long I should stir?", which is quite straightforward in the current context.

We have summarized this list of results in a smaller and much more concise way in Table 6.2. The rows in the table represent the *partner* and *sentence_type*, while the columns represent *verb2* and *domain_info*. Tables of this type can be constructed for all different modal verbs.

6.5 Conclusions and future work

In this chapter, we have shown a possible way of interpreting and formulating natural language utterances that a BDI-based system could use for its communication

with a human user. For this, we use Functional Discourse Grammar, which consists of a grammatical component with four levels (interpersonal, representational, morphosyntactic, and phonological), a conceptual component, which we take to be the BDI reasoning engine of the system, and a contextual component, which contains extra information that is needed to formulate or interpret the utterance, and in our framework is part of the beliefbase of the system.

In addition to a general tactic for interpreting natural language utterances, we have focused on the interpretation of indirect speech acts that contain modal verbs, and shown that it is possible to extract their intended meaning without considering the meaning of the modal verbs themselves. Given a number of contextual factors of the interaction, such as the type of dialogue (task-oriented, information-seeking, etc.), the meaning of an indirect speech act can be based on a small number of characteristics of the utterance. This appears to be a promising way to deal with these indirect speech acts.

As we have defined in previous chapters, anything that does not fit in the dialogue model is an error. In terms of expectations, that means that any input that cannot be matched with one of the expected responses in the current state is an error. In the next chapter, we will elaborate on the detection and handling of errors.

As future work, we would like to extend the parsing algorithm that we have presented in this chapter to make translations between all levels of FDG. Also, a more detailed implementation of the contextual component is in order; however, as we have explained in this chapter, the content of the contextual component is a complicated matter and may consist of a large semantic database.

A linguistic issue that we have encountered in human-human dialogues is the problem with different uses of 'okay' by the user after the system has given an instruction [66]. In some cases, the user means that he has performed the task in question and is ready for the next instruction, but in some cases it merely means that the user has understood the instruction and will start performing the task. A possible way to handle this is to set a default meaning for 'okay' and then leave the exception cases to an error repair module.

Another issue for future work is to extend the system for the indirect interpretation of speech acts that we have presented in Section 6.4. The table for the interpretation of utterances with the modal verb 'can' that we have presented in this chapter is only one of the 11 tables that make up the total system: nine tables for the nine modal verbs, one for the verb 'want', and a table for utterances that have a domain verb as the main verb. This also involves an additional case for error handling: the utterances that are marked as incorrect (a $*$ in the tables in this chapter) should be interpreted as errors and therefore require an appropriate response from the error handling module.

As we have mentioned in the introduction of this chapter, dialogues and dialogue acts are generally multifunctional [28]. In the presented algorithms, we have focused on eliciting the main domain-related function of discourse acts (e.g., the instruction). This model can be enriched by expanding the algorithm to elicit additional communicative functions (e.g., dialogue feedback such as time and turn allocation, politeness functions, and so on). Then, it would also be interesting to classify the dialogue acts and their communicative functions in the DIT^{++} taxonomy [28]. This will allow our work to be coupled to the ISO standard based on the taxonomy, which is currently

being developed [30]. This will provide a better way to compare our models and algorithms to other work on the subject and open possibilities to link our work to other algorithms and dialogue systems.

Utterance	VERB_CONS	result
I can tell you to turn on the microwave	(U, can, pos, U, tell, dec, method)	*
I can ask you to turn on the microwave	(U, can, pos, U, ask, dec, method)	*
I can know to turn on the microwave	(U, can, pos, U, know, dec, method)	*
I can turn on the microwave	(U, can, pos, U, nil, dec, method)	inform
Can I tell you to turn on the microwave?	(U, can, pos, U, tell, int, meth)	request
Can I ask you to turn on the microwave?	(U, can, pos, U, ask, int, method)	request
Can I know to turn on the microwave?	(U, can, pos, U, know, int, method)	*
Can I turn on the microwave?	(U, can, pos, U, nil, int, method)	query-if
You can tell me to turn on the microwave	(S, can, pos, S, tell, dec, method)	*
You can ask me to turn on the microwave	(S, can, pos, S, ask, dec, method)	*
You can know to turn on the microwave	(S, can, pos, S, know, dec, method)	*
You can turn on the microwave	(S, can, pos, S, nil, dec, method)	request
Can you tell me to turn on the microwave?	(S, can, pos, S, tell, int, method)	*
Can you ask me to turn on the microwave?	(S, can, pos, S, ask, int, method)	*
Can you know to turn on the microwave?	(S, can, pos, S, know, int, method)	*
Can you turn on the microwave?	(S, can, pos, S, nil, int, method)	request
I can tell you how long I should stir	(U, can, pos, U, tell, dec, prop-ref)	*
I can ask you how long I should stir	(U, can, pos, U, ask, dec, prop-ref)	*
I can know how long I should stir	(U, can, pos, U, know, dec, prop-ref)	*
I can how long I should stir	(U, can, pos, U, nil, dec, prop-ref)	*
Can I tell you how long I should stir?	(U, can, pos, U, tell, int, prop-ref)	*
Can I ask you how long I should stir?	(U, can, pos, U, ask, int, prop-ref)	query-ref
Can I know how long I should stir?	(U, can, pos, U, know, int, prop-ref)	query-ref
Can I how long I should stir?	(U, can, pos, U, nil, int, prop-ref)	*
You can tell me how long I should stir	(S, can, pos, S, tell, dec, prop-ref)	request
You can ask me how long I should stir	(S, can, pos, S, ask, dec, prop-ref)	*
You can know how long I should stir	(S, can, pos, S, know, dec, prop-ref)	*
You can how long I should stir	(S, can, pos, S, nil, dec, prop-ref)	*
Can you tell me how long I should stir?	(S, can, pos, S, tell, int, prop-ref)	request
Can you ask me how long I should stir?	(S, can, pos, S, ask, int, prop-ref)	*
Can you know how long I should stir?	(S, can, pos, S, know, int, prop-ref)	*
Can you how long I should stir?	(S, can, pos, S, nil, int, prop-ref)	*

Figure 6.3: *Verb-constructions and indirect interpretations for positive sentences with the modal verb 'can'*

7 Error Handling

> To err is human, but to really foul things up you need a computer.
>
> *Paul Ehrlich*

The framework that we presented in Chapter 5 can be used to construct much simpler versions of the example dialogue that we presented in Chapter 1. In this chapter, we will present error handling strategies that can detect and repair errors of different types, so that dialogues in which errors occur can still be successful. First, let us again consider the example dialogue from Chapter 1:

User: I would like to prepare French toast.
System: That's not possible, we don't have bread available.
User: Can you give me another recipe for breakfast?
System: Would you like to make a poached egg?
User: Alright, I would like to make a poached egg.
System: Okay, I'll teach you how to poach an egg. First, you have to boil some water.
User: Can you turn on the stove for me?
System: I'm not able to turn on the stove.
User: Okay, I will turn on the stove. (pause) I've boiled water.
System: Add some vinegar to the water.
(User opens a red vinegar bottle)
System: That's the wrong bottle. Please add white vinegar.
User: We don't have any white vinegar.
System: We do have white vinegar. I know this because I bought it.
User: Okay, you're right. I've added vinegar to the water.
System: Crack an egg into a soup ladle.
User: I've cracked the egg into a soup ladle.
System: Stir the boiling water.
User: How long should I stir the water?
System: I don't know.
User: Okay, I've stirred the water.

System: Drop the egg into the center of the water.
User: I've dropped the egg into the water.
System: Now wait 3-4 minutes.
(3 minutes later)
User: I've waited 3-4 minutes.
System: Remove the egg from the water.
User: I've removed the egg.
System: You have now poached an egg! Will you be able to do it by yourself next time?
User: Yes.
System: Okay, we're done!

Compare this with the dialogue that results from the rules in Chapter 5, which is much simpler:

User: I would like to prepare a poached egg.
System: I will teach you to poach an egg. Please boil some water.
User: I have boiled water.
System: Please add some vinegar to the water.
User: I have added some vinegar.
(etc., until the recipe is done)
System: You have now poached an egg. Will you be able to do it by yourself the next time?
User: Yes.
System: Okay, we're done!

Apart from the first two utterances and the last three dialogue turns, this second dialogue only consists of instructions (by the system) and confirmations that the instructions have been performed (by the user). In contrast, the first dialogue contains a considerable number of deviations from this 'flawless' dialogue. All of these deviations are handled successfully and after each of these deviations, the dialogue is resumed, eventually concluding successfully after the recipe is finished.

In this chapter, we present a framework for handling errors in cooperative task-oriented dialogues. We will present a set of rules that can detect and repair errors on the speech act level in dialogues that are produced by the system that we presented in Chapter 5. Just as in Chapter 5, we take communication actions with a speech act and a communicated content as primitives of communication. This means that we abstract from speech recognition errors and other purely communicative errors. Therefore, we will assume a perfect communication channel and matching vocabularies and not handle errors in those categories. In the first section of this chapter, we will elaborate on the different types of errors and specify which errors we will treat in this chapter.

The framework that we present is tailored specifically to address errors in cooperative task-oriented dialogues. Error handling in other types of dialogues may be different [103], depending on factors such as the roles of the participants, the type of goal or goals in the interaction, and the extent of cooperativity between the participants. Since we take a cooperative system as our basis, we will assume that there are

no errors pertaining to the goals of the system. This leaves the beliefs and intentions of the user and the system as possible sources of errors.

The general idea of errors is comparable to failed felicity conditions (cf. Austin [9]) of the contributions in the dialogue, mainly focusing on two possible problems: avoiding inconsistencies in the dialogue partners' mutual beliefs, and following the dialogue model for the applicable dialogue types as presented in Chapter 3. This means that any input that is inconsistent with the system's beliefbase or with one of the next steps in the dialogue model will have to be addressed as an error. When detecting an error, the error is categorized so that the types of errors can be connected to their respective repair strategies. Then, the repair is initiated.

Section 7.1 focuses on defining errors. First, we will discuss the scope of this chapter (7.1.1), and specify which types of errors we do not treat in this thesis (7.1.2). In 7.2, we present the types of errors that occur in the dialogues that our system will conduct: inconsistencies between the dialogue partners' mutual beliefs (7.2.1) and inconsistencies between the dialogue partners' intentions (7.2.2). In Sections 7.3 and 7.4, we will respectively treat the detection and the repair of these errors. Section 7.3 is divided in subsections in which we present rules for the detection of errors in different types of input: `inform` (7.3.1), `query-if` (7.3.2), `query-ref` (7.3.3), `request` (7.3.4), and performed actions (7.3.5). Section 7.4 focuses, again, on the two types of errors: repairing belief discrepancies (7.4.1) and repairing intention discrepancies (7.4.2). In Section 7.5, we treat some ideas for the handling of multimodal errors with the help of the dialogue score that we presented earlier in this thesis. In 7.6, we will present how we validated our work, showing how the example dialogue from Chapter 1 can be produced with the rules that we present in this thesis (7.6.1), discuss the implementation of error handling techniques (7.6.2), and like in Chapter 5, some issues with 2APL that were encountered in the implementation (7.6.3). Then, we will finish this chapter with conclusions and future work in 7.7.

7.1 Definition of errors

In order to be able to detect errors, we first have to define what errors exactly are. As we have seen in Chapter 2, there are different ways in which a dialogue can be successful or unsuccessful, most notably distinguishable in success at the utterance level and success at the dialogue level. We take the dialogue model in Figure 3.6 as the norm for successful dialogues.

As we have stated in Chapter 3, an erroneous dialogue move is a move that holds at least one of the following conditions:

1. It is an unexpected contribution; that is, it is inconsistent with the addressee's intentions.

2. It signals or introduces an inconsistency in the mutual beliefs of the dialogue partners.

The error handling tactics that we present in this chapter are general and unified, and can therefore be used to construct error handling modules for any task-oriented

dialogue system that adheres to the principles that we have presented in this thesis (requirement RE3).

7.1.1 Scope of this chapter

In this subsection, we distinguish different levels of communication (and therefore, potential errors) and explain which types of errors we will cover in this thesis. Clark [40] presents four *levels of action*:

Level 4 Proposal and consideration

Level 3 Signaling and recognition, or meaning and understanding

Level 2 Presentation and identification

Level 1 Execution and attention

If everything goes right, there are four joint actions that are performed by the speaker and hearer together. On level 1, the speaker is articulating words and the hearer is attending to the speaker's voice. On level 2, the speaker presents a signal and the hearer is identifying the words of the speaker. On level 3, the speaker is performing a speech act (e.g., asking a question) to the hearer and the hearer recognizes what the speaker means. On level 4, the speaker proposes a joint project (e.g., the continuation of a conversation) and the hearer accepts that proposal.

The upper three levels can be roughly compared to Austin's [9] terminology: level 2 represents the locutionary aspect of a dialogue act, level 3 represents the illocutionary aspect and level 4 represents the perlocutionary aspect. Clark's four levels of interaction are also similar to the ones that Allwood [6] distinguishes (contact, perception, understanding, and reaction) and the ones which are distinguished in DIT [27] (attention, perception, understanding, evaluation, and execution; presumably, evaluation and execution together correspond to Clark's Level 4).

On each of these four levels, something can go wrong. If none of the levels have been executed successfully, the hearer has not noticed that the speaker has executed communicative behavior. If only level 1 is successful, the hearer has only noticed that the speaker has executed some form of communicative behavior, but not understood any words (e.g., because of a noisy channel). If only level 1 and 2 are successful, the hearer has identified the speaker's presentation correctly, but not been able to construct the full meaning of the utterance (e.g., because the speaker mentions a person or object that the hearer cannot identify). If only levels 1, 2 and 3 are successful, the hearer has understood what the speaker meant by his utterance, but is not considering accepting the joint project (e.g., the hearer is uncooperative).

Errors on higher levels can be caused by errors on lower levels. If a contribution of dialogue partner A is verbal, possible causes for any problem are that dialogue partner B misheard A's utterance, that B misconstrued (misunderstood) A's utterance or that A made a slip of the tongue.

Errors are not always bad or unexpected. There are *communicative probes* [40] that are carried out with the expectation that they may not succeed. These communicative probes are specifically tailored to one of the levels that we mentioned above. An example of a communicative probe on level 1 (Hearing) is "Is anyone home?", which

may fail if there is nobody there that hears the utterance. On level 2 (Identification), one could ask "Do you speak English?" in order to test whether the hearer is able to identify the speaker's signal, and so forth for the higher levels. These probes can be compared to giving feedback on different levels with dialogue control acts [94].

Because the coding and decoding of messages (via these different layers) may be faulty, there can be 'feedback loops' [121] on every level of communication. For each level, these feedback loops can be different. For example, on level 4, if the addressee is unwilling to take up the joint project, the initiator of the joint project may try to convince the addressee, while a feedback loop on level 3 can involve an explanation by the initiatior that clarifies the unidentified referent.

In this work, we only deal with errors on level 3 and 4, although we will treat errors on level 1 and 2 briefly later in this section. A major difference between errors on level 1 or 2 and errors on level 3 or 4 is that the former levels pertain to the communication channel and not to the BDI configuration of the participating agents. For now, we abstract from errors on level 1 and 2, assuming that the interpretation on those levels is faultless (or that errors in those levels are solved by a different process).

In the terms of Allwood [5], we assume a correct *S-interpretation* (where S stands for symbol) of an utterance. S-interpretation does not take into account any belief or stand on the factual status of the information, but solely the content of the information itself. In other words, the hearer S-interprets a symbol if he (1) apprehends the information, and (2) apprehends the information as if it were conventionally tied to a symbol. Any further interpretation of the information represented by a signal (conclusion-drawing, inference, emotional states, attitudes, behavior) are not included in the S-interpretation of the signal.

7.1.2 Errors that fall outside the scope of this thesis

In this subsection, we will briefly specify which types of errors in dialogues we do not handle in our system.

Speech recognition errors

In speech recognition errors, we can distinguish two types of errors: *unhearings* and *mishearings* (misunderstandings). An example of an unhearing is:

A: "It costs five."
B: "How much did you say?"
A: "Five dollars."
B: "I'll take it." (Example from Goffman [62], p. 295)

An example of a mishearing:

A: "Have you ever had a history of cardiac arrest in your family?"
B: "We never had no trouble with the police."
A: "No. Did you have any heart trouble in your family?"
B: "Oh, that. Not that I know of." (Example from Goffman [62], p. 295)

An important difference between these two cases is who detects the error; in the first case (unhearing), the addressee detects the errors, while in the second case (mishearing/misunderstanding) it is the speaker who signals that something has gone awry, since he gets an incorrect response from the addressee [62]. The latter case is what Clark [40] calls 'third-turn repair': the first turn is the question, the second turn is the answer (which signals an incorrect construal), and the third turn is the repair.

Speech recognition errors are more likely to occur in noisy environments [2]. The situation gets even more difficult if we consider spontaneously spoken speech [82] [57], since colloquial language use by human speakers is often grammatically ill-formed in many ways. We do not deal with errors in speech recognition or in the parsing of spontaneously spoken speech in our system, since we take speech acts as our interaction language. For more about recognition-based errors, see e.g. Bourguet [17].

Turunen and Hakulinen [125] state that the detection of semantic errors is quite straightforward; if an error is detected, but the recognition confidence scores are good (above a certain treshold), then the error is probably semantic. Since we abstract from recognition errors, this way of narrowing down the detection of errors is not applicable in our error handling system, although it is a good starting point for future work if we want to implement an additional error handling module for speech recognition errors.

Error prevention and self-repair

An agent can attempt to prevent errors as much as possible by employing certain dialogue tactics. Grounding information (regularly giving feedback) will help with the detection of (potential) errors in an early stage. Thorough studies of feedback can be found in Allwood's work (e.g., on the topics of linguistic feedback [8] and gestural feedback [7]). Adjusting oneself to the dialogue partner syntactically and semantically is another way of preventing errors. Misunderstandings can also be avoided by a proactive approach in the dialogue; e.g., by asking specific questions [73]. On the other hand, to maintain a natural interaction, the agent must alternate between this active approach and a more passive approach [109].

Another way to prevent errors is to try to find out the dialogue partner's lack of understanding in an early phase. Breazeal [20] mentions that there are various *affective cues* that humans use in order to signal their understanding or misunderstanding, such as facial expressions, vocal prosody, and body posture. Breazeal contrasts these with *social cues* such as gaze direction and feedback gestures like nods and eyebrow movements. These cues can be used to infer the mental state of the interaction partner and can therefore regulate the "rate and content of information transferred".

Beun and Van Eijk [12] introduce the concept of *indirect mismatches* for preventing errors. When the system has a reasonable expectation that the user would adopt incorrect beliefs after receiving a planned utterance from the system, the system changes the utterance in order to ensure that the user would not infer any incorrect information from it. For example, if the user asks "Is this toxin poisonous?", the system can reasonably expect that if it replies "Yes", the user would infer that there are some toxins that are not poisonous. Instead, the system should reply "Yes, because toxins are always poisonous" or something along those lines, in order to cancel the implicature that the user could otherwise make. In contrast with these indirect mis-

matches, *direct mismatches* can be detected from the semantic content of the user's message. We will elaborate on direct mismatches later in this chapter.

Agents can also evaluate possible actions with respect to the consequences that they are expected to have in the current situation. Breazeal [20] states that this is a useful strategy; if an agent does not have such a 'sense of value', it cannot distinguish whether its actions are socially acceptable, especially when the social acceptability of action varies in different contexts. For example, calling someone by their first name is not socially acceptable in some formal situations – some situations are so formal that calling someone by their first name is unacceptable, even if it is someone that one would usually call by their first name; a PhD defense is a good example of such a situation. This tactic could be employed to prevent errors that the agent expects to occur when performing a certain action.

In human-human dialogues, errors can be prevented with dialogue control acts. For example, in one of the dialogues in our corpus, an instruction dialogue where A instructed B to build a LEGO® building from an example that only A had access to, a lot of errors occurred. After a few instructions, B asked A to adapt his way of giving instructions: "Just start by saying which block I need, because that's what I listen to first." For now, we leave error prevention for future work.

Besides error prevention, another type of errors that we do not treat in this thesis is *self-repair*. This occurs during the utterance of a dialogue contribution and may occur in situations where, for example, the speaker detects a slip of the tongue while speaking, or if the situation changes during the course of the utterance. Detection (by the hearer) of self-repair can also be helpful in order to prevent misunderstandings. Shriberg et al. [112] have found that using a combination of pattern matching, syntactic and semantic analysis and acoustics, repairs within an input sentence can be detected quite well.

Uncooperative behavior

Allwood [5] states that any failure to cooperate, whether intentionally or unintentionally, is a misunderstanding. When a receiver *intentionally* ignores the communicative intention of the sender that is more than the S-interpretation of the utterance, he is guilty of *literalism*: the receiver fails to interpret the sender's utterance pragmatically. For example, when a speaker utters "It's cold", meaning to get the addressee to close the window, the addressee is guilty of literalism when he replies "It surely is" instead of closing the window.

Allwood states that there is also *unintentional literalism*, which appears to contradict his definition of literalism as an intentional failure to cooperate. Unintentional literalism is not caused by unwillingness to cooperate, but by false inferences that are made by the addressee. We assume that all dialogue partners are cooperative and will therefore not intentionally fail to cooperate. Therefore, we will assume that all errors are of the type of false inferences.

Quantitative error research

Steidl et al. [113] predict success in dialogue steps and whole dialogues in a quantitative manner, based on some linguistic features. Walker et al. [130] also apply a

quantitative method in error handling, using a training database of dialogues with human users to learn the detection of problems in dialogues. While these are interesting alternative approaches, we will only treat the handling of errors in a qualitative way, in order to have a close coupling to our basic dialogue system as presented in Chapter 5.

7.2 Examples of the two types of errors

As we have stated in Chapter 3, we define an erroneous dialogue move as a move that holds at least one of the following conditions:

1. It is an unexpected contribution; that is, it is inconsistent with the addressee's intentions.
2. It signals or introduces an inconsistency in the mutual beliefs of the dialogue partners.

These conditions roughly correspond to Clark's fourth and third levels of action that we introduced earlier in this chapter: a contribution that is inconsistent with the addressee's intentions can be called a failure of proposal and consideration (level 4), while an inconsistency in mutual beliefs is roughly on the same level as signaling and recognition (level 3).

