Západočeská univerzita v Plzni
Fakulta aplikovaných věd

AGENTNÍ PŘÍSTUP K DIALOGOVÉMU ŘÍZENÍ

Tomáš Nestorovič

disertační práce
k získání akademického titulu
doktor
v oboru
Informatika a výpočetní technika

Školitel: Prof. Ing. Václav Matoušek, CSc.
Katedra informatiky a výpočetní techniky

Plzeň, 2015

University of West Bohemia in Pilsen
Faculty of Applied Sciences

AGENT-BASED DIALOGUE MANAGEMENT

Tomáš Nestorovič

doctoral thesis
in partial fulfillment of the requirements
for the degree of

Doctor of Philosophy

in specialization
Computer Science and Engineering

Supervisor: Prof. Ing. Václav Matoušek, CSc.

Department of Computer Science and Engineering

Pilsen, 2015

Prohlášení

Předkládám tímto k posouzení a obhajobě dizertační práci zpracovanou na závěr doktorského studia na Fakultě aplikovaných věd Západočeské univerzity v Plzni.

Prohlašuji tímto, že tuto práci jsem vypracoval samostatně s použitím odborné literatury a dostupných pramenů uvedených v seznamu, jež je součástí této práce.

V Plzni dne 7. března 2015 Tomáš Nestorovič

Abstract

This work focuses on dialogue management in human-computer interaction. Dialogue systems are considered an attractive topic nowadays and we may encounter them in many daily situations – they are in our cars, in our phones, and sometimes even control our homes. Conversational agents that incorporate principles of inter-human rationality and cooperation are highly preferred. Viewing a dialogue as an interaction between two intelligent entities, the Beliefs-Desires-Intentions (BDI) architecture has been the far most popular approach to create such agents over the past decade.

This work consists of three main parts that were all developed during the study and build upon each other. The first of them is an information management framework. Inspired by the Semantic Interface Language (SIL), this framework aims to represent detailed structure of knowledge. However, objects in this framework miss any implied meaning and taxonomy. We argue that objects receive their meaning and taxonomy within plans that deals with them, and that such design significantly facilitates complex operations with objects during cooperative dialogues, which the information management framework primarily targets.

The second part is a general cooperative dialogue framework called Daisy. It has been designed to host BDI conversational agents and provide "out of the box" solutions. It uses existing work in speech act theory and discourse analysis, namely the concepts of conversational acts and discourse segment intentions. The dialogue control flow is then derived automatically as the result of the BDI interpretation cycle. The Daisy framework has been developed from scratch during the study and among other things features the following functionalities: intention detection and management, dialogue length optimization, and complex utterances production – hence covers all major topics in dialogue systems.

The third part of the work is an experimental "Rogerian strategy", inspired by the idea of the so called Rogerian therapy. The essence of this strategy is to not push users to say what we want to hear from them but instead give them reasonable amount of freedom to say what they want. An experimental banking domain system has been developed to find out if this strategy performs better than the common mixed-initiative way. It's application resulted in shorter dialogues compared to classical mixed-initiative management. Naturally, strong constraints are discussed and put on this strategy.

Abstrakt

Práce se zaměřuje na dialogové řízení během interakce člověka s počítačem. Dialogové systémy jsou dnes atraktivním tématem a můžeme se s nimi setkat v mnoha každodenních situacích – jsou v našich automobilech, v našich telefonech a někdy jimi dokonce ovládáme naše domy. Konverzační agenti, kteří vykazují principy mezilidské racionality a spolupráce, jsou vysoce preferovány. Během poslední dekády byla pro implementaci takových agentů velice populární architektura Beliefs-Desires-Intentions (BDI), která pohlíží na dialog jako na interakci dvou inteligentních entit.

Práce sestává ze tří hlavních částí, které staví jedna na druhé. První z nich je framework správy informací, který našel inspiraci v Semantic Interface Language (SIL) a je orientován na detailní reprezentaci struktury znalostí. Objekty v tomto frameworku však postrádají jakoukoliv implicitní sémantiku a taxonomii, což odůvodňujeme skutečností, že objekty získají svou sémantiku a taxonomii v plánech, které s nimi nakládají, a že tento přístup výrazně usnadňuje složité operace s objekty během kooperativního dialogu, na nějž je náš framework správy informací primárně cílen.

Druhou částí naší práce je obecný framework kooperativního dialogu, nazvaný Daisy. Tento framework byl navržen jako běhové prostředí poskytující hotová řešení pro konverzační BDI agenty. Při jeho tvorbě byly použita teorie řečových aktů a analýza konverzace, konkrétně koncept konverzačních aktů a koncept záměrů v segmentech dialogu. Dialog je poté výsledkem interpretačního cyklu konverzačního BDI agenta. Framework byl celý vyvinut v průběhu studia a mimo jiné disponuje následujícími schopnostmi: rozpoznávání a správa záměrů, optimalizace délky dialogu a komplexní produkce promluv.

Třetí částí naší práce je experimentální "Rogeriánská strategie", inspirovaná tzv. Rogeriánskou terapií. Podstatou této strategie je netlačit uživatele, aby řekli, co chce systém slyšet, ale naopak dát jim rozumně velkou svobodu, aby řekli, co oni sami chtějí. Abychom ověřili, zda tato strategie účinkuje lépe než klasická smíšená iniciativa, vytvořili jsme experimentální bankovní systém. Použití této strategie vyústilo v kratší dialogy ve srovnání se smíšenou iniciativou. Samozřejmě klademe silná omezení na tuto strategii, která níže diskutujeme.

Contents

List of Figures

List of Tables

List of Definitions

List of Examples

List of Abbreviations

ASR	Automatic Speech Recognition
AVM	Atribute-Value Matrix
BDI	Beliefs-Desires-Intentions
CON	"Confirmation" event
CTA	Cognitive Task Analysis
CTS	Concept-to-Speech
DDM	Domain Data Model
DS	Discourse Segment
DSP	Discourse Segment Purpose
FIA	Form Interpretation Algorithm
HCI	Human-Computer Interaction
HMM	Hidden Markov Model
MDP	Markov Decision Process
MUFO	Multiple Utterance Field Object
NS	"Narrow" Strategy
OS	"Open" Strategy
PBX	Private Branch Exchange
PLN	"Plan interpretation" event
RL	Reinforement Learning
RS	"Rogerian" Strategy
SDL	Specification and Description Language
SIL	Semantic Interface Language
TTS	Text-to-Speech
UFO	Utterance Field Object
XML	Extensible Markup Language

This page intentionally left blank.

Chapter 1

Introduction

Over the past ten years, we witnessed an expansion of spoken language interfaces in various realms in our lives. They however are not a recent invention, as they have already quite a long history behind. Beginning with the 1960's when the first spoken interfaces started to emerge, they represented merely a scientific experimentation with natural language applied to simple, constraint, and sealed systems. One of such systems was Elisa [Wei66] which attempted to imitate a human thinking by leading a "plausible" conversation governed by a complex system of rules. Thus, by practically not regularly modeling a dialogue, Elisa embodied merely a reactive entity. Later on, the 1970's were represented by an intensive research of new approaches to understand the fluent natural language. The basic idea was to use knowledge-based systems to analyse and understand a speech. Along the way, also first complex analyses of dialogues emerged, focusing on the structure and underlying intentions. A well known system from that era is SHRDLU [Win72] to move objects on a screen from one place to another, allowing its users to operate it using fully natural sentences.

The early 1980's in general experienced a decline of interest in speech interfaces, mainly due to immaturity of hardware computational capacity. The renascence came during break of decades with advancements of ASR technologies performance that in turn led to rapid improvements in speech interfaces. The 1990's era is therefore characteristic with lots of commercial telephone-based dialogue services. For instance, the MIT Voyager [Zue89, Zue91] is considered as the first real spoken dialogue system to provide its users with detailed navigation of Cambridge. At the end of the decade, an improved version of it was released under the name JUPITER [Zue00]. Apart of that, the ATIS (Airline Travel Information System) [Hem90] was another important project of the 1990's on which many research institutes participated. Apart of providing users with airline information, the project also aimed to collect spontaneus utterances from users, annotate them, and analyse with respect to understanding the model of a dialogue context. The outcome was the statistical approach to dialogue management (discussed in the next chapter).

Nowadays, spoken dialogue interfaces have their established position in situations in which safety is the main concern. For instance, we barely now can think of a higher class car not equipped with a spoken interface to control the radio, navigation, or in-car phone. Telephone-based spoken dialogue systems have become quite a standard. A new term "conversational agent" emerges and is becoming more popular. Its essence indicates that speech interfaces cease to serve merely as an alternative way to put commands into an application. Instead, conversational agents are to take over a certain level of autonomy in solving tasks with users, for instance by proposing alternatives if no solution can be found [Jok10].

Conversational agents are on the rise in one speciffic family of applications: enterprise software [Les04]. Over the recent years, the demand for cost-effective solutions to the customer service problem has increased dramatically. Deploying automated solutions can signifficantly reduce the costs of company customer service. By exploiting the web technologies in conjunction with computational linguistics, conversational agents offer to companies the ability to provide customer service much more economically than with traditional human-human models. In customer-facing deployments, conversational agents interact directly with customers to help them obtain answers to their questions. In internal-facing deployments, they converse with customer service representatives to train them and help them assist customers.

In addition to that, with the wide expansion of mobile devices, speech interfaces have found themselves a brand new territory of usage, and this trend does not seem to fade out in the foreseeable future. As even the largest graphical displays suffer from relatively small dimensions, speech interfaces represent a reasonable and powerfull workaround to this limitation.

The spoken human-computer interaction has always been perceived as one of the artificial intelligence disciplines. Nonetheless, it should rather be understood as an inter-disciplinary interest as it spans various realms, including acustics, fonetics, information theory, signal processing, image recognition, and heuristic searching, among others [Oce98].

1.1 Thesis Goals

Due to our previous success with dialogue management and dialogue systems [Nes07] we would like to continue in this realm. The individual goals are as follows:

1. analysis of state-of-the-art approaches to dialogue management,
2. proposal of an extension to the dialogue management,
3. implementation of a subset of functionality and its validation using simple examples, and
4. evaluation and outline of possible future work.

Chapter 2

Dialogue System Architecture

2.1 Language as a Communication Medium: Pros and Cons

Before we begin our exploration of dialogue systems, let us concern with some major points in using the natural language as the communication means. Among many pros, there are also some cons to take into account [Eck95]. Let us first focus on them.

- Language provides a less quicker medium for information transfer than any visual interface. Especially in case of larger amounts of information, there is no such possibility like having a "quick look at the document". If one searches for a particular information in a message, one needs to listen to the message completely because language is a serial representation of information.

- Information contained in a spoken message is quickly forgotten and usually needs to be several times "refreshed", especially if the message is longer or difficult to grasp. This is caused by the language being a volatile medium [Boy99, Yan96] and humans having only a short-term memory [Gus02, Les04]. In contrast, visual refreshing is a much quicker process, requiring only a quick look at those parts of a scene which are likely to contain the searched information.

- One of the considerable downsides of language is also that it cannot be ignored, unlike with a computer screen [Hur05]. A talking computer may create an unacceptable work environment in which other coworkers are annoyed. One solution might be to localize such computer along with its operator to a sound-proof box.

- Being a human-exclusive communication channel, synthesized language will always be in direct competition with natural language. always being thoroughly judged against even the tiniest mistakes and factiousness [Oce98].

Apart of these limitations and challenges, there are also many applications for which speech interfaces are a much more effective (or the only) means. The following points show some of them.

- User's both eyes and hands are fully employed with other tasks. A typical example is when driving a car [Tsi12, Yam07]. Making use of a spoken interface enables the driver to fully concentrate on the surrounding traffic, leading to improved safety of all participants on the street.
- A spoken interface has the potential to understand complex intentional structures, so common for inter-human communication. Not randomly has therefore been language used in tutoring systems, in which an automated agent substitutes the role of a teacher, including explanation of a subject and testing the pupil [Lit06, Rot07, Gri13].
- The user is a handicapped person with impaired motion or sight. In these cases, dialogue systems can mediate an access to information that would otherwise be left unreachable to such people. Also their households can already be controlled using voice, including lighting, television, or temperature [Ger10, Pér06].
- Other reasons to opt for a spoken interface include: automation of a call service; remote control of a service with no visual alternative; requirement for a user to be mobile when interacting with a service; extension of a mobile phone service to incorporate additional functionality.

Hence, a speech interface provides also many notable advantages. However, there has still one crucial question left unanswered: How is one to know if a speech interface is a suitable way to extend an application with? Eckert proposes a short but valuable guideline that helps to answer it [Eck95]:

- Usage of a speech interface should bring in some benefits for the user. It should not be there just because it is currently "trendy" and eventually increases sales in a short-term horizon.
- The speech recognizer (briefly discussed later) should be as reliable as at least 95% for the user to be motivated to use the system again.
- It is very positively rewarded if users are provided with quick and non-ambiguous responses in order for them to gain the feeling that they are a real part of the communication process. The system should also provide enough feedback for them to gain the feeling of controlling the system.
- A cooperative system should be conceived as a user-friendly and robust entity. The system must be well prepared to deal with unexpected or unusual user input.

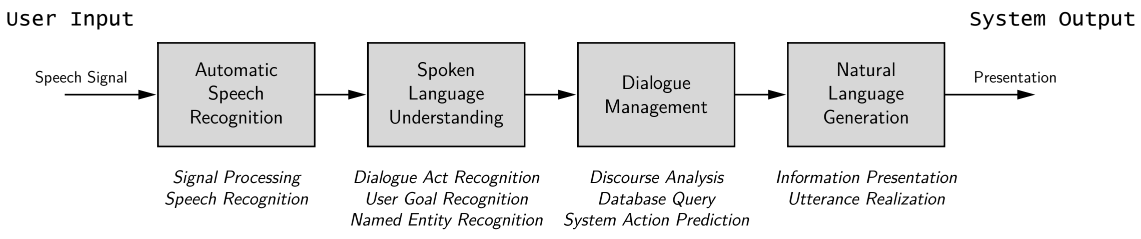


Fig. 2.1 Traditional architecture to uni-modal dialogue systems; adopted from [Lee10].

2.2 Dialogue System Architecture

A computer-based dialogue system can be defined as an *artificial participant in a dialogue* [Qu02]. Without engaging in much details, the creation of such participant means a long journey from the initial idea to the first real dialogue. It also usually takes a team of highly specialized people to successfully accomplish and deploy the system. Tasks to solve for are, for instance, system understanding of user's speech, recognition of user's intentions, or production of system reaction, among many other things. Hence people's qualification must span various realms, from Fourier transformation to search algorithms, and from raw data processing to abstract data types [Eck95].

These requirements imply that practically the only correct architecture of a dialogue system is a modular one. With such design, the complex task of human-computer interaction is decomposed into smaller pieces. Modules are to an extent independent on each other, which enables them to be evolved individually as needed. Last but not least, modularity also makes the components portable to other applications, possibly other dialogue systems.

Assuming now a uni-modal dialogue system (with a speech interface as the only input and output channels), the modules that constitute the system elemental skills are as follows (see also Fig. 2.1):

- *Automatic speech recognition* (*ASR*). This module is to perform all steps between processing an utterance raw speech signal and producing an equivalent textual representation. Additional functionality requirements may be put on this module, for instance, "barge-in" capability [Kle00] or a minimum recognition confidence for a given vocabulary [Sti01]. While the former one is a domain-independent and user-neutral property, the latter one is influenced by the number of users to interact with the system and the type of environment the system is to be used in [Pav09].
- *Spoken language understanding*. This module is fed in the textual representation from the ASR to analyze its content against the scope of a predefined domain. This analysis is carried out until a suitable symbolic representation of relevant information within the sequence of words has been found. There are two competitive branches of analysis, grammatical and stochastical. The *grammatical* approaches [Kni01, Jok10] rely on

a set of rules that describe possible sequences of words and produce a corresponding symbolic representation. In contrast, the *stochastical* approaches [Kon09] use for this procedure either neural networks or Hidden Markov Models (HMMs). Both of the branches naturally have their benefits and disadvantages. For instance, grammar-based parsers are transparent in representation but inflexible as for handling non-grammatical sentences; in contrast, stochastic parsers can deal with non-grammatical sentences but may be quite fuzzy to properly train.

- *Dialogue manager.* This module is responsible for coordinating actions between the system, user, and eventual back-end services. It takes over the symbolic representation from the semantic analysis and compares it with the past interaction to produce a suitable reaction. Depending on the complexity of the manager, the resulting dialogue exhibits various levels of naturalness. The dialogue management module is in closer detail presented in the next section, and in Chapter 4.

- *Natural language generation.* This module receives the dialogue manager reaction and transforms it into a corresponding speech signal to convey the message to the user. The dialogue manager might have produced either a textual representation of its response, in which case we talk about a text-to-speech synthesis (TTS, see also Chapter 4), or an abstract symbolic representation, in which case we talk about a concept-to-speech synthesis (CTS, see also Chapter 3).

These four modules constitute a common ground for spoken dialogue system. Depending on the complexity of the application domain (and other requirements put on the resulting dialogue system), each of the modules can be internally further divided into submodules. For instance, a system to perform isolated spoken commands is considerably simpler in terms of its internal architecture, than a sophisticated multi-modal agent that understands complex utterances. This thesis, however, does not concern with multi-modal dialogue systems and interaction.[1]

The final step in designing a dialogue system is to interconnect the modules with a communication channel. In the case of all components running locally, the *blackboard approach* is a sufficient way [Wal04]. With a chunk of memory serving as the shared blackboard, each module can write/read information to and from it, modify existing information, or erase it. Presumably, there also exists a superior control module to coordinate operation requests with the blackboard. However, in the opposite situation of dialogue system components running distributed across a heterogeneous environment (e.g., due to different programming languages, computers, operating systems, number endians, etc.), a network-based communication is one of the options [Tur05, Sto12, Boh07, All01]. With a hub at the centre, each module can request another module or

1 In essence, to extend a uni-modal dialogue system to a multi-modal one requires the basic workflow pipeline from Fig. 2.1 to be prepended with an *input modality fusion module*, melting partial input semantics down to a single compound semantic, further passed over to the dialogue manager. Eventually, the pipeline must also be appended with an *output modality fission module*, splitting the dialogue manager reaction into messages towards output modalities. An overview of these topics may be found, for instance, in [Ovi02, Bui06].

remote service to perform an operation by posting a message to the hub. The hub node then looks up the receiver of the message in its neighbourhood, passes the message on, and when the result is ready, notifies the sender.

2.3 Computational Models to Dialogue Management

In the remainder of this chapter, let us focus on the function of and approaches to the dialogue management module. The dialogue manager controls the overall interaction between the system and the user. The essential role of the dialogue manager may be summarized into the following intrinsic tasks [Cen04, Tra03]:

- Interpret observation (usually user input/s) in context, and update the internal representation of the dialogue.
- Provide context-dependent expectations for interpretation of upcoming responses.
- Interface with task/domain processing (e.g., database, planner, execution module, or other back-end subsystems) to coordinate dialogue and non-dialogue behaviour and reasoning.
- Determine the next action of the dialogue system, based on some dialogue management policy (with the aim to affect the mental state of the user).

Although all of these tasks are performed by virtually all dialogue managers, each of them is non-trivial and leads to a proliferation of different computational approaches. In addition, the dialogue manager accesses a number of *knowledge sources* which are sometimes collectively referred to as the "dialogue model". These sources may include the following types of knowledge relevant to the dialogue management [McT02]:

- *Dialogue history.* A trace of a dialogue observed and realized thus far. The representation should provide a basis for conceptual coherence and for the resolution of anaphora and ellipses.
- *Task description.* A representation of the solution to a particular task, including relevant pieces of information to be exchanged between the two participants.
- *Domain model.* A model with specific information about the domain in question (e.g., timetable domain).
- *Common knowledge model.* This model contains general background information that contributes to the commonsense reasoning of the system. For instance, the Christmas Eve is to be interpreted as December 24.
- *Generic model of conversational competence.* This includes knowledge about the principles of conversational turn-taking and discourse obligations; for instance, an appropriate response to a request for

information is to supply that information or provide a reason for not being able to supply it.

- *User model.* A model to contain relatively stable information about a user that may be relevant to the dialogue (e.g., user's age, preferences, previous experiences, etc.).

Hence, the expected capabilities of the dialogue manager span a relatively wide range. Over the past decades, many different approaches have emerged, ranging from simple finite state machines to Markov decision networks. However, their categorization has not yet been standardized. Hence, for instance, Xu *et al.* [Xu02] distinguishes among four groups of approaches: DITI (implicit dialogue model, implicit task model: like state-based models), DITE (implicit dialogue model, explicit task model: like frame-based models), DETI (explicit dialogue model, implicit task model), and DETE (explicit dialogue model, explicit task model). In contrast, Catizone *et al.* [Cat02] classifies approaches into mere three groups based on their underlying principles: dialogue grammars (approaches that put stress on the structure of dialogue, regardless of what controls the structure, be it a state automaton or a dialogue gaming framework), plan-based approaches (approaches that put stress on properly recognizing whatever intention a user may have, expressed or implied), and cooperative approaches (dialogue controlled by cooperative agents). Finally, Lee *et al.* [Lee10] groups approaches yet a different way: knowledge-based approaches (in which knowledge of the application domain plays the dominant role, including virtually everything between state-based and agent-based management), data-driven approaches (various learning strategies working in conjunction with various Markov decision processes), and hybrid approaches (supervised learning of optimal dialogue strategies). Despite the inconsistent divisions in the literature, the most commonly recognized approaches are the following ones [McT02, Ngu06b, Jok10]: (1) finite state machine approaches, (2) frame-based approaches, (3) plan-based approaches, (4) agent-based approaches, and (5) stochastic approaches. In the following sections, we will present them and discuss their properties in detail.

2.3.1 State-based Dialogue Management

Finite state models are the simplest models to base a dialogue manager on. The dialogue structure emerges implicitly by traversing a state transition network in which *nodes* represent system utterances and *edges* among nodes represent user's responses available at a given point in the dialogue [McT02, Chu05, Jok10]. The dialogue control is therefore system-driven and all the system utterances are predetermined. State-based approaches are adopted by most of the current commercial systems as they are suitable for applications in which the interaction is well-defined and can be structured as a sequential form-filling task or a tree, preferably with yes/no or short answers [Son06, Mel05]. Apart of these "classical" models, probabilistic finite-state automatons can also be used to learn optimal dialogue strategies automatically. As the design of such system is diametrically different from designing a "classical" state automaton, this family of approaches will closer be discussed below in Section 2.3.5.

The advantage of finite state models is that their background formalism is easy to understand and easy to implement. In this respect, designing a state-based system is relatively straightforward and intuitive. To further facilitate the development, several visualization toolkits have emerged over the years. One of the most popular ones is the Rapid Application Developer of the CSLU Toolkit [Sut98] which allows the designer to model the dialogue as a finite state automaton using a drag & drop interface.

In contrast, the main disadvantage is that a finite state approach typically leads to "unnatural dialogues" in which information is elicited from the user as a sequence of questions. Also, because the dialogue is controlled by the system, the dialogue flow is very inflexible: the user must strictly follow the structure of the dialogue and answer the system questions [Wil06]. No user initiative is permitted, and any additional information is ignored by the system. Each attempt to extend the system with a repair mechanism (reactions to misunderstandings, clarifications, etc.) lead to combinatorial explosion, as new states and edges among them are necessary to be added, thus making the system very hard to maintain [Mel05]. One possible workaround is to embed another finite-state network into one state, making the outer finite state automaton easier to understand and maintain [Mel05]. On a related note, there is practically none but explicit way of confirming user-specified information: the user has no possibility to initiate the correction, provided that after her or his misrecognized turn, the system has transited to another state. Explicit confirmations are commonly perceived as user unfriendly and lengthy [McT02]. One possible workaround to incorporate user-initiated corrections may be to enable the state automaton to track one state back [Ara03]. That way, the system may employ the more comfortable implicit confirmation, knowing that the user will eventually return back. Last but not least, the state methodology leads to systems tightly bound to a selected domain. That is, porting a finite state dialogue model to a new domain or application typically requires developing a brand new finite state automaton. The reason is that finite state systems lack a systematic delimitation between a task (i.e., what the dialogue manager wants to achieve) and the dialogue strategy (i.e., how the dialogue manager proceeds towards its goal) [Mel05].

2.3.2 Frame-based Dialogue Management

An extension to the state-based model has been developed to overcome the lack of flexibility. Hence, rather than building a dialogue according to a predetermined sequence of system utterances, the frame-based approach (sometimes also referred to as the "extended state automaton") takes on the analogy of a form-filling (or slot-filling) task in which a predetermined set of information is to be gathered. The cornerstone here is a *frame* (other authors use the terms *entity*, *form*, or *template*), consisting of a set of *slots* (alternatively termed as *items*, *fields*, or *attributes*). Each slot is related to a specific category of information recognized by the system. Given the notion of a frame, the approach already supports more flexible dialogues by allowing the user to fill in slots in different orders and different combinations. The frame is then to cumulate related pieces of information. Provided the current content of a frame, an accompanying *interpretation mechanism* is to select an action to do next. These actions usually cover the following situations [Mel05, Cen04]:

- *no input* – the user has not provided any utterance during the last turn,
- *no match* – the user answered but information provided does not fit in the frame (probably an "out-of-task" information),
- *value missing* – the mandatory slot misses a value,
- *request for repetition* – the user has asked for a repetition of the last system prompt,
- *request for help* – the user does not know how to answer the question and requires closer explanation,
- *start over* – the user wants to restart the task in focus, or eventually the whole interaction.