In this section, we present some examples of errors in human-human dialogues. As we have stated above, we treat two different types of errors: first of all, inconsistencies between the mutual beliefs of the dialogue participants (requirement RE5), and secondly, inconsistencies between the intentions of the dialogue partners. Since we developed our system according to the BDI architecture, there are three parts of the system where errors can occur: in the beliefs, in the goals, and in the intentions of the dialogue partners. Because our system functions in a cooperative setting, we assume that the dialogue partners have the same (joint) goals. This means that there can only be errors in the beliefs and in the intentions of the dialogue partners. In this section, we explore both of these error types based on occurrences in human-human dialogues. The goal of this categorization is to connect the types of errors to their respective repair strategies. In the next section, we will treat the detection of errors, and in Section 7.4, we will treat repair tactics for the errors.

For every dialogue contribution by the speaker (whether it is an external action that is being observed or a linguistic contribution), there are two possible problems: it does not fit in the dialogue models, or it is inconsistent with the addressee' beliefs. Because we can assume that the dialogue partners only say things that they hold to be true (cf. Grice's maxim of quality [69]), the latter means that there is an inconsistency between the dialogue partners' beliefs, which should be resolved as soon as possible, to avoid more errors in the future. Since the system can only conduct dialogues according to its dialogue model, it is also important that the actual dialogue deviates as little as possible from the model. So, to avoid dialogue breakdown if the dialogue partner's contribution does not fit in the dialogue models, the system should first address the contribution (e.g., if it is a question, answer it) and then bring the dialogue back to the dialogue model (Requirement RE2).

7.2.1 Inconsistencies between the dialogue partners' mutual beliefs

The belief inconsistencies that we treat in this thesis are of the type where the speaker's utterance contains or presupposes information that the hearer believes to be untrue. If the speaker's utterance presupposes certain information that the hearer does not believe (and neither its negation), we call this a *belief gap*. There are a number of possible approaches to take when encountering a belief gap. For example, the hearer may not accept the utterance (since believing the presupposition is a requirement for accepting it), or alternatively, simply accept the presupposition and add it to his beliefs. Although the case of a belief gap is an interesting one, we take a pragmatic approach and do not treat belief gaps deeply in this thesis. If the dialogue partner's utterance contains presuppositions that are not present in the agent's beliefs (and neither are their negations), the agent will simply add this information to its beliefs.

We will first give two examples of cases where the mutual beliefs of the participants do not match.

A: Hello, could you tell me at what time the AF 914 from Cairo will land?
[...]
B: Well, that's a flight from Paris...
A: Yes, but it's coming...
B: So they probably have to transfer there.

A: Something with vegetables, something with meat.
B: Lasagna with chicken and tomatoes?
A: That's not meat.
B: But chicken is meat.
A: No, that's chicken.

In both of these examples, an inconsistency is detected between the beliefs of A and B: whether AF 914 is a flight from Cairo (A's beliefs) or Paris (B's beliefs), and whether chicken is meat (B's beliefs) or not (A's beliefs). These two cases differ in the type of information in which an inconsistency is detected. In the first case, the flight number (AF 914) does not match the origin of the flight (Cairo, should be Paris), so it is factual information that A has incorrect beliefs about. In the second case, the inconsistency is on an ontological level: B believes that chicken is a type of meat, with which A disagrees.

The repairs that are being employed in these two examples are quite straightforward. In the first example, B detects the error and states the correct information ("that's a flight from Paris"). When A does not fully comply with this correction, B gives an explanation ("they probably have to transfer there") that is meant to convince A that B is right. In the second example, B also simply states information that he believes to be true that contradicts with A's utterance ("that's not meat"). Just like in the previous case, A also does not comply with the correction and instead states his own beliefs again ("chicken is meat"). B persists in his correction ("no, that's chicken").

It should be noted that the handling of this type of error is not necessarily about the factual truth of the conflicting information. It is simply a case of all participants reaching agreement on one or the other.

Another example of this type of error is when dialogue partner A gives dialogue partner B an instruction that B is not capable of performing. In this case, apparently A believed that B could perform the action (otherwise he would not have given B the instruction), with which B disagrees. Being an expert on his own capabilities, B should then inform A of his lack of capability to perform the action in question.

Another instantiation of this type of error is the following example: a situation where dialogue partner B performs an incorrect action (i.e. a different action than dialogue partner A intended him to perform). The cause of this error is that B apparently believes that he should perform another action.

> A: And that one goes in the middle on top of it. Only on those two pins. No, a bit to that side. No to– only on that edge. To the left. No, keep it in that orientation.

In the next example, A gives B an instruction that B is trying to follow, but B does not immediately perform the correct action (unfortunately we do not have any information about B's actions during this dialogue). As a repair, A keeps giving B new instructions during the performance of the action that are meant to get B to perform the correct action, and B keeps checking whether the action he is performing is the correct one.

> B: Like this?
> A: No, eh north–south.
> B: Like this?
> A: No, eh in north–south orientation.
> B: Like this?
> A: Yes, but on the west side.
> B: Like this?
> A: Yes.

It should be noted that a wrong action is not always due to incorrect beliefs. Norman [96] presents a classification of *action slips*: mistakes in actions that are not due to errors in beliefs, but that are purely accidental. These action slips can be sorted into three categories: errors in the formation of the intention, faulty activation of schemas, and faulty triggering. For example, action schemas are triggered by certain conditions and intentions; some variable may be incorrectly instantiated, which can lead to action slips such as putting a glass in the fridge and the milk in the cupboard. We do not treat these types of errors directly, but if the system observes the user performing an incorrect action, it could be that this was an action slip instead of an intentional action. The error can nevertheless be repaired in the same manner, as the repair will alert the user that he is performing an incorrect action (even though it is not caused by an error in the user's beliefs).

7.2.2 Inconsistencies between the dialogue partners' intentions

If the user performs a dialogue action that does not fit in the system's dialogue model at that point of the dialogue, the user apparently has different intentions than the system expected him to have. In this case, the system should attempt to get the dialogue back to the model (and thus to its intentions) as quickly as possible. After all, the system can only conduct constructive dialogues according to its own plans.

To illustrate this type of error, we will first give two examples.

> A: I would like to get an impression of the arrival times of airplaines from Dublin to Schiphol, could you give me some arrival times or not?
> B: Is that about airplanes that are going, eh, today?
> A: Airplanes from Dublin, at what time they're arriving in Amsterdam.
> B: Today?
> A: Eh, isn't that every day?
> B: Yes, but I have a lot from Dublin and sometimes there are delays and so on, so I would like to know whether you want to know the arrival times of today or just in general.
> A: Well it's about eh eh the day after tomorrow that's what I'm more interested in.

> A: What do you want to eat?
> B: Pasta pesto!
> A: Do you want red pesto?
> B: I want pasta!
> A: You want pasta with pesto. There's red pesto, there's green pesto...
> B: Eh, I would like green pesto with that.

In the first example, A does not answer B's question ("Is that about planes that are going today?"), but instead repeats part of his own question. A only answers B's question after B has explained why it is necessary for A to give him this information. In the second example, B also does not answer A's question ("Do you want red pesto?") until after B explains why he needs this information.

In the next example, B needs additional information from A in order to perform the action correctly.

> A: And on top of that, exactly on top, goes another green one.
> B: Also a square one?
> A: Yes, also a square one.

Since B apparently had multiple options for the object in question ("another green one"), he asked A for more specific information; apparently, B had a green square object and a green non-square, and therefore, expected to get sufficient information by asking this question ("Also a square one?").

Dialogue partners may also disagree about the state of the dialogue that they think they are in. In the next example, B presumes that the dialogue is finished after A's first utterance and proceeds to close the dialogue ("thank you, bye bye"), while A meant to give B more information. Fortunately, B has only signaled a close of the dialogue

and has not actually put down the phone, so A can give the additional information that he meant to give ("But you have to transfer in Dordrecht and Leiden").

> A: Those leave from Tilburg to Schiphol at two minutes past the whole hour and two minutes past the half hour.
> B: Yes thank you, bye bye.
> A: But you have to transfer in Dordrecht and Leiden.
> B: Oh yes, thank you, bye bye.

In our error handling framework, we do not analyze what the exact cause is of this type of error (too much information, too little information, ...), but instead, we will attempt to behave cooperatively by answering questions and fulfilling requests from the user at any point in the dialogue wherever possible, and additionally, alert the user of the path of the dialogue that the system expected him to follow (e.g., "I was expecting you to [...]").

7.3 Detecting errors

In this section, we will present our algorithm for the detection of errors. In the next section, we will elaborate on how we repair the errors.

As we have shown in the system architecture in Chapter 4, the error detection process in our system can be seen as a separate agent (i.e. separate reasoning engine) from the basic system that we presented in Chapter 5, and functions as a sort of filter or monitor that checks all incoming messages for certain properties which signal that there is a problem with the message in question. We can see this as similar to checking the felicity conditions [9] for all input from the user; the conditions that we present in this section are similar to felicity conditions, meaning that if they are not satisfied, a repair subdialogue should be initiated.

Because the detection rules work in this manner, we present our error detection rules sorted by type of input. In our system, the different speech acts are `inform`, `propose`, `query-if`, `query-ref`, and `request`. However, because `propose` is only used by the system, we do not treat it in this section. Additionally, we do treat the observation of an action a, `observe(done(a))`. In this section, we will treat the conditions that signal errors for each type of input, and activate the appropriate error repair tactic with the content of the input in which the error was detected as a parameter. In the next section, we will treat the different repair tactics ordered in the two categories of intention discrepancies and belief discrepancies (requirement RE1).

Belief discrepancies and deviations from the dialogue model may occur simultaneously. If this happens, the belief discrepancy should always be repaired first, and then the deviation from the dialogue model. Morante [94] shows that issues with (tentative) mutual beliefs always have to be handled first, because otherwise they are added to the mutual beliefs of the participants and believed to be common ground.

In the explanation of the possible errors that follows, the dialogue is between participants A and B, and A has just made a contribution to the dialogue. If B detects one of the problems stated below, the detected error will have to be resolved. We will treat these repairs in Section 7.4.

We separate the detection and repair of errors by having the error detection agent call PR-rules (plan revision rules) that stand for repair tactics. We do this in order to make it easier to re-use repair tactics if necessary. When an error is detected, the rest of the dialogue is temporarily suspended until the error is repaired (requirement RE4).

7.3.1 Detecting errors in `inform`

When A's contribution is `message(A, inform, p)`, there is a problem if `p` is inconsistent with B's beliefs about the mutual beliefs of A and B. The mutual beliefs of the participants should be consistent with their private beliefs, and if A says something and it is not attacked by either of the participants, it will become mutually believed by the participants. Therefore, to avoid inconsistencies between B's private beliefs and the mutual beliefs of A and B, if A informs B that `p`, and `p` is inconsistent with B's beliefbase, B should resolve this inconsistency immediately.

Presuppositions are partially taken into account, since belief inconsistencies are already detected if only part of p is inconsistent with the system's beliefbase. So, for example, consider the utterance "I saw Lucy's boyfriend yesterday." This utterance presupposes the proposition 'Lucy has a boyfriend'. If A says this utterance to B, while B believes that Lucy does not have a boyfriend, the communicated content of the utterance (`saw(A, X)` $\wedge$ `relationship(Lucy, X)`) is inconsistent with B's beliefs about Lucy's relationship status (`not relationship(Lucy, X)`).

If B did not know that Lucy has a boyfriend, the utterance is still felicitous, even though the presupposition was not (yet) in B's beliefs about the mutual beliefs of the dialogue partners and thus, A's beliefs about the mutual beliefs of the dialogue partners were incorrect. However, since B did not believe that Lucy does *not* have a boyfriend, this new information is not inconsistent with his private and mutual beliefs and therefore, B can can simply add it to his private and mutual beliefs after hearing A's utterance.

We do not treat the detection of presuppositions from utterances in this thesis, but we refer to Beun and Van Eijk [12], where the notion of presuppositions is compared to sequentiality. In the *interpretation phase*, the presuppositions are extracted from the utterance. We take `presup(p)` to be the set of presuppositions for utterance `p`.

We introduce some rules in our system that watch out for input that matches the conditions that we have just mentioned.

In the first two rules (21 and 22), belief inconsistencies are detected. If the system disagrees with the presuppositions of `p` (rule 21), this has to be addressed first, before the truth of `p` itself is debated (rule 22). After all, the system is not able to adequately interpret `p` if it does not agree with `p`'s presuppositions. For example, it is impossible to disagree with the utterance "I saw Lucy's boyfriend yesterday" if one does not believe that Lucy has a boyfriend; the referent *Lucy's boyfriend* can therefore not be instantiated and it is impossible to construct a proper representation of the utterance. Therefore, the order of the rules is such that the repair tactic for disagreement with presuppositions is called first, before the repair tactic for disagreement with the utterance itself. In both cases, the repair `untrue` is called, with the debated information as its argument.

```
(21)  message(U, inform, p) <- B(not presup(p)) |
      repair(presup(p), untrue)

(22)  message(U, inform, p) <- B(not p) |
      repair(p, untrue)

(23)  message(U, inform, p) <- not I(message(U, inform, _)) |
      repair(p, unexpected-inform)
```

Figure 7.1: *Detecting errors in* `inform`

The last rule (rule 23) is activated if it was not the system's expectation to receive an `inform` message from the user. This does not necessarily mean that the dialogue move itself is erroneous, but because the system can only conduct dialogues according to the dialogue models, we have introduced this repair rule to ensure that the user is aware of the reaction that the system was expecting of him.

An example of rule 21 is present in our example dialogue. The user says `send(S, inform, not available(white_vinegar))` ("We don't have any white vinegar"), which will arrive in the system's eventbase as `message(U, inform, not available(white_vinegar))`. The propositional content of the utterance, `not available(white_vinegar)`, is clearly inconsistent with the system's belief that white vinegar is available (`available(white_vinegar)`). In this case, the rule `repair(not available(white_vinegar), untrue)` is called.

7.3.2 Detecting errors in `query-if`

When A's contribution is `message(A, query-if, p)`, there is a problem if B does not know whether `p` is true or not (rule 24). Because we do not have a closed world assumption, the absence of `p` in B's Beliefbase does not imply that `not p` is true. In other words, there is a problem if neither `p`, nor `not p` is in B's beliefbase.

There is also a problem if B believes that the presuppositions of `p` are untrue (rule 25). This is the case when, for example, the question "Is Lucy visiting her boyfriend today?" is asked, and the addressee does not believe that Lucy has a boyfriend. Then, the same repair as in the previous subsection is called, namely `repair(presup(p), untrue)`.

For example, the user may ask "Do we have toilet paper?", which will arrive in the system's eventbase as `message(U, query-if, available(toilet_paper))`. Since neither `available(toilet_paper)` nor `not available(toilet_paper)` are in the beliefbase, the system has no information about the availability of toilet paper and should therefore inform the user of this.

As we have mentioned in Chapter 5, the only exception to this rule is in the case where `p` is `capable(X, Y)`. For this type of belief, we do have a closed world assumption: the lack of belief that `capable(X, Y)` means that X is not capable of

```
(24) message(U, query-if, p) <- not B(p) and not B(not p) and
     not (p = capable(_, _)) |
     repair(p, no-information)

(25) message(U, query-if, p) <- B(not(presup(p))) |
     repair(presup(p), untrue)
```

Figure 7.2: *Detecting errors in* `query-if`

doing Y. We have already presented rules for this in Chapter 5, but for completeness we also add the condition that `p` is not equal to `capable(X, Y)` in this error detection rule.

It should be noted that in order to be cooperative, the system should always answer questions from the user (requirement RE6). This means that for messages of the type `query-if` and `query-ref` (which we will treat in the following subsection), errors of the type of intention discrepancies do not exist. These questions should always be answered by the system, if possible, even if they appear to be irrelevant.

7.3.3 Detecting errors in `query-ref`

When A's contribution is `message(A, query-ref, info(a, Property, G))`, there is a problem if the system does not have the requested information (rule 26). As we have seen in Chapter 5, the proposition `info(a, Property, G)` refers to a belief of the form `Property(G, a, X)`, where a property of action `a` with respect to goal `G` has value `X`; for example, `duration(poach_egg, stir_water, briefly)` refers to the fact that the water should be stirred briefly if the current joint goal is to poach an egg.

If the system does not have a statement in the beliefbase that matches the requested `Property(G, A, X)`, the system should inform the user of this. Additionally, when the user disagrees with the presuppositions of the content of the question (rule 27), this should also be repaired. For example, "Where does Lucy's boyfriend live?" when the addressee does not believe that Lucy has a boyfriend.

```
(26) message(U, query-ref, info(A, Property, G)) <-
     not B(Property(G, A, _)) |
     repair(info(A, Property, G), no-information)
(27) message(U, query-ref, p) <- B(not(presup(p))) |
     repair(presup(p), untrue)
```

Figure 7.3: *Detecting errors in* `query-ref`

For example, the user may ask `send(S, query-ref, duration(stir_water))` ("How long should I stir the water?"), which will arrive in the system's eventbase as `message(U, query-ref, duration(stir_water))`. If there is no statement that matches `duration(poach_egg, stir_water, _)` in the Beliefbase, the system cannot answer the user's question.

Just like in the previous subsection w.r.t. messages of the type `query-if`, the system should always be cooperative in answering questions. Therefore, the second type of error, the intention discrepancy, also does not hold for messages of the type `query-ref`.

7.3.4 Detecting errors in `request`

A precondition of requesting an action is that the speaker believes that the addressee can perform the requested action [44]. So, if A requests action `a` from B, but B is not capable of performing this action (rule 28), there is an error in A's beliefs. In other words, when A's contribution is `message(A, request, next-action(a))`, there is a problem if `capable(B, a)` is not in B's beliefbase.

Another precondition of requesting an action is that the speaker believes that the action is practically possible at this time. If the addressee does not believe this to be true (if `checked-opportunity(a)` for the requested action `a` is not in the addressee's beliefbase) (rule 29), the action cannot be performed. This is the case when, for example, the speaker requests the addressee to take an egg from the fridge when the addressee believes that there are no eggs available. Then, the action of taking an egg from the fridge is of course not possible.

If the task is not possible at this time and/or if the system is not capable of performing the task, an error subdialogue is initiated, in which the system should inform the user of this problem. The order in which the rules are presented is important: if both the opportunity check and the capability check will fail, the system will first encounter the capability check (the rule that has `not capable(S, T)` as a condition) and inform the user that it is not capable of performing the task. We have chosen this order because the system's beliefs about its own capabilities are always correct, while there may be errors in its beliefs about the domain and therefore about the opportunity check. Therefore, stating of incapability to perform the requested task is a stronger reason why it will not perform the task.

A special case is when the user requests the goal to be aborted (rule 31). In this case, the system first checks if this is really the user's intention, and if it is, it removes the goal from its goalbase. Otherwise, it reminds the user of the next dialogue state that it was expecting.

In the example dialogue, the user asks "Can you turn on the stove?", which will arrive in the system's eventbase as `message(U, request, next-action(S, turn_on_heat_source))`. Since the system believes that it is not capable of turning on the stove (it does not believe `capable(S, turn_on_heat_source)`), this is an erroneous request and an error handling subdialogue has to be initiated.

Also, if the system believes that the action is not possible under the current circumstances, it should inform the user of this. Recall that the system has already checked whether its plan for the current goal is possible when he adopted the goal.

```
(28) message(U, request, next-action(S, a)) <- not B(capable(S, a)) |
     repair(a, action-incapable)

(29) message(U, request, next-action(S, a)) <- not
     B(checked-opportunity(a)) |
     repair(a, action-impossible)

(30) Beliefbase:
     inplan(T, G) :-
     makenestedlist(G, Gcomplete),
     flatten(Gcomplete, Gflat),
     member(T, Gflat).

(31) message(U, request, abort(G)) <- I(next_dialogue_state) |
     send(U, query-if, abort(G));
     if message(U, inform, abort(G)) then send(U, inform, accept);
     dropgoal(jointgoal(G))
     else send(U, inform, expected(next_dialogue_state))
```

Figure 7.4: *Detecting errors in* `request`

Therefore, an error of this type only happens if the user is executing a different plan than the system has, or if the circumstances have changed after the system calculated the opportunity check when it adopted the joint goal.

As for intention discrepancies, there are three different types of unexpected input in the case of `request`: first of all, when `a` is not in the system's plan for the current goal, secondly, when the action is in the current plan, but has already been performed, and thirdly, when the system was not expecting the user to request an action from the system. In each of these cases, the request can be seen as an unexpected request. However, since we want to approach error handling from a practical viewpoint and the system should be cooperative w.r.t. the user's goals, the system should simply perform the requested actions.

7.3.5 Detecting errors in performed actions

When A observes that task T has just been performed (`done(T)`) (rule 32), there is a problem if B did not intend A to perform `T`; in other words, if `T` is not in B's plan for the current joint goal `G`.

In the example dialogue, the user opens a red vinegar bottle instead of a white vinegar bottle. Thus, the system observes `done(open_red_vinegar_bottle)`, which is a subtask of the task `add_red_vinegar`. Since that action is not in the plan for the goal `poach_egg`, it is a wrong action and the system will address it as such.

If the system was not expecting the user to perform an action (`not`

```
(32)  observe(done(T)) <- jointgoal(G) and not inplan(T, G) |
      repair(T, wrongaction)

(33)  observe(done(T)) <- not I(observe(done(_)) |
      repair(T, unexpected-action)
```

Figure 7.5: *Detecting errors in performed actions*

`I(observe(done(_)))`) (rule 33), this does not necessarily mean that an incorrect action was performed; however, to make sure that the system and the user are still working towards the same joint goal, the system will still address the error.

7.4 Repairing errors

We have categorized the errors so that we can apply different repair strategies to errors in different categories: repairing a belief error requires a different strategy than addressing an an intention discrepancy.

7.4.1 Repairing belief discrepancies

The goal of repairing belief descrepancies is to keep the common ground of the dialogue partners consistent. Because common ground is "a sine qua non for everything that we do with others" (Clark [40]), "the repair of mismatches [...] is an essential part of the process that we call cooperative conversation" (Beun and Van Eijk [12]). As we have mentioned earlier, we assume that agents only say things that they believe to be true.

As we have seen in the previous section, there are five types of belief discrepancy repairs that we distinguish: untrue, no-information, action-incapable, action-impossible, and wrongaction.

Untrue

Generally, when two dialogue partners disagree about something, they can choose to resolve this difference or to simply take note of each other's beliefs. While it is not necessarily a problem for agents to disagree, the common ground of the agents should be consistent with their private beliefs, at least in a cooperative setting. When an agent has said something, we assume it to be mutually believed, unless an objection is made immediately (as we have stated in Chapter 2), and therefore any inconsistencies between an utterance and an agent's private beliefs should be solved immediately, to prevent the other agent from thinking that it is mutually believed by all participants.

According to McCoy [89] [90], a response to a misconception consists of three parts: first of all, a denial of the incorrect information, secondly, a statement of the

correct information, and thirdly, a justification for the denial and the correct response given.

```
(34) repair(capable(U, T), untrue) <- T = [Tfirst|Tail] |
     RemoveBelief(capable(U, T));
     if next-action(U, T) then RemoveIntention(next-action(U, T));
     AddIntention(next-action(U, [Tfirst|Tail]))
     else skip
(35) repair(p, untrue) <- reason(not p, X)|
     send(U, inform, not p),
     send(U, inform, reason(not p, X))
     if message(U, inform, accept) then skip
     else RemoveBelief(not p); AddBelief(p)
```

Figure 7.6: *Repairing errors of the type* `untrue`

When the user states that he is not capable of performing a task `T` (rule 34), the system simply removes this belief (`capable(U, T)`) from its beliefbase, since we assume that the participants always have accurate beliefs about their own capabilities. The user may have simply informed the user of his incapability to perform a certain task that is relevant to the current joint goal `G`, in which case nothing else needs to be done, but if the user informs the system that he is not able to perform the task that the system has just requested him to do, the system needs to give the user lower-level instructions. We do this by checking whether `T` is the next action that the user should perform, and if this is the case, we remove `T` from the list and replace it with the subtasks of `T`.

W.r.t. the notation of this rule, it should be noted that `T = [Tfirst|Tail]` does not mean that `T` is identical to the list `[Tfirst|Tail]`, but instead, the statement `T = [Tfirst|Tail]` is in the beliefbase, where `T` is the name of the recipe and the list is the list of instructions, of which `Tfirst` is the first instruction and `Tail` the remainder of the list.

When the system believes that the user has incorrect beliefs and the subject of disagreement `p` is *not* a capability (rule 35), the system simply states his beliefs about the matter in question. Then, the system searches its beliefbase for a reason why it holds `p` to be false, and then, according to McCoy's proposed strategy above, informs the user of its belief that `not p`, and the reason for this (`reason(not p, X)`).

This can have three different results: the user convinces the system of his beliefs, the system convinces the user of his beliefs, or they 'agree to disagree': both keep their own beliefs and add a new belief, stating that the dialogue partner believes differently (e.g., `believes(user, is(chicken, type_of(meat)))`, while keeping their own belief that `not is(chicken, type_of(meat))`). As for the first two options: because the user has the end responsibility for the preparation of the recipe, the system will only give its justifications for its beliefs and then wait for the user to accept or reject them. If the user rejects them, the system will adjust its beliefs accordingly; if

the user accepts them, the system expects the user to adjust his own beliefs. At this time, we have chosen to only implement the first two options; we leave the 'agree to disagree' option for future work (see e.g., Lebbink [83]).

This type of error occurs in the example dialogue:

User: "We dont have any white vinegar."
System: "We do have white vinegar. I know this because I bought it."
User: "Okay, you're right."

In his first utterance, the user says something that the system believes to be false (`not available(white_vinegar)`), since the system believes `available(white_vinegar)`. This means that a repair is activated, and the system states his beliefs about the matter and the reason for this [89]. The user accepts this and changes his beliefs about the matter, nothing changes in the system's beliefs, and the dialogue can continue as planned.

No-information

If the user asks the system for information that the system does not have, the system simply informs the user of this.

```
(36)  repair(p, no-information) <- true |
      send(U, inform, no-information(p))
```

Figure 7.7: *Repairing errors of the type* `no-information`

In this case, there is no fixed expected reaction from the user; he can try to find the answer elsewhere, or simply carry on with the task without the information he requested.

This type of error also occurs in the example dialogue:

System: "Stir the boiling water."
User: "How long should I stir the water?"
System: "I don't know."
User: "Okay."

When the user asks the system how long the water should be stirred, the system searches for information of the type `duration(poach_egg, stir_water, _)`. Since this information is not present in the beliefbase, this repair is called and the system replies with the fact that it has no information available about the requested property. The user accepts this and carries on with the task.

Action-incapable

When the user requests the system to perform a task that the system cannot perform, the system also simply informs the user of this.

```
(37)  repair(T, action-incapable) <- true |
      send(U, inform, not capable(S, T))
```

Figure 7.8: *Repairing errors of the type* `action-incapable`

This repair, similarly to the first case for `repair(capable(U, T), untrue)` above, is based on the assumption that all of the participants have accurate beliefs about their own capabilities, which means that there is no discussion possible about this type of belief. In this case, it is up to the user whether he wants to perform task `T` himself or abort the joint goal altogether.

User: "Can you turn on the stove for me?"
System: "I'm not able to turn on the stove."
User: "Okay."

In the example dialogue, the user requests the system to turn on the stove, which unfortunately it is not capable of doing. The system informs the user of this and the user accepts this information. (Contrary to the *untrue*-repair, in this case there is no reason for discussion about the truth of this statement or to state the reason for their beliefs about the matter, as all participants have accurate beliefs about their own capabilities.)

Action-impossible

When the user requests an action of the system that is not possible under the current circumstances, the system informs the user of this.

```
(38)  repair(T, action-impossible) <- true |
      send(U, inform, not possible(T))
```

Figure 7.9: *Repairing errors of the type* `action-impossible`

In this case, it is also up to the user whether he wants to find a different way to achieve task `T` or one of its supergoals, or abort the joint goal altogether. Alternatively, if the user wishes to disagree with the system about the possibility to perform `T`, he can simply state why he thinks `T` is possible, in which case the repair subdialogue

`Untrue` will be initiated. This means, as we have seen above, that the user may convince the system that he is right (and then the system will alter his beliefs) or vice versa.

Wrongaction

When the user performs an incorrect action, the system tells him that this is not the correct action and states what action it was expecting him to perform. The user may give an explanation for the action that he performed; after all, the user may have a different plan for the current (sub)goal, or the action may be unrelated to the goal, and it is therefore preliminary to condone the action as being incorrect.

```
(39)  repair(T, wrongaction) <- jointgoal(G) |
      send(U, inform, not next-action(U, T)),
      send(U, request, next-action(U, planned-action)),
      if message(S, inform, unrelated(T)) then skip
      else if message(S, inform, accept) then skip
      else if message(S, inform, inplan(T, Tsub)) then
      remove(G, Tsub, NewG); checked-done(Tsub)?; prepare(NewG)
```

Figure 7.10: *Repairing errors of the type* `wrongaction`

In general, there are several different reasons why the user may have performed the wrong action. First of all, the user may have performed an action that is not related to the current joint goal, in which case the action was not actually an incorrect action, it was just not planned by the system.

Secondly, the user may have misunderstood the system's previous instruction or made an action slip [96]; in this case, the action was really incorrect. In both of these cases, if the user informs the system that this was the reason, the dialogue can go on as planned.

The third case is a little more complicated: the user has a different plan for the current goal or one of its subgoals. If this is the case, the user should specify which (sub)goal `S` he will autonomously achieve, then the system will remove this subgoal from the stack of next actions, and continue the instruction from there. For example, when the system instructs the user to boil water, and the system only has a recipe for boiling water that involves boiling it in a pan on the stove, it appears to be a wrong action if the user takes the water boiler and starts to fill it with water. However, in this case the user can state that he will autonomously boil water and the system will wait until this is done, and then continue with the rest of the recipe.

In the example dialogue, the system is expecting the user to add white vinegar, but an error is detected when he grabs the red vinegar bottle:

System: "Add some vinegar to the water."
(User opens a red vinegar bottle)
System: "That's the wrong bottle. Please add white vinegar."
(...)
User: "Okay, you're right."

The example dialogue is a bit more complicated than this, since another error occurs in the middle of this one (the user informs the system that there is no white vinegar, which calls for a repair of the type `untrue`, as we have shown above), but for clarity we have extracted the part that is solely about the wrong action and presented it here. The user grabs the wrong bottle, which the system perceives. The system then informs the user that his action is incorrect. After repairing the `untrue` error, the user accepts the system's assessment of the incorrectness of the action and goes on to use the correct type of vinegar.

7.4.2 Repairing intention discrepancies

The main goal of repairing this type of error is to bring the dialogue back to the dialogue model as quickly as possible (requirement RE8), since the system is only capable of conducting dialogues according to those patterns. Therefore, all repairs end with a rule that informs the user what the system expected his next dialogue contribution to be. Since the dialogue is cooperative, the user is expected to be willing to return to the planned course of the dialogue.

Another consequence of the cooperative setting of the dialogue is the fact that there are only two types of intention discrepancies that we treat in this thesis: `unexpected-inform` and `unexpected-action`. As we have explained in Chapter 5, dialogue contributions that contain the other speech acts are never seen as errors, because the system will simply answer any questions and follow any requests from the user whenever possible. This means that this type of errors does not occur in these dialogue contributions, but solely in `inform` utterances and actions.

Unexpected-inform

When the user unexpectedly informs the system of something, the system adds the information to the beliefbase, informs the user of this, and then also informs the user of the input that it was expecting from him. Note that it is at this point never the case that the system believes `not p`, since belief discrepancies are always repaired before intention discrepancies.

For example, in the middle of a recipe preparation dialogue, the user may suddenly say: "We do not have any more vinegar." If the system disagrees with this (i.e. it believes that there is still vinegar), the `untrue` repair subdialogue that we presented above is activated, but if the system does *not* disagree with this statement (i.e. it already has this belief, or it has no belief about the matter in question), it will add this

```
(40) repair(p, unexpected-inform) <- I(next_dialogue_state) |
     addBelief(p),
     send(S, inform, accept(p)),
     send(S, inform, expected(next_dialogue_state))
```

Figure 7.11: *Repairing errors of the type* `unexpected-inform`

belief to its beliefbase, inform the user of this ("Okay, thanks for that information") and state the expected next dialogue state ("I was expecting you to [...]").

Unexpected-action

When the user performs an action at a stage in the dialogue where the system was not expecting the user to perform an action, the system informs the user of the dialogue state (or type of dialogue contribution) that it was expecting, just to make sure that the user is aware of this.

```
(41) repair(p, unexpected-action) <- I(next_dialogue_state) |
     send(S, inform, expected(next_dialogue_state))
```

Figure 7.12: *Repairing errors of the type* `unexpected-action`

This repair is not expected to occur very often in the dialogue, since as soon as the joint goal of preparing a certain recipe has been established, the interaction is completely focused on the performance of tasks that lead to the joint goal in question, and in this stage of the interaction, performed actions can only be (believed to be) incorrect, not unexpected. The only time in the interaction when actions can be unexpected is before a joint goal has been adopted by the participants.

7.5 Multimodal error handling

In a multimodal interaction, errors in actions *and* errors in utterances must be detected and handled (requirement RE7). Because actions and utterances may come simultaneously, as we have seen in Chapter 4, they should be interpreted as one event and possible errors should be handled accordingly. In this section, we will use our visualization of multimodal dialogues as we presented it in Chapter 4 to illustrate the detection and handling of errors in multimodal interaction.

There are two types of actions that may be relevant to error handling. First of all, there are domain actions, which are part of the task that is being performed by the participants of the dialogue to achieve their joint goal. Secondly, there are gestures and facial expressions which signal feedback to the dialogue partner. Although the

latter is an interesting topic of research that has been taken up by, among others, Allwood [7] and Beun & Bunt [11], we will treat actions of the first type and any additional meaning they can give to simultaneously occurring dialogue acts.

There are two different ways for the system to come to the conclusion that an error occurs: there is a maximum waiting time (if the user does not grab the white vinegar within a certain timespan after the instruction, the system should ask what is wrong) and if the dialogue partner performs another action than the system expected.

Generally, there are three different options at any point in the dialogue when input from the user is expected: the user may perform the desired (physical or communicative) action, the user may perform an undesired action, or the user performs no action at all (within a certain time span). The exact way of handling these types of input depends on the combination of the physical and communicative actions that occur simultaneously. We illustrate this tactic by closely examining an error in the example dialogue: the user takes a bottle of red vinegar instead of white vinegar.

System: "Add some vinegar to the water."
User grabs the red vinegar bottle.
System: "That's the wrong bottle, you need the white vinegar."
User: "Sorry, I'll get the white vinegar."
User grabs the white vinegar bottle.

In Figure 7.13, we show how undesired input is visualized in the music score visualization.

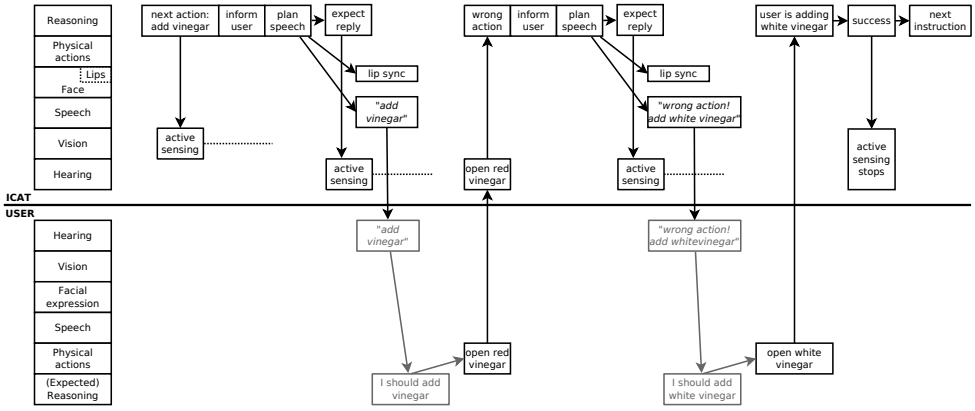

Figure 7.13: *An example dialogue where the user starts to open the wrong vinegar bottle, in the music score visualization.*

When the system observes the user taking the red vinegar bottle instead of the white vinegar bottle, it concludes that this is an undesired event. Several things may have gone wrong: the user may have misunderstood the system, the white vinegar may be finished or lost, or the user may just not know that there are two types of

vinegar and therefore mistakenly think that he got the correct vinegar bottle. While informing the user of his mistake, the system keeps watching out for the white vinegar, so that when the user eventually takes the white vinegar bottle, the system knows that the task in question has been completed, so that it can then go on to the next instruction.

In general, if something else happens instead of the desired event, the system has to adapt its behavior and plan a repair subdialogue. This is the difference between our representation scheme and an actual music score: while the latter is static, our dialogue score is flexible and can be adapted on-the-fly during the dialogue to suit undesired behavior. Table 7.1 shows how the system should handle desired and undesired input in general, based on its perception of physical actions combined with the communicative actions from the user. The table distinguishes between three different kinds of events for both modalities.

	Desired event	**Undesired event**	**No event**
Desired reply	Everything is ok (1)	Accidental wrong action (4)	Ask for event (7)
Undesired reply	Inputs clash (2)	Problem; backup plan (5)	Problem; no backup plan (8)
No reply	Everything is ok (3)	Accidental wrong action (6)	Timeout (9)

Table 7.1: *Handling of (un)desired input.*

In our example, the system can either see the user taking the correct vinegar bottle (the *desired event*), a wrong vinegar bottle (an *undesired event*), or performing no action at all (*no event*). The verbal response from the user is classified in *desired replies* (confirming answers, like "okay" or "I have the pan") and *undesired replies* (rejecting answers such as "we don't have any vinegar" or "I don't know what that is"); if the user says nothing, *no reply* is registered.

If the system receives the desired input in both modalities, we are in situation 1 in Table 7.1: *Everything is ok*. The dialogue will just proceed as planned. There seems to be no good explanation for the situation where the system sees the user performing the correct action (desired event) but the user gives a negative answer (undesired reply). In this situation (2: *Inputs clash*), the system should ask the user for explanation. If the system sees the user taking the white vinegar (desired event) but the user says nothing (no reply), the situation (3: *Everything is ok*) is similar to situation 1: there is no reason to suspect something is wrong and the dialogue can proceed as planned.

In case the system sees the user taking a red vinegar bottle (undesired event) and the user gives a positive answer (desired reply), we are in situation 4: *Accidental wrong action*. The user may have unknowingly grabbed the wrong bottle, and the system's reaction should be to inform the user of this mistake: "That's the wrong bottle, you need the white vinegar." The repair strategy `wrongaction` that we treated earlier in

this chapter can be used here.

If the user gives a negative answer (undesired reply) and the system sees the user performing an incorrect action (undesired event), the system concludes that the user has a problem in getting the vinegar (maybe it is lost or finished). However, since the user *did* perform an action, the user may have chosen to follow a backup plan: he (willingly) performs a different action (situation 5: *Problem; backup plan*). In this case, the system can give the user full responsibility for unexpected consequences of his action (e.g., "You have another type of vinegar than the recipe specifies, but if you think it's okay, you can use this one too."). This is the same course of action that is taken as part of the repair strategy `wrongaction` when the user replies that he has a different plan for the current (sub)goal.

If the user performs an incorrect action (undesired event) and says nothing (no reply), the system presumes that the user made a mistake (situation 6: *Accidental wrong action*), and should attempt to repair the situation just like in situation 2 described above: the repair strategy `wrongaction` can also be used here.

If the user gives a positive (desired) answer but the system does not see the user performing any action (no event), the system should ask the user whether he has really performed the desired action (situation 7: *Ask for event*): "I didn't see you getting the vinegar. Are you sure you have it?" Or, in terms of the repairs that we have presented in this chapter, the system can state the expected dialogue state.

In case of a negative answer (undesired reply) and no action (no event), apparently the plan cannot go ahead as expected and the user does not have a backup plan ready (situation 8: *Problem; no backup plan*). The system should inform the user of the expected dialogue state, just as in the case of `unexpected-inform`. It may also be helpful to ask the user if he has any alternatives or if he wants to stop or pause the process (e.g., to buy a bottle of vinegar).

Finally, if the system sees no action and hears no utterance from the user (situation 9: *Timeout*), it should wait for a while and then ask the user if he is still participating in the joint activity and, similar to before, inform the user of the expected dialogue state.

As this table is specifically made for the case where the system has just given the user an instruction, similar tables of multimodal responses can be made for other states in the dialogue. However, this is an especially prominent case, since it is a specific situation where both an action and a linguistic response are expected.

7.6 Implementation and issues with 2APL

7.6.1 The example dialogue

In Figure 7.14, we repeat the example dialogue that we have presented in Chapter 1. We will now show that this dialogue results from the rules that we have presented in this chapter and Chapter 5, in the same way that we have shown that a simpler dialogue (without errors) can result from the rules that we presented in Chapter 5.

While the example dialogue from Chapter 5 was almost boring in its straightforward generation by the simple rules that we presented there, there are some clear differences between this dialogue and the one that was generated by the basic system

1. User: I would like to prepare French toast.
2. System: That's not possible, we don't have bread available.
3. User: Can you give me another recipe for breakfast?
4. System: Would you like to make a poached egg?
5. User: Alright, I would like to make a poached egg
6. System: Okay, I'll teach you how to poach an egg. First, you have to boil some water.
7. User: Can you turn on the stove for me?
8. System: I'm not able to turn on the stove
9. User: Okay, I will turn on the stove. (pause) I've boiled water.
10. System: Add some vinegar to the water.
11. (User opens a red vinegar bottle)
12. System: That's the wrong bottle. Please add white vinegar.
13. User: We don't have any white vinegar.
14. System: We do have white vinegar. I know this because I bought it.
15. User: Okay, you're right. I've added vinegar to the water.
16. System: Crack an egg into a soup ladle.
17. User: I've cracked the egg into a soup ladle.
18. System: Stir the boiling water.
19. User: How long should I stir the water?
20. System: I don't know.
21. User: Okay, I've stirred the water.
22. System: Drop the egg into the center of the water.
23. User: I've dropped the egg into the water.
24. System: Now wait 3-4 minutes.
25. (3 minutes later)
26. User: I've waited 3-4 minutes.
27. System: Remove the egg from the water.
28. User: I've removed the egg.
29. System: You have now poached an egg! Will you be able to do it by yourself next time?
30. User: Yes.
31. System: Okay, we're done!

Figure 7.14: *The example dialogue from Chapter 1*

in Chapter 5. Several errors and unexpected actions occur, which have to be handled by the system.

First of all, in this dialogue, the joint goal is not possible, because one of the ingredients is not available. Then, in dialogue turn 7, the user requests a task from the system that it is not able to do. The system repairs this by stating that it does not have the capability to do so. In dialogue turn 11, the user performs an incorrect action, which is perceived by the system. The system then states that this is the wrong action and repeats the correct action (now in more detail). Then, in turn 13, the user asserts that there is no white vinegar available, which the system contradicts

in turn 14, also stating the reason why it believes this. The user accepts this (perhaps because he has found the bottle of white vinegar) and the dialogue can finally go on as planned. Then, in dialogue turn 19, the user asks a question that the system cannot answer, because it does not have the requested information.

In a similar manner to our example dialogue in Chapter 5, we also show here how this example dialogue is derived from the rules from Chapter 5 and this chapter. We will show this in Tables 7.2, 7.3, 7.4 and 7.5, which are placed at the end of this chapter.

In this dialogue, not all moves from the user are expected; as we have mentioned in Chapter 1, the unpredictability of humans is one of the reasons why error handling is necessary in human-computer interaction. Therefore, not all of the user's moves can be explained with the rules for the user agent that we have presented in Chapter 5. In these cases, the rule number for the user's moves are left out in the Tables 7.2 to 7.5.

7.6.2 Implementation

In the implementation [72] that we have referred to earlier (in Section 5.5, when we discussed the implementation of the basic system), some aspects of error handling were also implemented. Unfortunately, the development of a dedicated error handling agent was not possible within the available time. However, some error handling tactics have been included in the implementation.

First of all, when the user performs an incorrect action, the system tells the user that he is supposed to perform a different action, and also specifies which action this is. The system can even specify why the performed action is incorrect; for example, if the user selects a container that is too small, the system will tell the user that this is the problem (see Figure 7.15).