One of the well established representants of the so called *flat frames* is the VoiceXML platform.[2] Conceived within the Voice Browser Working Group of the World Wide Web Consortium (W3C), VoiceXML is a markup language based on XML that makes use of standard web programming techniques and languages, including for instance, Speech Recognition Grammar Specification (SRGS), Speech Synthesis Markup Language (SSML), Call Control Extensible Markup Language (CCXML), and ECMA-Script (some VoiceXML interpreters also support native code calls). The platform evolved as the result of various industry initiatives with the aim of providing a standardized way for development and deployment of speech applications. Hence, virtually all of the above actions have been incorporated into the current definition of VoiceXML. The mechanism to chose a suitable action (or *event*, in the VoiceXML terminology) is referred to as the Form Interpretation Algorithm (FIA). Provided the below short code snippet (adopted from [Jok10]) and assuming the whole `flight_info` form is initially empty, the FIA would opt for the value-missing event on the `source` field.

```
<form id="flight _ info">
     <field name="source">
          <grammar src="airports.grxml" />
          <prompt> Where are you flying from? </prompt>
     </field>
     <field name="destination">
          <grammar src="airports.grxml" />
          <prompt> Where are you flying to? </prompt>
     </field>
</form>
```

There are numerous variations to the basic flat frames and to the way of describing dialogue strategies. One of the variations is the *E-form*, standing for electronic form [God96]: slots are augmented with priorities and marked as mandatory or optional. E-forms have been used in the WHEELS dialogue

2 *http://www.w3.org/TR/voicexml21/*

system [Men96] to capture different user preferences about car parameters, like model, price, colour, etc., which usually do not have the same importance.

Another modification to the classical frame is a *hierarchical frame structure* (also sometimes referred to as *frame type hierarchy* or simply *nested frames*), in which one slot may be represented by a subframe. The underlying motivation here is that a hierarchy of frames better fits the structure of real world objects [vZa99, Hul96]. For instance, a slot *person* may be closer described by a nested frame containing slots *given_name*, *family_name*, *age*, etc. The mechanism to chose suitable action must take into account the nested structure, which makes it considerably more complex. However, one of the ways to determine the next action may be traversing through the structure in the top-down, left-to-right manner. Presuming frames are composed to reflect the structure of information within a task, an acceptably natural dialogue structure emerges.

The hierarchical frame structure has further been extended by Nestorovič [Nes10b, Nes09] with a set of *journals* to keep track of interrelated actions taken over the course of a dialogue. The motivation for this extension was to automate some commonly repeating routines, mostly related to causality tracking and subsequent error recovery. To account for these, the system designer would usually have to manually watch for slot updates and trigger corresponding reactions within *OnFilled*-like event handlers. However, this manual approach has a significant drawback: once the application logic gets more complicated, it is hard to keep track of where to "jump" next in a frame structure; there is also the threat of drawing in inconsistencies among these reactions. In contrast, by extending each frame with a journal, this procedure becomes automated. Full specification of the approach is attached on the CD.

The slot-filling approaches are far the most frequently used dialogue management techniques in practical systems [Pie09, vZa99, Son06, McT02, Cen04]. This is partly due to the frame-based management being still a simple enough approach with many available toolkits, for instance, VoiceXML (interpreted using OptimTalk[3]) or Philips HDDL (interpreted using Philips SpeechMania[4]). With using a frame-based management, we already can partly separate task and dialogue strategy: the task is defined by a (domain-specific) frame, whereas the strategy for filling in the frame is rather domain-independent (recall the above FIA, for instance) [Mel05].

On the other hand, even though task and dialogue strategies can be at least partially separated (which is beneficial for portability), it is an open question as for how scalable the approach is [McT02]. Extending an existing system with another useful dialogue strategy usually requires a considerable amount of hand-coding or may even be impossible: with handling a large number of rules or types of system reactions, it is difficult to predict all consequences of modifying an existing dialogue strategy [Mel05]. Another pitfall in frame-based systems is that they capture a dialogue as mere elicitation of several parameters in order to perform a task. However, dialogue is a more complex protocol, usually spanning multiple topics in a single conversation. In this respect, frame-based environments do not support mechanisms for topic detection, nor for explicit

3 *http://www.optimsys.com/en/products/application-platform-optimtalk*

4 *http://www.kbs.twi.tudelft.nl/People/Students/J.K.deHaan/Part%202%20Tools/06%20 SpeechMania/index.html*

representation of user goals (the goals are implicitly encoded in the structure of a frame).

2.3.3 Plan-based Dialogue Management

Dialogue management using a plan detection is part of complex dialogue systems exhibiting traces of free conversation. In the case of classical goal-oriented systems, individual plans basically match traversing through a state network or a frame structure. Such plans are of a short-term scope, with the aim to immediately elicit required information or immediately confirm uncertain information. In contrast, plans of a conversational agent may be considerably more abstract. For instance, in order to reach its objectives, the agent may adopt assertiveness as its long-term plan: if the user mentions that it would be nice to have X, then the agent assertively expresses an agreement of wanting X too [Wal01].

The key question in designing a plan-based system is the design of individual plans. Apart of the obvious empirical approach, the cognitive task analysis (CTA) [Hof98] is a much more sophisticated way. At its centre stands an *expert* in solving problems in a given domain, and an *interviewer*. The interviewer's goal is to gain information from the expert in order to clarify her or his reactions to observed or hypothetical situations. With a decent grain of salt, the CTA may be considered an analogy to filling the knowledge base of an automated expert system.

Over the course of a dialogue, the agent changes its strategies (plans) in accordance with the current state of the dialogue. This involves taking into account not only the convergence towards agent's objectives, but also changes in partner's detected intentions. Hence, the plan-based dialogue management has its underlying idea in the real world, in which it is the listener's objective to identify the speaker's intentions and respond to them accordingly [Cat02]. For instance, in response to a customer's question of "*Where are the steaks you advertised?*", a butcher's reply of "*How many do you want?*" is appropriate, because the butcher understands the customer's underlying plan to buy the steaks [Coh95]. On the other hand, the plan-based approaches have been widely criticized for their tight dependence on the plan identification which is considered their weakest point, provided that this process is computationally intractable in the worst case [Ric01] and more importantly unreliable [Bui06]. Another down side is the lack of any formal basis to lean the approaches on [Wil06].

2.3.4 Agent-based Dialogue Management

The agent-based approaches to dialogue management derive from the plan-based methods. Its essence therefore takes over all drawbacks, including weak parts in properly detecting partner's intentions. However, the agent-based approach puts numerous additional constraints to the plan-based methodology. That way, for instance, the detection of *unspoken* intentions[5] gets eliminated. From a certain perspective, these constraints represent the missing formal baseline. This

5 Sometimes also referred to as *hidden* intentions; see the customer-butcher example in Section 2.3.3.

baseline also includes typical agent characteristic [Zbo04, Woo00]: *reactivity* (ability to perceive the surrounding environment and react to it in a timely manner), *pro-activity* (ability to undertake goal-oriented actions in order to meet the objectives), *sociability* (ability to communicate and negotiate with other agents in the environment), and *mobility* (ability to perform actions at remote locations).

The essence of agent-based approaches is to view a dialogue as an interaction between two intelligent agents. In case of their collaboration, both of the agents work together to achieve a mutual understanding of the dialogue. The elemental cornerstone standing here behind this joint activity, is to handle classical dialogue phenomena such as confirmation or clarification [Bui06]. Hence, unlike with all the other approaches discussed above (including plan-based ones), the collaborative approaches try to capture the motivation behind a dialogue and the mechanisms of a dialogue itself.

The critical factor in designing agent-based applications is to find the proper tradeoff between agent's reactivity and pro-activity [Woo95, Rao95]. Continuously reacting to changes in the environment results in ceaselessly changing the direction of solution; contrarily, strictly insisting on a single direction puts the agent in threat of getting to nowhere. Applied to the dialogue management, a conversational agent must exhibit a certain level of pro-activity, in order to recover from errors in a dialogue, as well as reactivity, in order to meet user's objectives. Informally speaking, the pro-activity requirement may be compared with the system-initiative strategy, whereas the reactivity requirement corresponds with user-initiative one [Ngu06a].

Obviously, agent's behaviour is governed by its goals and knowledge about objectives to fulfill. These two components constitute agent's *mental state* [Zbo04]. One of popular implementations is the *Beliefs-Desires-Intentions* (*BDI*) architecture [Rao95]. In the BDI model, actions in the environment affect agent's beliefs. The agent in turn can reason about its beliefs, and thus formulate desires and intentions. Beliefs are how the agent perceives the environment, desires are how the agent would like the environment to be, and intentions are formulated plans of how to achieve these desires [Bra91]. Applying this again to the dialogue management, the three components of the BDI architecture take on the following responsibilities:

- *Beliefs* store a set of observations about a dialogue; for instance, the agent may believe that the user has chosen a train as the transportation means. This part of the architecture may therefore be perceived as the knowledge base of an expert system: a priori beliefs can be obtained from the ASR module, whereas a posteriori beliefs can be calculated from newly derived knowledge.

- *Desires* represent a collection of agent's top-level goals. These goals represent its motivation, and in turn influence how the agent plans its activity. In the case of a dialogue agent, desires are commonly organized as a stack [Gro86]. The result of such organization is a sequence of individual actions that the agent needs to carry out in order to meet its objectives.

- *Intentions* are another name for the "individual actions" (eventually also *moves*, in dialogue games terminology [Hul00]). Hence, this component may be seen as the storage of agent's planning procedure outcomes [Rao95].

Following the BDI architecture, a conversational agent must be provided with a means to deliberate. This promotes it to the so-called *deliberative agent*. One of the popular ways to provide such means assumes that each plan is a sequence of actions which together lead to the satisfaction of a particular desire. To following pseudo-code shows one possible realization of such deliberation [Woo00].

```
repeat {
      perceive the surrounding environment
      update the internal model of the environment
      select a desire
      compose a plan to satisfy the desire
      launch plan
}
```

As it can be seen, the resulting deliberative agent is governed by a cascade of actions enclosed in an infinite loop. This basic form fully suffices for the process of implementing the deliberation of a monolithic conversational agent. Eventual modification to the algorithm and underlying parent frameworks are overviewed, for instance, in [Zbo04].

One of the widely known agent-based collaborative dialogue managers is COLLAGEN (standing for COLLaborative AGENt) [Ric01]. It represents an application-independent platform to provide routine tools for dialogue management. Hence, when narrowed to a specific domain (by being supplied with a set of domain-specific "recipes"), it performs desire recognition, tracks the focus of attention, and maintains an "agenda" of actions that could satisfy the desire. The underlying representation of beliefs and intentions is based on the SharedPlan formalism by Grosz and Kraus [Gro96].

The SPA (standing for Smart Personal Assistant) [Ngu06b] is another agent-based dialogue system to interact with a mobile device a multimodal way. Covering two conversational domains (e-mail and calendar), SPA has been designed as a multi-agent system. The central agent, i.e. the dialogue agent, maintains the conversational context and other domain-specific knowledge as its internal beliefs. The upcoming dialogue processing is done automatically as the result of the BDI interpreter selecting and executing plans according to the current state of beliefs. Each such plan is a modular unit, handling either a discourse-level goal (such as recognizing the user's intention) or a domain-level aspect (such as performing a domain task). Thus, there is a separation between discourse-level and domain-level plans, which enables to reuse discourse-level plans also in additional domains in SPA, or applications other than SPA.

JASPIS [Tur05, Tur03] is another agent-based platform. Unlike with the previous two, JASPIS represents a decentralized approach to creating dialogue systems, with individual components running and communicating remotely. The architecture requires three types of components: agents, evaluators, and managers. Each agent specializes itself in a narrowed problem solving, such as speech output presentation or various dialogue situations handling (ideally one situation per agent; for instance, there may be multiple agents to propose different ways to present a particular system response). Evaluators are used to determine which agents are suitable for an observed situation (for instance, given a system response, which presentation agent fits user's priorities best). Finally, managers are used for execution and overall coordination of actions (for instance, sending the system response to the TTS module).

Hence as obvious from the discussion, agent-based approach is beneficial in that it provides a way for clearly separing *what* the system wants to achieve from *how* it really can achieve it. In other words, it is possible to extract general domain-independent behaviour as agent's initial knowledge base, this way fully supporting easier maintenance and portability to other domains. In addition, the agent-based approach also enables for dealing with more complex dialogues which may involve collaboration, problem solving, negotiation, and so on, either towards the user or among subagents in a multi-agent system. However, a disadvantage is that the agent-based approaches require much more complex resources and processing than any other way to dialogue management.

2.3.5 Probabilistic Dialogue Management

All of the above methodologies accounted for the traditional approach using a set of handcrafted rules, proposed by a dialogue designer on the basis of various decisions. For instance, to deal with potential ASR misrecognitions, the designer had to consider whether and when to confirm user's input (along with whether the ASR confidence score should be the influential factor) [Bui06]. Such expert design is naturally based on experience and commonly agreed guidelines. It also results in an iterative process of designing and testing until the optimal system has been produced [Eck95]. We will not investigate in detail here what the criterium of optimality is. However, the most straightforward criterium is the overall user satisfaction, although there may be other conditions depending on the purpose and nature of the resulting speech application (e.g., in a military application, the criterium of optimality might be the task success rate [Sti01]).

In contrast to the expert way of designing, the family of probabilistic approaches represents an effort to automate the process. Apart of that, it also aims to overcome the limitations observed in the state-based and frame-based approaches. The essence here is to use a corpus of dialogues to extract the necessary decisions of what to do next at each point in a task. One of the popular ways makes use of the *reinforcement learning* (*RL*) in conjunction with *Markov Decision Processes* (*MDPs*) [Hen08, Tsi12]. With the RL, the idea is to specify priorities for the system in terms of a real-valued *reward function* (or *utility function*); optimization then decides what action to take in a given state in order to maximize the immediate *reward* (or *utility*), as well as the total *return* associated with actions in the remainder of a dialogue [Jok10]. In other words, the optimal dialogue policy consists of choosing the best action at each state in a dialogue in order to achieve a given success metric, such as maximum user satisfaction, or a successful and efficient completion of a task.

The underlying MDP carries the *minimal and unambiguous information* to represent a dialogue. Similarly as with classical state automatons, increasing amount of information makes the MDP grow exponentially. In each state of the MDP, the system has generally several choices to pick among. These choices may, for instance, correspond to different dialogue strategies. More formally, the underlying MDP can be described as follows [Jok10]:

- $S = \{s_i\}$ is a set of system states, each representing one point in a dialogue,
- $A = \{a_i\}$ is a collective set of actions that the system can take,
- $P = P(s_i | s_{i-1}, a_j)$ is a probability function of transiting from state s_{i-1} to state s_i by taking action a_j,
- $R = R(s_i, a_j)$ is an immediate reward that is associated with taking a particular action at a given state.

In general, a dialogue conducted by a probabilistic manager begins in a state with all relevant information unknown (e.g., in case of a timetable domain, place of origin and destination, time of departure and arrival, etc.) Over the course of a dialogue, some of these attributes receive values which is reflected by the manager traversing through the dialogue state space, S. The transition from the current state s_i to the next state s_{i+1} is determined by the manager taking action $a_j \in A$ as a response to user's last action observed, and possibly other factors. For instance, these actions may cover questioning the user about the value of an unknown attribute, asking for validation of some known attributes, or clarifying some ambiguities or clashes among attributes [Jok10]. Eventually, the dialogue manager reaches the final state in which all relevant attributes are known and it can successfully query a database for corresponding results.

To determine which action is optimal in each state, the transitions between states must be assigned rewards, $R(s_i, a_j)$. The rewards should reflect consequences of taking an action. This reward may therefore be influenced either by some general principles (e.g., taking an action towards a final state results in a large positive reward), or by user feedback (e.g., user satisfaction as in [Wal98]). Arguably, by always taking the optimal action, the dialogue manager produces an optimal dialogue, finished with the optimal return (sum of rewards). During the phase of "learning", the RL is used to systematically explore the choices and compute the best policy for action selection based on rewards associated with each state transition, using empirical data such as interactions of real of simulated users with the system [Jok10]. Hence this phase of system design typically requires hundreds of dialogues to learn the optimal policy. In the ideal case, all of the possibilities have been explored the same number of times in the training corpora [Fil05, Lit02].

Once trained, there is always an optimal action to take in each state. Assuming state s_i, the optimal action a_{opt} contributes to the final return, Q, as follows [Bel57]:

$$Q(s_i, a) = R(s_i, a) + \gamma \cdot \max_{(a_k)} \sum_{(s_j)} P(s_j \mid s_i, a_k) \cdot Q(s_j, a_k)$$

where $\gamma \in \langle 0;1 \rangle$ is a coefficient to suppress or emphasize the weight of rewards received in the remainder of a dialogue.

One of the well known usages of the probabilistic approach is the NJFun dialogue system [Sin02], providing its users with information on entertainment in New Jersey. The system stores information about the state of a dialogue in the form of a 14-tuple feature vector. In this vector, one feature indicates if the system has greeted the user, and one feature informs which attribute is currently being elicited (entertainment type, place, or time); each of these attributes subsequently occupies four of the remaining features in the vector (attribute value, ASR confidence, number of attempts to elicit it, and strategy used to elicit it). Each of the underlying MDP states is fed in this attribute vector using another kind of vector, this time with seven features: system greeting, pointer to an attribute attempted to elicit, ASR confidence, success of elicitation, number of elicitation attempts, strategy used to elicit the attribute, and indicator of corrections of the attribute value by the user. Considering that each of the attributes in this 7-tuple vector takes on a discrete value, the underlying MDP should consist of 960 unique states. However, as many configurations are invalid (for instance, if the system has not yet greeted, any other actions are forbidden), the final MDP consists of mere 42 states. In each of them, the system determines between two elicitation strategies and two confirmation strategies of a particular information. The optimal dialogue policy was established in the MDP by conducting 311 dialogues with real users who evaluated their interactions as either "good" or "bad", this way contributing to the reward function determination.

The probabilistic approach has a significant benefit in that it does not require strong expertise in dialogue management – the "widely accepted" way of dialogue evolves automatically with enough training data. However, there are also several downsides accompanying this approach. The most notable one is that a trained dialogue manager is not portable to other domains and is also not open to eventual extensions. Hence, there is no other possibility to incorporate changes to an existing domain but retraining the manager. The other drawback is that the number of training dialogues should be great, and each MDP state should be explored ideally the same number of times. If these conditions are not met, the resulting manager is likely to be unprecise as for the optimal policy. Another important aspect is the MDP state space design. On one hand, it must be sufficiently rich in order to support for learning of accurate model, however on the other hand, it must be maximally compact to keep the number of states low [Jok10]. Is the number of states high, the policy might have not been trained thoroughly as some states or transitions might have been visited insufficient number of times. Hence, all of these restrictions make the probabilistic approach an ideal choice only for sealed dialogue systems with minimal to no future updates.

2.4 Summary

Dialogue systems are considered quite an appealing topic nowadays. This chapter has therefore aimed to briefly introduce the realm, and overview the widely established principles. After arguing for and against the use of speech interfaces, we briefly outlined the structure of a uni-modal, voice-controlled system. Then, we focused on the dialogue management module and discussed the "mainstream" approaches to how it can internally be accomplished. That is, our discussion has not covered some less popular approaches, as for instance, the script-based management (represented, for instance, by the Artificial Intelligence Markup Language, AIML; not included as this family of approaches does not construct a proper inner model of a dialogue – instead, it leads to mere reactive agents, usually used only as blind chatting bots), or information state-based management [Tra03] (not included as its extended and more popular variation is the probabilistic dialogue management, discussed in this chapter). We also could see that individual approaches are not strictly distinct, but rather overlap in some aspects. For instance, the E-form stores some additional information in attempt to push the resulting frame-based manager towards an agent. The agent-based approach in turn overlaps with the probabilistic approach in that the MDP may be viewed as the agent's fully expanded deliberation space, hence something that a rational agent constructs dynamically as it explores the surrounding environment.

With such a number of different approaches to dialogue management, it is reasonable to ask which one is the best for a particular application. Obviously, the tractable complexity of the task model, dialogue model, and domain application increases from finite state automatons towards agent-based approaches. Conversational agents that incorporate principles of inter-human rationality and cooperation would seem to be the obvious choice. Certainly, for applications that involve cooperative problem solving with negotiated solutions, the simpler types of dialogue control are not sufficient [Bui06]. On the other hand, for simple applications and for constrained subtasks with some applications, more basic techniques such as some kind of frames may be appropriate.

However, the main point of our discussion in this chapter has been to provide merely a comprehensive overview of approaches, shortlisted as possible candidates for our work (presented in Chapter 4, and evaluated in Chapter 5). This chapter has not aimed to provide an exhaustive coverage of the whole matters, hence the reader has been currently left puzzled with terms like dialogue move, dialogue act, dialogue strategy, or dialogue stack – they all will be explained along the way of describing our work in Chapter 4, where also necessary theories on dialogue modelling will be provided.

Chapter 3

Semantic Interface Language: Definition & Applications

The *Semantic Interface Language* (SIL) [McG91] is a recognizable contribution to the realm of dialogue systems. Developed as part of the SUNDIAL project[1][Fra93], SIL was originally intended only as a means to describe communication between the parser and the dialogue manager. Later on, it turned out that it is well possible to use it as a formalism to represent an arbitrary part of a natural environment by using a comprehensive symbolic notation. This chapter introduces ideas behind SIL, starting with definitions of basic concepts, covering representation of user's utterances (the so called *Utterance Field Objects*, abbr. "*UFOs*"), and heading towards complex principles of dialogue context processing. An exhaustive description of the SIL framework and its application in the realm of dialogue systems may then be found in [Eck95].

3.1 Definition

3.1.1 CoreSIL

As announced above, the SIL formalism will first be considered from the lowest level point of view, the so called SIL*def* concepts. A SIL*def* concept represents a certain object or event in possible use, detailing its structure and eventual operations on it. A set of these (interconnected) SIL*def* concepts then constitute a semantic network-like structure, representing a static real world environment model. More formally in the following definition.

Definition 3.1 (SILdef concept)
Let $\mathbf{C}$ be a set of SIL*def* concepts. A SIL*def* concept $C \in \mathbf{C}$ is an ordered tripple $C = (V, E, S)$, where $V \in C \cup \varnothing$ is a parent concept, $E = \{E_i\}$ is a set of

1 Speech UNderstading in DIALogue

Things	Objects	Living	Natural persons Corporate body
		Non-living	. . . Application-specific
	Properties	Number-related Place-related Time-related Dialogue-related Application-specific	
Events	State	. . .	
	Existence	. . .	
	Action	Mental	Want Believe Doubt
		Affective	Say Use Offer
		Motion	Location changing Departure Arrival

Fig. 3.1 A small excerpt of a possible system of concepts to represent the real world; adopted from [Eck95].

role edges, and $S = \{ S_i \}$ is a set of relations on C. Each role edge $E_i \in E$ is an ordered pair $E_i = (r_i, C_j)$, where r_i is a role in C. □

Based on the definition, we can characterize a SIL*def* concept as follows:

- It typifies a real world object (or event; for simplicity reasons, in the remainder of the text we will refer to any entity as an object).
- The structure of an object is described and further accessible via *roles*. These in turn are described using SIL*def* concepts, making up a hierarchy of nested objects.
- It derives from a parent SIL*def* concept, inherriting all of its properties. These properties involve not only roles but also a certain semantical meaning (see Fig. 3.1), that may be used for more elaborated processing, e.g., for utterance production [You92].
- Each possible operation on a given object is expressed using a *relation* (parametrized with roles and their eventual surroundings).

The definition further implies that given a non-empty set of SIL*def* concepts **C**, there always exists a root concept from which all other concepts in the structure infer. To avoid ambiguity, we will silently assume there exists exactly a single root concept $C_0 = (\emptyset, E_0, S_0)$.

*Example 3.1 (SIL*def *concepts)*
To illustrate the decomposition of the real environment into a set of SIL*def* concepts, let us model a simple time point information, fully specified by its

particular hour and minute. The corresponding SIL*def* concept of type C_{time_point} is appropriate to inherit from the built-in $C_{time\_property}$ parent concept (see Fig. 3.1), not only to gain all of its "prefabricated" common characteristics, but also to facilitate elliptical utterances resolving (covered later). The hour and minute specifics are modelled using edges E_1 and E_2, leading to nested SIL*def* concepts C_{hour} and C_{minute}, respectively, i.e. $E_1 = (\,thehour,\ C_{hour})$, $E_2 = (\,theminute,\ C_{minute})$. We will leave the set of relations empty as so far we do not need to handle C_{time_point} concept any way.

$$\begin{aligned} C_{time\_point} &= (\ V,\ \{E_1, E_2\},\ \varnothing\ ) \\ &= (\ C_{time\_property}, \\ &\qquad \{\ (thehour, C_{hour}),\ (theminute, C_{minute})\ \}, \\ &\qquad \varnothing \\ &\quad) \end{aligned}$$

To stick to the SIL notation throughout this text, each role will be prefixed with the article "the", whereas the concept names themselves will miss this prefix. □

Next, let us define several projection functions on SIL*def* concepts to extract their relevant components.

Definition 3.2 (Projection functions)
Let $C = (\ V,\ E,\ S)$ be a SIL*def* concept. Then, let $V(C) = V$ be the parent concept, $E(C) = E$ be the set of role edges, and $S(C) = S$ be the set of relations defined on C. Let further hold

$$E^*(C) = \begin{cases} E(C) \cup E^*(V(C)) & , \quad \text{if } V(C) \neq \varnothing \\ E(C) & , \quad \text{otherwise} \end{cases}$$

$$V^*(C) = \begin{cases} V^*(V(C)) & , \quad \text{if } V(C) \neq \varnothing \\ C & , \quad \text{otherwise} \end{cases}$$

□

There is a special group of SIL*def* concepts to bear elemental information, with no additional subinformation embedded. An example may be any of the built-in SIL*def* concepts to hold atomic information, i.e. $C_{integer}$, C_{char}, or C_{string}, among others. These concepts derive from the C_{data} super-concept, and their atomic information is accessible via the *value* role. To eliminate ambiguities, we again will silently assume that if there is an elemental concept, it contains no additional roles, as in the following (rather informal) definition.