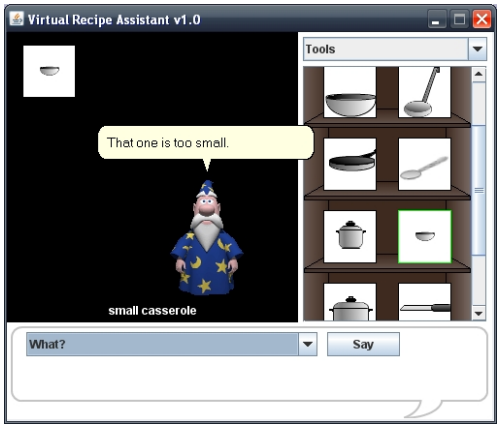

Figure 7.15: *The user selects a container that is too small.*

When the system believes that the user is capable of performing a subtask of the goal and gives the corresponding instruction, the user may at any point during

the preparation of this subtask ask what he needs to do next. Since the system observes the user's actions during the preparation of the subtask, it will then give the appropriate instruction.

Also, at any point during the preparation of a subtask, the user may inform the system that he does not know how to continue (i.e. that he is not capable of performing the subtask). Then, the system will remove the belief that the user is capable of performing the subtask in question, and it will start the instruction dialogue for the subtask (again, at the appropriate point).

7.6.3 Issues with 2APL

Just like in the implementation of the basic system in Chapter 5, we have also encountered some difficulties in the implementation of the error handling module w.r.t. the practicalities of using 2APL. The most notable issue is the fact that in 2APL, the system can only reason about its own beliefbase and not about other parts of its internals. For example, in this chapter we have used the notation `I(next_dialogue_state)` to denote the system's intentions about the dialogue state it is expecting next. Unfortunately, this is not possible in 2APL, since the system has no access to its planbase. Similarly, in our framework we add and remove information of the type `next-action` to and from the planbase of the agent, which is also not possible in 2APL.

Just like in Chapter 5, to solve some of these issues in 2APL, we need to keep a double administration of plans and goals by adding them to the beliefbase, in order to be able to reason about them. If the system instructs the user to perform a certain task `T`, it would always have to add `next-action(U, T)` to the beliefbase, and remove it as soon as the user has performed the action. This also goes for dialogue states; the set of expected dialogue moves that the user can perform should be added to the beliefbase and updated every time the user makes a dialogue move. However, there is still the problem of removing actions from the planbase, which is not possible in 2APL.

In order to correctly and completely implement the error handling module according to the rules that we have presented in this chapter, we should attempt to find workarounds for these issues in 2APL, or investigate other agent programming languages that may be better suitable for the implementation of the system.

7.7 Conclusions and future work

In this chapter, we have presented two different types of errors: belief discrepancies and intention discrepancies. We have presented a set of BDI-rules to detect and repair these errors, sorted by the type of input in which they may occur. We believe that these rules can be used in any task-oriented system in which the system instructs the user to perform a certain task that the user has requested.

In Table 7.16 we show that we have treated all requirements from Chapter 3 pertaining to error handling in this chapter. The page numbers in the table refer to the page in this chapter where the requirement in question is treated.

In the case of deviations from the dialogue state, it would be an interesting and informative expansion to also include a small subdialogue about the user's reasons for

Req	Page
RE1	156
RE2	152
RE3	148
RE4	157
RE5	152
RE6	159
RE7	168
RE8	167

Figure 7.16: *The page numbers in this chapter where the error handling requirements from Chapter 3 are treated.*

his dialogue contribution. After handling the contribution itself (e.g., answering the question), the system could then ask the user why he made the dialogue contribution in question (e.g., `send(U, query-ref, reason(p, X))`). However, because answers to 'why'-questions tend to be long and complex [127], the interpretation of such answers is something that we leave for future research.

If dialogue partners A and B disagree about p and wish to resolve this difference, they have to decide whether A will adopt B's beliefs about p or the other way around. Generally, the non-expert will adopt the expert's beliefs, and the agent who is not in charge will adopt the beliefs of the agent who is in charge. In our situation, however, the situation is more difficult: the system is the expert on the recipes but the user is the expert on the current circumstances in the kitchen. Also, the user is always in charge. In order to deal with this, we can mark each belief with the type of domain that it pertains to (the topic, as it were) and handle inconsistencies accordingly. We would also like to include the possibility to register belief discrepancies ('agree to disagree' [83]).

Alternatively, instead of the simple handling of belief discrepancies that we have presented in this chapter, it would be a useful expansion to implement the possibility to hold more advanced discussions about belief discrepancies. For this, we will have to look into argumentation theory and persuasion dialogues [100].

An interesting addition to our error handling would be to assess whether an error really needs to be repaired or if it would be better to let it slide; in other words, if the cost of repairing is higher than the cost of not repairing the error. Menzel [93], for example, states that in spontaneously spoken speech, it is easier to find an interpretation for a grammatically ill-formed utterance than to detect and correct the error. In order to choose an optimal tactic, we need some sort of algorithm to estimate what the consequences of both approaches are. We can also use the two different attitudes for error repairing that Beun and Van Eijk [12] discern: the *lazy* attitude and the *eager* attitude. An eager system will explain a mismatch more specifically. Depending on the exact needs of the user, the optimal strategy may vary. Also, it may be better to let some errors slide; for example, if the dialogue partner's utterance starts with "My internet ...", one could respond along the lines of "The internet does not belong to you", but alternatively, one could also choose not

to address this error.

Finally, in a similar fashion as with the basic system that we presented in Chapter 5, we have encountered some issues in the implementation of the error handling module in 2APL. In order to fully implement a functional system, we will have to assess the issues that we have found and either develop workarounds for them or investigate the suitability of other agent programming languages.

Actor: Rule no.	**Communicative act, PC-rule or BeliefUpdate**	**Action or utterance**
U	Starting state: `goal(poach_egg)`	
U: 16	`send(S, request, jointgoal(french_toast))`	User: I would like to prepare French toast
S: 1	`adopta(jointgoal(french_toast))`	
S: 5c	`send(U, inform, not possible(french_toast))`	System: That's not possible
S: 5c (contd.)	`send(U, inform, reason(not possible(french_toast, not available(bread))))`	System: We don't have bread
S: 5c (contd.)	`dropgoal(jointgoal(french_toast))`	
U.	`send(S, request, recipe(breakfast))`	User: Do you have another breakfast recipe for me?
S: 3	`send(U, propose, poach_egg)`	System: How about poached egg?
U.	`send(S, inform, accept(poach_egg))`	User: Alright, I would like to make a poached egg
S: 3 (contd.)	`adopta(jointgoal(poach_egg))`	
S: 5b	`send(U, inform, possible(poach_egg))`	System: Okay
U: 16	`adopta(jointgoal(poach_egg))`	
S: 9b	`send(U, inform, start-learning(U, poach_egg))`	System: I will teach you how to poach an egg
S: 9b (contd.)	`teachprepare(poach_egg, [boil_water, add_white_vinegar, crack_egg, stir_boiling_water, drop_egg, wait_3-4_minutes, remove_egg] )`	
U: 17	`tentativeRecipe(poach_egg = [])`	
S: 10b	`send(U, request, next-action(U, boil_water))`	System: Please boil some water
U: 18b	`tentativeRecipe(poach_egg = [boil_water])`	
U: 18b (contd.)	`@kitchen(boil_water, performed(U, boil_water), Time-out)`	(User boils water)
U: 18b (contd.)	`send(S, inform, performed(U, boil_water))`	User: I have boiled water
S: 10b (contd.)	`checked-done(boil_water)`	

Table 7.2: *The example dialogue (part 1)*

Actor: Rule no.	Communicative act, PC-rule or BeliefUpdate	Action or utterance
S: 10b (contd.)	`teachprepare(poach_egg, [add_white_vinegar, crack_egg, stir_boiling_water, drop_egg, wait_3-4_minutes, remove_egg] )`	
S: 10b	`send(U, request, next-action(U, add_white_vinegar))`	System: Please add some vinegar to the water
U.	`perform(open_red_vinegar_bottle)`	(User opens a red vinegar bottle)
S: 32	`repair(open_red_vinegar_bottle, wrongaction)`	
S: 39	`send(U, inform, not next-action(U, open_red_vinegar_bottle))`	System: That's the wrong bottle.
S: 39 (contd.)	`send(U, request, next-action(U, add_white_vinegar))`	System: Please add white vinegar.
U.	`send(S, inform, not available(white_vinegar))`	User: We don't have any white vinegar
S: 22	`repair(not available(white_vinegar), untrue)`	
S: 35	`send(U, inform, available(white_vinegar))`	System: We do have white vinegar
S: 35 (contd.)	`send(U, inform, reason(available(white_vinegar), bought(white_vinegar)))`	System: I know this because I bought it
U.	`send(S, inform, accept)`	User: Okay, you're right
U: 39 (contd.)	`skip`	
U: 18b	`tentativeRecipe(poach_egg = [boil_water, add_white_vinegar])`	
U: 18b (contd.)	`@kitchen(add_white_vinegar, performed(U, add_white_vinegar), Time-out)`	(User adds vinegar to the water)
U: 18b (contd.)	`send(S, inform, performed(U, add_white_vinegar))`	User: I have added some vinegar
S: 10b (contd.)	`checked-done(add_white_vinegar)`	
S: 10b (contd.)	`teachprepare(poach_egg, [crack_egg, stir_boiling_water, drop_egg, wait_3-4_minutes, remove_egg] )`	

Table 7.3: *The example dialogue (part 2)*

Actor: Rule no.	Communicative act, PC-rule or BeliefUpdate	Action or utterance
S: 10b	`send(U, request, next-action(U, crack_egg))`	System: Please crack an egg into a soup ladle
U: 18b	`tentativeRecipe(poach_egg = [boil_water, add_white_vinegar, crack_egg])`	
U: 18b (contd.)	`@kitchen(crack_egg, performed(U, crack_eggr), Time-out)`	(User cracks egg into soup ladle)
U: 18b (contd.)	`send(S, inform, performed(U, crack_egg))`	User: I have cracked an egg into a soup ladle
S: 10b (contd.)	`checked-done(crack_egg)`	
S: 10b (contd.)	`teachprepare(poach_egg, [stir_boiling_water, drop_egg, wait_3-4_minutes, remove_egg] )`	
S: 10b	`send(U, request, next-action(U, stir_boiling_water))`	System: Please stir the boiling water
U.	`send(s, query-ref, info(stir_water, duration, poach_egg))`	User: How long should I stir the water?
S: 26	`repair(info(stir_water, duration, poach_egg), no-information)`	
S: 36	`send(U, inform, no-information(info(stir_water, duration, poach_egg)))`	System: I don't know
U: 18b	`tentativeRecipe(poach_egg = [boil_water, add_white_vinegar, crack_egg, stir_boiling_water])`	
U: 18b (contd.)	`@kitchen(stir_boiling_water, performed(U, stir_boiling_water), Time-out)`	(User stirs boiling water)
U: 18b (contd.)	`send(S, inform, performed(U, stir_boiling_water))`	User: I have stirred the boiling water
	(etc., until teachprepare-list is empty)	
S: 10a	`send(U, inform, finished(poach_egg))`	System: You have now poached an egg

Table 7.4: *The example dialogue (part 3)*

Actor: Rule no.	**Communicative act, PC-rule or BeliefUpdate**	**Action or utterance**
U: 19b	`RemoveBelief(tentativeRecipe( poach_egg = [boil_water, add_white_vinegar, crack_egg, stir_boiling_water, drop_egg, wait_3-4_minutes, remove_egg])`	
U: 19b (contd.) U: 19b (contd.)	`AddBelief(poach_egg = [boil_water, add_white_vinegar, crack_egg, stir_boiling_water, drop_egg, wait_3-4_minutes, remove_egg])` `AddBelief(capable(U, poach_egg))`	
S: 9b (contd.)	`send(U, query-if, capable(U, poach_egg))`	System: Will you be able to do it by yourself the next time?
U: 20a	`send(S, inform, capable(U, poach_egg))`	User: Yes
S: 9b (contd.)	`AddBelief(capable(U, poach_egg))`	
S: 9a	`send(U, inform, finished(poach_egg))`	System: Okay, we're done!
U: 19a	`dropgoal(poach_egg); dropgoal(jointgoal(poach_egg))`	
S: 9a (contd.)	`dropgoal(poach_egg)`	

Table 7.5: *The example dialogue (part 4)*

8 Conclusions

Alles hat ein Ende, nur die Wurst hat zwei.

Stephan Remmler

As we have stated in Chapter 1, this thesis focuses on:

> *a BDI-based framework for a dialogue system that can instruct a user to perform tasks consisting of predefined sequences of actions, based on a joint goal that the user and the system have together, focusing on various tactics to make the system more robust.*

Briefly, we have developed this framework by investigating human-human dialogues, cooperation, and dialogue systems. Based on this, we have gathered a set of requirements that we believe any such system should adhere to. Then, we have constructed a framework for such a system, consisting of a basic system and an error handling module. We have also investigated a technique for natural language processing and generation.

We summarize the conclusions in this final chapter, presenting the results as answers to the research questions that we posed in Chapter 1 in Section 8.1. In Section 8.2, we will present a discussion of our results, the validation of the results (8.2.1), point out some directions of future work (Section 8.2.2), and finish with our vision for the future (Section 8.2.3).

8.1 Results

In this section, we present the results of our research, in the form of answers to the research questions.

8.1.1 Cooperative dialogue

> **Question 1:** *How does a cooperative dialogue (or interaction) result from a joint goal?*

In Chapter 2, we have presented a definition of joint goals that require the participants of the joint goal to communicate about the status of the goal and of its subtasks. This means that we can implement cooperative interaction in a principled way that is based on the mutual adoption of a joint goal by the participants of the dialogue.

To come to this definition of joint goals and cooperation, we have first investigated activities, in the broadest sense. Activities may only involve one agent that performs actions without taking into account any other agents. Since we consider agents, these actions are goal-directed, meaning that they are initiated by the agent in order to achieve a goal.

Then, we have refined the concept of activities to *joint* activities: activities in which more than one agents is involved. These activities are initiated to achieve joint goals. For joint activities, it is very useful to have the concept of mutual belief. The actions that are performed as part of joint activities may still be single-agent actions, but may also be joint actions, performed by multiple agents together. Both of these types of actions are participatory actions, meaning that they are part of a joint activity.

We have then refined joint activities to dialogues: joint activities that are mainly language-based. Beside linguistic actions, which are performed in dialogue turns, dialogues may also contain non-linguistic actions such as gestures. Both linguistic and non-linguistic dialogue actions need to be interpreted by the dialogue partner, requiring the concept of *grounding*.

Cooperative dialogues are a subset of dialogues in which the participants adhere to norms of cooperation. This mainly implies that they are sincere and helpful with respect to each other and to their joint goal and that they do not hold any hidden agendas. Participants may have private goals, as long as they do not conflict with any joint goals. Finally, we have defined successful dialogues, which are dialogues in which joint goals are reached.

8.1.2 Requirements

> **Question 2:** *What are the requirements for a robust cooperative task assistant?*

In Chapter 3, we have explored the context and setting of our research aim of constructing a robust, BDI-based task assistant. In order to do this, we have studied five different aspects of such a system: the participants, the domain, the interaction, specifically multimodal interaction, and error handling. To ensure that the requirements are generic for any system that uses sequences of actions (cf. recipes), we added an additional domain for which the requirements should also hold: besides the cooking assistant, we also studied a system that can instruct a user to set up a weblog.

First of all, we studied the participants (the system and the user), focusing on reasoning and on properties of companion robots. We have investigated the requirements that companion robots should fulfill. Some domain-specific requirements were also encountered, which pertain specifically to cooking or to setting up weblogs. For example, the system should be able to handle the concept of different recipes for the

same goal (e.g., boing water in a pan vs. in the microwave) and should be able to reason about the capabilities of itself and the other participants.

Secondly, we studied the domain, specifically focusing on recipes and actions. For this, we first studied recipes, discovering that they are essentially nested sequences of actions. This also resulted in a list of requirements, including the concepts of open and closed delegation, meaning that participants in the dialogue can be instructed to perform atomic actions or subgoals (which are sequences of tasks, such as the subgoal 'boil water' for the goal 'poach egg'), and reasoning about whether it is possible to perform certain actions given the current circumstances.

Thirdly, we studied the interaction between the system and the user. To account for different aspects of the interaction, we studied different types of dialogues: information-seeking dialogues, tutoring dialogues, and task-oriented dialogues. Having analyzed such dialogues, we constructed dialogue models for the three types of dialogues, subsequently combining them into one dialogue model for a recipe dialogue, which contains aspects of all three dialogue types. The framework that we present in this thesis should follow this dialogue model in the dialogues that it produces. The dialogue models are represented as flow charts in Section 3.3.

Fourthly, we focused specifically on multimodal aspects of interaction. Multimodal interaction, which may contain simultaneous in- or output on different modalities, requires synchronization. Furthermore, we believe that an intuitive representation of multimodal interaction facilitates the implementation of such a system by visualizing the connections and relations between in- and output on different modalities.

Fifthly and finally, we studied error handling. This involved establishing a definition of errors that can occur in a system that uses speech acts with a communicated content as in- and output (instead of natural language), thus abstracting from any errors in the interpretation and formulation of speech or text. This resulted in two types of errors: unexpected contributions (which do not fit into the dialogue model that we presented earlier) and dialogue contributions which signal or introduce inconsistencies in the mutual beliefs of the participants.

Our investigation of these five aspects of a robust cooperative task assistant has resulted in a set of requirements for such a system, specifically a system that assists the user in the preparation of recipes and/or setting up weblogs. The requirements are sorted into five lists pertaining to the five different aspects of the system: reasoning, the domain, the interaction, a separate list for multimodal (aspects of) interaction, and error handling. The lists can be found in the tables in Chapter 3.

8.1.3 Basic framework

Question 3: *How can we implement a basic cooperative task assistant?*

In Chapters 4 and 5, we have presented a framework for a basic cooperative task assistant. The framework is based on the BDI architecture and uses notions of joint goals, which are tightly coupled with communication, as we have seen in Chapter 2, and capabilities and opportunities for reasoning about pre- and postconditions of tasks that have to be performed in order to achieve the joint goals. We have also introduced a visualization for multimodal dialogues that can help in the development of multimodal dialogue systems.

Chapter 4 focuses mainly on the translation of the requirements from Chapter 3 to more specific aspects in the architecture of the system. Except for the error handling requirements, which are treated in Chapter 7 (see below, under research question 5), all of the requirements are fulfilled in the architecture that we present. We have presented an instantiation of a generic architecture for a companion robot which contains (only) the parts that we treat in this thesis, leaving the implementation of additional aspects such as the emotion synthesizer for future work. The system is represented as a BDI agent, with separate reasoning engines for the main interaction process and for the error handling process.

We also present a formalization of recipes that fulfills the requirements from Chapter 3. Recipes are represented as lists of tasks; tasks denoting either identifiers of atomic actions, or names of other lists of tasks. Atomic actions are represented with pre- and postconditions, allowing the system to reason about the feasibility of actions in recipes, based on postconditions of previous actions. The interaction is always initiated with the introduction of a joint goal by one of the participants. From this joint goal, communicative and/or physical actions result, based on the beliefs and rules that the participants have. Dialogue actions are modeled as FIPA-ACL communicative acts.

In Chapter 5, we have presented a set of rules that fit into the architecture from Chapter 4. These rules, together with a set of beliefs and goals, can generate dialogues. The rules are presented in a 2APL-like pseudocode. We have sorted the rules into three phases of the dialogue: joint goal selection, the planning phase, and the instruction phase. Additionally, we have added rules for answering questions and following requests from the user. While it is true that these dialogue contributions are not included in the dialogue model, we feel that any cooperative system should react in a cooperative way to these contributions, as we have stated in the requirements in Chapter 3.

Although not all of the rules that we have presented in Chapter 5 are suitable to be directly implemented in 2APL, we have simulated the generation of a dialogue with these rules. For this, we have constructed a set of rules that form a simple user agent. The rules mainly pertain to complying with instructions from the system. Furthermore, we have added a set of beliefs and goals, to which we applied the rules of our framework. From this, a simple example dialogue resulted, without error handling (which we treated later, in Chapter 7), but in which a recipe preparation was successfully simulated.

8.1.4 Language generation and interpretation

> **Question 4:** *What robust approach to language generation and interpretation can we use?*

In Chapter 6, we have discussed a layered approach for language generation and interpretation in the form of Functional Discourse Grammar (FDG), which uses four levels: the phonological level, the morphosyntactic level, the representational level (pertaining to semantic aspects) and the interpersonal level (pertaining to pragmatic aspects). Also, we have presented a framework that uses syntactic features of utterances to elicit the indirect interpretation of speech acts.

Although we have constructed our framework on the level of speech acts, we have also briefly treated natural language. In Chapter 6, we gave a brief introduction to FDG, a theory of grammar which encompasses the complete process of formulation of natural language utterances from communicative intentions. Because these communicative intentions are quite similar to the communicative actions that result from our framework that we present in this thesis, we believe that FDG can function as an appropriate grammar theory to be coupled with our basic system.

FDG consists of four main components: the *conceptual component*, which is similar to the reasoning enging in 2APL and which delivers communicative intentions as its output; the *contextual component*, which is similar to the beliefbase of a BDI agent, containing information that has an effect on the natural language output of the system; the *output component*, which articulates the natural language output to the outside world; and most importantly, the *grammatical component*, which contains the aforementioned four levels of representation and is connected to all other components. The grammatical component translates the communicative intentions from the conceptual component to a natural language utterance, which it then delivers to the output component. In this process, it uses information from the contextual component to form the utterance.

While most FDG literature treats the formulation of utterances, in this thesis we focus on the interpretation of utterances, starting with the natural language utterance itself and parsing it to a communicative intention. While the completion of this algorithm for the whole process is still a topic of future work, a partial algorithm was constructed, which can translate structures on the morphosyntactic level to structures on the representational level. We believe that similar processes can be used to make translations on different levels to complete the whole algorithm.

One of the problems that we encountered was the interpretation of indirect speech acts; that is, utterances that have a non-literal meaning, such as "Can you tell me the time?", which has the syntactic structure of a yes/no-question, but is usually meant as a request. In Chapter 6, we also present a simple algorithm for the interpretation of such utterances, based solely on syntactic properties of the utterance and the type of dialogue.

8.1.5 Error handling

> **Question 5:** *How can we augment a basic cooperative task assistant with an error handling module?*

In Chapter 7, we have proposed rules to handle two types of errors in cooperative dialogues: situations where the beliefs of the system and the user do not match, and situations where the intentions of the system and the user do not match. We have sorted these rules by the type of input in which they may occur, and presented tactics for detecting and repairing these errors.

We augment our basic system, that we presented in Chapter 5, with a generic set of rules that can detect and repair errors in cooperative, task-oriented dialogues. By comparing our original example dialogue from Chapter 1 to the simpler version of the example dialogue that resulted from the rules that we presented in Chapter 5, we can

see a number of deviations from the planned course of the dialogue; the errors that occur in the dialogue and are successfully handled by the system.

After a recap of the definition of errors that we presented in Chapter 3, we presented a short overview of the main types of errors that we do *not* treat in this thesis: speech recognition errors, error prevention, self-repair, and uncooperative behavior. We also briefly explain that we have chosen a qualitative approach to error handling to make a tight coupling to our basic system, although quantitative error research is a promising field of research. Then, we give some examples of the two types of errors that we treat in this thesis.

In Sections 7.3 and 7.4, we present the main error handling process: first, the detection of errors in 7.3, and then the repair in 7.4. We have presented the rules for the detection of errors sorted by type of input: four speech acts: inform, query-if, query-ref, request; and performed actions. For each of these input types, a number of conditions are given that indicate an error. For all of these errors, different repair strategies are activated. The error repairs in Section 7.4 are sorted by type of error: belief discrepancies and intention discrepancies. Where possible, we have illustrated all of the repair strategies with examples from the example dialogue.

Finally, we treat multimodal error handling. With the same visualization method for multimodal interaction that we presented in Chapter 4, we visualize an exerpt from the example dialogue where an error occurs. We show how to detect errors multimodally by combining two input modalities: action and speech. On each of these modalities, three different types of events can happen: the desired event, an undesired event, or nothing at all. Because we consider two modalities, there are nine different combinations. We connect these combinations with an error handling strategy that we presented earlier in the chapter.

We finish the chapter by validating our error handling rules in the same way as in Chapter 5: we simulate the generation of a dialogue based on the presented rules, together with a set of beliefs and goals and a user agent. For this version of the example dialogue, we have not presented a set of rules for a user agent, since some errors occur because not all of the (communicative or physical) actions from the user are logically explainable. We conclude that the example dialogue can successfully be generated from the rules that we present in this thesis.

8.2 Discussion

In this section, we discuss some positive and some negative general aspects of our research. Then, we will again treat each separate research question, discussing the results in a broader context; e.g., what does it mean w.r.t. the research aim, what can we infer from the results, and what are the limitations of the results or of the methods used.

Generally, we believe that this research contributes a broad generic basis for cooperative task-oriented dialogue systems, both for further research into the subject and for implementation of such a system. We have discussed various aspects of the system, building up a framework from scratch, looking at multimodal interaction, companion robots, dialogue theory, natural language, and error handling.

In the process of constructing this framework, we have taken care to make it as robust as possible, since we feel that errors and breakdowns are some of the major issues in human-computer dialogue systems. As we have stated in Chapter 1, we believe that robustness is an interesting research topic for two main reasons: first of all, from the point of view of human interaction, this thesis helps to study how humans handle errors in interaction, and secondly, this thesis provides a framework for developing a dialogue system that can produce more robust interaction.

Although there is plenty of dialogue systems research, dialogue systems are not very widely used for practical purposes outside of the research domain. If instruction systems become more robust and can handle errors in a natural, constructive way, we believe that such systems are more likely to be actually used. We have attempted to keep this in mind in the construction of our framework, and hope that this framework can be used to make steps towards the implementation of a practical, robust task-oriented system.

One of the strong points of this thesis is that it is strongly multidisciplinary, treating dialogue theory and natural language as well as BDI logic and agent systems. This thesis provides a general basis for the implementation of cooperative task-oriented dialogue systems and treats all of the involved topics in the broader context of this subject. An important contribution of this thesis is the connection between these fields. Because of this, this thesis can assist researchers that are specialized in only one of these fields to form a basic understanding of the other topics and give them some pointers for developing a complete framework or implementation.

In Chapter 2, we studied how a cooperative interaction results from a joint goal. The contribution of this chapter to the general research aim of developing a robust dialogue system is the connection between the concepts of joint goals and cooperation on the one hand, and task-oriented dialogues on the other hand. We have shown that such dialogues can be generated based on principles from literature about joint goals and cooperation.

In Chapter 3, we studied dialogue systems and, specifically, requirements for a cooperative task-oriented system. We believe that the main contribution of this chapter is the dialogue model for cooperative task-oriented dialogues. This dialogue model adheres to the principes of cooperative dialogue that we presented in Chapter 2, and combines features of information-seeking, tutoring, and task-oriented dialogues. A more advanced version of such a dialogue model could be formed by keeping the separate dialogue models for the different types of dialogue, and for each dialogue action choosing the appropriate dialogue model. However, we believe that for the current purposes, the presented dialogue model is sufficient.

In Chapter 4 and 5, we studied how to translate the requirements from Chapter 3 into a basic cooperative task-oriented system. The contribution of this chapter is the idea of a relatively simple framework that generates dialogues based on joint goals that the participants have together. The rules that this framework contains, are based on the principles of cooperation and joint goals that we presented in Chapter 2. The generation of a dialogue based on these rules shows that the framework is sufficient for this purpose. The establishment of formal specifications of the architecture and the implementation and a formal proof of the relationship between them is not the focus of this thesis and has therefore been left as a topic for future work.

In Chapter 6, we studied the generation and interpretation of natural language, focusing on two topics: Functional Discourse Grammar and the interpretation of indirect speech acts. The contribution of this chapter to the general research aim is the idea that we can use multiple expectations (in this case, on the different levels of FDG) to make language interpretation more robust. We believe that FDG is an appropriate theory for the language generation and interpretation modules in our system, because the Conceptual Component of FDG is comparable to the BDI reasoning engine that our system is based on.

In Chapter 7, we studied the detection and handling of errors in a cooperative task-oriented system. The contribution of this chapter to the general research aim of developing a robust dialogue system is the idea that with a separate module, any basic cooperative task-oriented system can be augmented with error handling and made more robust. Just like in Chapter 5, we validated this framework by presenting an example dialogue that is generated by the rules of the framework. We believe that our error handling framework can be used to augment any BDI-based cooperative instruction system, possibly with a few minor adjustments. In order to check this, the error handling module has to be implemented and tested on various instantiations of such systems.

In our view, the biggest shortcoming of the research in this thesis is that the presented framework is not completely implemented and tested with human users. Without this, we cannot yet give a complete and proper assessment of the value of the framework. Testing such a system may be easier for the weblog setup task that we mentioned earlier in this thesis (most notably in Chapter 3), since the implementation of a fully functioning vision processing algorithm that can correctly perceive kitchen actions is a difficult task. It is easier to implement a system that is connected to a computer and observes actions that are performed on this computer.

8.2.1 Validation

Our basic framework was almost completely implemented, showing that it is indeed a valid framework. However, of our error handling module, only some aspects are implemented, leaving the rest of the implementation and validation for future work. As a preliminary validation of our framework, we have presented a dialogue in which errors are detected and repaired, which can be constructed with the rules that we present in this thesis. As we have stated above, a more definitive validation can be given if we implement the complete system and test it with human users. The value of the framework can be fully assessed after testing whether users can perform tasks better, faster and easier with a task assistant than with, for example, a manual.

However, this does not mean that this thesis presents only a shot in the dark. We believe that this thesis can aid researchers in agent systems and/or dialogue systems in the development of a robust task assistant, by providing generic frameworks for various aspects of the process. Most notably, this consists of the requirements analysis, the architecture, the basic system, a starting point for a natural language parser in FDG, and an error handling module. The framework that we present is for a cooperative, flexible system that is not only reactive, but also acts proactively in assisting the user, by reasoning about resources and capabilities, and by detecting

and repairing errors in the interaction.

To check that this framework is at least functional as a basis for implementation, we have emulated the construction of a dialogue by our system. We did this by applying the rules of our framework to a set of beliefs and goals; first only the rules from Chapter 5 to construct a basic instruction dialogue, which was not capable of properly responding to errors, and then the rules from Chapters 5 and 7, which resulted in a dialogue in which several different errors were detected and repaired. Applying different sets of beliefs and goals to the framework and using different recipes in the recipe database will result in different, but similar, dialogues.

Also, the fact that a large part of our framework was implemented in 2APL, and that not too many issues were encountered in this process, leads us to believe that our framework is sufficient to be used for the construction of task-oriented dialogue systems.

For a more thorough validation, there are two more steps to be taken. First of all, the complete system should be implemented in an agent programming language. Then, this system should be tested in a real kitchen setting with human users, who use the system as a recipe assistant in the preparation of recipes. Then, the quality of the framework can be tested according to various metrics: amount of successfully prepared recipes, amount of dialogues that are concluded successfully, and average pleasantness of the dialogues as perceived by the users are three examples of such metrics that can be used to evaluate the implemented system.

8.2.2 Future work

There are many points on which we can expand the system. Some of these were originally planned, but abandoned with the ending of the Dutch Companion project, such as the implementation of emotions, which would be an interesting addition to the system (for a comprehensive work on the formalization of emotions, see Steunebrink [115]). Emotions can be used not only to augment the system's output with, for example, facial expressions, but also to influence the system's decisions. These emotions can be caused by cognitive-level events, such as plans failing (disappointment), goal achievement (joy), and perceived emotions from the user (if negative: pity). Reactive emotions like startle or disgust can also influence a robot's emotional state. Moreover, emotions can manifest themselves in many different ways; e.g., facial expressions, speech prosody, selecting or abandoning certain plans, etc.

As we have pointed out in Chapter 3, an interesting topic of future work is the incorporation of features of social dialogues into the interaction. Especially in combination with the implementation of emotion heuristics, this feature has the potential of making the system more pleasant and intuitive to interact with.

Another aspect of the task assistant that we were not able to develop collaboratively with the Dutch Companion project partners is an advanced level of multimodality of the dialogue system, with vision and sound processing systems and a sound awareness module that can perceive and identify various kitchen sounds. Also, in order to appear more natural and human-like, a companion robot could exhibit some low-level reactive behaviors, such as blinking and following the user's face, and fast reactive behaviors such as startling when subjected to a sudden loud noise.

An expansion that would be helpful in the interaction is a more advanced recipe selection phase. The user should then be able to specify a number of criteria for his desired recipe, varying from simple to complicated: e.g., something vegetarian, a dessert with chocolate, something healthy, or something that's fitting for a warm summer evening. Then, the system searches its recipe database for a recipe that satisfies these criteria. In addition, the user should be able to add or retract constraints during the recipe selection dialogue.

We would also like to implement a more sophisticated treatment of recipe trees, possibly involving GPGP [86]. Then, we could also implement agents as resources in order to make parallel planning possible, especially for situations with multiple users. This involves investigating possibilities for parallel performing of tasks by the participants, by having a notion of interleaved plans and interruptions. For example, cooking spaghetti bolognese consists of two main subtasks: cooking the spaghetti and cooking the bolognese sauce. However, a lot of interleaving occurs between these two recipes: first put water to boil, then start with the sauce while waiting for the water to boil, then when the water boils, put spaghetti in the boiling water, then continue making the sauce, then when the spaghetti is done, drain the spaghetti, etcetera.

The task of implementing a parser that can automatically extract pre- and post-conditions from natural language recipes (possibly in a structured markup language) is a difficult task that requires domain knowledge, but we believe that it is certainly an interesting and promising research path. It even seems to be of interest to parties such as Google, who are currently working on a recipe representation format called Google Recipes[1].

In order to complete the path that we have set out upon in Chapter 6, parsing natural language utterances using Functional Discourse Grammar, we will need to expand the current algorithm that only translates structures from the morphosyntactic level to the representational level to the complete FDG framework of four levels. In order to elicit the correct interpretation of indirect speech acts, we will need to complete the construction of tables for all specified verbs in Section 6.4.

Another useful and interesting expansion is to make our error handling module more advanced, which can be done in various different ways. For example, we may implement subdialogues about the user's reasons for making an error; we can add persuasion dialogues or rules for different types of beliefs to make the handling of belief discrepancies more advanced; or we may add the possibility to reason about whether an error is worth solving (possibly including a distinction between lazy and eager error handling attitudes).

Finally, an important issue is sufficiently and accurately evaluating this work, as we have mentioned above. Therefore, a crucial next step would be to completely implement the system and then to test it in an experimental setting with real users.

8.2.3 Vision

If some of the important issues are resolved, it is very well possible that task assistants will get more and more prevalent on many domains: in households and workplaces,

[1] `http://www.google.com/support/webmasters/bin/answer.py?hl=en&answer=173379`, retrieved June 14th, 2010

as personal assistants, shop assistants, secretaries, etcetera. Users will be able to acquire different task domain packages; for example, a task assistant can have several functions, not only as a recipe assistant, but also a school tutor, a do-it-yourself instructor, a sewing help, and an arts and crafts guide.

Ideally, task assistants are connected to the internet, allowing them access to larger databases of information. Semantic processing can enable them to acquire new information. An internet connection also allows the assistants to communicate with each other, functioning as sort of a mediated social network. This is especially interesting for systems functioning as personal assistants, who can then perform tasks such as setting up meetings with others through their personal assistants.

Task assistants may or may not take the form of a physical robot. A virtual assistant has the advantage of being more portable; for example, an online task assistant can be accessed anywhere via the user's smartphone or other mobile device. An advantage of physical robots is that they can help the user perform actions. On the other hand, this does make them less versatile; an embroidery robot and a wood-cutting robot are hardly interchangable. A solution to this is to make multiple 'embodiments' that can be connected to a user's task assistant.

It's ten o'clock in the morning on a Sunday. Lucy walks into the kitchen. She's in a cheerful mood and wants to make something special for breakfast. She turns on her new recipe robot, iCat.
Lucy: Good morning iCat, can you tell me how to make Eggs Benedict?
iCat: Good morning Lucy! I'm afraid we can't make Eggs Benedict, since we don't have English muffins.
Lucy: Oh, but I have this bag of muffins that I bought yesterday.
iCat: English muffins aren't the same as regular muffins. Do you want me to explain the difference?
Lucy: Not really, or maybe later. Do you have another suggestion for breakfast?
iCat: We can make toast and poached eggs, that's quite similar to Eggs Benedict but a lot less complicated. Also, we need to finish the eggs, since the expiration date is tomorrow.
Lucy: Good thinking, iCat. Also, it'll be nice to practice poaching an egg again.
iCat: Okay, I'll turn on the toaster, let me know if you need help poaching the egg.
Lucy: Alright! *[starts cooking]*
iCat: In the meantime, I can tell you that today is going to be sunny but cold, and that your brother asked you to confirm your attendance at his birthday next week.
Lucy: Thanks. I'll call him back after breakfast, but you can already let him know that I'll be there.
iCat: *[looks alarmed]* Wait a second, don't forget the vinegar before you put the egg in!
Lucy: Oh, that's right on time! You're keeping a good eye on me, aren't you?
iCat: *[winks]* I sure am! The vinegar is not absolutely mandatory, but it helps the egg stay together.
Lucy: Okay, I just put the egg in the water, can you warn me when it's done?
iCat: Sure. It's probably about time that you put the bread in the toaster now.
Lucy: Ah, thanks for reminding me. [puts bread in toaster]

iCat: Don't forget to take a cup of coffee, I started the coffee machine when you walked into the kitchen.
Lucy: I didn't even notice! Thanks, iCat. *[opens fridge]* Oh, but we don't have milk...
iCat: I'm sorry, I didn't realize that you'd want milk in your coffee. Hang on, let me check with the neighbor's kitchen assistant... Yes, they have milk, and they're at home and awake now. If you're quick, you can go and borrow a cup of milk before your egg is done.
Lucy: Good idea! *[2 minutes later]* I'm back.
iCat: The egg is about done by now, but if you want to butter your toast before you put your egg on it, I recommend you do that first, so the egg won't get cold.
Lucy: *[takes toast out of toaster and butters it]* Thanks for the advice, iCat!
iCat: You're welcome. Enjoy your breakfast, Lucy!

Appendix

GLOSSARY OF LINGUISTIC TERMS

Admonitive A speech act that denotes a warning (e.g., "Be careful not to break the egg yolk.").

Anaphora An expression that refers to another (preceding) expression; e.g., "We do have white vinegar. I know this because I bought it", where the anaphor 'it' refers to the referent 'white vinegar'.

Background See *Focus/Background*.

Clause A group of words that consist of a subject and a predicate.

Comment See *Topic/Comment*.

Commissive A speech act that denotes the making of a commitment (e.g., "I will teach you how to poach an egg.").

Communicative act See *Move*.

Communicative intention The intention that a speaker has in performing a communicative act.

Construal The hearer's interpretation of an utterance or event.

Context All information outside the current utterance.

Conversation Spoken form of discourse [40]; see also *Dialogue*.

Declarative A sentence that is a statement (e.g., "I have turned on the stove.").

Dialogue A mostly linguistic interchange of information between two (or more) persons, usually initiated in order to reach some goal(s); see also *Conversation*.

Dialogue control act A dialogue act that does not pertain to the domain, but to the interaction itself.

Directive A speech act that denotes an order (e.g., a request or command; "Can you turn on the stove for me?").

Discourse A type of joint activity in which conventional language plays a prominent role [40]

Discourse act The smallest identifiable unit of communicative behavior [79].

Encoding (FDG) The process that converts pragmatic and semantic representations into morphosyntactic and phonological ones.

Focus/Background The focus is the most important/salient part of the utterance; the background is the rest of the utterance. (Similar to, but not necessarily the same as *Topic/Comment*.)

Formulation (FDG) The process that determines valid pragmatic and semantic representations in a language, based on a communicative intention; the strategic arrangement of the communicative intention into a temporal sequence of discourse acts.

Grounding Assuring success on three different levels: attending to, hearing and understanding each other; establishing something as part of common ground well enough for current purposes; integrating an utterance's meaning with one's beliefs [94].

Hortative A speech act that denotes encouragement (e.g., "Let's go!").

Illocution (Austin), illocutionary act See *Speech act*.

Illocution (FDG) See *Sentence type*.

Imperative A sentence that is a command (e.g., "Turn on the stove!").

Implicature Information that is suggested by an utterance, though not (conventionally) entailed by it. E.g., "Lucy got up and made breakfast" suggests that "Lucy got up and then (subsequently) made breakfast", even though the utterance, being a conjunction, is logically equivalent to "Lucy made breakfast and got up."

Interrogative A sentence that is a question (e.g., "Can you turn on the stove for me?").

Language The creation and usage of symbols; a verbal form of interaction; the method of human communication, either spoken or written, consisting of the use of words in a structured and conventional way (New Oxford American Dictionary).

Locutionary act The utterance of a sentence with determinate sense and reference [88].

Mirative A speech act that denotes wonder (e.g., "How beautifully she sings!").

Modal verb A type of auxiliary verb that is used to express modality (e.g., can, should, may).

Modifier (FDG) An element that lexically modifies a move (e.g., "to cut a long story short").

Move A participatory action; the minimal free unit of discourse that is able to enter into an exchange structure [74].

Operator (FDG) An element that grammatically modifies a move (e.g., duration, instrument).

Optative A speech act that denotes a wish or hope (e.g., "Enjoy your meal!").

Parsing The process of analyzing text to extract its syntactic structure.

Performative A linguistic act that does not have a truth value, but constitutes the performance of the action that it describes.

Perlocutionary act The bringing about of effects on the audience by means of uttering the sentence, such effects being special to the circumstances of utterance [88].

Phonology The systematic use of sound to encode meaning in any spoken human language.

Phrase A group of words functioning as a single unit in the syntax of a sentence.

Pragmatics The way in which context contributes to meaning.

Presupposition "Presuppositions can be considered as a kind of background assumptions that should be fulfilled to understand the meaning of the message." [118] [99]

Prosody The rhythm, stress, and intonation of speech.

Referent The object or (part of an) utterance that is referred to, e.g. by anaphora or a deictic gesture.

Relevance The property of having a purpose with respect to the current joint activity or to the interaction itself.

Saliency The state of (part of) an utterance of being especially noticeable or prominent.

Semantics The study of how meaning is attached to natural language, and of relations between different linguistic units.

Sentence type The grammatical mood of a sentence; e.g., declarative, interrogative, or imperative.

Speech act The making of a statement, offer, promise etc. in uttering a sentence, by virtue of the conventional force associated with it [88].

Syntax The principles and rules for constructing sentences from words in natural languages.

Topic/Comment The topic is what the utterance is about; the comment is what is being said about the topic. (Similar to, but not necessarily the same as *Focus/Background*.)

Transitional-relevance place A point in a dialogue turn where the dialogue partner may begin a dialogue turn.

Turn-constructional unit A word or clause that ends in a transitional-relevance place.

Utterance A deliberate linguistic act that conveys meaning, usually bounded by silence.

Bibliography

[1] R.M.C. Ahn, R.J. Beun, T. Borghuis, H. C. Bunt, and C.W.A.M. Van Overveld. The DenK-architecture: a fundamental approach to user interfaces. *Artificial Intelligence Review*, 8(2):431–445, 1995.

[2] W.A. Ainsworth and S.R. Pratt. Feedback strategies for error correction in speech recognition systems. *International Journal of Man-Machine Studies*, 36(6):833–842, 1992.

[3] G. Airenti, B.G. Bara, and M. Colombetti. Failures, exploitations and deceits in communication. *Journal of Pragmatics*, 20(4):303–326, 1993.

[4] J. F. Allen, B. W. Miller, E. K. Ringger, and T. Sikorski. Robust understanding in a dialogue system. In *Proceedings of the 34th Annual Meeting of the Association for Computational Linguistics*, pages 62–70, 1996.

[5] J. Allwood. *Linguistic communication as action and cooperation.* PhD thesis, Göteborg University, 1976.

[6] J. Allwood. On dialogue cohesion. *Gothenburg Papers in Theoretical Linguistics*, 65, 1992.

[7] J. Allwood and L. Cerrato. A study of gestural feedback expressions. In P. Paggio, K. Jokinen, and A. Jönsson, editors, *Proceedings of the First Nordic Symposium on Multimodal Communication*, pages 7–22. Citeseer, 2003.