Definition 3.3 (Elemental concepts)
We say $C \in \mathbf{C}$ is an elemental SIL*def* concept, if $V^*(C) = C_{data} \wedge E^*(C) = \varnothing$. □

*Example 3.2 (SIL*def *concepts continued)*
To finish with the time point example, let us define both C_{hour} and C_{minute} as elemental concepts, deriving from the built-in $C_{integer}$:

$$C_{hour} = (\ C_{integer},\ \varnothing,\ \varnothing\ )\ ,\quad C_{minute} = (\ C_{integer},\ \varnothing,\ \varnothing\ )\ .$$ □

$$\begin{bmatrix} id : tp17 \\ type : time\_point \\ thehour : \begin{bmatrix} id : hour17 \\ type : hour \\ value : 8 \end{bmatrix} \\ theminutes : \begin{bmatrix} id : mins17 \\ type : minutes \\ value : 30 \end{bmatrix} \end{bmatrix}$$

Fig. 3.2 SIL structure representing a time point of 8:30.

3.1.2 SIL Expressions

The above created set of SIL*def* concepts can be considered a passive, static information skeleton, of which any part may be "instantiated", resulting in a so called SIL expression. The following definition formally describes such instances.

Definition 3.4 (SIL expressions)
A SIL expression of an instance of the SIL*def* concept C is an ordered triple $I = I(C) = (D, C, E)$, where D is a unique identifier and $E = \{E_i\}$ is a set of edges. Each edge is an ordered pair $E_i = (r_i, I_j)$ representing a relationship with a (nested) SIL expression $I(C_j)$ via the r_i role. □

Let us make several observations regarding SIL expressions.

- Each instance $I = I(C)$ must comply with its underlying SIL*def* concept C. In other words, roles of I must be a subset of roles of C.
- It is reasonable that each of its edges leads to a unique instance.
- The instance may be subjected projection functions $V(.)$, $E(.)$, $S(.)$, $E^*(.)$, and $V^*(.)$ (see Definition 3.2) in such a way that these are applied to the underlying concept C.
- SIL expressions are finite because their leaves are instances of elemental concepts, containing no further roles.
- Instance uniqueness is guaranteed by its identifier. From the technical point of view, identifiers are best approached by direct memory pointers.

Example 3.3 (SIL expressions)
Pondering Fig. 3.2, we may recognize *tp17*, *hour17*, and *mins17* as particular identifiers for instances of the SIL*def* concepts C_{time_point}, C_{hour}, and C_{minute}, respectively. By no means we say, however, that the instance *time17* is always to have subinstances of both C_{hour} and C_{minute}. According to Definition 3.4, any concept of C_{time_point} is legally instantiated even if it has none of the possible roles. As we will see later, any eventual ambiguity caused by a missing role may be resolved by supplying default values. □

Definition 3.5 (SIL expression projection functions)
Let $I_i = (D, C, E)$ be a SIL expression. Instance I_j of role r_j can be extracted from I_i as $I_j = \Pi(I_i, [r_i])$ if it holds

$$\Pi(I_i,[r_i]) \;=\; \begin{cases} I_j & , \quad \text{if } E_j \in E(I_i) \wedge E_j = (r_j, I_j) \\ \mathit{error} & , \quad \text{otherwise} \end{cases}$$

A path $\Pi(I_i, [r1, \ldots, r_k])$ from instance I_i leads over roles $r1, \ldots, r_k$ to a single SIL expression (accessible from within I_i):

$$\Pi(I_i,[r_1,r_2,\ldots,r_k]) \;=\; \begin{cases} I_i & , \quad \text{if } k = 0 \\ \Pi(I_j,[r_2,\ldots,r_k]) & , \quad \text{if } \exists I_j : I_j = \Pi(I_i,[r_1]) \\ \mathit{error} & , \quad \text{otherwise} \end{cases}$$

We furthermore say I_i is a *parent* of the SIL expression I_j if:

$$\exists r_i : \Pi(I_i, [r_i]) = I_j \,.$$

We say I_i is the *root* of the SIL expression I_j if:

$$\forall I_j \, \exists r_1, \ldots, r_k : \Pi(I_i, [r_1, \ldots, r_k]) = I_j \,.$$

Finally, let $D(I) = D$ be a projection function to extract instance identifier. □

Example 3.4 (SIL expression projection functions)
Given the below SIL expression, we may access its particular instances as follows:

$$\begin{bmatrix} \mathit{id} : \mathit{time}17 & \\ \mathit{type} : \mathit{time} & \\ \mathit{thetimepoint} : & \begin{bmatrix} \mathit{id} : \mathit{tp}17 & \\ \mathit{type} : \mathit{time\_point} & \\ \mathit{thehour} : & \begin{bmatrix} \mathit{id} : \mathit{hour}17 \\ \mathit{type} : \mathit{hour} \\ \mathit{value} : 8 \end{bmatrix} \end{bmatrix} \end{bmatrix}$$

$\Pi(\mathit{time17}, [\mathit{thetimepoint}]) = \mathit{tp17}$, and
$\Pi(\mathit{time17}, [\mathit{thetimepoint}, \mathit{thehour}, \mathit{value}]) = \Pi(\mathit{tp17}, [\mathit{thehour}, \mathit{value}]) = 8$. □

There is an interesting property in the SIL formalism called the *local closure* (*lokale Abgeschlossenheit*, in German). It forbids two unrelated SIL expressions (e.g., user's two utterances) to have a common part. However, this reasonable constraint has to be breached as soon as inevitably recurrent structures are brought into play (covered later).

Definition 3.6 (Local closure)
Let I be the root of a SIL expression. Also, let

$$D^*(I) = \bigcup_{d_1,\ldots,d_k} D(\, \Pi(I_i,[r_1,\ldots,r_k]) \,)$$

be the set of all identifiers used in I. Then I is locally closed if it holds

$$\forall I', \; D(I') \notin D^*(I) : D^*(I) \cap D^*(I') = \varnothing \,. \qquad □$$

Before exemplifying, let us focus on another aspect of the SIL formalism, particularly on SIL*def* relations. *Relations* represent knowledge on how defined concepts depend on each other, and eventually how new knowledge can be inferred from existing one. Let us therefore summarize the characteristics of this component in the following, rather informal definition (more formally in [Eck95]).

Definition 3.7 (Relations)
An n-tuple relation $S_i(x_1, \ldots, x_n) = S_i \in S(C)$ is a projection χ such that χ: $X_1 \times \ldots \times X_n \rightarrow \{fail, success\}$, where X_i is the domain of x_i for $i \in \{1, \ldots, n\}$. □

Let us make several observations regarding relations:

- Each parameter in the S_i relation takes on the form of a path.
- Mutual parameters are usually in a logical relationship rather than in a functional relationship (i.e., they are best approached and further handled using a Prolog-based interpretation system).
- The relation S_i is a local property of the concept C. The relation S_i is evaluated by binding given parameters with instances accessible from within the instance $I(C)$ that triggered it.
- The application of S_i leads either to a "fail" or "success". The only side effect of a "successful" application may be newly bound variables (which eventually may lead to instantiating inferred knowledge).

Relations are the building block of any (dialogue) system that is to incorporate the SIL framework as a means for its knowledge representation.

Example 3.5 (Relations)
Let us extend the C_{time_point} concept defined in Section 3.1 with a relation $S_{hour\_minute\_time}$ that composes the two currently separate pieces of information (hour and time) into a single integer as: $time = 100 \cdot hours + minutes$. By affiliating $S_{hour\_minute\_time}$ to C_{time_point}, we make it accessible for any other concept that infers from C_{time_point}. The relation synopsis may be defined as $S_{hour\_minute\_time}(\ Time,\ Hours,\ Minutes)$. If triggered on instance *tp17* from Fig. 3.2 with parameters taking on the form

$$S_{hour\_minute\_time}(\ [tp17,\ cvalue]\ ,\ [hour17,\ value]\ ,\ [minute17,\ value]\ )\ ,$$

its application is "successful" with the new *cvalue* role of 830 being inferred (Fig. 3.3). For completeness sake, let us add that the *cvalue* role has been inherited from the parent $C_{time\_property}$ built-in concept. Thus, although the relation had the *Time* variable unbound at the beginning, it was capable to determine its value given the remaining bound variables *Hours* and *Minutes*. In addition, given the Prolog-like notion of relations, the natural result of applying $S_{hour\_minute\_time}$ to *tp17* in Fig. 3.3 yields the same SIL expression and the relation finishes with "success". Analogously, it is possible to use $S_{hour\_minute\_time}$ in the "reverse" way by binding the *Time* variable, and asking about the *Hours* and *Minutes* variables. Thus, each relation can be considered an implementation of a set of functions. □

$$\begin{bmatrix} id : tp17\,,\ \ type : time\_point\,,\ \ cvalue : 830 \\ thehour : \begin{bmatrix} id : hour17\,,\ \ type : hour\,,\ \ value : 8 \end{bmatrix} \\ theminutes : \begin{bmatrix} id : mins17\,,\ \ type : minutes\,,\ \ value : 30 \end{bmatrix} \end{bmatrix}$$

Fig. 3.3 SIL expression time point 8:30 with *cvalue* property defined.

3.2 Application 1: Utterance Semantics

3.2.1 Utterance Field Objects

SIL is capable to describe not only grammatically correct utterances but also spontaneous, grammatically incorrect ones. In other words, it is possible to extract and represent merely those units of user's utterance that can be considered most consistent. The utterance can then be represented as a concatenation of these partial pieces. We therefore introduce the so called *utterance field objects* (*UFOs*) which can cover a great part of spontaneous speech phenomenons, and *multiple UFOs* to express the semantics of the whole utterance.

Definition 3.8 (Multiple UFO, MUFO)
Let I_1, …, I_n be roots of SIL expressions and D_1, …, D_n identifiers. A multiple utterance field object (MUFO) is

$$U = \begin{bmatrix} D_1 : \begin{bmatrix} syntax : \begin{bmatrix} string : \text{character\_sequence}_1 \end{bmatrix} \\ semantics : \begin{bmatrix} I_1 \end{bmatrix} \end{bmatrix}, \\ \ldots \\ D_n : \begin{bmatrix} syntax : \begin{bmatrix} string : \text{character\_sequence}_n \end{bmatrix} \\ semantics : \begin{bmatrix} I_n \end{bmatrix} \end{bmatrix} \end{bmatrix}$$

where I_i is the semantical representation of *character_sequence*$_i$. A MUFO with $n = 1$ is denoted as a single utterance field object (UFO). □

An example of MUFO that fulfills this definition is shown in Fig. 3.4.

A dedicated position in the SIL formalism have co-referential expressions, as in the fractional utterance "*the train, which arrives in Erlangen at five, departs from Schwandorf*". Obviously, the nested sentence "[*train*,] *which arrives in Erlangen at five*" extends properties of the train introduced in the primary sentence. Provided that the parser can determine the train object once it has encountered the conjunction "*which*", the utterance is described as shown in Fig. 3.5. This expression is essentially a composition of two separate SIL expressions, each covering one of the two sentences in the utterance [Eck95]. The fact that the primary and referred train objects are the same is caught by co-indexing both of them by the same identifier, *trn16.*

The most notable point in describing co-references is that their resulting SIL expressions are cyclic; for instance, Π(*go16*, [*thevehicle*]) = *trn16* = Π(*trn16*, [*thedesc*, *thevehicle*, *thedesc*, *thevehicle*]). This observation formally impacts some techniques presented thus far (e.g., an expression is now to have a *set of parents*, instead of merely a single one). Elaboration of these impacts is beyond the scope of this text and interested reader may refer to [Eck95] for more details.

3.2.2 Informational Content

Once we have the semantical representation of user's utterance conveyed by a sequence of UFOs, we need to spot and extract those pieces of information that are suitable for a given task or domain. In other words, we need to find the contribution of user's utterance to the problem currently in question.

While this issue is sometimes considered already at lower processing phases of the semantical interpretation, it turns out that this may not be the correct approach, given that certain information conveyed in the natural speech may get lost (especially as for partial UFOs). Therefore by taking this approach, user's utterance may result in supplying no relevant information. This is the immediate cause of all eventual contexts to put this utterance into, be already disposed.

It is therefore introduced the term *eigen information* (*eigentliche Information*, in German) that delimits relevant pieces within the SIL structure. These pieces are projected onto roles of a special-purpose SIL*def* concept that mediates the transfer of them from the input semantics to the dialogue context (using transport relations, e.g., $S_{equality}$). More formally in the following two definitions.

Definition 3.9 (Eigen information, A-parameter)
The set of application-relevant values, A-parameters, is defined as a set of roles $\psi_1, \ldots, \psi_n$ of the concept $C_\psi \in \mathbf{C}$, along with C_ψ-affiliated relations to determine their corresponding values. These resulting values are then holders of eigen information in a particular domain. □

Definition 3.10 (Eigen information extraction)
Let I be a SIL-expression and $I_\psi = I(C_\psi) = \Pi(I, [r_1, \ldots, r_j])$ be an instance accessible from within I. Then eigen information contained in I can be retrieved as $\psi(I) = \cup_{(i)} \langle \psi_i, \Pi(I_\psi, \psi_i) \rangle$. □

Obviously, $\psi(I)$ represents the set of parameter-value pairs. In the special case of I containing no instance of C_ψ, the set $\psi(I)$ is empty, implying I contains no eigen information. Last but not least, the A-parameters themselves can be used to measure the understanding capabilities of a given system.

3.3 Application 2: Dialogue Context

Up to this point, we have concerned ourselves with defining the SIL formalism from the semantic information representation point of view. Beginning with this section, it will be shown how this general framework may be used to process sequential user's utterances. We will first start with merely a single *fully specified* utterance, i.e. an utterance that is not ambiguous and whose content can thus be clearly determined. Afterwards, we will show how a context may be modeled,

$$
U = \left[\begin{array}{ll}
ufo_1 : & \left[\begin{array}{ll}
syntax : & \left[string : \text{to Erlangen} \right] \\
semantics : & \left[\begin{array}{ll}
id : go17 & \\
type : go & \\
thegoal : & \left[\begin{array}{ll}
id : loc17 & \\
type : location & \\
thecity : & \left[\begin{array}{l}
id : city17 \\
type : city \\
value : \text{Erlangen}
\end{array}\right]
\end{array}\right]
\end{array}\right]
\end{array}\right] \\
ufo_2 : & \left[\begin{array}{ll}
syntax : & \left[string : \text{at eight o'clock} \right] \\
semantics : & \left[\begin{array}{ll}
id : time17 & \\
type : time & \\
thehourpoint : & \left[\begin{array}{ll}
id : hp17 & \\
type : hour\_point & \\
thecity : & \left[\begin{array}{l}
id : hour17 \\
type : hour \\
value : 8
\end{array}\right]
\end{array}\right]
\end{array}\right]
\end{array}\right]
\end{array}\right]
$$

Fig. 3.4 MUFO example.

$$
U = \left[\begin{array}{ll}
syntax : & \left[string : \text{the train, which arrives in Erlangen at five, departs from Schwandorf} \right], \\
semantics : & \left[\begin{array}{ll}
id : go16 \ , \ type : go & \\
thevehicle : & \left[\begin{array}{ll}
id : trn16 \ , \ type : train & \\
thedesc : & \left[\begin{array}{ll}
id : go116 \ , \ type : go & \\
thevehicle : & \left[id : trn16 \right] \\
thegoal : & \left[\boxed{\text{Erlangen}} \right] \\
thegoaltime : & \left[\boxed{500} \right]
\end{array}\right]
\end{array}\right] \\
thesource : & \left[\boxed{\text{Schwandorf}} \right]
\end{array}\right]
\end{array}\right]
$$

Fig. 3.5 SIL co-referential expression.

$$
\left[\begin{array}{ll}
id : go43 \ , \ type : go & \\
thegoal : & \left[id : loc43 \ , \ type : location \ , \ value : \text{Erlangen} \right]
\end{array}\right]
$$

$$+$$

$$S_{inverse}(\ [go43],\ [go43, thejourney, thejourneyevent]\ )$$

$$\downarrow$$

$$
\left[\begin{array}{ll}
id : go43 \ , \ type : go & \\
thegoal : & \left[id : loc43 \ , \ type : location \ , \ value : \text{Erlangen} \right] \\
thejourney : & \left[\begin{array}{ll}
id : sjn43 \ , \ type : single\_journey & \\
thejourneyevent : & \left[id : go43 \ , \ type : go43 \right]
\end{array}\right]
\end{array}\right]
$$

$$+$$

$$S_{equality}(\ [go43, thejourney, thearrival, theplace],\ [go43, thegoal]\ )$$

$$\downarrow$$

$$
\left[\begin{array}{ll}
id : go43 \ , \ type : go & \\
thegoal : & \left[id : loc43 \ , \ type : location \ , \ value : \text{Erlangen} \right] \\
thejourney : & \left[\begin{array}{ll}
id : sjn43 \ , \ type : single\_journey & \\
thejourneyevent : & \left[id : go43 \ , \ type : go43 \right] \\
thearrival : & \left[\begin{array}{l}
id : arr43 \ , \ type : arrival \\
theplace : loc43
\end{array}\right]
\end{array}\right]
\end{array}\right]
$$

Fig. 3.6 $S_{inverse}$ relation application example.

given a sequence of user's utterances. Finally, we will return to the ambiguous utterances problem and revise our approaches to accommodate the solution. At the end of this chapter, our investigation will therefore result into a representation of a general (collaborative) dialogue context.

3.3.1 Elaborating User's Utterance Semantics

As outlined above, let us first start with showing how a single utterance needs to be handled in order for it to be further processable in the dialogue context scale. The approach presented here uses elemental and locally operating relations, defined and exemplified in Section 3.1.2. To describe the approach, let us first categorize each relation, based on its purpose in the system:

- *Necessary processing relation* represents an axiom or theorem valid for a given environment. For instance, time information of *seventeen ten* may be generally split into hour and minute values (and this way $C_{time\_property}$ upcasted to C_{time_point}), and vice versa (downcasting; see also Fig. 3.3).

$$seventeen\ ten \quad \Leftrightarrow \quad \begin{Bmatrix} hour = 17 \\ minute = 10 \end{Bmatrix} \quad \Leftrightarrow \quad 17:10$$

- *Default-value relation* supplies a default value for a particular undefined concept role, reducing the degree of freedom of an underspecified object. In other words, this kind of relation may be considered describing common knowledge. For instance, time information of *thirteen o'clock* may be explained by the system as a demand to set *minutes* = 0.

$$thirteen \quad \Leftrightarrow \quad \begin{Bmatrix} hour = 13 \\ minute = 0 \end{Bmatrix} \quad \Leftrightarrow \quad 13:00$$

- *Identification relation* ($S_{equality}$) is to extend one object with another by establishing a *reference* to it. In the following example, the identification relation assures a propagation of the built-in *thegoal* role value to the *thearrival* custom role.

$$\begin{bmatrix} id : go33\ ,\ type : go \\ thegoal : \begin{bmatrix} id : loc33\ ,\ type : location\ ,\ value : \text{Erlangen} \end{bmatrix} \end{bmatrix}$$

$$+$$

$$S_{equality}(\ [go33, thejourney, thearrival, theplace],\ [go33, thegoal]\ )$$

$$\downarrow$$

$$\begin{bmatrix} id : go33\ ,\ type : go \\ thegoal : \begin{bmatrix} id : loc33\ ,\ type : location\ ,\ value : \text{Erlangen} \end{bmatrix} \\ thejourney : \begin{bmatrix} id : sjn33\ ,\ type : single\_journey \\ thearrival : \begin{bmatrix} id : arr33\ ,\ type : arrival \\ theplace : loc33 \end{bmatrix} \end{bmatrix} \end{bmatrix}$$

- *Recurrent identification relation* ($S_{inverse}$) is to create cyclic structures, required to properly represent utterances with cross- and co-references. The example in Fig. 3.6 shows an application of the $S_{inverse}$ relation to create a co-reference (middle structure), followed by application of the above already shown $S_{equality}$ relation (bottommost structure).

These kinds of relations in fact describe the whole interpretation mechanism. Their particular order in the list represents their importance in the interpretation frame (thus for instance, no *default-value* relation must be used as long as there are *necessary* relations fired).

Definition 3.11 (Inferrention)
Let **C** be a SIL*def* concept hierarchy and $I = I(C)$ an instance of concept $C \in \mathbf{C}$. The inferential step $I \rightarrow I'$ is achieved using relation $S \in S(C)$ whose side effect causes a role from I to be bound ($I' \supset I$). In addition, we say I is maximally inferred if $\forall S \in S(C)$: $I \rightarrow I$. We say I^* is the maximal interpretation of I if I^* is maximally inferred. □

Apparently, for each SIL-expression $I = I(C)$ there exists its maximal interpretation I^* resulting from the application of relations S_1, ..., S_n such that

$$I \xrightarrow{S_1} I' \xrightarrow{S_2} \dots \xrightarrow{S_n} I^* \ .$$

Assuming I is the root of user's particular utterance, then by maximally interpreting it, we have brought it to the stable state from where no further knowledge can be inferred. Of course the validity and entirety if this inferrention is not guaranteed by the SIL framework itself but is always implied from and dependent on a given design of a particular domain. We may call this the "consistency" - the system of rules is "consistent" as long as its underlying set of relations is neither underspecified (some real world objects relationships are ignored) nor over-specified (some real world relationships work against each other). Despite this is an important aspect in properly designing a SIL-based system, we will leave this topic out due to space reasons. An interesting discussion on impacts and recovery from both of these special cases may be found in [Eck95].

3.3.2 Representing Dialogue Context

The above outlined interpretation mechanism easily allows user's utterance to achieve a stable expanded state using a set of relations. It is important to note that this expansion is strictly monotonous, meaning no modification nor negation can be considered. During the course of a dialogue, on the other hand, informational state on either partner's side may be modified or retracted. The monotonous interpretation therefore turns out to provide a too weak approach for a dialogue context representation and needs to be further revised in order to accommodate the mentioned dialogue phenomena.

To overcome the implied monotony, it is introduced the so called *interpretation worlds.* This new axis sets each incoming utterance into its "own" interpretation world where it can be elaborated in a monotonous way. Using this approach, no object is necessary to be modified, retracted, or even negated (as by committing any of these operations would cause the whole sequence knowledge to cease to be valid). Using a new blank world, the utterance maximal interpretation can be determined without clashing with any of the previous interpretations.

Interpretation worlds are stacked onto each other. By elaborating an utterance in its own world, a new view is made and incorporated into the knowledge base. The current state of a dialogue can then be determined by projecting these layers, making sure that recent objects always hide older ones.

The stacked structure exposes the following apparent properties.

- The set **C** of SIL*def* concepts is located at the bottom of the stack and constitutes the so called *basic view*. In this basic view, apparently no new instances are created - it serves merely to describe the real world objects and their relations. Once this world has been initialized at the beginning, it remains unchanged until the end of the dialogue.

- A world always inherits all objects (along with their mutual relations) from the world laying beneath it. In the simplest case, if an object has a particular role value undefined, the projection process makes sure that it receives a value from the top-most "compatible" object - we say the objects are *unified*. Obviously, the case of binding undefined and defined role values is merely a special case of extending one value with another. This more general case arises, for instance, when precising one time information ("*tomorrow*") with another ("*morning*") to assign the result to the object corresponding role ("*tomorrow morning*"). At this point, let us recall that we assume here user's fully specified utterances with no ambiguities; the determination of meaning of underspecified objects will be covered in the next section, and in more detail then in [Eck95].

- Instances in a given world always overlay incompatible instances in the world beneath it. We may call such two incompatible instances "concurrent". Concurrent instances usually occur when the user corrects previously wrong understood piece of information. This correction involves the rejection of the actual interpretation, implying no previous information must be used.

Example 3.6 (Compatible objects)
To illustrate the idea of "stacked worlds", ponder the dialogue snippet shown below (adopted from [Eck95]). In utterance U_1, the instance *go1* with the role *thesource* is created with Bonn as the determined place of departure (see Fig. 3.7 and Fig. 3.8). After interpreting U_2, a new world with the instance *go2* is created with the description of the place of arrival. Finally, once interpreted U_3 in another new world, there are three compatible *go*-typed objects that can be unified in order to put all three originally separated pieces of information into perspective (the user wants to go from Bonn to Erlangen at eight o'clock).

U_1	I want to go from Bonn.
S_1	From Bonn. Where do you want to go to?
U_2	To Erlangen.

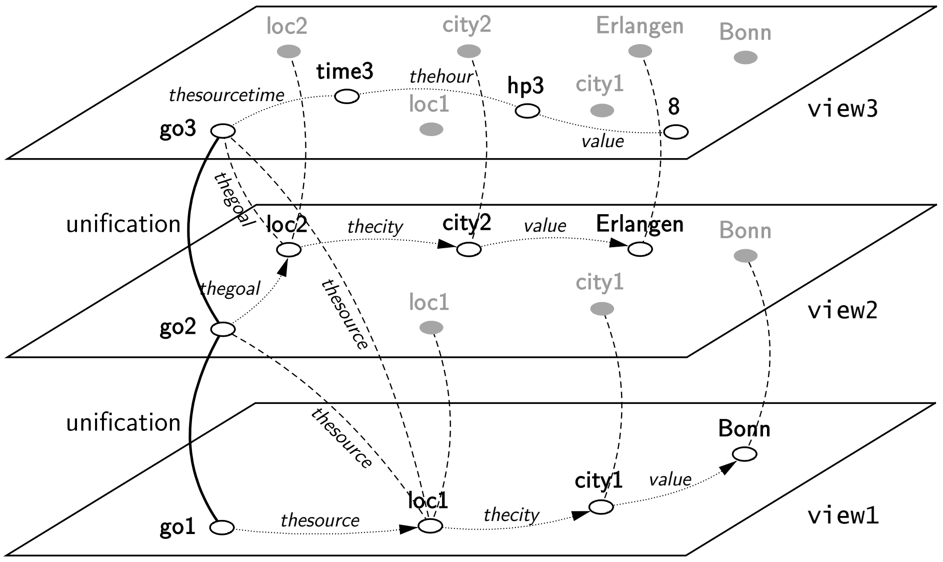


Fig. 3.7 Compatible objects example.