[8] J. Allwood, J. Nivre, and E. Ahlsén. On the semantics and pragmatics of linguistic feedback. *Journal of semantics*, 9(1):1–26, 1992.

[9] J.L. Austin. *How to Do Things With Words.* Oxford University Press, 1962.

[10] R.J. Beun, M. Baker, and M. Reiner, editors. *Dialogue and Instruction: Modeling Interaction in Intelligent Tutoring Systems.* Springer, 1995.

[11] R.J. Beun and H. Bunt. Multimodal cooperative communication. In *Cooperative Multimodal Communication*, pages 1–10. Springer, 2001.

[12] R.J. Beun and R.M. van Eijk. Repairing conceptual mismatches in human-computer dialogue. *Discourse processes*, 44(3):213–243, 2007.

[13] R.J. Beun, R.M. van Eijk, J-J.Ch. Meyer, and N.L. Vergunst. A computational approach to the interpretation of indirect speech acts. In V.P. Guerrero-Bote, editor, *International Conference on Multidisciplinary Information Sciences and Technologies*, pages 311–315, Mérida, 2006. Open Institute of Knowledge.

[14] R.J. Beun and P. Piwek. Pragmatische features in DenK: Pragtags. *DenK report*, 97/29, 1997.

[15] T. Bickmore and J. Cassell. Social dialogue with embodied conversational agents. In C.J. van Kuppevelt, L. Dybkjaer, and N. Bernsen, editors, *Natural, Intelligent and Effective Interaction with Multimodal Dialogue Systems*. Kluwer Academic, 2004.

[16] D. Bohus. *Error awareness and recovery in task-oriented spoken dialogue systems*. PhD thesis, PhD Thesis proposal, Carnegie Mellon University, Pittsburgh, 2004.

[17] M.L. Bourguet. Towards a taxonomy of error handling strategies in recognition-based multi-modal human–computer interfaces. *Signal Processing*, 86(12):3625–3643, 2006.

[18] H.P. Branigan, M.J. Pickering, and A.A. Cleland. Syntactic coordination in dialogue. *Cognition*, 75:B13–B25, 2000.

[19] M.E. Bratman. *Intentions, Plans and Practical Reasoning*. Harvard University Press, Cambridge, MA, 1987.

[20] C. Breazeal. Robot in society: Friend or appliance? In *Proceedings of the Autonomous Agents Workshop on Emotion-Based Agent Architectures*, pages 18–26, 1999.

[21] C. Breazeal. *Designing sociable robots*. MIT Press, 2002.

[22] C. Breazeal and J. Velásquez. Toward teaching a robot 'infant' using emotive communication acts. In B. Edmonds and K. Dautenhahn, editors, *Proceedings of the Simulated Adaptive Behavior Workshop on Socially Situated Intelligence*, pages 25–40, 1998.

[23] A.J.N. van Breemen. iCat: experimenting with animabotics. In *Proceedings of the AISB 2005 Creative Robotics Symposium*, 2005.

[24] P. Brown and S.C. Levinson. *Politeness: Some universals in language usage*. Cambridge University Press, 1987.

[25] H. Bunt. Context and dialogue control. *Think Quarterly*, 3(1):19–31, 1994.

[26] H. Bunt. Dynamic interpretation and dialogue theory. In M.M. Taylor, D.G. Bouwhuis, and F. Néel, editors, *The Structure of Multimodal Dialogue*, volume 2, pages 139–166. Elsevier, 1995.

[27] H. Bunt. Dialogue pragmatics and context specification. *Abduction, Belief and Context in Dialogue: studies in computational pragmatics*, pages 81–150, 2000.

[28] H. Bunt. The DIT++ taxonomy for functional dialogue markup. In D. Heylen, C. Pelachaud, R. Catizone, and D. Traum, editors, *Proceedings of the AAMAS 2009 Workshop "Towards a Standard Markup Language for Embodied Dialogue Acts"*, pages 13–24, 2009.

[29] H. Bunt, R. Ahn, R.J. Beun, T. Borghuis, and K. van Overveld. Multimodal cooperation with the DenK system. *Multimodal Human-Computer Communication: systems, techniques and experiments*, pages 39–67, 1998.

[30] H. Bunt, J. Alexandersson, J. Carletta, J.W. Choe, A.C. Fang, K. Hasida, K. Lee, V. Petukhova, A. Popescu-Belis, L. Romary, C. Soria, and D. Traum. Towards an ISO standard for dialogue act annotation. In *Proceedings of LREC 2010, the Seventh International Conference on Language Resources and Evaluation*, pages 2548–2558, 2010.

[31] H. Bunt, R. Morante, and S. Keizer. An empirically based computational model of grounding in dialogue. In *Proceedings of the 8th SIGDIAL Conference on Discourse and Dialogue*, pages 283–290, 2007.

[32] H.C. Bunt. Information dialogues as communicative action in relation to partner modelling and information processing. In M.M. Taylor, D.G. Bouwhuis, and F. Néel, editors, *The structure of multimodal dialogue*, volume 1, pages 47–74. Elsevier, 1989.

[33] H.C. Bunt. Dialogue control functions and interaction design. *Dialogue and Instruction*, pages 197–214, 1995.

[34] C.S. Butler. Interpersonal meaning in the noun phrase. *The Noun Phrase in Functional Discourse Grammar*, pages 221–261, 2008.

[35] L. Carlson. *Dialogue games: An approach to discourse analysis.* Reidel, 1983.

[36] J. Cassell, T. Bickmore, L. Campbell, H. Vilhjálmsson, and H. Yan. More than just a pretty face: Conversational protocols and the affordances of embodiment. *Knowledge-Based Systems*, 14(1-2):55–64, 2001.

[37] J. Cassell, H.H. Vilhjálmsson, and T. Bickmore. BEAT: the behavior expression animation toolkit. In *Proceedings of the 28th annual conference on Computer graphics and interactive techniques*, pages 477–486, 2001.

[38] C. Castelfranchi. Modelling social action for AI agents. *Artificial Intelligence*, 103:157–182, 1998.

[39] C. Castelfranchi and R. Falcone. From task delegation to role delegation. In *Lecture Notes in Artificial Intelligence*, pages 278–289. Springer, 1997.

[40] H.H. Clark. *Using Language.* Cambridge University Press, Cambridge, MA, 1996.

[41] P.R. Cohen and H.J. Levesque. Intention is choice with commitment. *Artificial Intelligence*, 42(2-3):213–261, 1990.

[42] P.R. Cohen and H.J. Levesque. Confirmations and joint action. In *Proceedings of the 12th international joint conference on Artificial intelligence*, pages 951–957, 1991.

[43] P.R. Cohen, J.L. Morgan, and M.E. Pollack. *Intentions in Communication*. MIT Press, 1990.

[44] P.R. Cohen and C.R. Perrault. Elements of a plan-based theory of speech acts. *Cognitive Science: A Multidisciplinary Journal*, 3(3):177–212, 1979.

[45] K.M. Colby, S. Weber, and F.D. Hilf. Artificial paranoia. *Artificial Intelligence*, 2:1–25, 1971.

[46] J.H. Connolly. The question of discourse representation in functional discourse grammar. *A new architecture for Functional Grammar*, pages 89–116, 2004.

[47] H.G.W. van Dam. *Dialogue Acts in GUIs*. PhD thesis, TU Eindhoven, 2006.

[48] M.M. Dastani, D. Hobo, and J.-J.Ch. Meyer. Practical extensions in agent programming languages. In *AAMAS '07: Proceedings of the 6th international joint conference on Autonomous agents and multiagent systems*, pages 1–3. ACM Press, 2007.

[49] S.C. Dik. *The theory of functional grammar*. Mouton de Gruyter, 1997.

[50] P. Doherty and J.-J.Ch. Meyer. Towards a delegation framework for aerial robotic mission scenarios. *Cooperative Information Agents XI*, pages 5–26, 2007.

[51] A.F. Dragoni, P. Giorgini, and L. Serafini. Mental states recognition from communication. *Journal of Logic and Computation*, 12(1):119–136, 2002.

[52] C. Fellbaum. *WordNet: An electronic lexical database*. MIT Press, 1998.

[53] G. Ferguson, J. Allen, and B. Miller. TRAINS-95: Towards a mixed-initiative planning assistant. In *Proceedings of the Third Conference on AI Planning Systems*, pages 70–77, 1996.

[54] G. Ferguson and J.F. Allen. TRIPS: An integrated intelligent problem-solving assistant. In *Proceedings of the National Conference on Artificial Intelligence*, pages 567–573, 1998.

[55] Foundation for Intelligent Physical Agents. FIPA-ACL communicative act library specification. Website, 2002. `http://www.fipa.org/specs/fipa00037/SC00037J.html`.

[56] FormalSystems. RecipeML, 2000. `http://www.formatdata.com/recipeml/`.

[57] J. Foster and C. Vogel. Parsing ill-formed text using an error grammar. *Artificial Intelligence Review*, 21(3):269–291, 2004.

[58] R.E. Frederking. Grice's maxims: "do the right thing". In *Computational Implicature workshop at the AAAI-96 Spring Symposium Series*, pages 21–26, 1996.

[59] S. Garrod and A. Anderson. Saying what you mean in dialogue: a study in conceptual and semantic co-ordination. *Cognition*, 27(2):181–218, 1987.

[60] S. Garrod and M.J. Pickering. Why is conversation so easy? *Trends in Cognitive Science*, 8(1):8–11, 2004.

[61] G. Gazdar. *Pragmatics: Implicature, Presupposition and Logical Form*. Academic Press, New York, 1979.

[62] E. Goffman. Replies and responses. *Language in society*, 5(03):257–313, 1976.

[63] B. Goodman, A. Soller, F. Linton, and R. Gaimari. Encouraging student reflection and articulation using a learning companion. In *International Journal of Artificial Intelligence in Education*. IOS Press, 1997.

[64] A.C. Graesser, X. Hu, and D.S. McNamara. Computerized learning environments that incorporate research in discourse psychology, cognitive science, and computational linguistics. *Experimental cognitive psychology and its applications: Festschrift in honor of Lyle Bourne, Walter Kintsch, and Thomas Landauer*, pages 183–194, 2005.

[65] A.C. Graesser, S. Lu, G.T. Jackson, H. Mitchell, M. Ventura, A. Olney, and M.M. Louwerse. Autotutor: A tutor with dialogue in natural language. *Behavioral Research Methods, Instruments, and Computers*, 36(2):180–193, 2004.

[66] A. Gravano, S. Benus, J. Hirschberg, S. Mitchell, and I. Vovsha. Classification of discourse functions of affirmative words in spoken dialogue. In *Proceedings of Interspeech 2007*, pages 1613–1616, 2007.

[67] H.P. Grice. Meaning. *Philosophical Review*, 66(3):377–388, 1957.

[68] H.P. Grice. Logic and conversation. In P. Cole and J.L. Morgan, editors, *Syntax and Semantics 3: Speech Acts*. Academic Press, New York, 1975.

[69] H.P. Grice. *Studies in the Way of Words*. Harvard University Press, Cambridge, MA, 1989.

[70] B. Grosz and C. Sidner. Plans for discourse. In P.R. Cohen, J. Morgan, and M.E. Pollack, editors, *Intentions and Plans in Communication and Discourse*. MIT Press, 1990.

[71] B.J. Grosz and S. Kraus. Collaborative plans for complex group action. *Artificial Intelligence*, 86(2):269–357, 1996.

[72] R. Hagenaars. The virtual recipe assistant. Bachelor's Thesis, Hogeschool Utrecht, 2010.

[73] P.A. Heeman, M. Johnston, J. Denney, and E. Kaiser. Beyond structured dialogues: Factoring out grounding. In *Fifth International Conference on Spoken Language Processing*, 1998.

[74] K. Hengeveld and J.L. Mackenzie. *Functional Discourse Grammar: A typologically-based theory of language structure.* Oxford University Press, Oxford, 2008.

[75] J. Hintikka and E. Saarinen. Information-seeking dialogues: Some of their logical properties. *Studia Logica*, 38(4):355–363, 1979.

[76] K. Hone. Empathic agents to reduce user frustration: The effects of varying agent characteristics. *Interacting with Computers*, 18(2):227–245, 2006.

[77] S. Kopp, B. Jung, N. Lessmann, and I. Wachsmuth. Max – a multimodal assistant in virtual reality construction. *KI – Künstliche Intelligenz*, 4(3):11–17, 2003.

[78] S. Kopp and I. Wachsmuth. Model-based animation of coverbal gesture. In *Proceedings of Computer Animation*, pages 252–257. IEEE Press, 2002.

[79] C. Kroon. *Discourse Particles in Latin. A study of nam, enim, autem, vero and at.* Gieben, 1995.

[80] R. Lamers. Automated natural language processing using functional discourse grammar. Master's Thesis Cognitive Artificial Intelligence, 2009.

[81] S. Larsson and D.R. Traum. Information state and dialogue management in the trindi dialogue move engine toolkit. *Natural language engineering*, 6(3&4):323–340, 2000.

[82] A. Lavie. GLR*: A robust grammar-focused parser for spontaneously spoken language. *Proceedings of ESSLLI-96 Workshop on Robust Parsing*, 1996.

[83] H.J. Lebbink. *Dialogue and decision games for information exchanging agents.* PhD thesis, Utrecht University, Utrecht, The Netherlands, 2006.

[84] O. Lemon, L. Cavedon, and B. Kelly. Managing dialogue interaction: A multi-layered approach. In *Proceedings of the 4th SIGDIAL Workshop on Discourse and Dialogue*, pages 168–177, 2003.

[85] A.M. Leslie. Spatiotemporal continuity and the perception of causality in infants. *Perception*, 13(3):287–305, 1984.

[86] V. Lesser, K. Decker, T. Wagner, N. Carver, A. Garvey, B. Horling, D. Neiman, R. Podorozhny, M. Prasad, A. Raja, R. Vincent, P. Xuan, and X. Zhang. Evolution of the GPGP/TAEMS domain-independent coordination framework. *Autonomous Agents and Multi-Agent Systems*, 9(1):87–143, 2004.

[87] W.J.M. Levelt. *Speaking: From Intention to Articulation.* MIT Press, Cambridge, MA, 1989.

[88] S.C. Levinson. *Pragmatics*. Cambridge University Press, Cambridge, 1983.

[89] K.F. McCoy. Reasoning on a highlighted user model to respond to misconceptions. *Computational Linguistics*, 14(3):52–63, 1988.

[90] K.F. McCoy. Generating context-sensitive responses to object-related misconceptions. *Artificial Intelligence*, 41(2):157–195, 1989.

[91] M.F. McTear. Using the CSLU toolkit for practicals in spoken dialogue technology. In *MATISSE Workshop on Method and Tool Innovations for Speech Science Education*, 1999.

[92] M.F. McTear. *Spoken dialogue technology: toward the conversational user interface*. Springer, 2004.

[93] W. Menzel. Robust processing of natural language. *KI-95: Advances in Artificial Intelligence*, pages 19–34, 1995.

[94] R. Morante Vallejo. *Computing meaning in interaction*. University of Tilburg, 2007.

[95] K. Morik. User models and conversational settings: modeling the user's wants. In A. Kobsa and W. Wahlster, editors, *User Models in Dialog Systems*, pages 364–385. Springer Verlag, 1989.

[96] D.A. Norman. Categorization of action slips. *Psychological review*, 88(1):1–15, 1981.

[97] S. Oviatt. Ten myths of multimodal interaction. *Communications of the ACM*, 1999.

[98] T. Paek. Toward a taxonomy of communication errors. In *Workshop on Error Handling in Spoken Dialogue Systems*, pages 53–58, 2003.

[99] P. Piwek and E. Krahmer. Presuppositions in context: Constructing bridges. *Formal aspects of context*, 20, 2000.

[100] H. Prakken. Formal systems for persuasion dialogue. *The Knowledge Engineering Review*, 21(02):163–188, 2006.

[101] L.M. Reeves, J. Lai, J.A. Larson, S. Oviatt, T.S. Balaji, S. Buisine, P. Collings, P. Cohen, B. Kraal, J.C. Martin, M. McTear, T.V. Raman, K.M. Stanney, H. Su, and Q.Y. Wang. Guidelines for multimodal user interface design. *Communications of the ACM*, 47(1):57–59, 2004.

[102] J. Rickel and W.L. Johnson. Task-oriented collaboration with embodied agents in virtual worlds. In J. Cassell, J. Sullivan, and S. Prevost, editors, *Embodied Conversational Agents*, pages 95–122. MIT Press, 2000.

[103] V. Rieser and J.D. Moore. Implications for generating clarification requests in task-oriented dialogues. In *Proceedings of the 43rd Annual Meeting on Association for Computational Linguistics*, pages 239–246. ACL, 2005.

[104] J. Ruppenhofer, M. Ellsworth, M.R.L. Petruck, C.R. Johnson, and J. Scheffczyk. FrameNet II: Extended theory and practice, 2010. `http://framenet.icsi.berkeley.edu/book/book.pdf`.

[105] H. Sacks, E.A. Schegloff, and G. Jefferson. A simplest systematics for the organization of turn-taking for conversation. *Language*, 50(4):696–735, 1974.

[106] B. Scasselati. Theory of mind for a humanoid robot. *Autonomous Robots*, 12(1):13–24, 2002.

[107] R. Schäfer. Multidimensional probabilistic assessment of interest and knowledge in a noncooperative dialog situation. In *Proceedings of ABIS-94: GI Workshop on Adaptivity and User Modeling in Interactive Software Systems*, pages 46–62, 1994.

[108] E. A. Schegloff and H. Sacks. Opening up closings. *Semiotica*, 8(4):289–327, 1973.

[109] L. Seabra Lopes and A. Teixeira. Human-robot interaction through spoken language dialogue. In *Proceedings of the International Conference on Intelligent Robots and Systems*, pages 528–534, 2000.

[110] J.R. Searle. *Speech acts: An essay in the philosophy of language*. Cambridge University Press, 1969.

[111] J.R. Searle. Indirect speech acts. In P. Cole and J.L. Morgan, editors, *Syntax and Semantics, Volume 3: Speech Acts*, pages 59–82. Academic Press, 1975.

[112] E. Shriberg, J. Bear, and J. Dowding. Automatic detection and correction of repairs in human-computer dialog. In *Proceedings of the Workshop on Speech and Natural Language*, pages 419–424, 1992.

[113] S. Steidl, C. Hacker, C. Ruff, A. Batliner, E. Nöth, and J. Haas. Looking at the last two turns, i'd say this dialogue is doomed – measuring dialogue success. In *Text, Speech and Dialogue*, pages 629–636. Springer, 2004.

[114] A. Stein and E. Maier. Structuring collaborative information-seeking dialogues. *Knowledge-Based Systems*, 8(2-3):82–93, 1995. Special Issue on Human-Computer Collaboration.

[115] B.R. Steunebrink. *The Logical Structure of Emotions*. PhD thesis, Universiteit Utrecht, 2010.

[116] B.R. Steunebrink, N.L. Vergunst, Chr. P. Mol, F.P.M. Dignum, M.M. Dastani, and J-J.Ch. Meyer. A generic architecture for a companion robot. In J. Filipe, J.A. Cetto, and J.-L. Ferrier, editors, *Proceedings of the 5th International Conference on Informatics in Control, Automation and Robotics*, pages 315–321, Madeira, Portugal: Funchal, 2008.

[117] M. Stone. Linguistic representation and gricean inference. In *Proceedings of the International Workshop on Computational Semantics*, pages 5–21, 2003.

[118] P.F. Strawson. On referring. *Mind*, 59(235):320–344, 1950.

[119] J.A. Taylor, J. Carletta, and C. Mellish. Requirements for belief models in cooperative dialogue. *User Modeling and User-Adapted Interaction*, 6(1):23–68, 1996.

[120] M.M. Taylor. Layered protocols for computer-human dialogue. i: Principles. *International Journal of Man-Machine Studies*, 28(2-3):175–218, 1988.

[121] M.M. Taylor. Response timing in layered protocols: a cybernetic view of natural dialogue. *The structure of multimodel dialogue*, pages 159–172, 1989.

[122] K.R. Thórisson. Layered action control in communicative humanoids. In *Proceedings of Computer Graphics Europe*, pages 134–143, 1997.

[123] D. Traum and S. Larsson. The information state approach to dialogue management. *Current and New Directions in Discourse and Dialogue*, pages 325–353, 2003.

[124] D.R. Traum. *A computational theory of grounding in natural language conversation.* PhD thesis, University of Rochester, 1994.

[125] M. Turunen and J. Hakulinen. Agent-based error handling in spoken dialogue systems. In *Seventh European Conference on Speech Communication and Technology*, 2001.

[126] W. van der Hoek, B. van Linder, and J.-J.Ch. Meyer. An integrated modal approach to rational agents. *Foundations of rational agency*, pages 37–75, 1998.

[127] S. Verberne. *In Search of the Why.* PhD thesis, Tilburg University, 2010.

[128] N. Vergunst, Lamers R., and Dignum F. Using functional discourse grammar in an agent-based dialogue system. In M. Dall'Aglio-Hattnher, M. Chondrogianni, and K. Hengeveld, editors, *13th International Conference on Functional Grammar*, 2008.

[129] N.L. Vergunst, B.R. Steunebrink, M.M. Dastani, F.P.M. Dignum, and J.-J.Ch. Meyer. Towards programming multimodal dialogues. In *Proceedings of the workshop on communication between human and artificial agents (CHAA'07)*, 2007.

[130] M. Walker, I. Langkilde, J. Wright, A. Gorin, and D. Litman. Learning to predict problematic situations in a spoken dialogue system: experiments with how may i help you? In *Proceedings of the North American meeting of the Association for Computational Linguistics conference*, 2000.

[131] J. Weizenbaum. ELIZA – a computer program for the study of natural language communication between man and machine. *Communications of the ACM*, 9(1):36–45, 1966.

[132] Y. Wilks, editor. *Close Engagements with Artificial Companions: Key Social, Psychological, Ethical and Design Issues*. John Benjamins Publishing Company, 2010.

[133] D. Wilson and D. Sperber. Relevance theory. In G. Ward and L. Horn, editors, *Handbook of pragmatics*. Blackwell, Oxford, 2002.

[134] T. Winograd. *Understanding natural language*. Academic Press, Inc., 1972.