U_1 : I want to go from Bonn.

S_1 : From Bonn. Where do you want to go to?

$$\mathit{view}_1 : \begin{bmatrix} \mathit{ufo}_1 : \begin{bmatrix} \mathit{syntax} : \begin{bmatrix} \mathit{string} : \text{from Bonn} \end{bmatrix} \\ \mathit{semantics} : \begin{bmatrix} \mathit{id} : \mathit{go1} \;,\; \mathit{type} : \mathit{go} \\ \mathit{thesource} : \begin{bmatrix} \mathit{id} : \mathit{loc1} \;,\; \mathit{type} : \mathit{location} \\ \mathit{thecity} : \begin{bmatrix} \mathit{id} : \mathit{city1} \;,\; \mathit{type} : \mathit{city} \\ \mathit{value} : \text{Bonn} \end{bmatrix} \end{bmatrix} \end{bmatrix} \end{bmatrix} \end{bmatrix}$$

U_2 : To Erlangen.

S_2 : At what time do you want to go to Erlangen?

$$\mathit{view}_2 : \begin{bmatrix} \mathit{ufo}_2 : \begin{bmatrix} \mathit{syntax} : \begin{bmatrix} \mathit{string} : \text{to Erlangen} \end{bmatrix} \\ \mathit{semantics} : \begin{bmatrix} \mathit{id} : \mathit{go2} \;,\; \mathit{type} : \mathit{go} \\ \mathit{thegoal} : \begin{bmatrix} \mathit{id} : \mathit{loc2} \;,\; \mathit{type} : \mathit{location} \\ \mathit{thecity} : \begin{bmatrix} \mathit{id} : \mathit{city2} \;,\; \mathit{type} : \mathit{city} \\ \mathit{value} : \text{Erlangen} \end{bmatrix} \end{bmatrix} \end{bmatrix} \end{bmatrix} \\ \mathit{ufo}_1 : \begin{bmatrix} \mathit{syntax} : \begin{bmatrix} \mathit{string} : \text{from Bonn} \end{bmatrix} \\ \mathit{semantics} : \begin{bmatrix} \mathit{id} : \mathit{go1} \;,\; \mathit{type} : \mathit{go} \\ \mathit{thesource} : \begin{bmatrix} \mathit{id} : \mathit{loc1} \;,\; \mathit{type} : \mathit{location} \\ \mathit{thecity} : \begin{bmatrix} \mathit{id} : \mathit{city1} \;,\; \mathit{type} : \mathit{city} \\ \mathit{value} : \text{Bonn} \end{bmatrix} \end{bmatrix} \end{bmatrix} \end{bmatrix} \end{bmatrix}$$

U_3 : At eight o'clock.

$$\mathit{view}_2 : \begin{bmatrix} \mathit{ufo}_3 : \begin{bmatrix} \mathit{syntax} : \begin{bmatrix} \mathit{string} : \text{eight o'clock} \end{bmatrix} \\ \mathit{semantics} : \begin{bmatrix} \mathit{id} : \mathit{go3} \;,\; \mathit{type} : \mathit{go} \\ \mathit{thesourcetime} : \begin{bmatrix} \mathit{id} : \mathit{tp3} \;,\; \mathit{type} : \mathit{time\_point} \;,\; \mathit{cvalue} : 800 \\ \mathit{thehour} : \begin{bmatrix} \mathit{id} : \mathit{hour3} \;,\; \mathit{type} : \mathit{hour} \\ \mathit{value} : 8 \end{bmatrix} \end{bmatrix} \end{bmatrix} \end{bmatrix} \\ \mathit{ufo}_2 : \begin{bmatrix} \mathit{syntax} : \begin{bmatrix} \mathit{string} : \text{to Erlangen} \end{bmatrix} \\ \mathit{semantics} : \begin{bmatrix} \mathit{id} : \mathit{go2} \;,\; \mathit{type} : \mathit{go} \\ \mathit{thegoal} : \begin{bmatrix} \mathit{id} : \mathit{loc2} \;,\; \mathit{type} : \mathit{location} \\ \mathit{thecity} : \begin{bmatrix} \mathit{id} : \mathit{city2} \;,\; \mathit{type} : \mathit{city} \\ \mathit{value} : \text{Erlangen} \end{bmatrix} \end{bmatrix} \end{bmatrix} \end{bmatrix} \\ \mathit{ufo}_1 : \begin{bmatrix} \mathit{syntax} : \begin{bmatrix} \mathit{string} : \text{from Bonn} \end{bmatrix} \\ \mathit{semantics} : \begin{bmatrix} \mathit{id} : \mathit{go1} \;,\; \mathit{type} : \mathit{go} \\ \mathit{thesource} : \begin{bmatrix} \mathit{id} : \mathit{loc1} \;,\; \mathit{type} : \mathit{location} \\ \mathit{thecity} : \begin{bmatrix} \mathit{id} : \mathit{city1} \;,\; \mathit{type} : \mathit{city} \\ \mathit{value} : \text{Bonn} \end{bmatrix} \end{bmatrix} \end{bmatrix} \end{bmatrix} \end{bmatrix}$$

Fig. 3.8 Compatible objects interpretation worlds

S_2 At what time do you want to go to Erlangen?
U_3 At eight o'clock. □

Example 3.7 (Concurrent objects)
A simple example is shown in Fig. 3.9 with the simplified resulting structure depicted in Fig. 3.10 (adopted from [Eck95]). In *view2*, the role *thesource* of the instance *go2* was assigned a new value. Therefore from now on, the instance *go1* in *view1* is overlaid and inaccessible whenever targeted via [*go*, *thesource*, *thecity*, *value*] - the new value Nürnberg will be used instead. □

3.3.3 Disambiguing User's Utterances

Once we have seen how user's fully specified (i.e. unambiguous) utterances are represented in the form of stacked worlds, we need to further extend this approach by allowing for processing of elliptical (i.e. ambiguous) utterances.

In a dialogue system, the processing of elliptical utterances is a crucial feature as such utterances are very often spoken by users: for instance, being asked for

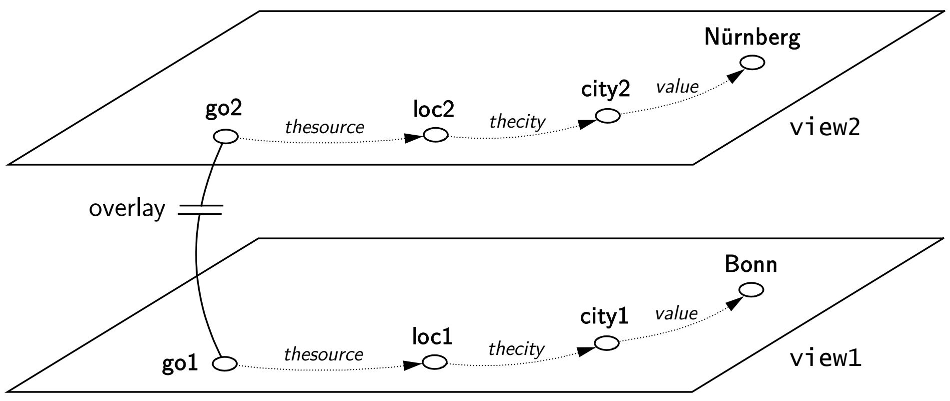


Fig. 3.9 Incompatible (concurrent) objects example.

U_1 : I want to go from Bonn.

$$view_1 : \begin{bmatrix} syntax : \begin{bmatrix} string : \text{from Bonn} \end{bmatrix} \\ semantics : \begin{bmatrix} id : go1 \ , \quad type : go \\ thesource : \begin{bmatrix} id : loc1 \ , \quad type : location \\ thecity : \begin{bmatrix} id : city1 \ , \quad type : city \\ value : \text{Bonn} \end{bmatrix} \end{bmatrix} \end{bmatrix} \end{bmatrix}$$

S_1 : Do you want to depart from Bonn?
U_2 : From Nürnberg.

$$view_2 : \begin{bmatrix} syntax : \begin{bmatrix} string : \text{from Nürnberg} \end{bmatrix} \\ semantics : \begin{bmatrix} id : go2 \ , \quad type : go \\ thesource : \begin{bmatrix} id : loc2 \ , \quad type : location \\ thecity : \begin{bmatrix} id : city2 \ , \quad type : city \\ value : \text{Nürnberg} \end{bmatrix} \end{bmatrix} \end{bmatrix} \end{bmatrix}$$

Fig. 3.10 Incompatible (concurrent) objects interpretation worlds.

$$(a)\quad \begin{bmatrix} \mathit{syntax}: \begin{bmatrix} \mathit{string}: \text{departure at ten} \end{bmatrix} \\ \mathit{semantics}: \begin{bmatrix} \mathit{id}: \mathit{dep}45, \quad \mathit{type}: \mathit{depart} \\ \mathit{thetime}: \begin{bmatrix} \mathit{id}: \mathit{tp}45, \quad \mathit{type}: \mathit{time\_point} \\ \boxed{\text{at ten}} \end{bmatrix} \end{bmatrix} \end{bmatrix}$$

$$(b)\quad \begin{bmatrix} \mathit{syntax}: \begin{bmatrix} \mathit{string}: \text{departure at <timePlaceholder>} \end{bmatrix} \\ \mathit{semantics}: \begin{bmatrix} \mathit{id}: \mathit{dep}45, \quad \mathit{type}: \mathit{depart} \\ \mathit{thetime}: \begin{bmatrix} \mathit{id}: \mathit{tp}45, \quad \mathit{type}: \mathit{time\_point} \end{bmatrix} \end{bmatrix} \end{bmatrix}$$

$$(c)\quad \begin{bmatrix} \mathit{syntax}: \begin{bmatrix} \mathit{string}: \text{at ten} \end{bmatrix} \\ \mathit{semantics}: \begin{bmatrix} \mathit{id}: \mathit{tp}61, \quad \mathit{type}: \mathit{time\_point} \\ \mathit{thehour}: \begin{bmatrix} \mathit{id}: \mathit{hour}61, \quad \mathit{type}: \mathit{hour} \\ \mathit{value}: 10 \end{bmatrix} \end{bmatrix} \end{bmatrix}$$

Fig. 3.11 SIL-based disambiguation; (a) fully specified information, (b) system expectation, (c) result of disambiguation.

the departure time by a timetable dialogue system, the user may simply respond "*at five*", instead of "*I want to depart at five*". Hence, there are two essential questions at this point: how can be SIL applied to predict actual context for a response, and how can such prediction be further used to resolve user's elliptical utterances. Before answering either of them, let us summarize our problem in the following two definitions.

Definition 3.12 (Interpretability)
We denote an utterance I as interpretable if it conveys any eigen information, i.e. it holds that $\psi(I) \neq \varnothing$. □

Definition 3.13 (Disambiguation)
An utterance I can be disambiguated using a dialogue context I_c if I on its own is not interpretable but in conjunction with I_c yields an unambiguous interpretation:

$$\psi(I) = \varnothing \quad \wedge \quad \psi(I_c) = \varnothing \quad \wedge \quad \psi(I \cup I_c) \neq \varnothing. \qquad \square$$

Typically, disambiguing an utterance I using a dialogue context I_c means making I accessible from within I_c. That is, I_c can be understood as the logical surrounding of I:

$$\exists s_1, \ldots, s_k \colon \Pi(\, I_k,[s_1, \ldots, s_k]\,) = I\,.$$

To demonstrate the principle, ponder Fig. 3.11 in whose upper part (a), user's fully specified information "*departure at ten*" is shown. In this case, the user not only supplies the particular time information (*at ten*) but also narrows its scope to *departure*. Reversing our thoughts, should the user say merely "*at ten*" (utterance I with $\psi(I) = \varnothing$) at the moment the system asked for a departure time (context I_c with $\psi(I_c) = \varnothing$), user's response could be explained as conveying departure time (because $\psi(I \cup I_c) \neq \varnothing$). This leads to the SIL-expression (b) that is the result of replacing the particular time information with a placeholder, or formally *anchor point* (*Ankerpunkt*). Technically, the placeholder $I_a = \mathit{tp45}$ of type $C_{\mathit{time\_point}}$ is instantiated but does not contain any specific roles nor meaningful value. More importantly, it is compatible with any upcoming elliptical time concept, and can therefore be unified with $I = tp61$ shown in (c). We call such kind of unification *anchoring* (*Verankerung*) and will say that I has been anchored in context I_a. From this perspective, the process of anchoring can be perceived as prepended to the utterance interpretation step (covered in Section 3.3.1), as depicted in Fig. 3.12.

Over the course of a dialogue, there of course may be multiple dialogue contexts candidates I_{C1}, …, I_{Cn} that correspond to different anchorings. Application of any of these contexts must naturally result into a unique set of eigen information, i.e.

$$\forall \mathrm{I}: \ \psi(I_{Ci} \cup I) \neq \varnothing \quad \wedge \quad \psi(I_{Cj} \cup I) \neq \varnothing \quad \rightarrow \quad \psi(I_{Ci} \cup I) = \psi(I_{Cj} \cup I) \ .$$

The process of anchoring can therefore be summarized as follows [Eck95]:

- Isolately interpretable utterances (with $\psi(I) \neq \varnothing$) do not need to be anchored.
- Only non-leaf concepts may serve as anchoring points (e.g., $C_{location\_property}$ in Fig. 3.1). Contrarily, leaf concepts cannot be used as anchoring points.
- The process of anchoring is possible by instantiating a *virtual context* whose description contains an empty information (placeholder) expected in the very next user's utterance. Once supplied, the elliptical concept then takes on the corresponding role within the virtual context.

3.4 Application 3: System Utterance Semantics

Receiving user's turn, the dialogue manager constructs a set of the so called *moves*, i.e. possible continuations in the dialogue with respect to its current state. Each of the moves is to inform the user about certain conceptual content. Therefore each such move has assigned the communicative purpose indicated by a particular *dialogue act* label, while the conceptual content is indicated by a reference to the dialogue context. Consider the sentence "*You want to travel from Schwandorf to Erlangen. What time do you want to travel on*?", consisting of two moves, each expressed using its own UFO:

$$\begin{bmatrix} \begin{array}{ll} id : ufo4 & \\ card : 2 & \\ ufo_1 : & \begin{bmatrix} \begin{array}{ll} id : ufo5 & \\ dialogue : & \begin{bmatrix} dact : confirm \end{bmatrix} \\ semantics : & \begin{bmatrix} \begin{array}{l} id : dbtrain1 \\ sourcecity : \boxed{\text{Schwandorf}} \\ goalcity : \boxed{\text{Erlangen}} \end{array} \end{bmatrix} \end{array} \end{bmatrix} \\ ufo_2 : & \begin{bmatrix} \begin{array}{ll} id : ufo6 & \\ dialogue : & \begin{bmatrix} dact : open\_request \end{bmatrix} \\ semantics : & \begin{bmatrix} \begin{array}{l} id : dbtrain1 \\ thesourcetime : \begin{bmatrix} id : tp6 \ , \quad type : time\_point \end{bmatrix} \end{array} \end{bmatrix} \end{array} \end{bmatrix} \end{array} \end{bmatrix}$$

The first move (*ufo1*) intends to confirm the departure and arrival cities (dialogue act *confirm*, and conceptual content targeting the two cities in the

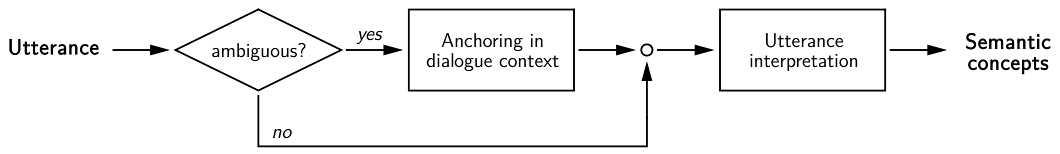


Fig. 3.12 Utterance processing with and without dialogue context.

dialogue context), while the second move (*ufo2*) aims to elicit the departure time (dialogue act *open_request*, and instance *tp6* with undefined value).

Provided the abstract description of the intents in a system utterance, the next step is the production of the system utterance semantics itself. Simple planning rules are used to specify the type of semantic description to be extracted from the dialogue context. For example, *ufo1* triggers a rule which is to describe a number of objects, O_1, …, O_n (in our case $n = 2$) that in the simplest case have a common parent object O_p. The resulting description is then the SIL representation of O_p that contains sub-descriptions for all of O_1, …, O_n. The production of the SIL representation is driven by a grammar-lexicon. The underlying algorithm can be understood as the process of finding a lexical candidate that best matches the input, and then recurrently generate its arguments.

Example 3.8 (Utterance semantics production)
Fig. 3.13 shows an example of a lexicon entry for the C_{arrive} SIL*def* concept. Given the inheritance paradigm, this entry is applicable for any instance that is of or derives from C_{arrive}. Naturally, in derived concepts, the descriptive content may be overridden. For instance, while it is reasonable to use the verb "*go*" when expressing surface traveling, it is more appropriate to substitute it with "*fly*" when speaking about flying. This may be captured by two sibling concepts, both inheriting a shared part from C_{arrive}. In the case of C_{arrive}, this shared part are the syntactical arguments, each of which with its own *syntax*, *semantics*, *order* (to indicate the surface position with respect to the head, i.e. "lexical parent"), etc. For instance, *theplace* argument is optional and can occur at any position after the head. □

Youd and McGlashan [You92] further describe the application of the SIL formalism as a means for utterance production. The impacts on and implied feedback to the dialogue manager module are then further discussed in [Eck95].

3.5 Summary

This chapter conceived with the Semantic Interface Language (SIL) formalism. With roots in the SUNDIAL project [Fra93], SIL was developed as a methodology for modeling semantic contents with focus on its use to be maximally application- and language-independent. To prove the universality of the formalism, three possible applications comprising different parts of a dialogue system were presented in this chapter: utterance semantics (original aim of SIL), dialogue context representation, and finally system utterance production.

$$
\begin{bmatrix}
\mathit{lexeme} : \mathit{arrive} \\
\mathit{syntax} : \begin{bmatrix}
\mathit{head} : \mathit{major} : \mathrm{v} \\
\mathit{args} : \left\langle
\begin{bmatrix}
\mathit{syntax} : \begin{bmatrix} \mathit{head} : \mathit{major} : \mathrm{n} \end{bmatrix} \\
\mathit{semantics} : \begin{bmatrix} \mathit{id} : A \\ \mathit{type} : \mathit{object} \end{bmatrix} \\
\mathit{order} : \begin{bmatrix} \mathit{dir} : \mathit{pre} \\ \mathit{adj} : \mathit{any} \\ \mathit{opt} : \mathit{oblig} \end{bmatrix}
\end{bmatrix}
\begin{bmatrix}
\mathit{syntax} : \begin{bmatrix} \mathit{head} : \mathit{major} : \mathrm{prep(at)} \end{bmatrix} \\
\mathit{semantics} : \begin{bmatrix} \mathit{id} : B \\ \mathit{type} : \mathit{location} \end{bmatrix} \\
\mathit{order} : \begin{bmatrix} \mathit{dir} : \mathit{post} \\ \mathit{adj} : \mathit{any} \\ \mathit{opt} : \mathit{opt} \end{bmatrix}
\end{bmatrix}
\begin{bmatrix}
\mathit{syntax} : \begin{bmatrix} \mathit{head} : \mathit{major} : \mathrm{prep(at)} \end{bmatrix} \\
\mathit{semantics} : \begin{bmatrix} \mathit{id} : C \\ \mathit{type} : \mathit{time} \end{bmatrix} \\
\mathit{order} : \begin{bmatrix} \mathit{dir} : \mathit{post} \\ \mathit{adj} : \mathit{any} \\ \mathit{opt} : \mathit{opt} \end{bmatrix}
\end{bmatrix}
\right\rangle
\end{bmatrix} \\
\mathit{semantics} : \begin{bmatrix}
\mathit{id} : \mathrm{G} \\
\mathit{type} : \mathit{arrive} \\
\mathit{thetheme} : \begin{bmatrix} \mathit{id} : A \\ \mathit{type} : \mathit{object} \end{bmatrix} \\
\mathit{theplace} : \begin{bmatrix} \mathit{id} : B \\ \mathit{type} : \mathit{location} \end{bmatrix} \\
\mathit{thetime} : \begin{bmatrix} \mathit{id} : C \\ \mathit{type} : \mathit{time} \end{bmatrix}
\end{bmatrix}
\end{bmatrix}
$$

Fig. 3.13 A lexicon entry for the C_{arrive} SIL*def* concept.

Chapter 4

daisy Dialogue Management Framework

In this chapter, the Daisy dialogue management framework, developed during the study, will be presented in detail. The framework has been designed to provide (a priori) uni-modal support for dialogue management, however, additional modalities can be intuitively added by simply incorporating them into particular dialogue plans. The framework follows an agent-based approach in which the dialogue model is explicitly represented in agent's plans and beliefs. The dialogue agent uses its beliefs to maintain information about the dialogue, including dialogue history and the salient list, as well as the domain-specific knowledge. Plans then govern the interaction with both the user and other parts of the system.

This chapter begins with arguing for the decision for the agent-based approach to dialogue management. The description of the framework itself then begins with presenting the representation of the real world dialogue environment that has been to an extent motivated by the SIL formalism [Eck95, Fra93]. The remainder of the chapter then thoroughly concerns with the dialogue processing as it occurs in the Daisy framework. The description can be informally divided into three major parts targeting agent's beliefs, intentions, and desires. The investigation is enriched with sample dialogue snippets to illustrate problems in question as they arise at different stages of the dialogue processing.

4.1 Reasons to Opt for Agent-based Approach

As discussed in Chapter 2, most of business applications are focused on practical tasks, as for instance telephone-based information inquiry or booking of different kinds. These applications employ relatively simple dialogue models, in which the dialogue state may be well modelled using finite state machines or a hierarchy of frames. Nonetheless, state-based dialogue models are very limited because of their inflexibility and high costs required to overcome this constraint. Although this significant limitation is addressed by frame-based dialogue models, also they suffer from different disadvantages. One of them is the lack of modularity – principles that model the intended dialogue behaviour cannot be shared, unless they exist in duplicitous instances. In conjunction with stochastic approaches, all of these dialogue models have one in common: they are difficult to extend, e.g. with another task or another common behaviour.

Using this method of exclusion, the agent-based approach seems to be of the best usability with respect to creating a general-purpose dialogue environment, as was the goal. The essential notion of an agent also fits the needs of a dialogue management in a natural way. In particular, the dialogue is supposed to be mainly user-driven. Nonetheless, system initiative is also essential in order to clarify user's requests. Hence, from the agent's point of view, the dialogue management requires some degree of pro-activeness to recover from errors, and at the same time also reactiveness to fulfill user's requests. The fact of having to be pro-active as well as reactive in different situations to meet the dialogue objectives is a significant sign to opt for an agent-based approach, specifically the BDI architecture (Section 2.3.4). Another benefit of taking this course is the possibility to decompose a system into a set of autonomous, specialized agents, each dedicated to handle a certain isolated part of the system, as for instance shown in [Tur05, Ngu06b]. Last but not least, unlike with frame-based or stochastic approaches, agent's behaviour can be well controlled and eventually reused between different applications, as for instance shown in [Gus03, Tur05].

4.2 Domain Data Model (DDM)

To begin our investigation of the Daisy framework, let us first introduce a way of defining the "static" part of a domain, in charge of structuring agent's beliefs – the *domain data model* (*DDM*). From a certain point of view, DDM can be understood as a modified *Core*SIL. The common goal of these two approaches is to model (a subset of) the real environment using a network of interconnected concepts. The essential distinction is, however, that while in SIL these concepts take on some inherited meaning along with corresponding taxonomy, in DDM these concepts are of a purely "data-centric" notion with any meaning being given to them no earlier than within a particular plan instance.

The loose coupling of data and meaning allows for creation of explicit and strongly tied structures of information. We call these structures *collections*. The purpose of a collection is to group "similar" pieces of information, this way allowing for more sophisticated dialogue processing at later stages.

This section covers the definition and syntax of DDM. Despite all distinctions, DDM may be well described in a very similar fashion like *Core*SIL. Given this fact, some of the upcoming definitions will merely slightly differ from those already encountered in Chapter 3. However, the main distinction will be in defining the model not only from the elemental concepts standpoint but also from their grouping collections standpoint.

Definition 4.1 (DDM concept, DDM collection, and data type)
Let $\mathbf{C}$ be a set of domain data model (DDM) concepts. A *DDM concept*, $C \in \mathbf{C}$, is identified by its domain-wide unique name and has no further parameters, i.e. $C_{name} = ()$.

Let $\mathbf{K}$ be a set of domain data model collections. A *DDM collection*, $K \in \mathbf{K}$, is identified by its domain-wide unique name and represented as an ordered triple $K_{name} = (C, E, T)$, where $C = \{C_i\}$ is a non-empty set of DDM concepts, $E = \{E_i\}$ is a set of edges to parent DDM concepts, and T is a data type for each DDM concept $C_i \in \mathbf{C}$ to hold. Furthermore, an edge $E_i \in E$ is an ordered pair $E_i = (C_i, R_i)$, where C_i is a DDM concept, and $R_i \in \mathbf{N}^+$ is the maximum cardinality of K_{name} with respect to C_i.

A data type, $T_i = \{f_j\}$, is a set of functions that completely and unambiguously define the range and operations with an eigen information ψ_i. □

Before exemplifying, let us point out several characteristic about DDM concepts and collections:

- Concepts represents real-world or abstract objects. Each concept is an exclusive part of its containing collection. For instance, given that a *train* and *bus* are two similar real world objects intended for transportation, both may be groupped under a *transportation means* collection and cannot be members of any other collection.

- Concepts contain subcollections. This recurrent pattern is further constraint by each concept being allowed to contain at most one collection of a given name.

- Each edge is assigned a cardinality resembling an "ERA model-like" relationship "1:1" or "1:N" (the remaining relationships, "N:1" and "M:N", are performed as needed automatically by the framework over the course of a dialogue).

Example 4.1 (DDM concepts and DDM collections)
To demonstrate the DDM terms, let us revisit the time point object we already encountered in Example 2.1 when examining the SIL formalism. Recall that a time point is specified by its particular hour and minute information. A time point DDM concept is identified by its DDM-wide unique name, i.e.

$$C_{time\_point} = () \ .$$

It is reasonable for us to define the C_{time_point} concept as an *exclusive* part of the K_{time_point} collection. This collection contains no other concepts, i.e. C_{time_point} is not *collectible* with any other object:

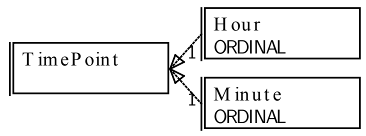
$$\begin{aligned} K_{time\_point} &= (\ C,\ E,\ T\ ) \\ &= (\ \{\ C_{time\_property}\ \}, \\ &\qquad \varnothing, \\ &\qquad \varnothing \\ &\quad) \end{aligned}$$

The name of the collection has been chosen identical with the concept name, which is not in contradiction with the demand on uniqueness among all collection names. The collection currently has no parents, and no type (as apparently the essential information is the hour and minute).

The hour and minute information is defined by the K_{hour} and K_{minute} collections containing the only C_{hour} and C_{minute} concepts, respectively. Given that both of the pieces of information are of a discrete nature, they can be represented by the built-in $T_{ordinal}$ data type (equivalent to $C_{integer}$ concept found in SIL). Finally, the name of the "time point" suggests it contains at most one piece of hour and minute information, leading to both of these being bound with a "1:1" relationship to the C_{time_point} concept:

$$K_{hour} = (\ \{\ C_{hour}\ \},\ \{\ E_{time\_point}\ \},\ T_{ordinal}\ ) \qquad K_{minute} = (\ \{\ C_{minute}\ \},\ \{\ E_{time\_point}\ \},\ T_{ordinal}\ )$$

Finally, the resulting K_{time_point} collection may graphically be depicted as:



□

To further handle DDM contents, let us define several projection functions analogous to their SIL counterparts.

Definition 4.2 (Projection functions)
Let $K = (\ C,\ E,\ T)$ be a DDM collection. Let then $C(K) = C$ be the set of concepts, $E(K) = E$ the set of edges to parent concepts, and $T(K) = T$ the type of information each concept is to hold. In addition, let C be a DDM concept. Let then $K(C) = K$: $C \in C(K)$ be the reference to the collection K that contains the concept C, and $P(C) = \{\,K_i$: $(\ C,\ R_j) \in E(K_i)\,\}$ the set of all subcollections directly accessible from C. Let further be

$$P^*(C) = \begin{cases} P(C) \cup \Big(\bigcup\limits_{\substack{K_i \in P(C) \\ C_j \in C(K_i)}} P^*(C_j) \Big) & \text{, if } P(C) \neq \varnothing \\ \varnothing & \text{, else} \end{cases}$$

$$P^*(K) = \bigcup_{C \in C(K)} P^*(C)$$

a set of all subcollections recurrently available from concept C and collection K, respectively. Finally, each projection function f which is applicable to collection K is applicable to a concept $C \in C(K)$ using $f(C) = f(\ K(C)\ )$. □

Before exemplifying, let us define three key components of the DDM: root, topics, and paths, all of which will be extensively used at different stages of the dialogue processing. The *root* is, presumably, the "starting point" of the model from which any other concept may be addressed. However, at most of the times, the root will be considered a rather logical unit of its own existence instead of a physically present entity. The collection K_{time_point} from Example 4.1 is therefore not to be taken as the root of the DDM (the upcoming Example 4.2 will show it).

A *topic* is supposed to be tightly related with a certain task in a dialogue system (or generally a *set of similar* tasks). Such task may be, for instance, providing information on departure times, in case of a timetable dialogue system. The notion of a topic is therefore to serve as a representative of a given task while grouping its related information. This ambiguous usage has already been proposed in [Rei81]. Apparently, the properties of a topic are best approached by a root-bound collection within the DDM. The collection K_{time_point} from Example 4.1 is therefore a topic (see also upcoming Example 4.2).

A *path* takes on an identical notion like a path in SIL, i.e. its ultimate purpose is to address a particular DDM collection or concept. However, unlike with SIL, there are two distinct and non-interchangeable kinds of paths: one addressing the DDM itself, and another addressing DDM expressions (covered next). The following two definitions formalize all of the presented terms (except for expression paths), i.e. paths, topics, and root.

Definition 4.3 (DDM path)
Let **C** be a set of DDM concepts. A concept C_T is *immediately* targeted from a source concept C_S if there exists exactly one edge E_i such that

$$\Pi(C_S,C_T) = \begin{cases} \{\ [C_S,C_T]\ \} & , \quad \text{if } E_i(C_S,R_i) \in E(C_T) \\ \mathit{error} & , \quad \text{otherwise} \end{cases}$$

A concept C_T is *addressable* from a concept C_S if $K(C_T) \in P^*(C_S)$, i.e. if there exists a set of distinct paths over concepts $C_1, \ldots, C_N$:

$$\Pi(C_S,C_T) = \left\{ \begin{array}{ll} [C_S,C_1,\ldots,C_N,C_T]: & E_1(C_S,R_1) \in E(C_1) \quad \wedge \\ & \forall i \in \{2,\ldots,N\} : E_i(C_{i-1},R_i) \in E(C_i) \quad \wedge \\ & E_{N+1}(C_N,R_{N+1}) \in E(C_T) \end{array} \right\}$$

Let **K** be a set of DDM collections. A collection K_T is addressable from a collection K_S if there exists a set of distinct paths over collections $K_1, \ldots, K_N$:

$$\Pi(K_S,K_T) = \bigcup_{[C_S,C_1,\ldots,C_N,C_T] \in \Pi(C_S,C_T)} [K_S,K(C_1),\ldots,K(C_N),K_T]$$

where $C_S \in C(K_S)$ and $C_T \in C(K_T)$. □

Definition 4.4 (DDM root and DDM topic)
We call the DDM *root* a collection $\rho = K_\rho = (\{\, C_\rho\}, \varnothing, \varnothing)$. We call a DDM *topic* a collection $\tau = K_\tau = (\{\, C_\tau\}, \{\, E_\rho\}, \varnothing)$. □

Example 4.2 (Projection functions, topic, and DDM paths)
Ponder the timetable domain data model in Fig. 4.1. It shows that a given *timetable* schedule consists of *bus* and *train* connections. Each of these connections *departs* and *arrives* at a particular *time point*, modeled by the above C_{time_point}.

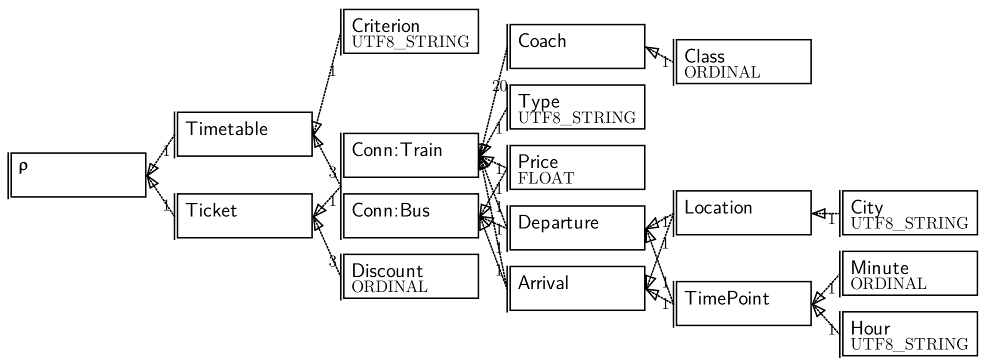


Fig. 4.1 Simplified timetable domain data model.

Notice the cardinality of 3 in $E(C_{timetable}, 3)$. It is to limit the maximum number of connections that are actively dealt with (and possibly presented to the user) in a given task. All other edges are left to their expected "1:1" relationships.

It holds that $C(K_{connection}) = \{C_{bus}, C_{train}\}$, i.e. the set of connections is generally a mixture of buses and trains, i.e. buses and trains are *collectible*. For C_{train} it holds that $P(C_{train}) = \{K_{arrival}, K_{departure}\}$, i.e. both $K_{arrival}$ and $K_{departure}$ are direct subcollections of C_{train}. Furthermore, for $K_{connection}$ it holds that $P^*(K_{connection}) = P^*(C_{bus}) = P^*(C_{train}) = \{K_{arrival}, K_{departure}, K_{time\_point}, K_{hour}, K_{minute}\}$ are subcollections of $K_{connection}$.

The root ρ is explicitly shown in this example. The DDM has a single topic, $\tau = \tau_{timetable} = K_{timetable}$, leading currently merely to information on connections $K_{connection}$. Given the absence of meaning, this topic may be used by multiple tasks focused on providing timetable information (e.g., departure times, arrival times, route planning, etc.). Let us recall that the DDM is necessary to be understood as a purely passive component that only describes how domain objects (real or abstract) relate to each other.

There are two paths from $K_{connection}$ to K_{time_point}: $\prod(K_{connection}, K_{time\_point}) = \{[K_{connection}, K_{arrival}, K_{time\_point}], [K_{connection}, K_{departure}, K_{time\_point}]\}$. Furthermore, it holds that $|\prod(K_{timetable}, K_{hour})| = 2 = |\{[K_{timetable}, K_{connection}, K_{arrival}/K_{departure}, K_{time\_point}, K_{hour}]\}|$, whereas $|\prod(C_{timetable}, C_{hour})| = 4 = |\{[C_{timetable}, C_{bus}/C_{train}, C_{arrival}/C_{departure}, C_{time\_point}]\}|$. □

Finally, to facilitate further use of the DDM, let us make several assumptions about its structure, guaranteeing the resulting DDM to be *correct*.

Definition 4.5 (Correct DDM)
Let **K** be a set of DDM collections, and **C** be a set of DDM concepts. We say the DDM consisting of **K** and **C** is *correct* if all of the below conditions are met:

- All concepts within a collection K Î K are either leaves or "intermediate":

$$\forall C_i \in C(K): \begin{cases} P(Ci) \neq \varnothing, & \text{if } P(K) \neq \varnothing \text{, i.e. } K \text{ is intermediate,} \\ P(Ci) = \varnothing, & \text{if } P(K) = \varnothing \text{, i.e. } K \text{ is leaf.} \end{cases}$$

- Each eigen information (see Definition 3.9) is stored exclusively in leaf collections, i.e. it holds

$$\forall K_i \in \mathbf{K}\colon\ T(K_i) \neq \varnothing \quad \rightarrow \quad P(K_i) = \varnothing\ .$$

- Each two path alternatives between distinct collections K_i and K_j are mutually non-unifiable, i.e. no alternative emerged from another by omitting one or more of its "intermediate" collections.

- The resulting structure is an acyclic graph. □

It can be easily prooved that the DDMs from Example 4.1 and Example 4.2 conform to the definition of correctness.

4.3 DDM Expressions

A correct DDM can be instantiated as the so called DDM expressions (compare with principles in SIL). Given the new informational axis (collections), DDM expressions need to be covered from both concepts and collections point of view. Provided that no concept features any taxonomy, DDM expressions are defined as not necessarily having to follow the exact structure of their underlying model.

Definition 4.6 (DDM expression)
We call a DDM expression an instance $I = I(C)$ of a DDM concept C, or an instance $Y = Y(K)$ of a DDM collection K. An instance of a concept C is an ordered triple $I = (D, C, X)$, where D is a unique identifier and X is a value of type $T(C)$. An instance of a collection K is an ordered triple $Y = (K, I, F)$, where $I = \{I_i\}$ is a non-empty set of instances of concepts $C_i \in C(K)$, and $F = \{F_i\}$ is a set of edge instances $F_i = (E_i, I_i) = ((C_i, R_i), I_i)$ each of which leads to a recurrently accessible parent concept instance I_i, i.e. $F = \{F_i\colon K \in P^*(C_i)\}$. Furthermore, the following constraints apply to each DDM expression:

- The number of concept instances within a collection satisfies the cardinality of each binding edge, i.e.

$$\forall Y = (K, I, F)\colon |I| \leq \min(R_j\colon ((C_j, R_j), I_j) \in F)\ .$$

- Concept instances are not to be shared among collections, i.e.

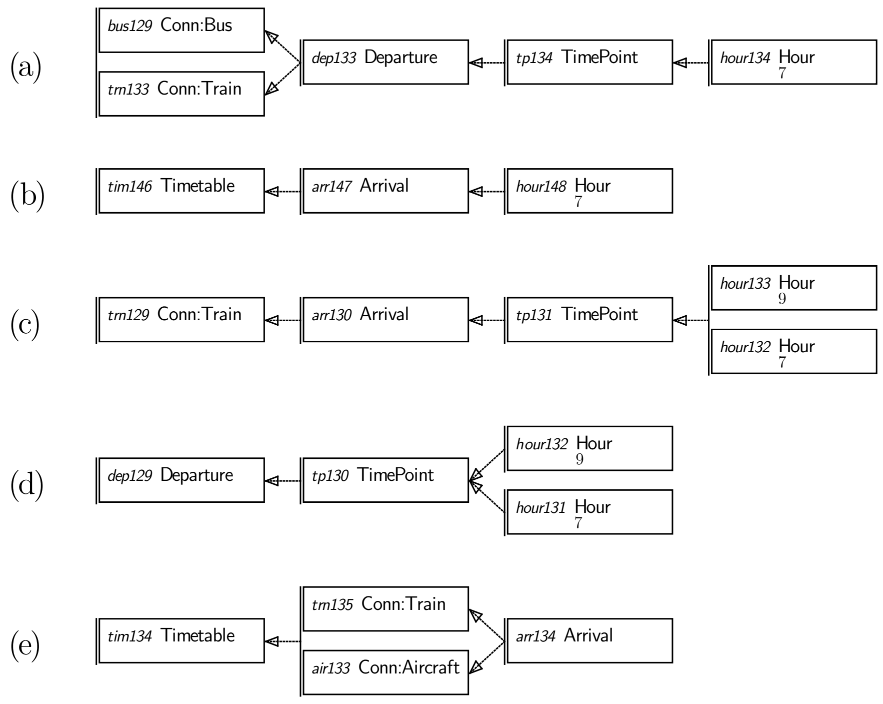
$$\forall Y_i = (K, I_i, F_i),\ Y_j = (K, I_j, F_j),\ Y_i \neq Y_j\colon\ I_i \cap I_j = \varnothing\ .$$

- Each concept instance has at most one subcollection of a given type K,

$$\forall Y_i = (K, I_i, F_i),\ Y_j = (K, I_j, F_j),\ Y_i \neq Y_j\colon F_i \cap F_j = \varnothing\ .$$ □

Example 4.3 (DDM expressions)
Naturally, each instance, along with all its parametrization, must entirely conform to its underlying definition. Of the following DDM expressions, only (a) and (b) are *correct* and comply with the data model in Fig. 4.1. In the incorrect expression (c), the collection Y_{hour} violates the prescribed number of concept

instances dictated by the underlying model; in expression (d), I_{time_point} breaches the rule of containing at most one K_{hour} by containing two instances instead; finally, as for expression (e), *air133* is of unknown type as no $C_{aircraft}$ concept exists in the underlying model.



□

Definition 4.7 (DDM expression projection functions)
Let $Y = (K, I, F)$ be a DDM collection instance. Let then $K(Y) = K$ be the underlying DDM collection, $I(Y) = I$ the set of concepts, $F(Y) = F$ the set of edge instance. Each function f applicable to K is applicable to Y as $f(Y) = f(\, K(Y)\,)$. In addition, let $I = (\, D, C, X)$ be a DDM concept instance. Let then $D(I) = D$ be the concept instance identifier, $C(I) = C$ the underlying DDM concept, and $X(I) = X$ the data type particular value instantiated using I. Also, let $Y(I) = Y\colon I \in I(Y)$ be the reference to the collection Y that contains the concept instance I, and $P(I) = \{\, Y_i\colon (\, E_j, I) \in F(Y_i) \}$ the set of all subcollection instances directly accessible from I. Let further be

$$P^*(I) = \begin{cases} P(I) \cup (\bigcup\limits_{\substack{Y_i \in P(I) \\ I_j \in I(Y_i)}} P^*(I_j)) & \text{, if } P(I) \neq \varnothing \\ \varnothing & \text{, else} \end{cases}$$

$$P^*(Y) = \bigcup_{I \in I(Y)} P^*(I)$$

a set of all subcollections recurrently available from concept instance I and collection instance Y, respectively. Finally, each function f applicable to C is applicable to I as $f(I) = f(\, C(I)\,)$, and each function applicable to a collection Y is applicable to $I \in I(Y)$ using $f(I) = f(\, Y(I)\,)$. □

All of the above projection functions are analogous to those already presented for the data model (Section 4.2) – they will be therefore left without demonstration, similarly as DDM expression paths, defined next. Presumably, expression paths allow us to comfortably address particular instances.

Definition 4.8 (DDM expression path)
A concept instance I_T is *immediately* targeted from a source concept instance I_S if there exists exactly one edge instance F_i such that

$$\Pi(I_S, I_T) \quad = \quad \begin{cases} \{\ [I_S, I_T]\ \} & , \quad \text{if } F_i(E_i, I_S) \in F(I_T) \\ error & , \quad \text{otherwise} \end{cases}$$

A concept instance I_T is *addressable* from a concept instance I_S if $Y(I_T) \in P^*(I_S)$, i.e. if there exists a set of distinct paths over concept instances $I_1, \ldots, I_N$:

$$\Pi(I_S, I_T) \quad = \quad \left\{ \begin{array}{ll} [I_S, I_1, \ldots, I_N, I_T]: & F_1(E_1, I_S) \in F(I_1) \quad \wedge \\ & \forall i \in \{2, \ldots, N\} : F_i(E_i, I_{i-1}) \in F(I_i) \quad \wedge \\ & F_{N+1}(E_{N+1}, I_N) \in F(I_T) \end{array} \right\}$$

A collection instance Y_T is addressable from a source collection instance Y_S if there exists a set of distinct paths over collection instance $Y_1, \ldots, Y_N$:

$$\Pi(Y_S, Y_T) \quad = \quad \bigcup_{[I_S, I_1, \ldots, I_N, I_T] \in \Pi(I_S, I_T)} [Y_S, Y(I_1), \ldots, Y(I_N), Y_T]$$

where $I_S \in I(Y_S)$ and $I_T \in I(Y_T)$. □

4.4 Semantics Representation

The input semantics representation has been designed to easily allow for all further processing stages, described and discussed in the upcoming sections (context representation, user's intentions detection, deliberation, etc.). As already partly revealed earlier, the dialogue context makes an extensive use of DDM expressions. It is therefore reasonable that *each input semantics is represented as a sequence of DDM expressions.*

With processing-related benefits on one side, this approach introduces drawbacks in representing some linguistical constructs, that, on the other hand, may easily be modeled using the SIL formalism (see Fig. 3.1). Given that the domain data model cannot account for meaning of contained objects, it is impossible, for instance, to represent the mere sentence "*Yes*". A more complex situation arises when attempting to represent a general content. For instance, recall the sentence "*the train, which arrives in Erlangen at five, departs from Schwandorf*" where a known train is referred (by its arrival) and further described (by its departure), resulting in a co-referential SIL expression discussed in Section 2.2.1. Last but not least, the SIL formalism has also the potential of representing (although not directly detecting) implicit intentions in user's *statements* (e.g., "*I want to know the nearest train to Erlangen*" instead of "*When does the nearest train to Erlangen go?*") that are usually pre-processed yet before the dialogue manager

Table 4.1 Application-neutral directives to modify the meaning of enclosed DDM expressions.

Directive	Description
declarative *interrogative* *imperative*	utterance dialogue act classification
agree *disagree*	confirmation or basic mental state
reference	explicit (parser-detectable) cross-reference
timeout *help* *repeat*	local dialogue control acts

module by detecting the corresponding "shape" of the sentence and transforming it to a manager-aware "question-like" token [Eck95, Boh09, Ngu06b].

The above plotted limitations are of a crucial importance. Solving them is further constrained by having to always conform with the given DDM in a *global manner* (unlike with SIL whose subconcept roles are of a *local nature*). Therefore, the implication for the simple sentence "*Yes*" is that it cannot be represented as a concept instance, unless C_{yes} has been involved into the DDM with a domain-wide agreed meaning of representing an agreement. To work around this constraint, the semantics representation includes the ability to decorate arbitrary portion of it with one or more pre-defined *directives*. Each directive is to supply a simple (elemental) meaning to the content it encapsulates, preventing resulting expression from violating the underlying DDM. Table 4.1 gives an overview of currently available directives, and the following example shows their usage.

Example 4.4 (Semantics directives usage)
In its raw form, semantics is to be written using a "Prolog-style" notation, see Appendix A.1 for syntax grammar. We will stick to this notation throughout the text as it is sufficiently comprehensive; we also will enclose directives between underscores to visually distinguish them from regular concepts:

(a) *_agree_*
Example sentences: "*Yes.*", "*I want.*", "*I agree.*"

(b) *_imperative_*(*Arrival*(*Location*(*City*:"Ostrava")))
"*I said to Ostrava!*"

(c) *_disagree_*(*Departure*(*City*:"Praha") , *_agree_*(*Arrival*(*Location*)))
*_disagree_*(*Departure*(*City*:"Praha")) , *_agree_*(*Arrival*(*Location*)) ♣
*_agree_*(*Arrival*(*Location*) , *_disagree_*(*Departure*(*City*:"Praha")))
"*I didn't say from Praha but to there.*"

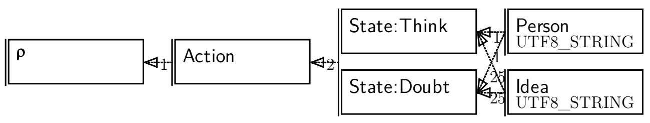
(d) *_help_*
"*Help.*", "*I'm lost.*" □

In the above example, the middle semantics in (c) (marked with ♣) may be in some sense taken as a concatenation of *_disagree_* and *_agree_* UFO-like units. Generally, DDM expressions are not intended to imitate the notion of UFOs as they are a weaker tool. The reason implies from the constraints put on the underlying model. As already discussed, the model cannot account for any object meaning as objects receive their meaning within a particular plan. For a semantics represented using DDM expressions, it is therefore difficult to capture, for instance, speaker's opinion or mental state – recall the workaround of using directives to express the opinion of *disagreeing* with departure from Praha. From the framework point of view, all of the above built-in directives share a common aspect: their meaning can be *immediately* reflected by accordingly updating the dialogue context. For instance, the above mentioned disagreement in (c) immediately results in Praha ceasing to be the location of departure. However, should a "permanent" meaning be represented, the local character of directives would not suffice. The following example attempts for a "permanent" meaning along with elaborating a "proxy" solution.

Example 4.5 (Attempting for semantics and taxonomy using DDM)
Let us try to model the situation that a *person* may *think* or *doubt* about an *idea* while performing an *action* (compare with Fig. 3.1):

(a) *I think of the idea.*
(b) *I doubt of the idea.*
(c) *I think that I doubt of the idea.*
(d) *I doubt that I think of the idea.*

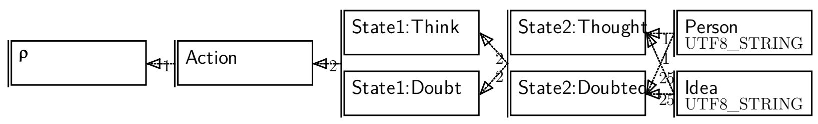
The essential problem here is the lack of object taxonomy. While it is reasonable to represent the *action* as a topic in the model, it is difficult to figure out the right order of nested subconcepts to produce a sensible DDM structure. The first approach we may try out is to define that *thinking* and *doubting* are mental states requiring exactly one agent (*person*) and at least one object (*idea*):



Apparently, C_{think} and C_{doubt} attempt to imitate SIL defining a domain-wide agreed meaning and taxonomy. Our model is correct as long as C_{think} and C_{doubt} cannot act as subobjects in each other. While this holds true for sentences (a) and (b), it is not met with sentences (c) and (d):

(a) *Think*(*Person* , *Idea*)
(b) *Doubt*(*Person* , *Idea*)
(c) *Think*(*Person* , *Doubt*(*Person* , *Idea*))
(d) *Doubt*(*Person* , *Think*(*Person* , *Idea*))

Neither of expressions (c) and (d) complies with our model. Unfortunately, we cannot redefine it correspondingly, as objects must not be recurrently nested (see Definition 4.5). Nonetheless, if we constrain ourselves to these two sentences, we find that a *person* may *think* or *doubt*, and that an *idea* may be *thought* or *doubted*. Furthermore, if we accept that *thinking* and *doubting* are person's opinions, we can remodel our situation as follows:



To avoid the recurrency, each C_{idea} is now to be "marked" (or "prefixed") with a $C_{thought}$ or $C_{doubted}$ concepts. In addition, notice edges with cardinality of two. For instance, $E(C_{action}, 2) \in I(K_{state1})$ guarantees that a person can think and doubt of different ideas at the same time as the model allows both C_{think} and C_{doubt} be instantiated at the same time. Hence, according to this model, our four motivational sentences can be expressed as follows:

(a) *Person*(*Think*(*Thought*(*Idea*)))
(b) *Person*(*Doubt*(*Thought*(*Idea*)))
(c) *Person*(*Think*(*Doubted*(*Idea*)))
(d) *Person*(*Doubt*(*Thought*(*Idea*)))

Obviously, this approach puts additional requirements on the semantic parser (or semantics post-processing): if the *person* has not explicitly announced *doubting*, any supplied *idea* is to be taken implicitly *thought*.