[135] M. Wooldridge. *Introduction to Multiagent Systems*. John Wiley & Sons, Inc., 2002.

[136] M. Wooldridge and N.R. Jennings. Intelligent agents: Theory and practice. *Knowledge engineering review*, 10(2):115–152, 1995.

Summary

Planning dialogues is a notoriously difficult task, consisting of several non-trivial components. Because of this, programming human-computer dialogues is also a complicated task. Humans are unpredictable, make mistakes, vary their pronunciation, change their minds halfway through utterances or actions, and so on. Although not all errors can be solved easily, we can at least attempt to avoid total communication breakdown.

In this thesis, we present a number of tactics to make the generation of human-computer dialogues more robust. We focus on task-oriented dialogues, since they have a relatively high level of predictability and a clear goal. More specifically, we have studied the generation of dialogues in which the robot iCat instructs the user to perform a task, most notably the preparation of a recipe that is chosen by the user at the beginning of the dialogue.

The framework that we present in this thesis is modeled as a BDI agent. Being an agent, it is autonomous, social, reactive and goal-directed. Its internals are represented according to the BDI (Belief-Desire-Intention) paradigm, which enables us to consider the program and the human user in similar terms. This allows us to simulate human-computer dialogue by representing the user as a simple BDI agent itself.

We have developed this framework by investigating human-human dialogues, principles of cooperation, and dialogue systems. Based on this, we have gathered a set of requirements that we believe any such system should adhere to. Then, we have constructed a framework for such a system, consisting of a basic system and an error handling module. We have also investigated a technique for natural language processing and generation.

In order to make the system robust and generic, we present an error handling module that is separate from the basic dialogue system. The basic dialogue system itself is only capable of producing and interpreting instructions that are part of the task at hand, based on principles from dialogue theory and cooperation. The error handling module can deal with errors such as misunderstandings and erroneous actions by the user.

Our basic framework was almost completely implemented, showing that it is indeed a valid framework. Of our error handling module, only some aspects were implemented, leaving the rest of the implementation and validation for future work.

As a preliminary validation of our framework, we have presented a dialogue in which errors are detected and repaired, which can be constructed with the rules that we present in this thesis. A more definitive validation can be given if we implement the complete system and test it with human users. The value of the framework can be fully assessed after testing whether users can perform tasks better, faster and easier with a task assistant than with, for example, a manual.

Samenvatting

Het plannen van dialogen is buitengewoon lastig. Dialogen bestaan immers uit verschillende niet-triviale componenten. Het programmeren van mens-computerdialogen is daarom ook een complexe taak. Mensen zijn onvoorspelbaar, maken fouten, variëren de uitspraak van woorden, veranderen van gedachten halverwege zinnen of acties, en zo verder. Hoewel niet alle fouten gemakkelijk kunnen worden opgelost, kunnen we in ieder geval proberen te voorkomen dat communicatie compleet mislukt.

In dit proefschrift worden een aantal tactieken gepresenteerd om de generatie van mens-computerdialogen meer robuust te maken. We richten ons op taakgerichte dialogen, aangezien zij een relatief hoge mate van voorspelbaarheid en een duidelijk doel hebben. Meer in het bijzonder behandelen we het genereren van dialogen waarin de robot iCat de gebruiker instrueert een taak uit te voeren, namelijk het bereiden van een recept dat door de gebruiker gekozen is aan het begin van de dialoog.

Het framework dat we in dit proefschrift presenteren, is gemodelleerd als een BDI (Belief-Desire-Intention) agent. Een agent is autonoom, sociaal, reactief en doelgericht. Zijn innerlijke werking wordt gerepresenteerd volgens het BDI-paradigma, dat ons in staat stelt het programma en de menselijke gebruiker in soortgelijke termen te beschouwen. Dit stelt ons in staat mens-computerdialoog te simuleren door de gebruiker als een eenvoudige BDI-agent te representeren.

Om dit framework te ontwerpen, hebben we mens-mensdialogen, principes van samenwerking en dialoogsystemen onderzocht. Op basis hiervan hebben we een pakket van eisen samengesteld waaraan een dergelijk systeem moet voldoen. Vervolgens hebben we een conceptueel framework geconstrueerd, bestaande uit een basissysteem en een foutafhandelingmodule. We hebben ook een techniek voor de interpretatie en productie van natuurlijke taal onderzocht.

Om het systeem robuust en generiek te maken, presenteren we een foutafhandelingmodule die gescheiden is van het basisdialoogsysteem. Het fundamentele dialoogsysteem zelf is alleen geschikt voor het produceren en interpreteren van instructies die deel uitmaken van de taak in kwestie, gebaseerd op de beginselen van de dialoogtheorie en principes van samenwerking. De foutafhandelingmodule kan omgaan met fouten als misverstanden en onjuiste handelingen van de gebruiker.

Het framework is bijna volledig geïmplementeerd, waaruit blijkt dat het inderdaad een valide framework is. Van de foutafhandelingmodule zijn vooralsnog slechts enkele

aspecten geïmplementeerd, waardoor de rest van de implementatie en validatie ruimte bieden voor toekomstig onderzoek. Als een eerste validatie van ons framework hebben wij een dialoog gepresenteerd waarin fouten worden opgespoord en gerepareerd. Deze dialoog kan worden geconstrueerd met de regels die in dit proefschrift gepresenteerd worden. Een meer definitieve validatie kan worden gegeven als het systeem compleet wordt geïmplementeerd en wordt getest met menselijke gebruikers. De waarde van het framework kan volledig worden beoordeeld door te testen of gebruikers taken beter, sneller en makkelijker kunnen uitvoeren met een taak-assistent dan met, bijvoorbeeld, een handleiding.

Dankwoord

- Mag ik u hartelijk bedanken voor deze fijne voorstelling?
- Gaat uw gang.
- Ik wil u hartelijk bedanken voor deze fijne voorstelling!
- Gaat uw gang.
- Hartelijk bedankt voor deze fijne voorstelling!
- Geen dank.
- En ik had het nog zo gevraagd!
- Ja, d'r is niks meer aan te doen.
- Oh, nou, u wordt bedankt!
- Ja, dat idee kreeg ik al!

Herman Finkers,
Het meisje van de slijterij

Er zijn talloze overeenkomsten tussen het voeren van een dialoog en het schrijven van een proefschrift. Beide hebben over het algemeen een hoop woorden nodig, zijn lastig te plannen, verre van triviaal, en onmogelijk om in je eentje te doen. Ik wil hier even de tijd nemen om enkele woorden van dank uit te spreken voor de vele helpende handen die, direct of indirect, dit proefschrift mede mogelijk hebben gemaakt.

Allereerst wil ik mijn begeleiders hartelijk bedanken. John-Jules heeft mij altijd kunnen motiveren, ook in de tijden dat mijn onderzoek wat minder soepel liep, en wist hierbij altijd enthousiast en geïnteresseerd te blijven. Robbert-Jan is er erg goed in geslaagd om mij scherp te maken en te houden: geen enkel woord of zinsdeel dat niet precies kloppend was is aan zijn arendsoog ontsnapt. Dat heeft me absoluut geholpen om beter te leren schrijven. Soms was ik wat overweldigd door mijn begeleiders, maar dan was er altijd weer Rogier die vroeg: "Maar wat vind je er zelf eigenlijk van?" Zijn tip om discussie te zien als een Oosterse vechtsport (meebuigen, niet forceren) heeft me geholpen om mijn mannetje te kunnen staan in wetenschappelijke discussies. Ook wil ik Frank Dignum bedanken voor de vele goede inhoudelijke tips en adviezen.

Mijn leescommissie wil ik graag hartelijk bedanken voor de feedback waardoor ik net even de puntjes op de i heb kunnen zetten.

Verder wil ik met name grote waardering uitspreken voor mijn collega's Tom, Susan, Hado, Maaike, Joost, Eric en Nick, voor alle gezelligheid, de borrels en de uitjes. Speciale woorden van dank voor Bas en Chris, die mij al die jaren als kamergenoot hebben weten te verdragen (en andersom). Ook de andere collega's uit de IS-groep, collega's uit andere vakgroepen, SIKS-collega's en vele conferentiegenoten wil ik bedanken voor alle gezelligheid en leuke (al dan niet inhoudelijke) discussies.

Annemarie, Renske, Lisanne, Jaii, Lotte en alle andere vrienden en kennissen: bedankt! Aan de ene kant moesten jullie soms op je tenen lopen om niet per ongeluk 'het p-woord' te noemen, en aan de andere kant werden jullie minstens net zo vaak door mij overstelpt met (al dan niet onbegrijpelijk) gebabbel over mijn onderzoek. Ik ben heel blij dat jullie me nog steeds niet zat zijn na al die tijd.

Cobi, Dick, Arjan en de rest van de familie: bedankt voor de overweldigende steun op alle gebieden. Ik wil ook nog mijn oprechte bewondering uitspreken voor de onuitputtelijke pogingen om mijn onderzoek inhoudelijk te begrijpen. Dat doen niet veel mensen buiten het vakgebied jullie na.

Mijn grootste dank gaat uit naar Nils. I hold a whole bool from you.

Utrecht,
23 januari 2011

Nieske Vergunst

SIKS Dissertation Series

1998

1998-1 | **Johan van den Akker** (CWI), *DEGAS - An Active, Temporal Database of Autonomous Objects.*

1998-2 | **Floris Wiesman** (UM), *Information Retrieval by Graphically Browsing Meta-Information.*

1998-3 | **Ans Steuten** (TUD), *A Contribution to the Linguistic Analysis of Business Conversations within the Language/Action Perspective.*

1998-4 | **Dennis Breuker** (UM), *Memory versus Search in Games.*

1998-5 | **E.W. Oskamp** (RUL), *Computerondersteuning bij Straftoemeting.*

1999

1999-1 | **Mark Sloof** (VU), *Physiology of Quality Change Modelling; Automated modelling of Quality Change of Agricultural Products.*

1999-2 | **Rob Potharst** (EUR), *Classification using decision trees and neural nets.*

1999-3 | **Don Beal** (UM), *The Nature of Minimax Search.*

1999-4 | **Jacques Penders** (UM), *The practical Art of Moving Physical Objects.*

1999-5 | **Aldo de Moor** (KUB), *Empowering Communities: A Method for the Legitimate User-Driven Specification of Network Information Systems.*

1999-6 | **Niek J.E. Wijngaards** (VU), *Redesign of compositional systems.*

1999-7 | **David Spelt** (UT), *Verification support for object database design.*

1999-8 | **Jacques H.J. Lenting** (UM), *Informed Gambling: Conception and Analysis of a Multi-Agent Mechanism for Discrete Reallocation.*

2000

2000-1 | **Frank Niessink** (VU), *Perspectives on Improving Software Maintenance.*

2000-2 | **Koen Holtman** (TUE), *Prototyping of CMS Storage Management.*

2000-3 | **Carolien M.T. Metselaar** (UVA), *Sociaal-organisatorische gevolgen van kennistechnologie; een procesbenadering en actorperspectief.*

2000-4 | **Geert de Haan** (VU), *ETAG, A Formal Model of Competence Knowledge for User Interface Design.*

2000-5 | **Ruud van der Pol** (UM), *Knowledge-based Query Formulation in Information Retrieval.*

2000-6 | **Rogier van Eijk** (UU), *Programming Languages for Agent Communication.*

2000-7 | **Niels Peek** (UU), *Decision-theoretic Planning of Clinical Patient Management.*

2000-8 | **Veerle Coup** (EUR), *Sensitivity Analyis of Decision-Theoretic Networks.*

2000-9 | **Florian Waas** (CWI), *Principles of Probabilistic Query Optimization.*

2000-10 | **Niels Nes** (CWI), *Image Database Management System Design Considerations, Algorithms and Architecture.*

2000-11 | **Jonas Karlsson** (CWI), *Scalable Distributed Data Structures for Database Management.*

2001

2001-1 | **Silja Renooij** (UU), *Qualitative Approaches to Quantifying Probabilistic Networks.*

2001-2 | **Koen Hindriks** (UU), *Agent Programming Languages: Programming with Mental Models.*

2001-3 | **Maarten van Someren** (UvA), *Learning as problem solving.*

2001-4 | **Evgueni Smirnov** (UM), *Conjunctive and Disjunctive Version Spaces with Instance-Based Boundary Sets.*

2001-5 | **Jacco van Ossenbruggen** (VU), *Processing Structured Hypermedia: A Matter of Style.*

2001-6 | **Martijn van Welie** (VU), *Task-based User Interface Design.*

2001-7 | **Bastiaan Schonhage** (VU), *Diva: Architectural Perspectives on Information Visualization.*

2001-8 | **Pascal van Eck** (VU), *A Compositional Semantic Structure for Multi-Agent Systems Dynamics.*

2001-9 | **Pieter Jan 't Hoen** (RUL), *Towards Distributed Development of Large Object-Oriented Models, Views of Packages as Classes.*

2001-10 | **Maarten Sierhuis** (UvA), *Modeling and Simulating Work Practice BRAHMS: a multiagent modeling and simulation language for work practice analysis and design.*

2001-11 | **Tom M. van Engers** (VUA), *Knowledge Management: The Role of Mental Models in Business Systems Design.*

2002

2002-01 | **Nico Lassing** (VU), *Architecture-Level Modifiability Analysis.*

2002-02 | **Roelof van Zwol** (UT), *Modelling and searching web-based document collections.*

2002-03 | **Henk Ernst Blok** (UT), *Database Optimization Aspects for Information Retrieval.*

2002-04 | **Juan Roberto Castelo Valdueza** (UU), *The Discrete Acyclic Digraph Markov Model in Data Mining.*

2002-05 | **Radu Serban** (VU), *The Private Cyberspace Modeling Electronic Environments inhabited by Privacy-concerned Agents.*

2002-06 | **Laurens Mommers** (UL), *Applied legal epistemology; Building a knowledge-based ontology of the legal domain.*

2002-07 | **Peter Boncz** (CWI), *Monet: A Next-Generation DBMS Kernel For Query-Intensive Applications.*

2002-08 | **Jaap Gordijn** (VU), *Value Based Requirements Engineering: Exploring Innovative E-Commerce Ideas.*

2002-09 | **Willem-Jan van den Heuvel** (KUB), *Integrating Modern Business Applications with Objectified Legacy Systems.*

2002-10 | **Brian Sheppard** (UM), *Towards Perfect Play of Scrabble.*

2002-11 | **Wouter C.A. Wijngaards** (VU), *Agent Based Modelling of Dynamics: Biological and Organisational Applications.*

2002-12 | **Albrecht Schmidt** (UVA), *Processing XML in Database Systems.*

2002-13 | **Hongjing Wu** (TUE), *A Reference Architecture for Adaptive Hypermedia Applications.*

2002-14 | **Wieke de Vries** (UU), *Agent Interaction: Abstract Approaches to Modelling, Programming and Verifying Multi-Agent Systems.*

2002-15 | **Rik Eshuis** (UT), *Semantics and Verification of UML Activity Diagrams for Workflow Modelling.*

2002-16 | **Pieter van Langen** (VU), *The Anatomy of Design: Foundations, Models and Applications.*

2002-17 | **Stefan Manegold** (UVA), *Understanding, Modeling, and Improving Main-Memory Database Performance.*

2003

2003-01 | **Heiner Stuckenschmidt** (VU), *Onotology-Based Information Sharing In Weakly Structured Environments.*

2003-02 | **Jan Broersen** (VU), *Modal Action Logics for Reasoning About Reactive Systems.*

2003-03 | **Martijn Schuemie** (TUD), *Human-Computer Interaction and Presence in Virtual Reality Exposure Therapy.*

2003-04 | **Milan Petkovic** (UT), *Content-Based Video Retrieval Supported by Database Technology.*

2003-05 | **Jos Lehmann** (UVA), *Causation in Artificial Intelligence and Law - A modelling approach.*

2003-06 | **Boris van Schooten** (UT), *Development and specification of virtual environments.*

2003-07 | **Machiel Jansen** (UvA), *Formal Explorations of Knowledge Intensive Tasks.*

2003-08 | **Yongping Ran** (UM), *Repair Based Scheduling.*

2003-09 | **Rens Kortmann** (UM), *The resolution of visually guided behaviour.*

2003-10 | **Andreas Lincke** (UvT), *Electronic*

Business Negotiation: Some experimental studies on the interaction between medium, innovation context and culture.

2003-11 | **Simon Keizer** (UT), *Reasoning under Uncertainty in Natural Language Dialogue using Bayesian Networks.*

2003-12 | **Roeland Ordelman** (UT), *Dutch speech recognition in multimedia information retrieval.*

2003-13 | **Jeroen Donkers** (UM), *Nosce Hostem - Searching with Opponent Models.*

2003-14 | **Stijn Hoppenbrouwers** (KUN), *Freezing Language: Conceptualisation Processes across ICT-Supported Organisations.*

2003-15 | **Mathijs de Weerdt** (TUD), *Plan Merging in Multi-Agent Systems.*

2003-16 | **Menzo Windhouwer** (CWI), *Feature Grammar Systems - Incremental Maintenance of Indexes to Digital Media Warehouses.*

2003-17 | **David Jansen** (UT), *Extensions of Statecharts with Probability, Time, and Stochastic Timing.*

2003-18 | **Levente Kocsis** (UM), *Learning Search Decisions.*

2004

2004-01 | **Virginia Dignum** (UU), *A Model for Organizational Interaction: Based on Agents, Founded in Logic.*

2004-02 | **Lai Xu** (UvT), *Monitoring Multi-party Contracts for E-business.*

2004-03 | **Perry Groot** (VU), *A Theoretical and Empirical Analysis of Approximation in Symbolic Problem Solving.*

2004-04 | **Chris van Aart** (UVA), *Organizational Principles for Multi-Agent Architectures.*

2004-05 | **Viara Popova** (EUR), *Knowledge discovery and monotonicity.*

2004-06 | **Bart-Jan Hommes** (TUD), *The Evaluation of Business Process Modeling Techniques.*

2004-07 | **Elise Boltjes** (UM), *Voorbeeldig onderwijs; voorbeeldgestuurd onderwijs, een opstap naar abstract denken, vooral voor meisjes.*

2004-08 | **Joop Verbeek** (UM), *Politie en de Nieuwe Internationale Informatiemarkt, Grensregionale politiële gegevensuitwisseling en digitale expertise.*

2004-09 | **Martin Caminada** (VU), *For the Sake of the Argument; explorations into argument-based reasoning.*

2004-10 | **Suzanne Kabel** (UVA), *Knowledge-rich indexing of learning-objects.*

2004-11 | **Michel Klein** (VU), *Change Management for Distributed Ontologies.*

2004-12 | **The Duy Bui** (UT), *Creating emotions and facial expressions for embodied agents.*

2004-13 | **Wojciech Jamroga** (UT), *Using Multiple Models of Reality: On Agents who Know how to Play.*

2004-14 | **Paul Harrenstein** (UU), *Logic in Conflict. Logical Explorations in Strategic Equilibrium.*

2004-15 | **Arno Knobbe** (UU), *Multi-Relational Data Mining.*

2004-16 | **Federico Divina** (VU), *Hybrid Genetic Relational Search for Inductive Learning.*

2004-17 | **Mark Winands** (UM), *Informed Search in Complex Games.*

2004-18 | **Vania Bessa Machado** (UvA), *Supporting the Construction of Qualitative Knowledge Models.*

2004-19 | **Thijs Westerveld** (UT), *Using generative probabilistic models for multimedia retrieval.*

2004-20 | **Madelon Evers** (Nyenrode), *Learning from Design: facilitating multidisciplinary design teams.*

2005

2005-01 | **Floor Verdenius** (UVA), *Methodological Aspects of Designing Induction-Based Applications.*

2005-02 | **Erik van der Werf** (UM), *AI techniques for the game of Go.*

2005-03 | **Franc Grootjen** (RUN), *A Pragmatic Approach to the Conceptualisation of Language.*

2005-04 | **Nirvana Meratnia** (UT), *Towards Database Support for Moving Object data.*

2005-05 | **Gabriel Infante-Lopez** (UVA), *Two-Level Probabilistic Grammars for Natural Language Parsing.*

2005-06 | **Pieter Spronck** (UM), *Adaptive Game AI.*

2005-07 | **Flavius Frasincar** (TUE), *Hypermedia Presentation Generation for Semantic Web Information Systems.*

2005-08 | **Richard Vdovjak** (TUE), *A Model-driven Approach for Building Distributed Ontology-based Web Applications.*

2005-09 | **Jeen Broekstra** (VU), *Storage,*

Querying and Inferencing for Semantic Web Languages.