To conclude our investigation, let us add that this approach also allows for retracting person's opinion by simply "re-binding" an idea (along with its $C_{thought}$ or $C_{doubted}$ prefix) to the competitive opinion:

I no longer think I doubt the idea. ≡ *I think I no longer doubt the idea.*
Person(*Think*(*_disagree_*(*Doubted*(*_agree_*(*Idea*))) , *_agree_*(*Thought*)))

Finally, despite its feasibility, the resulting model can reasonably be considered cumbersome. However, it is only *one of possible solutions* to capture meaning using DDM, and there are more sophisticated solutions that, however, exceed our current initial investigation. □

As the example has shown, the absence of meaning may require different approaches to modeling with SIL and DDM. A similar situation is encountered when taking into account co-referential (and in DDM case also cross-referential) expressions: "*the person, that thinks the idea, goes to Praha*". The absence of meaning has already been discussed to have a limiting character to the underlying model, hence it will not be revisited here and such referential phrases will not be considered. We instead constrain ourselves to cases in which such semantical entities do not appear, as, for instance, in "*the train, which arrives in Erlangen at five, departs from Schwandorf*", already roughly analyzed in Section 2.2.1 (and closer elaborated in [Eck95]).

Similarly as with SIL, referential DDM expressions rely on proper structure of objects describing a reference. Due to the absence of taxonomy, the transformation of user's reference into a DDM expression is a "one-way" operation – the result cannot be approximated back to user's original utterance as with SIL (Section

3.4). Instead, related instances are bound in a "specified-specifying" relationship, i.e. each concept instance narrows the characteristics of its parent. Therefore, the DDM expression that describes "*the train, which arrives in Erlangen at five, departs from Schwandorf*" takes on the following form:

*_ref_*(*Train*(*Arrival*(*City*:"Erlangen") , *Departure*(*City*:"Schwandorf")))

Note that the absence of $C_{location}$ is not a mistake here (recall that DDM expressions do not have to strictly follow the underlying model). Nevertheless, if this reference is to be resolved, there must exist a train in the dialogue context for which it holds that it both arrives in Erlangen and departs from Schwandorf. Also notice the prefixing *_ref_* directive, here to denote an *explicit reference*. This directive is optional and can be omitted in most of the cases.[1] This directive is necessary, however, to express references using unexpressed nominative, as in:

S The ticket costs 10 coins. Do you want to purchase it?
U Yes, I buy it.
Semantics: *_agree_* , *Purchase*(*_ref_*)

Last but not least, references can be nested. For the "most outer" reference it holds that its successful resolution is conditioned by resolving first all its nested subreferences, as for instance in:

S The ticket costs 10 coins. Do you want to purchase it?
U No. How much is a ticket for the train with the first class coach?
Semantics: *_disagree_* ,
Ticket(*Price*:— ,
*_ref_*(*Train*(*_ref_*(*Coach*(*Class*:"1"))))
)

Finally, references can be performed in different genders (*ten*, *ta*, *to* in Czech, or *der*, *die*, *das* in German) to precise their resolution. This feature has not been included in Definition 4.1 in Section 4.2 and thus will be not discussed and used further (however, see Appendix A.1 for syntax of gender-specific references).

The last topic to cover is intention description using DDM expressions. Naturally, with respect to the absence of meaning, their descriptive capabilities are limited to only question-like *explicit intentions* [Gro86, Coh95]. However, similarly as with SIL, DDM expressions may convey mere fragmented intentions that resulted from either poor recognition or incompletely formulated requests. The word "potential" has been used intentionally here. Eckert in his work [Eck95] uses a very flat approach to convey and further deal with intentions, or more precisely, task identification: is there an empty eigen information ψ_i encountered in the input semantics, then ψ_i is considered the information user has asked about (e.g., departure time). As Eckert argues, this approach suffices: if the user has called a timetable information system, then it is quite certain that she or he will want to query about one of the timetable-related services [Eck95].

Generally, semantics may be understood as a mixture of task and task-related information, as in the following situation:

1 In fact, it is merely intended to take a precise control over the reference resolution mechanism.

S There is a train at 14 o'clock and a bus at 15 o'clock going to Praha.
U When does the train arrive there?
Semantics: *_ref_*(*Train*(*Arrival*(*Time*:—)))

In the dialogue snippet, the user wants to know the arrival time of the particular transportation means. There is a similarity with Eckert's approach as for instantiating the C_{time} concept with an empty value to indicate information to find. Note, however, that such case is only one of clues to estimate user's intentions (more on their detection in Section 4.7). With respect to the performed reference, this sample situation already becomes relatively complex to resolve.

4.5 Information Management

The dialogue snippets in the previous section have shown quite a broad variety of possible ways to convey information: ellipses, intentions, references, corrections, etc. For us to deal with them, it is necessary to first know how information is *internally* organized in the framework. No earlier than after this has been explained we will be able to proceed with particular processing of ellipses "anchoring", intentions detection, and references resolving. This section will therefore concern with pure *information management*. At this point, we will assume input information is fully specified (non-elliptical), does not contribute to intentional shift, and contains no references. However, it may contain user's corrections, covered later in this section.

The information management is a procedure too complex to cover in a single section or conceive in a single definition. It therefore will be presented in an iterative manner, with each iteration revising its predecessor to pinpoint problems and present solutions. This section is organized so that it first infers the initial approach, and afterwards refines it several times by incorporating additional "requirements on functionality". At the end of this section, we will have a working model of information management to subsequently represent the dialogue context within the next section.

4.5.1 Initial Approach

The initial approach to the information management is best compared with creating a SIL *view* (see Section 3.3.2). Recall that the view is a "projection" of dialogue history using operations of unification and hiding to concepts instances that are compatible and concurrent, respectively (see Fig. 3.7 and Fig. 3.9). Hence, information that was originally scattered across different interpretation worlds, is now put together to produce a compact information representation.

As Eckert points out, the information representation is far from being a trivial task, given that the semantics is strongly oriented towards the linguistical structure of utterances [Eck95]. Therefore, presuming there are no further operations beyond the unification and hiding, there probably have to be additional rules (built-in or domain-specific) to overcome the incompatibility of otherwise compatible objects, caused by one being nested in a linguistical concept while the other not, like in the following illustrative snippet:

S Where do you want to go to?
U I want to Erlangen.
S When do you want to go to Erlangen?
U Eight o'clock.

$$\begin{bmatrix} id : want116 \ , \quad type : want \\ value : \begin{bmatrix} id : go116 \ , \quad type : go \\ thegoal : \begin{bmatrix} \boxed{\text{Erlangen}} \end{bmatrix} \end{bmatrix} \end{bmatrix} \qquad \begin{bmatrix} id : go61 \ , \quad type : go \\ thesourcetime : \begin{bmatrix} \boxed{800} \end{bmatrix} \end{bmatrix}$$

Intuitively, there is a complication with unifying the *go116* and *go61* concepts due to one being part of a C_{want} concept instance. Unfortunately, such singular cases have not been addressed in Eckert's work and it is therefore difficult to estimate the behaviour of a SIL-based dialogue manager.

With DDM expressions, the above plotted situation cannot occur as they must comply with an arbitrary complex but strongly structured correct model. As seen earlier, any user's meaning is to be supplied using directives or explicit concepts. This fact significantly simplifies otherwise complex operations (e.g., corrections), discussed later.

The most notable distinction between the SIL and our information management is the way information itself is stored. While in the former case, information is kept scattered across the dialogue history, in the latter case information is maintained in an "already projected state", comprising the so called *information pool*. This factual distinction also changes the set of operations to manipulate concept instances to inferrention and disposal. Exemplified shortly, given an instantiated concept in the pool, the goal of the *inferrention* is to derive a new concept of the same type and update its state in accordance with the input semantics. The goal of the *disposal* is to determine if the original concept is further necessary to be held in the pool, and if not, remove it (either by marking it as historical[2] or disposing it permanently). However, before putting these operations into a perspective, let us formally define the term information pool.

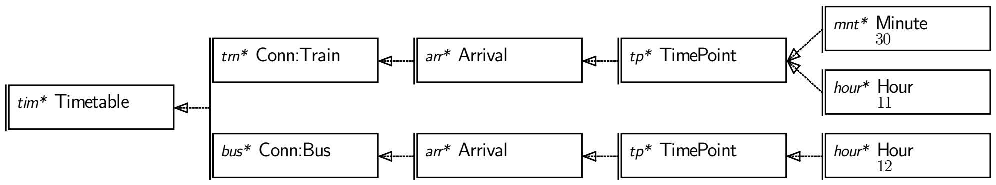
Definition 4.9 (Information pool)
An information pool $\mathbf{Y}$ is a set of DDM expressions for which it holds:

$$\forall Y \in \mathbf{Y},\ \forall Y_i \in P^*(Y):\ Y_i \in \mathbf{Y}\ . \qquad \square$$

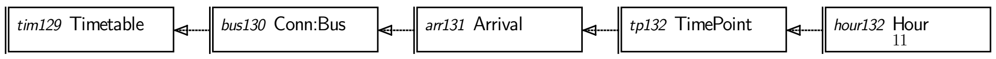
Thus, the information pool is a compact entity: if a collection instance is part of it, then all its subcollections must be as well. The root instance of the information pool, $I(\rho)$, then navigates to the open topics currently in focus.

Example 4.6 (Motivational)
To begin with, ponder the below shown information pool. It may be considered a single expression with a single open topic $\tau_{timetable}$ (multi-expression and multi-topic information pools to follow). Its structure represents a train and a bus which arrive at 11.30 and 12 o'clock, respectively. Particular identifiers are unimportant at this moment.

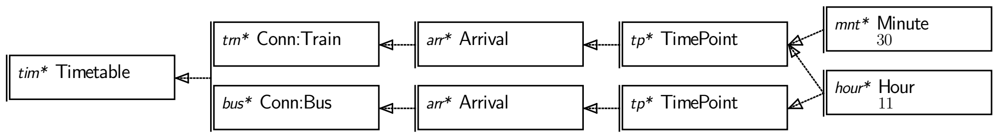
2 A collection instance (i.e. not merely a separate concept) may become historical if it is "sealed", meaning there has left nothing to discuss about it and it is recurrently confirmed.



For demonstrative purposes, consider the bus arrival time has to be changed to 11 o'clock (say, to reflect user's new wish). This can be simply done by "updating" merely its hour information. The following semantics carries such update:



Presumably, the expected result is as follows (IDs are again unimportant):



□

Given the above motivation, the initial approach may be simply put as follows:

- Start from the respective roots of the information pool and semantics.
- For each immediate subconcept of the semantics root, find an equivalent subconcepts in the information pool root. Infer a new subconcept.
- Take the new subconcept and recurrently process the rest of the semantics.
- Once backtracking, attempt to dispose the original subconcept.

This recurrent algorithm is closer elaborated in Fig. 4.2. Its particular steps that have transformed the initial information pool from Example 4.6 to its final state are then caught in Fig. 4.3 (see the figure legend for more information).

Thus, the initial approach can handle whatever a semantics that carries a fully specified information and "incorporates" it into the information pool using the inferential and disposal operations, applied recurrently. Once done, an arbitrary sequence of initially isolated pieces of information (isolated input semantics) is stored in a "flat" manner to avoid the SIL-like projection. The algorithm can also deal with collections of information in such a way that inferred collections replace their original counterparts or eventually absorb their concepts (Lines 12–15 in Fig. 4.2). Finally, the algorithm on its own is naturally of a very limited usability. The upcoming sections will therefore aim to improve it by accommodating features to deal with regular cooperative dialogue *requirements*.

4.5.2 Requirement 1: Dialogue Is a Shared Space

This requirement aims to equip the user and the system with the same possibility of influencing the dialogue information state. The fact that we are dealing here

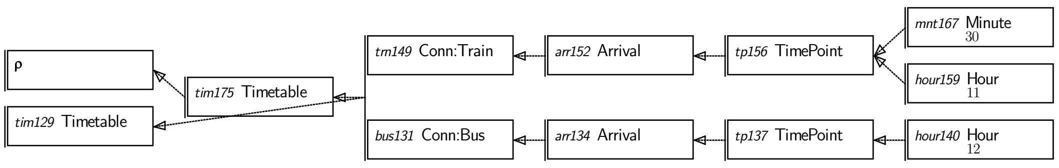
```
    procedure $Incorporate( Y(K_S) = Y_S \in Semantics , I(C_{parent}) = I_{parent} \in \mathbf{Y} )$ {
        // instantiate a new "empty" collection $Y$ (not yet a DDM expression) of the type $K_S$
1       $Y := ( K_S, \varnothing, \{( E_{parent}, I_{parent})\} )$
        // find in and remove from $I_{parent}$ the original collection instance $Y_{orig}$
2       $Y_{orig} := ( K_S, I_{orig}, F_{orig})$: $\exists F_{parent} = ( E_{parent}, I_{parent}) \in F_{orig}$
3       if $Y_{orig} \neq \varnothing$ {
4           $Y_{orig} := Y_{orig} \setminus F_{parent}$
        }
        // re-instantiate $I(Y_S)$ in $Y$ (raising $Y$ to a DDM expression)
5       $\forall I_S = ( D_S, C_S, X_S) \in I(Y_S)$ {
            // re-instantiate $I_S$ as $I$
6           $I := I(C_S)$
7           $I(Y) := I(Y) \cup I$
            // inherit all subcollections from an original $I_0 = ( D_S, C_S, X_S) \in I_{orig}$ (if any)
8           $\forall Y_{sub} = ( K_{sub}, I_{sub}, F_{sub}) \in P(I_0)$ {
9               $F_{sub} := F_{sub} \cup ( E, I)$, where $E = ( C_S, R) \in K_{sub}$
            }
            // recurrently process
10          $\forall Y_{sub} = ( K_{sub}, I_{sub}, F_{sub}) \in P(I_S)$ {
11              $Incorporate( Y_{sub}, I)$
            }
        }
        // absorb uncontained concept instances from $Y_{orig}$ into $Y$
12      if $I(Y) \subseteq I_{orig}$ {          // no new concept introduced by $Y_S$
13          $I(Y) := I(Y) \cup \{ I = ( D, C, X): I \in I_{orig} \wedge I \notin I(Y) \}$
        }
        // attempt to dispose $Y_{orig}$
14      if $Y_{orig}$ no longer referred {
15          $\mathbf{Y} := \mathbf{Y} \setminus Y_{orig}$
        }
        // add $Y$ into $\mathbf{Y}$
16      $\mathbf{Y} := \mathbf{Y} \cup Y$
    }

Usage: $Incorporate( Y(\rho) \in Semantics , I(\rho) \in \mathbf{Y} )$
```

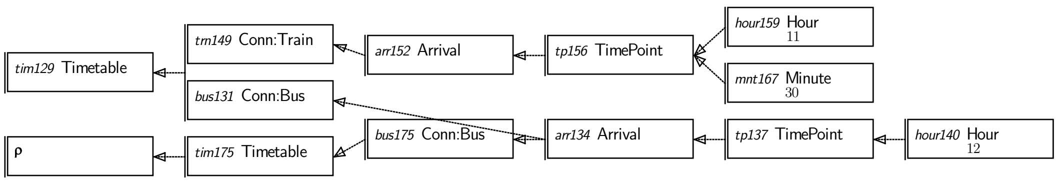
Fig. 4.2 Information management initial approach algorithm.

with a collaborative dialogue facilitates our situation. The proposed information pool usage cases are follows:

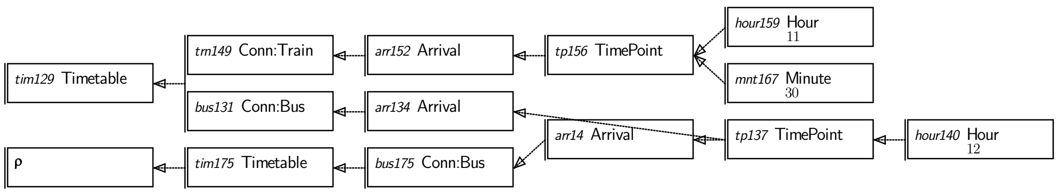
1. *User describes brand new information about the world* (e.g., wants to find a train that goes to Praha at around 8 o'clock). The term "brand new" is significant here as references to "already known" information will be discussed later. Recall that similar pieces of information may be collected. To eliminate ambiguity in upcoming processing, user's collections may only consist of at most one instance of each concept type. For example, the user may evolve information regarding only one train and/or only one bus (see the DDM in Fig. 4.1). Naturally, the user is given the possibility to evolve only a selected concept, leaving the rest of the collection untouched, as seen in Example 4.6.



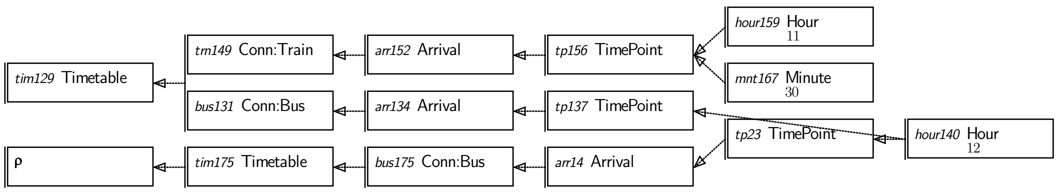
(a) derrived new timetable instance *tim175*, sharing subcollections with the original, *tim129*.



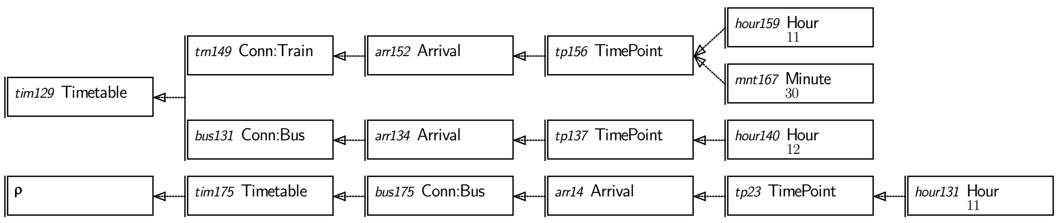
(b) derrived new bus instance *bus175*, sharing subcollections with the original, *tim131*.



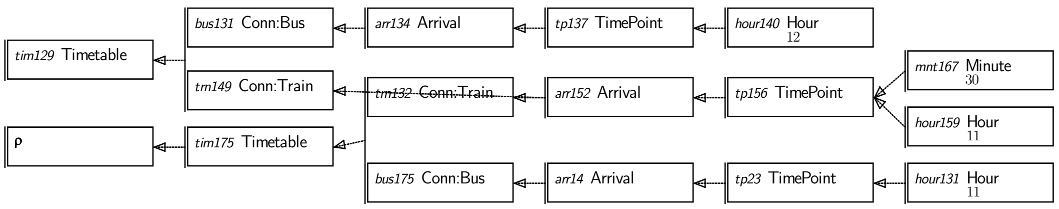
(c) derrived new arrival instance *arr14*, sharing subcollections with the original, *arr134*.



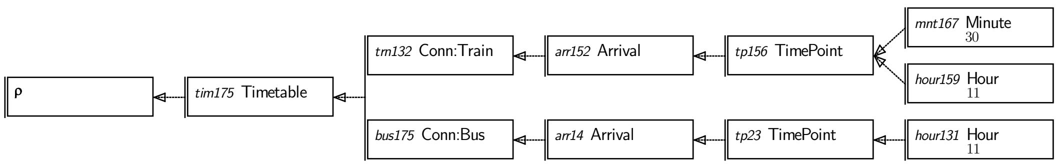
(d) derrived new time point instance *tp23*, sharing subcollections with the original, *tp137*.



(e) created new hour instance *hour131*; initiation of back tracking.



(f) added new train instance *trn132*; original collection {*bus131*, *trn149*} still referred from *tim129*.



(g) original timetable *tim129* not referred from anywhere – disposed along with all its subcontent.

Fig. 4.3 Information management initial approach demonstration.

2. *System supplies new information about the world that may override user's one* (e.g., finds that there is a matching train at quarter past eight). Such information is to *override* user's eventual one. Nevertheless, the overriding must be temporal to meet the notion of the information pool serving as a shared space. However, unlike with user's collections, system ones may contain multiple instances of the same concept (supported by the fact that *the system is never wrong*) which cannot be further evolved (supported by the fact that *the system is always accurate*).

3. *User re-specifies his demands, eventually rejects system current information* (e.g., yet wants a seat in the first class coach). Under such conditions, user's imagination of the world must override the system one (compare with processing a sequence of utterances in SIL).

To allow for the two cases of information overriding, it is necessary to introduce a function that determines *who* of the participants has provided the information. We term this the *information content type function.* The overriding itself is then implemented as *user information hiding.* The following definition clarifies these two functions.

Definition 4.10 (Information content type, and user information hiding functions) Let $Y = Y(K) \in \mathbf{Y}$ be a DDM collection instance. The *information content type function* is a projection A: $\mathbf{Y} \to \{ user, system, dereferenced \}$ with the following characteristics:

$$A(Y) = \begin{cases} user & , \quad \text{if } Y \text{ has been provided by the user} \\ system & , \quad \text{if } Y \text{ has been provided by the system} \\ dereferenced & , \quad \text{if } Y \text{ has been used to resolve user's reference} \end{cases}$$

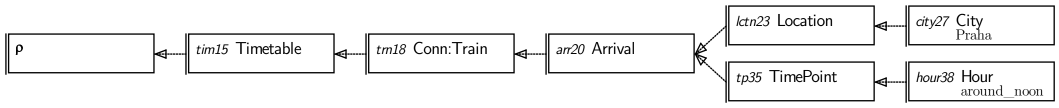
Let $\tau \in \mathbf{Y}$ be a topic instance. Let it hold $\Pi(\tau, Y_1, \ldots, Y_N, K(Y)) = \{ [\tau, Y_1, \ldots, Y_N, Y], [\tau, Y_1, \ldots, Y_N, Y_U] \}$, $Y \neq Y_U$. Then the *user information hiding* function is a projection U: $\mathbf{Y} \to \mathbf{Y}$ such that

$$U(Y) = \begin{cases} Y & , \quad \text{if } A(Y) = user \\ Y_U & , \quad \text{otherwise} \end{cases}$$

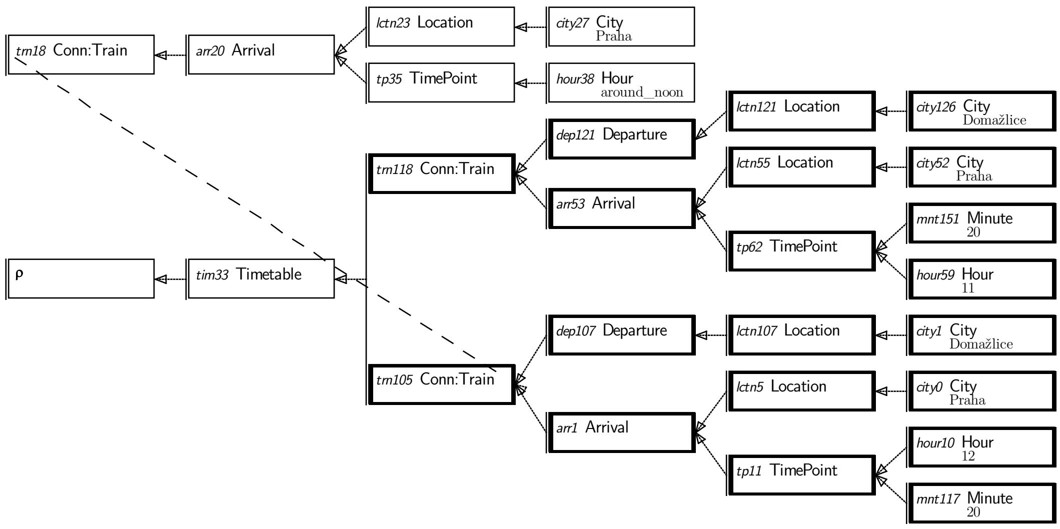
□

The user information hiding function is to "mine" user-provided information from a particular "address" (path) in the information pool. Apparently, if the information Y addressed by a given path has been provided by the user, then we are done and return Y. Otherwise, Y has been introduced by the system and eventually hides user-provided information, Y_U. As we will see in the revised algorithm, system-provided information is removed from the information pool once the system has been overridden (an operation outside the $U(.)$ function), hence there is no implied ambiguity. Fig. 4.4 illustrates the idea (see the figure legend for more information).

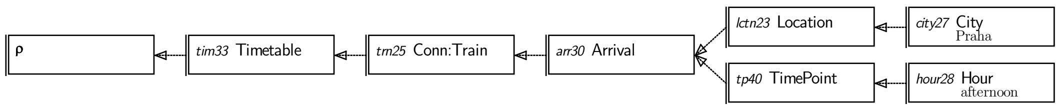
However, before presenting the updated information management approach (Appendix A.2.1), let us introduce one more term from Grosz and Sidner's work



(a) user has introduced brand new information into a blank information pool



(b) system has introduced new information (thick frames); this information overrides user's original one (system "backs up" the user; dashed line)



(c) user has made changes to its imagination of the world, disposing system content, if any

Fig. 4.4 Shared space extension demonstration.

on discourse attention [Gro86], the salience. The *salience* is a meta-information on how "recent" a given object is. Let us represent the salience as a number with the following semantic: the *higher* the number the *older* the object, i.e. the *less* salient it is, and vice versa.

Definition 4.11 (Salience and related projection functions)
Let $I = I(C)$ be a DDM concept instance. The salience is a projection S: $\mathbf{Y} \rightarrow \Re^+$ such that $S(I) < S(I_i)$, if I has been uttered later than $I_i = I(C_i)$, and $S(I) = 0$, if I has not been uttered yet. In addition, let $\mathbf{M} = \{I_i\}$ be a set of concept instances. Then $S(\mathbf{M}) = \{\, S(I_i) \,\}$ (set of corresponding saliences). Finally, let $\mathrm{S}(\varnothing) = 0$. □

The new parts in the approach account for: 1) information overriding (Lines 5–12), 2) subinformation spreading (Lines 16–21), and 3) meta-information setting (Line 23). While the first and last ones are self-explanatory, the second one desires a short explanation. The subinformation spreading is motivated by the fact that users are unlikely to repeat one piece of information for each sibling concept, like in "*I am searching for a bus to Praha or a train to Praha*". They rather use an ellipsis to intuitively specify both siblings, as in "*I am searching for a bus or train to Praha*", semantically represented as

Timetable(*Bus*, *Train*(*Arrival*(*Location*(*City*:"Praha")))) .