2005-10 | **Anders Bouwer** (UVA), *Explaining Behaviour: Using Qualitative Simulation in Interactive Learning Environments.*

2005-11 | **Elth Ogston** (VU), *Agent Based Matchmaking and Clustering - A Decentralized Approach to Search.*

2005-12 | **Csaba Boer** (EUR), *Distributed Simulation in Industry.*

2005-13 | **Fred Hamburg** (UL), *Een Computermodel voor het Ondersteunen van Euthanasiebeslissingen.*

2005-14 | **Borys Omelayenko** (VU), *Web-Service configuration on the Semantic Web; Exploring how semantics meets pragmatics.*

2005-15 | **Tibor Bosse** (VU), *Analysis of the Dynamics of Cognitive Processes.*

2005-16 | **Joris Graaumans** (UU), *Usability of XML Query Languages.*

2005-17 | **Boris Shishkov** (TUD), *Software Specification Based on Re-usable Business Components.*

2005-18 | **Danielle Sent** (UU), *Test-selection strategies for probabilistic networks.*

2005-19 | **Michel van Dartel** (UM), *Situated Representation.*

2005-20 | **Cristina Coteanu** (UL), *Cyber Consumer Law, State of the Art and Perspectives.*

2005-21 | **Wijnand Derks** (UT), *Improving Concurrency and Recovery in Database Systems by Exploiting Application Semantics.*

2006

2006-01 | **Samuil Angelov** (TUE), *Foundations of B2B Electronic Contracting.*

2006-02 | **Cristina Chisalita** (VU), *Contextual issues in the design and use of information technology in organizations.*

2006-03 | **Noor Christoph** (UVA), *The role of metacognitive skills in learning to solve problems.*

2006-04 | **Marta Sabou** (VU), *Building Web Service Ontologies.*

2006-05 | **Cees Pierik** (UU), *Validation Techniques for Object-Oriented Proof Outlines.*

2006-06 | **Ziv Baida** (VU), *Software-aided Service Bundling – Intelligent Methods & Tools for Graphical Service Modeling.*

2006-07 | **Marko Smiljanic** (UT), *XML schema matching – balancing efficiency and effectiveness by means of clustering.*

2006-08 | **Eelco Herder** (UT), *Forward, Back and Home Again – Analyzing User Behavior on the Web.*

2006-09 | **Mohamed Wahdan** (UM), *Automatic Formulation of the Auditor's Opinion.*

2006-10 | **Ronny Siebes** (VU), *Semantic Routing in Peer-to-Peer Systems.*

2006-11 | **Joeri van Ruth** (UT), *Flattening Queries over Nested Data Types.*

2006-12 | **Bert Bongers** (VU), *Interactivation – Towards an e-cology of people, our technological environment, and the arts.*

2006-13 | **Henk-Jan Lebbink** (UU), *Dialogue and Decision Games for Information Exchanging Agents.*

2006-14 | **Johan Hoorn** (VU), *Software Requirements: Update, Upgrade, Redesign - towards a Theory of Requirements Change.*

2006-15 | **Rainer Malik** (UU), *CONAN: Text Mining in the Biomedical Domain.*

2006-16 | **Carsten Riggelsen** (UU), *Approximation Methods for Efficient Learning of Bayesian Networks.*

2006-17 | **Stacey Nagata** (UU), *User Assistance for Multitasking with Interruptions on a Mobile Device.*

2006-18 | **Valentin Zhizhkun** (UVA), *Graph transformation for Natural Language Processing.*

2006-19 | **Birna van Riemsdijk** (UU), *Cognitive Agent Programming: A Semantic Approach.*

2006-20 | **Marina Velikova** (UvT), *Monotone models for prediction in data mining.*

2006-21 | **Bas van Gils** (RUN), *Aptness on the Web.*

2006-22 | **Paul de Vrieze** (RUN), *Fundaments of Adaptive Personalisation.*

2006-23 | **Ion Juvina** (UU), *Development of Cognitive Model for Navigating on the Web.*

2006-24 | **Laura Hollink** (VU), *Semantic Annotation for Retrieval of Visual Resources.*

2006-25 | **Madalina Drugan** (UU), *Conditional log-likelihood MDL and Evolutionary MCMC.*

2006-26 | **Vojkan Mihajlovic** (UT), *Score Region Algebra: A Flexible Framework for Structured Information Retrieval.*

2006-27 | **Stefano Bocconi** (CWI), *Vox Populi: generating video documentaries from semantically annotated media repositories.*

2006-28 | **Borkur Sigurbjornsson** (UVA), *Focused Information Access using XML Element Retrieval.*

2007

2007-01 | **Kees Leune** (UvT), *Access Control and Service-Oriented Architectures.*

2007-02 | **Wouter Teepe** (RUG), *Reconciling Information Exchange and Confidentiality: A Formal Approach.*

2007-03 | **Peter Mika** (VU), *Social Networks and the Semantic Web.*

2007-04 | **Jurriaan van Diggelen** (UU), *Achieving Semantic Interoperability in Multi-agent Systems: A Dialogue-based Approach.*

2007-05 | **Bart Schermer** (UL), *Software Agents, Surveillance, and the Right to Privacy: a Legislative Framework for Agent-enabled Surveillance.*

2007-06 | **Gilad Mishne** (UVA), *Applied Text Analytics for Blogs.*

2007-07 | **Natasa Jovanovic** (UT), *To Who It May Concern - Addressee Identification in Face-to-Face Meetings.*

2007-08 | **Mark Hoogendoorn** (VU), *Modeling of Change in Multi-Agent Organizations.*

2007-09 | **David Mobach** (VU), *Agent-Based Mediated Service Negotiation.*

2007-10 | **Huib Aldewereld** (UU), *Autonomy vs. Conformity: an Institutional Perspective on Norms and Protocols.*

2007-11 | **Natalia Stash** (TUE), *Incorporating Cognitive/Learning Styles in a General-Purpose Adaptive Hypermedia System.*

2007-12 | **Marcel van Gerven** (RUN), *Bayesian Networks for Clinical Decision Support: A Rational Approach to Dynamic Decision-Making under Uncertainty.*

2007-13 | **Rutger Rienks** (UT), *Meetings in Smart Environments; Implications of Progressing Technology.*

2007-14 | **Niek Bergboer** (UM), *Context-Based Image Analysis.*

2007-15 | **Joyca Lacroix** (UM), *NIM: a Situated Computational Memory Model.*

2007-16 | **Davide Grossi** (UU), *Designing Invisible Handcuffs. Formal investigations in Institutions and Organizations for Multi-agent Systems.*

2007-17 | **Theodore Charitos** (UU), *Reasoning with Dynamic Networks in Practice.*

2007-18 | **Bart Orriens** (UvT), *On the development an management of adaptive business collaborations.*

2007-19 | **David Levy** (UM), *Intimate relationships with artificial partners.*

2007-20 | **Slinger Jansen** (UU), *Customer Configuration Updating in a Software Supply Network.*

2007-21 | **Karianne Vermaas** (UU), *Fast diffusion and broadening use: A research on residential adoption and usage of broadband internet in the Netherlands between 2001 and 2005.*

2007-22 | **Zlatko Zlatev** (UT), *Goal-oriented design of value and process models from patterns.*

2007-23 | **Peter Barna** (TUE), *Specification of Application Logic in Web Information Systems.*

2007-24 | **Georgina Ramrez Camps** (CWI), *Structural Features in XML Retrieval.*

2007-25 | **Joost Schalken** (VU), *Empirical Investigations in Software Process Improvement.*

2008

2008-01 | **Katalin Boer-Sorbn** (EUR), *Agent-Based Simulation of Financial Markets: A modular,continuous-time approach.*

2008-02 | **Alexei Sharpanskykh** (VU), *On Computer-Aided Methods for Modeling and Analysis of Organizations.*

2008-03 | **Vera Hollink** (UVA), *Optimizing hierarchical menus: a usage-based approach.*

2008-04 | **Ander de Keijzer** (UT), *Management of Uncertain Data - towards unattended integration.*

2008-05 | **Bela Mutschler** (UT), *Modeling and simulating causal dependencies on process-aware information systems from a cost perspective.*

2008-06 | **Arjen Hommersom** (RUN), *On the Application of Formal Methods to Clinical Guidelines, an Artificial Intelligence Perspective.*

2008-07 | **Peter van Rosmalen** (OU), *Supporting the tutor in the design and support of adaptive e-learning.*

2008-08 | **Janneke Bolt** (UU), *Bayesian Networks: Aspects of Approximate Inference.*

2008-09 | **Christof van Nimwegen** (UU), *The paradox of the guided user: assistance can be counter-effective.*

2008-10 | **Wauter Bosma** (UT), *Discourse oriented summarization.*

2008-11 | **Vera Kartseva** (VU), *Designing Controls for Network Organizations: A Value-Based Approach.*

2008-12 | **Jozsef Farkas** (RUN), *A Semiotically Oriented Cognitive Model of Knowledge Representation.*

2008-13 | **Caterina Carraciolo** (UVA), *Topic Driven Access to Scientific Handbooks.*

2008-14 | **Arthur van Bunningen** (UT), *Context-Aware Querying; Better Answers with Less Effort.*

2008-15 | **Martijn van Otterlo** (UT), *The Logic of Adaptive Behavior: Knowledge Representation and Algorithms for the Markov Decision Process Framework in First-Order Domains.*

2008-16 | **Henriette van Vugt** (VU), *Embodied agents from a user's perspective.*

2008-17 | **Martin Op 't Land** (TUD), *Applying Architecture and Ontology to the Splitting and Allying of Enterprises.*

2008-18 | **Guido de Croon** (UM), *Adaptive Active Vision.*

2008-19 | **Henning Rode** (UT), *From Document to Entity Retrieval: Improving Precision and Performance of Focused Text Search.*

2008-20 | **Rex Arendsen** (UVA), *Geen bericht, goed bericht. Een onderzoek naar de effecten van de introductie van elektronisch berichtenverkeer met de overheid op de administratieve lasten van bedrijven.*

2008-21 | **Krisztian Balog** (UVA), *People Search in the Enterprise.*

2008-22 | **Henk Koning** (UU), *Communication of IT-Architecture.*

2008-23 | **Stefan Visscher** (UU), *Bayesian network models for the management of ventilator-associated pneumonia.*

2008-24 | **Zharko Aleksovski** (VU), *Using background knowledge in ontology matching.*

2008-25 | **Geert Jonker** (UU), *Efficient and Equitable Exchange in Air Traffic Management Plan Repair using Spender-signed Currency.*

2008-26 | **Marijn Huijbregts** (UT), *Segmentation, Diarization and Speech Transcription: Surprise Data Unraveled.*

2008-27 | **Hubert Vogten** (OU), *Design and Implementation Strategies for IMS Learning Design.*

2008-28 | **Ildiko Flesch** (RUN), *On the Use of Independence Relations in Bayesian Networks.*

2008-29 | **Dennis Reidsma** (UT), *Annotations and Subjective Machines - Of Annotators, Embodied Agents, Users, and Other Humans.*

2008-30 | **Wouter van Atteveldt** (VU), *Semantic Network Analysis: Techniques for Extracting, Representing and Querying Media Content.*

2008-31 | **Loes Braun** (UM), *Pro-Active Medical Information Retrieval.*

2008-32 | **Trung H. Bui** (UT), *Toward Affective Dialogue Management using Partially Observable Markov Decision Processes.*

2008-33 | **Frank Terpstra** (UVA), *Scientific Workflow Design; theoretical and practical issues.*

2008-34 | **Jeroen de Knijf** (UU), *Studies in Frequent Tree Mining.*

2008-35 | **Ben Torben Nielsen** (UvT), *Dendritic morphologies: function shapes structure.*

2009

2009-01 | **Rasa Jurgelenaite** (RUN), *Symmetric Causal Independence Models.*

2009-02 | **Willem Robert van Hage** (VU), *Evaluating Ontology-Alignment Techniques.*

2009-03 | **Hans Stol** (UvT), *A Framework for Evidence-based Policy Making Using IT.*

2009-04 | **Josephine Nabukenya** (RUN), *Improving the Quality of Organisational Policy Making using Collaboration Engineering.*

2009-05 | **Sietse Overbeek** (RUN), *Bridging Supply and Demand for Knowledge Intensive Tasks - Based on Knowledge, Cognition, and Quality.*

2009-06 | **Muhammad Subianto** (UU), *Understanding Classification.*

2009-07 | **Ronald Poppe** (UT), *Discriminative Vision-Based Recovery and Recognition of Human Motion.*

2009-08 | **Volker Nannen** (VU), *Evolutionary Agent-Based Policy Analysis in Dynamic Environments.*

2009-09 | **Benjamin Kanagwa** (RUN), *Design, Discovery and Construction of Service-oriented Systems.*

2009-10 | **Jan Wielemaker** (UVA), *Logic programming for knowledge-intensive interactive applications.*

2009-11 | **Alexander Boer** (UVA), *Legal Theory, Sources of Law & the Semantic Web.*

2009-12 | **Peter Massuthe** (TUE, Humboldt-Universitaet zu Berlin), *Perating Guidelines for Services.*

2009-13 | **Steven de Jong** (UM), *Fairness in Multi-Agent Systems.*

2009-14 | **Maksym Korotkiy** (VU), *From ontology-enabled services to service-enabled ontologies.* making ontologies work in e-science with ONTO-SOA

2009-15 | **Rinke Hoekstra** (UVA), *Ontology Representation - Design Patterns and Ontologies that Make Sense.*

2009-16 | **Fritz Reul** (UvT), *New Architectures in Computer Chess.*

2009-17 | **Laurens van der Maaten** (UvT), *Feature Extraction from Visual Data.*

2009-18 | **Fabian Groffen** (CWI), *Armada, An Evolving Database System.*

2009-19 | **Valentin Robu** (CWI), *Modeling Preferences, Strategic Reasoning and Collaboration in Agent-Mediated Electronic Markets.*

2009-20 | **Bob van der Vecht** (UU), *Adjustable Autonomy: Controling Influences on Decision Making.*

2009-21 | **Stijn Vanderlooy** (UM), *Ranking and Reliable Classification.*

2009-22 | **Pavel Serdyukov** (UT), *Search For Expertise: Going beyond direct evidence.*

2009-23 | **Peter Hofgesang** (VU), *Modelling Web Usage in a Changing Environment.*

2009-24 | **Annerieke Heuvelink** (VUA), *Cognitive Models for Training Simulations.*

2009-25 | **Alex van Ballegooij** (CWI), *"RAM: Array Database Management through Relational Mapping".*

2009-26 | **Fernando Koch** (UU), *An Agent-Based Model for the Development of Intelligent Mobile Services.*

2009-27 | **Christian Glahn** (OU), *Contextual Support of Social Engagement and Reflection on the Web.*

2009-28 | **Sander Evers** (UT), *Sensor Data Management with Probabilistic Models.*

2009-29 | **Stanislav Pokraev** (UT), *Model-Driven Semantic Integration of Service-Oriented Applications.*

2009-30 | **Marcin Zukowski** (CWI), *Balancing vectorized query execution with bandwidth-optimized storage.*

2009-31 | **Sofiya Katrenko** (UVA), *A Closer Look at Learning Relations from Text.*

2009-32 | **Rik Farenhorst and Remco de Boer** (VU), *Architectural Knowledge Management: Supporting Architects and Auditors.*

2009-33 | **Khiet Truong** (UT), *How Does Real Affect Affect Affect Recognition In Speech?.*

2009-34 | **Inge van de Weerd** (UU), *Advancing in Software Product Management: An Incremental Method Engineering Approach.*

2009-35 | **Wouter Koelewijn** (UL), *Privacy en Politiegegevens; Over geautomatiseerde normatieve informatie-uitwisseling.*

2009-36 | **Marco Kalz** (OUN), *Placement Support for Learners in Learning Networks.*

2009-37 | **Hendrik Drachsler** (OUN), *Navigation Support for Learners in Informal Learning Networks.*

2009-38 | **Riina Vuorikari** (OU), *Tags and self-organisation: a metadata ecology for learning resources in a multilingual context.*

2009-39 | **Christian Stahl** (TUE, Humboldt-Universitaet zu Berlin), *Service Substitution – A Behavioral Approach Based on Petri Nets.*

2009-40 | **Stephan Raaijmakers** (UvT), *Multinomial Language Learning: Investigations into the Geometry of Language.*

2009-41 | **Igor Berezhnyy** (UvT), *Digital Analysis of Paintings.*

2009-42 | **Toine Bogers** (UvT), *Recommender Systems for Social Bookmarking.*

2009-43 | **Virginia Nunes Leal Franqueira** (UT), *Finding Multi-step Attacks in Computer Networks using Heuristic Search and Mobile Ambients.*

2009-44 | **Roberto Santana Tapia** (UT), *Assessing Business-IT Alignment in Networked Organizations.*

2009-45 | **Jilles Vreeken** (UU), *Making Pattern Mining Useful.*

2009-46 | **Loredana Afanasiev** (UvA), *Querying XML: Benchmarks and Recursion.*

2010

2010-01 | **Matthijs van Leeuwen** (UU), *Patterns that Matter.*

2010-02 | **Ingo Wassink** (UT), *Work flows in Life Science.*

2010-03 | **Joost Geurts** (CWI), *A Document Engineering Model and Processing Framework for Multimedia documents.*

2010-04 | **Olga Kulyk** (UT), *Do You Know What I Know? Situational Awareness of Co-located Teams in Multidisplay Environments.*

2010-05 | **Claudia Hauff** (UT), *Predicting the Effectiveness of Queries and Retrieval Systems.*

2010-06 | **Sander Bakkes** (UvT), *Rapid Adaptation of Video Game AI.*

2010-07 | **Wim Fikkert** (UT), *A Gesture interaction at a Distance.*

2010-08 | **Krzysztof Siewicz** (UL), *Towards an Improved Regulatory Framework of Free Software. Protecting user freedoms in a world of software communities and eGovernments.*

2010-09 | **Hugo Kielman** (UL), *Politiële gegevensverwerking en Privacy, Naar een effectieve waarborging.*

2010-10 | **Rebecca Ong** (UL), *Mobile Communication and Protection of Children.*

2010-11 | **Adriaan Ter Mors** (TUD), *The world according to MARP: Multi-Agent Route Planning.*

2010-12 | **Susan van den Braak** (UU), *Sensemaking software for crime analysis.*

2010-13 | **Gianluigi Folino** (RUN), *High Performance Data Mining using Bio-inspired techniques.*

2010-14 | **Sander van Splunter** (VU), *Automated Web Service Reconfiguration.*

2010-15 | **Lianne Bodenstaff** (UT), *Managing Dependency Relations in Inter-Organizational Models.*

2010-16 | **Sicco Verwer** (TUD), *Efficient Identification of Timed Automata, theory and practice.*

2010-17 | **Spyros Kotoulas** (VU), *Scalable Discovery of Networked Resources: Algorithms, Infrastructure, Applications.*

2010-18 | **Charlotte Gerritsen** (VU), *Caught in the Act: Investigating Crime by Agent-Based Simulation.*

2010-19 | **Henriette Cramer** (UvA), *People's Responses to Autonomous and Adaptive Systems.*

2010-20 | **Ivo Swartjes** (UT), *Whose Story Is It Anyway? How Improv Informs Agency and Authorship of Emergent Narrative.*

2010-21 | **Harold van Heerde** (UT), *Privacy-aware data management by means of data degradation.*

2010-22 | **Michiel Hildebrand** (CWI), *End-user Support for Access to Heterogeneous Linked Data.*

2010-23 | **Bas Steunebrink** (UU), *The Logical Structure of Emotions.*

2010-24 | **Dmytro Tykhonov** (), *Designing Generic and Efficient Negotiation Strategies.*

2010-25 | **Zulfiqar Ali Memon** (VU), *Modelling Human-Awareness for Ambient Agents: A Human Mindreading Perspective.*

2010-26 | **Ying Zhang** (CWI), *XRPC: Efficient Distributed Query Processing on Heterogeneous XQuery Engines.*

2010-27 | **Marten Voulon** (UL), *Automatisch contracteren.*

2010-28 | **Arne Koopman** (UU), *Characteristic Relational Patterns.*

2010-29 | **Stratos Idreos** (CWI), *Database Cracking: Towards Auto-tuning Database Kernels.*

2010-30 | **Marieke van Erp** (UvT), *Accessing Natural History - Discoveries in data cleaning, structuring, and retrieval.*

2010-31 | **Victor de Boer** (UVA), *Ontology Enrichment from Heterogeneous Sources on the Web.*

2010-32 | **Marcel Hiel** (UvT), *An Adaptive Service Oriented Architecture: Automatically solving Interoperability Problems.*

2010-33 | **Robin Aly** (UT), *Modeling Representation Uncertainty in Concept-Based Multimedia Retrieval.*

2010-34 | **Teduh Dirgahayu** (UT), *Interaction Design in Service Compositions.*

2010-35 | **Dolf Trieschnigg** (UT), *Proof of Concept: Concept-based Biomedical Information Retrieval.*

2010-36 | **Jose Janssen** (OU), *Paving the Way for Lifelong Learning; Facilitating competence development through a learning path specification.*

2010-37 | **Niels Lohmann** (TUE), *Correctness of services and their composition.*

2010-38 | **Dirk Fahland** (TUE), *From Scenarios to components.*

2010-39 | **Ghazanfar Farooq Siddiqui** (VU), *Integrative modeling of emotions in virtual agents.*

2010-40 | **Mark van Assem** (VU), *Converting and Integrating Vocabularies for the Semantic Web.*

2010-41 | **Guillaume Chaslot** (UM), *Monte-Carlo Tree Search.*

2010-42 | **Sybren de Kinderen** (VU), *Needs-driven service bundling in a multi-supplier setting - the computational e3-service approach.*

2010-43 | **Peter van Kranenburg** (UU), *A Computational Approach to Content-Based Re-*

trieval of Folk Song Melodies.

2010-44 | **Pieter Bellekens** (TUE), *An Approach towards Context-sensitive and User-adapted Access to Heterogeneous Data Sources, Illustrated in the Television Domain.*

2010-45 | **Vasilios Andrikopoulos** (UvT), *A theory and model for the evolution of software services.*

2010-46 | **Vincent Pijpers** (VU), *e3alignment: Exploring Inter-Organizational Business-ICT Alignment.*

2010-47 | **Chen Li** (UT), *Mining Process Model Variants: Challenges, Techniques, Examples.*

2010-48 | **Milan Lovric** (EUR), *Behavioral Finance and Agent-Based Artificial Markets.*

2010-49 | **Jahn-Takeshi Saito** (UM), *Solving difficult game positions.*

2010-50 | **Bouke Huurnink** (UVA), *Search in Audiovisual Broadcast Archives.*

2010-51 | **Alia Khairia Amin** (CWI), *Understanding and supporting information seeking tasks in multiple sources.*

2010-52 | **Peter-Paul van Maanen** (VU), *Adaptive Support for Human-Computer Teams: Exploring the Use of Cognitive Models of Trust and Attention.*

2010-53 | **Edgar Meij** (UVA), *Combining Concepts and Language Models for Information Access.*

2011

2011-01 | **Botond Cseke** (RUN), *Variational Algorithms for Bayesian Inference in Latent Gaussian Models.*

2011-02 | **Nick Tinnemeier** (UU), *Organizing Agent Organizations. Syntax and Operational Semantics of an Organization-Oriented Programming Language.*

2011-03 | **Jan Martijn van der Werf** (TUE), *Compositional Design and Verification of Component-Based Information Systems.*

2011-04 | **Hado van Hasselt** (UU), *Insights in Reinforcement Learning - Formal analysis and empirical evaluation of temporal-difference learning algorithms.*

2011-05 | **Base van der Raadt** (VU), *Enterprise Architecture Coming of Age - Increasing the Performance of an Emerging Discipline.*

2011-06 | **Yiwen Wang** (TUE), *Semantically-Enhanced Recommendations in Cultural Heritage.*

2011-07 | **Yujia Cao** (UT), *Multimodal Information Presentation for High Load Human Computer Interaction.*

2011-08 | **Nieske Vergunst** (UU), *BDI-based Generation of Robust Task-Oriented Dialogues.*