The C_{bus} instance is left without any closer specification. To allow for the common-sense understanding of the arrival in Praha relating to both the train and bus, the $C_{arrival}$ subinformation must be spread from C_{train} to C_{bus}. Presuming an empty information pool, Rule 4 fires and carries out such update, drawing the semantics to fhe following state,[3]

Timetable(*Bus*(*Arrival*(*Location*(*City*:"Praha"))),
Train(*Arrival*(*Location*(*City*:"Praha")))) .

4.5.3 Requirement 2: Information Error Recovery Approach (Corrections)

Dialogue is an error-prone environment, causing inconsistencies between the user and system dialogue models. It is reasonable to claim that the earlier such inconsistency is revealed the easier it is to recover from it. Naturally, the best way to avoid inconsistencies is to confirm each incoming information immediately; nevertheless, this leads to boring interactions [McT02]. Alternatively, a reliable user model may be employed to predict obstacles in conversation, hence again avoid inconsistencies [Hja05].

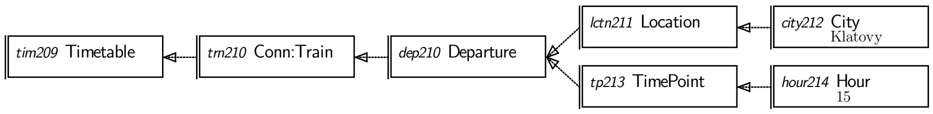
The Daisy framework currently contains only limited possibilities to predict errors. The source of predictions is rather a blind process constraining merely to a selection of one of dialogue strategies. In Section 4.9, we will see different strategies to switch between over the course of a dialogue in order to improve the dialogue flow *after errors have already been observed*. With the proper strategy selected, the system confirmation behaviour can be adjusted.

Taking this limitation into account, the Daisy framework has been equipped with a strong error recovery approach at the level of information management. The approach assumes the worst case: a late discovery of an error and multiple corrections in a single utterance. To begin our investigation, ponder the below dialogue snippet. As it can be seen, there were two errors in the dialogue. Although they both appeared at different times, the framework managed to recover from them to a consistent state.

3 The information spreading may be perceived analogous to the SIL semantics elaboration. However, unlike with SIL, this process is hard-coded as it carries out a single-purpose operation only.

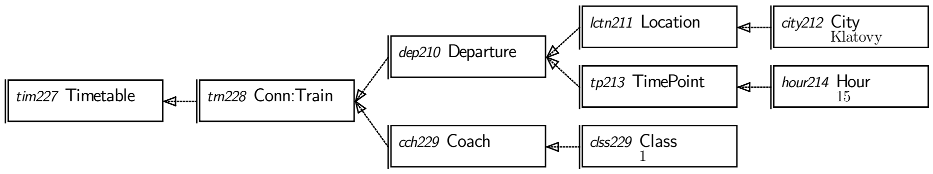
U_1 I want to go by train to Klatovy at fifteen o'clock.

Timetable(*Train*(**Departure**(← **error**
Location(*City*:"Klatovy") ,
TimePoint(*Hour*:"15")
)))



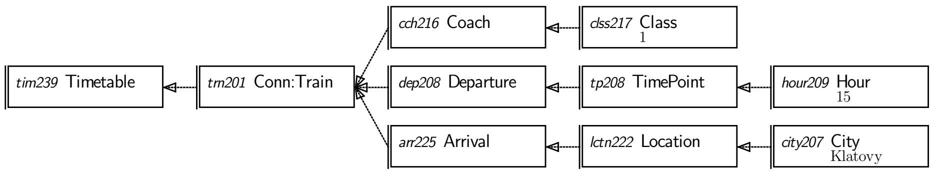
S_1 What class do you want to travel in?
U_2 Second.

Timetable(*Train*(*Coach*(**Class**:"1"))) ← **error**



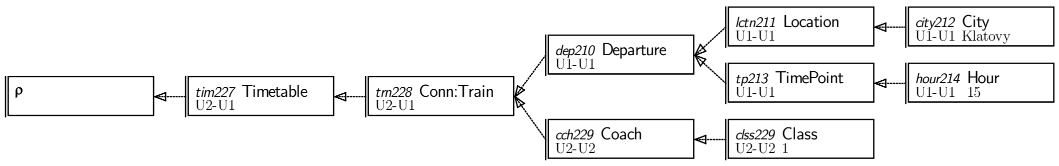
S_2 I understood you want to depart at 15 o'clock from Klatovy by the first class train. Where do you want to go to?
U_3 No, I do not want to depart from there but to arrive.

```
1   _disagree_ ,
2   Timetable( Train(
3               _disagree_( Departure( Location ) ) ,
4               _agree_( Arrival )
5   ) )
```

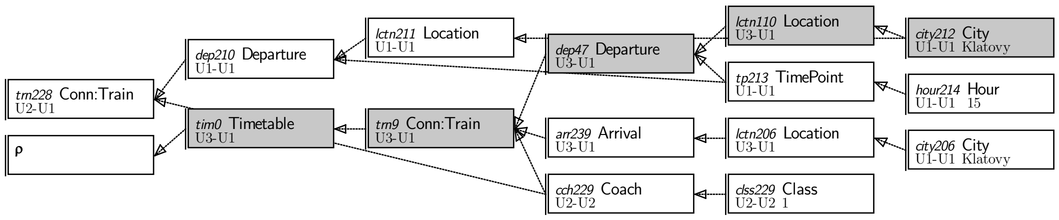


The error recovery is a non-trivial process. The main reason is that the information pool is of a flat nature, with information originating from different utterances. In general, user's corrective intention is recognized by the co-occurrence of the following two features in a semantics:

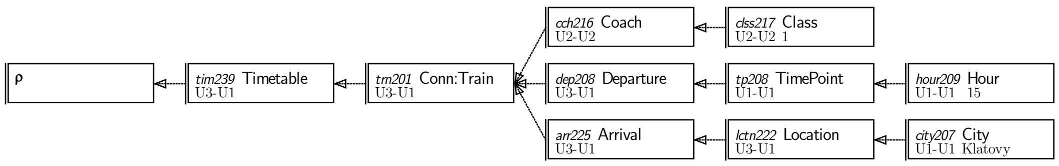
- *Disagreement* or *imperative utterance type* (either required). Accepted is either a blank disagreement whenever the system has *not* posed a Yes-No question (e.g., Line 1 in U_3 semantics, corresponding to the initial "*No*"), or a closer specified disagreement (e.g., Line 3, corresponding to "*I don't want to depart*").
- *Agreement* (optional). Any agreed (or more specifically "not explicitly disagreed") information is considered to carry correct information to replace the disagreed one, if any. To "link" the agreed and disagreed pieces together is the objective of the error recovery algorithm itself.



(a) initial state with misunderstood information



(b) information pool with disagreed portion of a semantics marked



(c) corrected information pool (disagreed portion extracted and re-incorporated to the correct place)

Fig. 4.5 Processing of user's utterance "*I do not want to depart from there but to arrive*"; instances are annotated with *utterance–last_update* pairs.

The essential clue for recovering from an error is the observation of user's disagreement. In the ideal case, the user pinpoints the particular information s/he disagrees with (as in "*I do not want to depart from there*"). When incorporating such disagreement into the information pool, the corresponding concept instances get marked correspondingly, as in Fig. 4.5b (notice *city212* has been marked despite not explicitly disagreed by the user; we will clear that below). When an agreed part of the semantics is then encountered (Line 4 in U_3 semantics), the correction processing melts down to a simple rule of finding an object (*dep47* in Fig. 4.5b) that contains "compatible" disagreed subobjects (*loc110*). It is reasonable to replace the path to these disagreed subobjects with a path from the agreed portion in the semantics (i.e. replacing Π(…, *dep47*, *loc110*, *city212*) with Π(…, *arr225*, *loc222*, *city207*) as shown in Fig. 4.5c).

In the case of the user having not explicitly specified disagreed information, the above rule of searching compatible disagreed subobjects fails. Consider the following alternative dialogue continuation to the previous example:

S_2	I understood you want to depart at 15 o'clock from Klatovy by the first class train. Where do you want to go to?
U_4	No, I want to go there.

*_disagree_* ,
Timetable(*Train*(*Arrival*(*Location*)))

While the core principle remains (i.e. searching for *compatible subobjects* to change their paths with agreed portions of the semantics), the underlying "linking" rule has to be reformulated. For the above situation, the missing disagreement may be naturally understood as "disagreeing with the most recently used information compatible with location", or generally speaking, "disagreeing with the most recently used compatible subobject". Given that the user's intention is the same in both U_3 and U_4, application of this rule marks the same objects as in Fig. 4.5b, now also including *city212* – it is the most recently used information within *loc110* (both *loc110* and *city212* have been first introduced in U_1).

Let us now make several notes regarding the underlying recovery algorithm itself (Fig. 4.6). Apart of the self-explanatory set of rules (*RULES*), the first point to make is that it can operate in two complementary modes (*MODE*): *correct*, to extract the correct part of a DDM expression, i.e. the part which has not been marked as disagreed, and *incorrect*, to extract only the marked portion of the DDM expression. To briefly illustrate, given the marked instances in Fig. 4.5b, the *incorrect*-mode result is *Timetable*(*Train*(*Departure*(*Location*(*City*:"Klatovy"))))), while the *correct*-mode result is the "remainder". The result is always stored in an auxiliary concept ($I_{EXTRACT}$) to allow for further handling of the returned expression (e.g., in revised algorithm in Appendix A.2.2).

The recovery algorithm is based on applying extraction rules to subobjects of a given parent concept instance. The application occurs at two different levels: *collection instances* (to determine if the collection as a whole is "relevant" in the given mode), and *concept instances* (soft-grained handling and recurrent processing). The implementation of this process then accounts for yet one level (*edge instances*) which is, however, due to simplicity reasons not included in the algorithm outline in Fig. 4.6, nor are included some singular extraction situations. Interested reader is therefore advised to trace the `Extrakce` procedure in `Vrstva.pas` to gain a complete view at the extraction process.

4.5.4 Requirement 3: Representing Information Spanning Multiple Intentions

We so far assumed a dialogue consisted of merely a single intention that was collaboratively solved and then replaced by another intention (or the dialogue finished). This is rarely the case, unless considering state-based approaches [McT02, Jok10]. The agent-based approach usually uses some variant of Grosz and Sidner's work on collaborative dialogues [Ric01, Ngu06b, Rot07] which holds also in our case. We nonetheless will not consider their whole framework at this moment but focus ourselves merely on the following two aspects:

- organizing information of multiple intentions,
- passing information between two intentions.

Thus our current aim is not the recognition of user's intentions but rather mere management of information that relates to them.

```
$RULES$ = set of prioritized filtration rules to extract relevant information := {
        [ $priority$: highest,  $rule$: "subcollection contains disagreed concept instances" ],
        [ $priority$: medium,  $rule$: "subcollection was uttered in system last utterance" ],
        [ $priority$: lowest,   $rule$: "subcollection was last updated
                                        at the same moment as the parent instance" ]
}
$MODE$ = extraction mode to generate a DDM expression $\in \{correct, incorrect\}$
$I_{EXTRACT} = I(\varnothing, \rho, \varnothing)$

procedure $\_Xtract\_( I_{parent} = I(C_{parent}) )$ {
    $Rule$ := highest priority rule in $RULES$ that fires for any of $I_{parent}$ subcollections
    $\forall Y_{sub} := U(Y_{sub}) = (K, I_{sub}, F_{sub}) \in P(I_{parent})$ {
        // check if user subcollection $Y_{sub}$ is relevant for the DDM expression to extract
        if $Rule(Y_{sub})$ not fired {
            if $MODE = incorrect$ { $F_{sub} := F_{sub} \setminus (E_{parent}, I_{parent})$ }
            skip $Y_{sub}$
        }
        // check if $Y_i$ is entirely relevant, incl. subcollections (defererenced objects only)
        if $A(Y_{sub}) = dereferenced \wedge MODE = correct$ { skip $Y_{sub}$ }
        // check which of $Y_{sub}$ concept instances are relevant
        $Y_{dup} = (K, \varnothing, \varnothing)$
        $\forall I_i = (D_i, C_i, X_i) \in I(Y_{sub})$ {
            // duplicate $I_i$ (incl. all edges to nested subcollections)
            $I_{dup} = I(C_i)$
            // check if $I_i$ is relevant
            if $Rule(I_i)$ not fired {
                if $MODE = correct$ { $I(Y_{dup}) := I(Y_{dup}) \cup I_{dup}$ }
                skip $I_i$
            }
            // recurrently process and evaluate result
            $\_Xtract\_( I_{dup} )$
            if $MODE = correct \wedge P(I_{dup}) = \varnothing$ { skip $I_i$ }
            // add $I_{dup}$ to $Y_{dup}$
            $I(Y_{dup}) := I(Y_{dup}) \cup I_{dup}$
        }
        // determine if $Y_{dup}$ contains concept instances (and is thus a DDM expression)
        if $I(Y_{dup}) \neq \varnothing$ {
            $F(Y_{dup}) := F(Y_{dup}) \cup (E_{parent}, I_{parent})$     // add $Y_{dup}$ as subcollection of $I_{parent}$
        }
        $F_{sub} := F_{sub} \setminus (E_{parent}, I_{parent})$     // delete edge to "original" subcollection $Y_{sub}$
    }
}

procedure $Extract( Y = (K, I, F) \in \mathbf{Y}, Mode \in \{correct, incorrect\} )$ {
    $MODE = Mode$
    $F := F \cup F_{EXTRACT}$, where $F_{EXTRACT} = (E_{EXTRACT}, I_{EXTRACT})$
    _Xtract_( $I_{EXTRACT}$ )     // the result of extraction is stored in $I_{EXTRACT}$
}
```

Usage: $Extract(Y, correct/incorrect), Y \in \mathbf{Y}$

Fig. 4.6 Algorithm to extract either the "correct" part of a DDM expression or the "incorrect" part ($MODE$ switch)

To begin with, ponder the dialogue snippet in Fig. 4.7. As it can be seen, the dialogue consists of user's two sub-intentions (beginning with U_1 and U_2) and two system sub-intentions (beginning with S_1 and S_5) that together define the logical *segmentation* of the dialogue into the so called *discourse segments* (*DSs*), labeled as $DS_{1...4}$. As Grosz and Sidner propose in their work, each of these segments is assigned a *focus space*, "recording the objects, properties, and relations that are salient in its scope – either because they have been mentioned explicitly in the segment or because they became salient in the process of producing or comprehending the utterances in the segment" [Gro86]. Each focus space also includes a *discourse segment purpose* (*DSP*) to keep track of "why" given information is being discussed in the dialogue. Fig. 4.8 illustrates the linkage between the focus spaces and intentions – they are proposed to constitute a stack. Information in lower spaces is therefore accessible from higher ones but "less" than the information in higher spaces. In terms of salience, the lower positioned focus spaces are of lower salience than their higher positioned counterparts. Note that we at this moment will not examine why DSP2 has been placed "above" DSP1 or why DSP4 has "replaced" DSP3 in the stack (the necessary management foundation will be covered later).

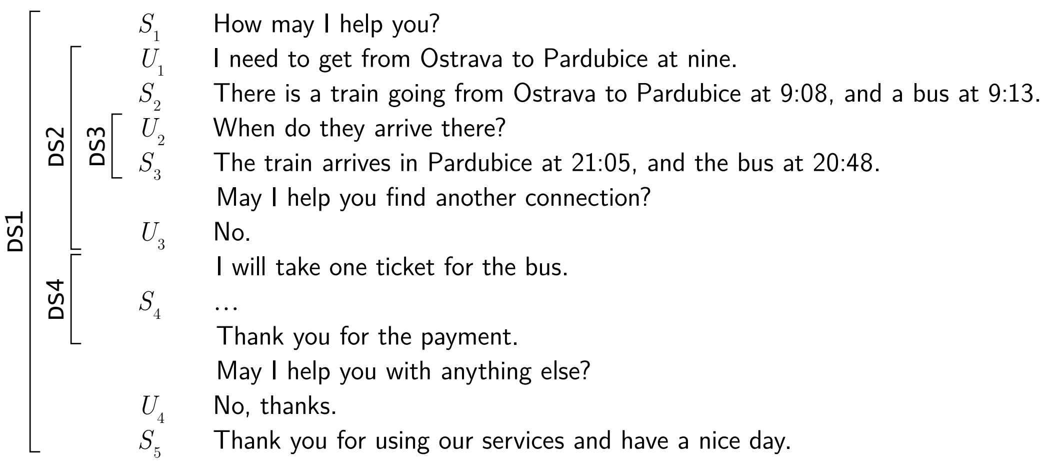
The fact that focus spaces are stacked onto each other has natural implications on interpreting user's utterances. For instance, if an airplane existed in DS2 and another different one in DS1, then user's sentence "*the airplane*" uttered in DS2 would be understood to address the particular airplane in DS2. Contrarily, if a ship existed only in DS1, user's sentence "*the ship*" uttered in DS2 would be understood as referring to that ship in DS1.

Definition 4.12 (DSP function)
Let **F** denote the set of tasks (funtionality) of a given system. Let $Y = Y(K) \in \mathbf{Y}$ be a topic instance. A *discourse segment purpose function* is a projection *DSP*: $\mathbf{Y} \rightarrow \mathbf{F}$. □

Hence, Grosz and Sidner's work is approached using topics. Recall from Section 4.2 that topics are an abstraction of a task or a group of similar tasks. Different DSPs then distinguish instances of the same topic from each other. Of course, the information pool plays here merely a role of a blob of known information grouped into topics; these topics are not organized into a stack. However, this can be worked around by keeping topics salience up-to-date with their most salient related object.

Fig. 4.9 demonstrates this idea. As it can be seen, the train and bus instances (*trn28* and *bus23*) have been *passed over* from DS1 (*tim34*) to DS2 (*tim1*), being still affiliated to DSP1 just as expected. The information pool also allow for *objects overriding*. In such case, objects are first passed over and then changed, resulting in the unmodified part of them being shared among DSPs while the changed part being specific only for the DSP that triggered the modification (notice two arrows directing from system K_{conn} to *tim34* and *tim1* topics). This operation is naturally in coherence with Grosz and Sidner's work.

Finally, see Appendix A.2.3 for updates to the information management approach, accommodating now information spanning multiple intentions.



S_1 How may I help you?
U_1 I need to get from Ostrava to Pardubice at nine.
S_2 There is a train going from Ostrava to Pardubice at 9:08, and a bus at 9:13.
U_2 When do they arrive there?
S_3 The train arrives in Pardubice at 21:05, and the bus at 20:48.
May I help you find another connection?
U_3 No.
I will take one ticket for the bus.
S_4 …
Thank you for the payment.
May I help you with anything else?
U_4 No, thanks.
S_5 Thank you for using our services and have a nice day.

Fig. 4.7 Segmented multiple-tasks dialogue.

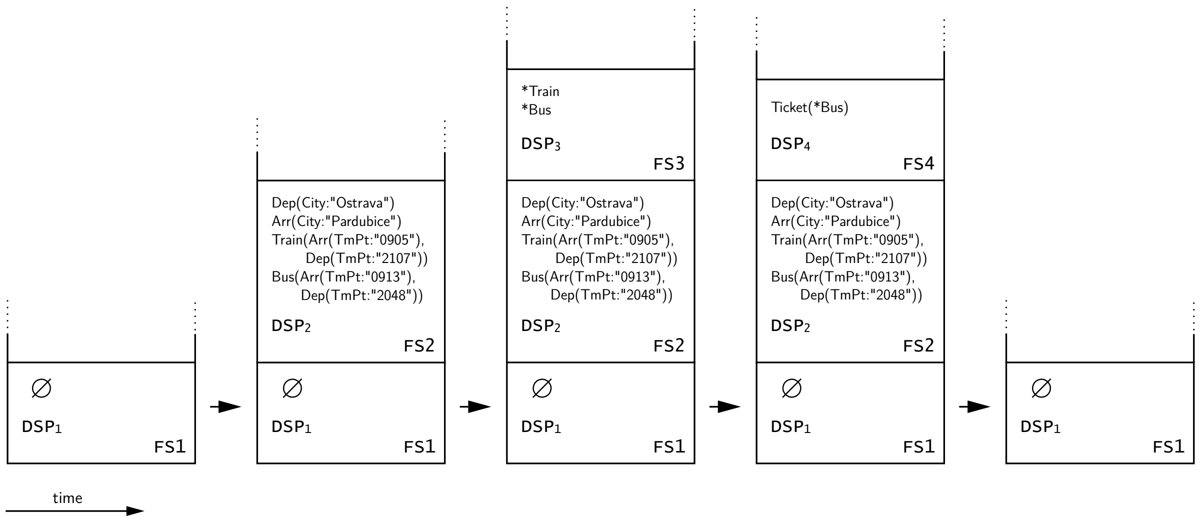


Fig. 4.8 Focus stack transitions between utterances S_1 and S_5 in dialogue from Fig. 4.7.

4.5.5 Requirement 4: Representing User's Underspecified Information

This section is on representing information that *cannot* be resolved using the dialogue context or dialogue history. With respect to the DDM in Fig. 4.1, such case occurs, for instance, in the following dialogue snippet:

S_1 How may I help you?
U_1 I want to go to Praha.
Semantics: *Location*(*City*:"Praha") ← **error**
($C_{timetable}$ and $C_{arrival}$ not recognized)

Here, error recognition caused user's information originally fully specified to be observed as underspecified. Assuming the information pool is empty at the beginning, the system cannot resolve if the user talks about a departure or an arrival, nor if the information contributes to a timetable search or ticket

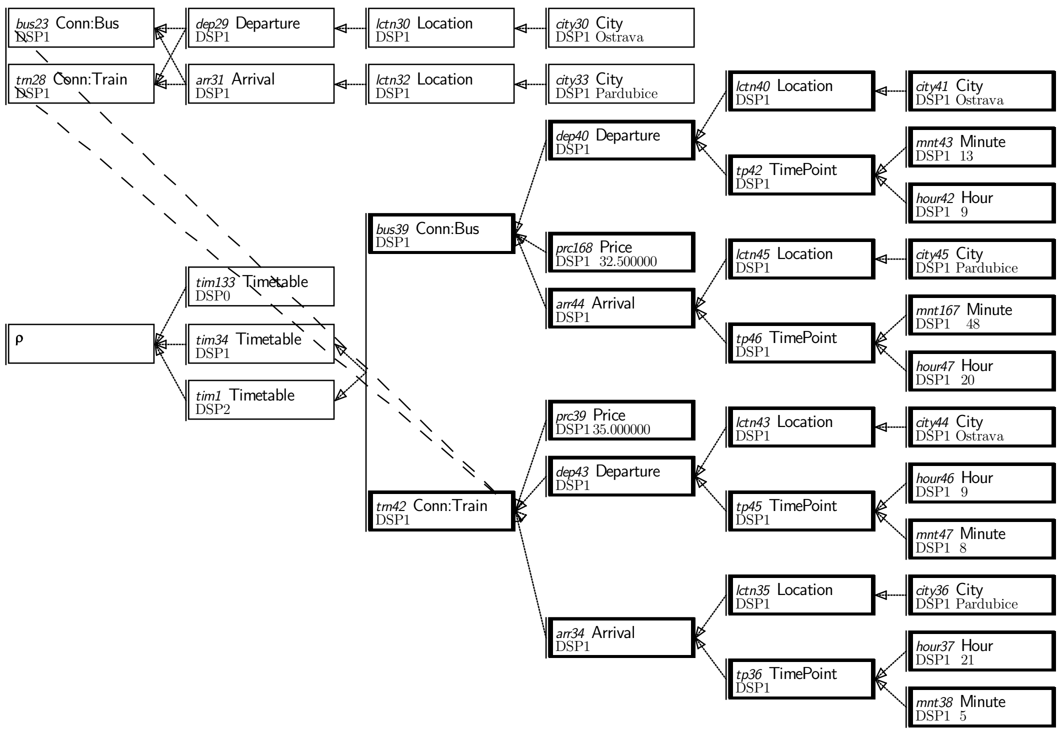
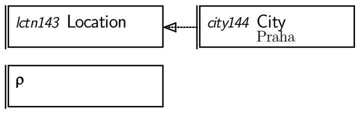


Fig. 4.9 Multi-topics information pool with three different discourse segment purposes.

purchase. One of the anticipated dialogue continuations in such situation could be (dialogue planning covered later in Section 4.8):

S_2 Should the city of Praha regard your departure or arrival?

Each piece of underspecified information is in the information pool represented as an "unbound" DDM expression, with root having no parent, i.e. Y: $F(Y) = \emptyset$. For the above snippet, the following information pool shows the result:



It is necessary to point out that the system perceives unbound expressions as mostly vague pieces of information which eventually may not even contribute to the current DSP in focus (e.g., due to ASR failure). Hence, the management of such information is additionally constraint as follows:

- Unbound expressions are not subject of object passing between discourse segments (they therefore cannot be shared nor overridden in order to not spread the possible error that may stand behind them).
- Unbound expressions exist in a single instance within a given discourse segment (it is therefore impossible to have two unbound city locations within a DS – should such situation arise, the current unbound location replaces the previous one).

- Unbound expressions disappear as soon as a concept instance that can contain them, emerges (e.g., for the above sample snippet, as soon as the user decides Praha being the location of either departure or arrival). The containing instance can in turn become an unbound expression.

Despite the fact that the representation of unbound expressions is very straightforward, they mean a significant update to the information management. To spare on space, only updates to the *Incorporate* procedure are shown in Appendix A.2.4. Interested reader is suggested to have a closer look at the `Extrakce` procedure in `Vrstva.pas` to gain a complete view at the extraction process for unbound DDM expressions.

4.5.6 Requirement 5: Information Scalability

So far, we concerned with merely simple types with strictly disjunctive values; for instance, we assumed integer values (as in *Class*:"1") or string values (as in *City*:"Cheb"). Such values were reasonably supposed to have nothing in common, implying one could always be easily replaced with the other (*Class*:"1" $\rightarrow$ *Class*:"2", or *City*:"Cheb" $\rightarrow$ *City*:"Praha"). The mentioned integer and string are part of the elemental *built-in data types* set, natively provided by the framework (see Table 4.2).[4] These intrinsic types are, however, of a limited usability. For instance, it is impossible to adjust the wrong understood city of Hradec Králové simply by saying "*No, I mean u Stoda*" to change the location to Hradec u Stoda, as for the string type it holds *City*:"Hradec Králové" $\rightarrow$ *City*:"u Stoda".

Generally, a particular object value is the result of an information type-specific operation [Men96]: a new value can not only replace but can also extend or infer from an old value; for instance, we can replace the number of passengers on a ticket, merge ticket discounts, and evolve the name of a city by combining two pieces of information. Given that the listed sample operations are all of distinct nature, the information management does not provide any "combinatorial pattern" to determine a new value. Instead, it passes this responsibility to external resources – domain-specific libraries that define the so called *external data types*.

As already seen in Definition 4.1, each data type T is a set of functions that together completely and unambiguously define the range and operations with eigen information ψ, i.e. $T = \{f_i\}$. The functions proposed to provide such properties are listed in Table 4.3.[5] For instance, the above outlined problem of

4 For optimization reasons, these types have been chosen with respect to their local language independence, and do not contain types whose range is a subset of another type – for instance, *Char* $\subset$ *String* $\subset$ *Utf8String* (the input semantics lexical analysis guarantees that any later comparison of two strings is as quick as a comparison of two characters) or *Boolean* $\rightarrow \{0,1\} \subset$ *Integer*.

5 From the technical point of view, note that each of the binary functions is considered a *tolerance* with reflexive and asymmetric properties. The reflexivity eliminates randomness from processing by pertaining any *old* value if related to itself. The asymmetry says the order of arguments matters. This is important when developing external data types. As a rule, the left argument is always the comparer while the right argument is always the comparee, as, for instance, in the *IsContainedIn* function whose semantics is "true if inf_2 is contained in inf_1, otherwise false".

Table 4.2 Daisy framework intrinsic data types; the Size and Capacity columns apply to x86 platform.

Name	**Size**	**Capacity**
Ordinal	4 Bytes	-2147483648 … 2147483647
Double	8 Bytes	ten digits of precision
UTF8 String	1 Byte minimum	unlimited

Table 4.3 Data type definition functions.

Function C-style synopsis and description
void **Create* (char **description*)   Creates information based on its textual description.
int *GetCardinality* (void **inf*)   Returns the cardinality of the specified information.
bool *Equal* (void $*inf_1$, $*inf_2$)   Returns true if both pieces of information are equal, otherwise false.
bool *IsCombinable* (void $**inf_1$, int $nInf_1$, void $*inf_2$)   Returns true if inf_1 can be combined with inf_2, otherwise false.
void **Combine* (void $*inf_1$, $*inf_2$)   Returns the result of combining inf_2 with inf_1.
void **Negate* (void **inf*)   Returns the negation of the specified information.
bool *IsInstantiable* (void **inf*)   Returns true if the specified information is instantiable, otherwise false.
bool *IsContainedIn* (void $*inf_1$, $*inf_2$)   Returns true if inf_2 is fully contained in inf_1.
bool *IsUndefined* (void **inf*)   Returns true if the specified information does not contain value.
char **ToText* (void **inf*)   Returns TTS module-processable form of the specified information.
void *Destroy* (void **inf*)   Destroys specified information.

city name misunderstanding can easily be solved for by defining an external data type with the following combinatorial pattern (schematic):

$$X(City_{old}) \oplus X(City_{new}) \;=\; \begin{cases} X(City_{old} \oplus City_{new}) & \text{, if } City_{old} \oplus City_{new} \text{ is a known name,} \\ X(City_{new}) & \text{, otherwise} \end{cases}$$

where $\oplus$ is a value-combination operator and $X(.)$ is the value projection function as introduced in Definition 4.7.

Apparently, the "completeness" requirement for a data type operations refers to covering all possible combinatorial situations. Given that each combination always accounts for an *old* and a *new* information, each of which may be either agreed or disagreed, the combinatorial situation falls into the Cartesian space

Table 4.4 Information combining behaviour for different mutual relationships of Old and New information; $\oplus$ is the information-combining operator, and subscripted D denotes a disagreed portion of information (to be removed from the containing list).

Rule (Condition $\rightarrow$ white-list [attributes] ; black-list [attributes])
$(\mathit{Old} \oplus \mathit{New}) = \varnothing \quad \rightarrow \quad (\mathit{Old}) \; ; \; —$
Old and *New* information do not have anything in common – they will exists in parallel as they cannot be combined.
$(\mathit{Old} \oplus \mathit{New}) = \mathit{Old} \quad \rightarrow \quad (\mathit{Old}) \; ; \; —$
New information is fully contained in *Old* information.
$(\mathit{Old} \oplus \mathit{New}) \neq \varnothing \quad \rightarrow \quad (\mathit{Old} \oplus \mathit{New}) \; ; \; —$
General combination of agreed *Old* and *New* information.
$(\mathit{Old} \oplus \neg \mathit{New}) = \varnothing \quad \rightarrow \quad (\mathit{Old})_{\mathrm{D}} \; ; \; —$
Disagreed $\neg$*New* object completely contradicts *Old* object, hence *Old* is necessary to be marked as disagreed.
$(\mathit{Old} \oplus \neg \mathit{New}) = \mathit{Old} \quad \rightarrow \quad (\mathit{Old}) \; ; \; —$
Non-disagreed portion of *New* supports *Old*.
$(\mathit{Old} \oplus \neg \mathit{New}) \neq \varnothing \quad \rightarrow \quad (\mathit{Old} \oplus \neg \mathit{New}) \, , \, (\mathit{Old} \oplus \mathit{New})_{\mathrm{D}} \; ; \; —$
General combination of disagreed $\neg$*New* with white-listed *Old* information.
$(\neg \mathit{Old} \oplus \mathit{New}) = \varnothing \quad \rightarrow \quad — \; ; \; (\neg(\mathit{Old} \oplus \neg \mathit{New})) \, , \, (\neg \mathit{New})_{\mathrm{D}}$
New partially contradicts with $\neg$*Old*.
$(\neg \mathit{Old} \oplus \mathit{New}) = \neg \mathit{Old} \quad \rightarrow \quad — \; ; \; (\neg(\mathit{Old} \oplus \neg \mathit{New})) \, , \, (\neg \mathit{New})_{\mathrm{D}}$
New partially contradicts with $\neg$*Old*.
$(\neg \mathit{Old} \oplus \mathit{New}) \neq \varnothing \quad \rightarrow \quad — \; ; \; (\neg(\mathit{Old} \oplus \neg \mathit{New})) \, , \, (\neg(\mathit{Old} \oplus \mathit{New}))_{\mathrm{D}}$
General combination of agreed *New* with blacklisted $\neg$*Old* information.
$(\neg \mathit{Old} \oplus \neg \mathit{New}) = \varnothing \quad \rightarrow \quad — \; ; \; (\neg \mathit{Old})$
Disagreed $\neg$*New* information does not contradict with blacklisted $\neg$*Old*, i.e. both can exist in parallel.
$(\neg \mathit{Old} \oplus \neg \mathit{New}) = \neg \mathit{Old} \quad \rightarrow \quad — \; ; \; (\neg \mathit{Old})$
Disagreed $\neg$*New* is a superset of blacklisted $\neg$*Old*, i.e. both can exist in parallel resulting in a union of disagreements).
$(\neg \mathit{Old} \oplus \neg \mathit{New}) \neq \varnothing \quad \rightarrow \quad — \; ; \; (\neg \mathit{Old})$
General combination of disagreements is replaced by both of them coexisting in parallel, resulting in a union of disagreements.

of $\{agreed, disagreed\} \times \{agreed, disagreed\}$. Furthermore, for each such situation, we distinguish its three significant results: (1) both pieces of information have nothing in common, (2) are equal, or (3) overlap. Table 4.4 overviews these relationships along with short explanations of their meaning.

4.6 Dialogue Context

4.6.1 Problem Identification

As already mentioned, our approach to dialogue context follows Grosz and Sidner's work on discourse [Gro86]. They propose that the attentional component in a dialogue can be recognized as noun and pronoun phrases. Contrarily, the recognition of intentions at the level of DSPs is a large research problem in itself. As it turns out, the clues for a proper intention recognition come from a variety of sources that must be put together to fully identify the intention [Gro86]:

- *Cue phrases.* Phrases like "*excuse me*", "*by the way*", etc. are the most apparent linguistical means to indicate discourse segment boundaries and thus the beginning or end of a DSP. Grosz and Sidner divide these phrases into categories (e.g., attentional change, interruption, etc.) that further help organize the structure of the dialogue. However, cue phrases are ambiguous; e.g., if several interruptions have been made in a dialogue, the cue phrase "*but anyway*" indicates a return to some previously interrupted discourse, but does not specify which one [Gro86].

- *Utterance-level intention.* Intention is to describe utterance meaning [Gri69]. Generally, a DS consists of several utterance-level meanings, which must be combined in some way to produce the overall DSP. This is a quite complex process as it is necessary to recognize that subsequent utterances do not bear standalone purposes but should be used in the same context.

- *Shared knowledge about the domain.* This refers to both conversational partners knowing the taxonomy of domain tasks and how different tasks can be "nested" into each other.

The intention recognition process must be capable of operating on partial information. It thus must allow for incrementally constraining the range of possibilities as more information becomes available over the course of a particular discourse segment.

Hence, this section aims to spot the general problem of properly representing a dialogue context with respect to user's intention recognition. Grosz and Sidner's work has been partially implemented in the past (e.g., in [Ric01, Ngu06b, Rot07]), with "partially" referring to different workarounds to utterance-level intentions. However, the generally preferred way to representing and recognizing intentions is by analyzing utterance linguistical structure for dedicated "request" tokens [Eck95, Boh09, Ngu06b]. Presumably, both of these competitive approaches have their pros and cons: while Grosz and Sidner's approach plays a rather universal role, it is unnecessarily over-scaled for common collaborative dialogues (e.g., as simplifications in [Ngu06b] illustrate); contrarily, the "request" token-approach is simple enough to be easily accounted for, but lacks flexibility as for evolution of user's intentions.[6]

Presumably, for our framework to be as general as possible, Grosz and Sidner's work has been partially adopted too. Before describing, let us introduce the term *dialogue act* and its importance for maintaining a coherent dialogue with the user.

4.6.2 Dialogue Acts

Dialogue acts have their roots in theory of purpose and effect of speech. More particularly, by making an utterance, the speaker in a dialogue intends to perform some action known as a *speech act* [Sea69]. In this respect, the important application of speech acts is to give a clue for a proper recognition of speaker's intentions by observing their performed speech acts. They may be therefore seen as the cornerstone in natural speech processing, be it at the level of automated conversational agents, or during an annotation post-processing of existing dialogue transcripts [Eck95]. For practical reasons, conversational partners' utterances need to be described a more abstract way in order to account for the necessary terms like *clarification* or *response*, dealt within a dialogue. These abstract terms are referred to as *dialogue acts*. Each dialogue act is fully characterized by a *semantic content* (information conveyed or requested by the particular utterance) and a *communicative function* (the purpose of saying that utterance[7]). Thus, in order for a conversational agent to maintain a coherent dialogue with the user, it is necessary to first recognize the associated dialogue acts that lead to understanding user's underlying intentions.

Given that the agent must allow for a mixed-initiative style of interaction, the proposed set of dialogue acts must be modeled correspondingly. Our proposition can be seen in Table 4.5. The notion of the mixed-initiative is captured by giving a user the possibility to take a turn when requesting a task (*Request*) or confirming information (*Confirm*); contrarily, the system can take a turn when requesting clarification (*Request-clarify*) or informing about task results, thus finishing the task and the corresponding discourse segment (*Respond*). This mixed-initiative behaviour occurs automatically as the result of the agent selecting appropriate dialogue strategies by observing a known context situation patterns (more on them in Section 4.9). The following snippet shows a dialogue with sentences annotated with their corresponding dialogue acts.

S	Welcome to the Timetable Information System.	*Politeness*
	How may I help you?	*Request*
U	When does the next train go to Hradec?	*Request*
S	Do you mean Hradec Králové or Hradec u Stoda?	*Request-clarify*
U	Hradec u Stoda.	*Clarify*

6 This means that their requests have to be expressed a "parser-aware" way within a single turn.
7 This function is usually related to a task, but often some of the function is to maintain the interaction process. Dialogue acts with such notion are called *dialogue control acts* [All92].

Table 4.5 Dialogue acts.

Act type	**Description**
Politeness	General politeness padding
Request	Ask the dialogue parter to fulfil a request
Request-yn	Same as Request but with confirmation accepted
Respond	Give results to a request
Request-clarify	Ask the dialogue partner to clarify some ambiguities
Clarify	Clarify some ambiguities
Request-confirm	Ask the dialogue partner to confirm some proposition
Confirm	Confirm some proposition
Help	Request contextual help
Repeat	Request repetition of dialogue partner's last proposition
Ack	Express acknowledgement
Rogerian-psychologist	Ask the dialogue partner to clarify some ambiguities

S	There is a train going to Hradec u Stoda at ...	*Respond*
	Do you want to buy a ticket now?	*Request-yn*
U	No, thanks.	*Confirm*
S	Thank you for using our services and have a nice day.	*Politeness*

At the beginning, the user asks for information about departure times. Since "*Hradec*" is an ambiguous city name, the system performs a *Request-clarify* act to gain more detailed information from the user. The task can then be completed successfully and the system performs a *Respond* act to inform about the connections found, immediately followed by a *Request* act to propose possible continuation in the interaction. With the user refusing, the system performs a farewell *Politeness* act and terminates the dialogue. In this regard, let us note that we group agent's all polite phrases under the *Politeness* dialogue act, despite the common way of classifying them individually, e.g., *Greeting*, *Goodbye*, *Please*, etc.

4.6.3 Recognizing Dialogue Acts

With the dialogue acts defined, the arising question is how to recognize them in user's utterances. Different approaches may be used, depending on the overall complexity of the dialogue management and desired functionality of the resulting system. For instance, in information state-based systems, the way of recognizing dialogue acts is relatively easy to conduct. Given the observable effects of utterance on the underlying information state, user's dialogue acts (or set of) can be simply induced by the update they have caused [Tra03, Bui06]. Contrarily, in a SIL-based dialogue manager, a set of flat rules is proposed to extract dialogue act(s) from user's current utterance and corresponding UFOs

Table 4.6 Nguyen's [Ngu06b] sample heuristic rules for dialogue act type determination.

Rule	User's last act	Agent's last act	Utterance type	Result
1	*Request*	*Respond*	imperative	*Request*
2	*Clarify*	*Respond*	wh-question	*Request*
3	*Cancel*	*Ack*	yn-question	*Request*
4	*Request*	*Request-clarify*	declarative	*Clarify*
5	*Request*	*Request-clarify*	imperative	*Cancel*
6	*Request*	*Request-confirm*	declarative	*Confirm*

Table 4.7 Daisy framework sample heuristic rules for dialogue act type determination; acts marked with an asterisk must have been performed on intentional layer, and dash denotes unimportant parameter.

Rule	Agent's last act	Utterance type	Result
1	—	interrogative	*Request*
2	*Request / Request-yn*	declarative	*Request*-suspected
3	*Request-clarify**	—	*Request*-suspected
4	*Rogerian-psychologist**	—	*Request*-suspected
5	*Request-clarify*	declarative	*Clarify*
6	*Request-confirm*	declarative	*Confirm*

(e.g., if an UFO contains an A-parameter that is part of a wh-question, then the user has performed a *Request* dialogue act) [Eck95].

Nevertheless, in the case of the Daisy framework, we adopt Nguyen's approach consisting of a set of heuristic rules fired under distinct combinations of both participants' last dialogue acts and user's current utterance syntactic class (Table 4.6). However, we made two modifications to her approach. First, we do not consider the *Cancel* dialogue act; provided that cancellation is a destructive operation, we think it should not be roughly estimated merely upon observing imperative utterance as in Rule 5. Instead, we propose it to be modeled as a meta-task with a special phrase as a trigger. And second, we do not take into account user's last dialogue act. The reason is that this feature is redundant in Nguyen's approach. Table 4.7 shows our resulting set of rules.

Probably the most notable point to make regards the "*Request*-suspected" dialogue meta-act. As its name suggests, it is to indicate those utterances that *possibly* may be recognized as *Request* dialogue acts (with additional analysis of their semantics, covered next). The following dialogue snippet with four alternative user responses shows the motivation behind this approach:

S	...There is a train and bus going to Cheb at 7 o'clock.	*Respond*

	Do you want to buy a ticket now?	*Request-yn*
U_{alt1}	Is there anything departing later?	*Request*
U_{alt2}	I am quite sure there was yet something later.	*Request*-suspected $\rightarrow$ *Clarify*
U_{alt3}	Yes.	*Request*-suspected $\rightarrow$ *Confirm*
U_{alt4}	Say me their arrival times, please.	*Request*-suspected $\rightarrow$ *Request*

As it can be seen (currently wihtout further explanation), while U_{alt1} describes a direct question caught by Rule 1 as a *Request* dialogue act, U_{alt2} and U_{alt3} show a false suspicion as their respective contents are recognized by Rule 2 and additional analysis as *Clarify* and *Confirm* acts. Finally, U_{alt4} shows a successful suspicion of a *Request* dialogue act, caught also by Rule 2. However, before we can cover the additional analysis on determining if a sentence contributes to the current discourse segment or creates a new one (or possibly returns to a dominant one), we need to first describe the overall structure of our adoption of Grosz and Sidner's work.

4.6.4 Approaching Grosz and Sidner's Work: Two-layered Representation

Grosz and Sidner's work on discourse [Gro86] is widely agreed as it puts together and further develops theory from different realms of dialogue processing: focusing in discourse, utterance-level intentions, and discourse intention recognition. They suggest that a discourse is a combination of three components: linguistical structure, intentional structure, and attentional state. The *linguistical structure* refers to recognizing the boundaries of discourse segments (DSs), i.e. sequences of utterances that "fulfil certain functions with respect to the overall discourse". We already saw example of dialogue segmentation in Fig. 4.7. The *intentional structure* describes discourse purposes (DPs) that stay behind engaging in the particular discourse. At the level of DSs, the overall DP is decomposed into discourse segment purposes (DSPs) each of which specifying the contribution to achieving the DP. Finally, the *attentional state* is a stack of focus spaces as they were described in Section 4.5.4.

These three components do not act as separate entities but rather mutually depend on and influence each other. For instance, the linguistical structure is affected by determining mutual relationships between subsequent DSPs (the so called *dominance* and *satisfaction-precedence* relationships). On the other hand, although there is an infinite number of intentions a discourse participant may have, there is only a small number of intentions relevant to the current discourse structure [Gro86].

For a successful application in automated task-oriented systems, Grosz and Sidner's work may be significantly simplified. Given the domain-wise orientation of nowadays dialogue system, there is always a fixed number of tasks the user may intend to discuss. We also can presume that the user always makes the intention *explicit* so that it can be recognized (i.e. the user does not have hidden intentions, unrecognizable from the system standpoint). Hence, these individual tasks can be considered the discourse segment purposes. Given the assumed collaborative behaviour of the user, the overall discourse purpose may be understood as to fulfill a domain task. In addition, each user's utterance in the dialogue plays at least one of the two roles: specifying her of his intention in the current segment (i.e. a task to be performed) or adjusting the recognition of that intention (e.g., clarifying or giving more information).

The above mentioned two roles of an utterance play a significant clue within our dialogue context model. They in fact determine its layout as two distinct "layers" of inter-related information regarding currently open tasks (*task layer*) and their related "parametrizations" (*data layer*) [Nes10a, Nes13]. Naturally, the data layer approaches the attentional state in Grosz and Sidner's work, and not surprisingly may be represented using the information pool developed in Section 4.5. However, when designing the task layer, our aim has been to overcome the common limitation in dialogue systems, namely that tasks have to be recognizable from a single utterance [Eck95, Tur05, Boh09, Sin02, Wal97] so that eventually a new discourse segment is created, and recognized task set as its purpose [Ngu06b, Gus02]. Despite sufficient in majority of cases, different circumstances may cause this simple approach to fail, as illustrated in the following snippet:

S_1	How may I help you?
U_1	Karlovy Vary, please.
S_2	I understood Karlovy Vary. Should that be your departure or target location?
U_2	Target location.
S_3	Can I help you find a connection to Karlovy Vary or do you want to proceed to the ticketing service?
U_3	Find connection, please.

Here, the system attempts to recognize user's intention by elaborating so far known, hence narrowing the set of possible intentional candidates. An alternative reaction to the incomplete request in U_1 might be to list all available system functionality in S_2, e.g., "*I can offer you information on arrival times, departure times, ticketing service, local weather, tourism, ...*", eventually categorized and browsed as a hierarchy of menus.

The second motivational example shows a situation of the system posing a confirmation on the uncertainly recognized task, responded to by the user with a disagreement and a correction:

S_1	How may I help you?
U_1	When does the next train to Karlovy Vary depart?
S_2	Pardon me, I am not sure now, have you asked for arrival information?
U_2	No, departure, please.
S_3	There is a train going to Karlovy Vary at ...

As it can be seen, in both of the above dialogue snippets, the task recognition spans multiple utterances. Therefore, to allow for an incremental evolution of the task specification, the task layer is represented using an information pool as well. In this information pool, Requirements 1 and 3 from Section 4.5 are naturally

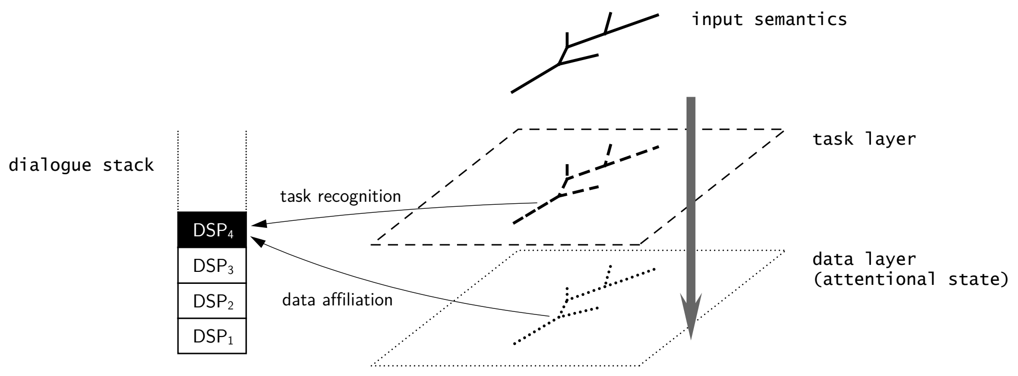


Fig. 4.10 Two-layered approach to task-oriented dialogue context representation.

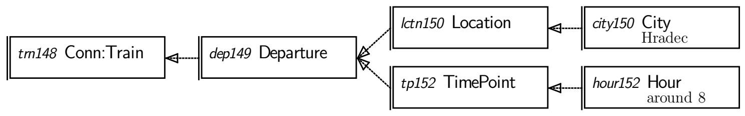
left unexploited (as we do not distinguish who of the two dialogue participants initiated the given task, nor is there the need to organize recognized tasks into advanced structures – the order of tasks is implicitly given by saliences of their corresponding instances); however, the remaining Requirements 2, 4, and 5 in their respective order guarantee that, apart of spanning multiple utterances, each task specification may be corrected, be specified at least partially, and target any custom information.

With task and data layers approached by information pools (Fig. 4.10), let us denote them $\mathbf{Y}_T$ and $\mathbf{Y}_D$, respectively. At this moment, we are facing the problem of finding those pieces of information in the input semantics that correspond to the notion of either of the layers. Apparently, detecting the *Request* (or *Request*-suspected) dialogue act is insufficient in this regard. For instance, although user's utterance "*When does the next train to Karlovy Vary depart?*" is recognizable as a *Request* using Rule 1 in Table 4.7, the utterance not only specifies a new task (departure time request) but it also brings in task-related information (next train to Karlovy Vary).

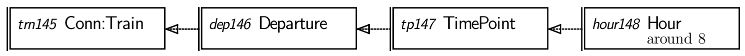
4.6.5 Fragmenting User's Utterance Semantics

The idea behind finding either of the two kinds of information targets splitting the input semantics into two distinct *fragments*. These are not necessarily disjunctive, however, presumably homomorfic with the input semantics. We call the process of splitting the *fragmentation*, and its results the *task fragment* and the *data fragment*, respectively, denoted as $\mathbf{F}_T$ and $\mathbf{F}_D$.

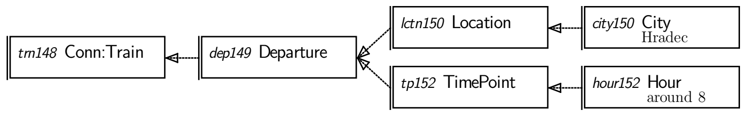
The essential constraint put on the fragmentation is to operate with paths from semantics root to semantics leaves, i.e. $\Pi(\ Y(\rho), Y(K_1), \ldots, Y(K_N)\ )$, where $Y(K_N)$ is a leaf iff $P(\ Y(K_N)\ ) = \varnothing$. Apparently, a semantics leaf does not necessarily has to be the underlying model leaf (e.g., $Y(K_{train})$ instance is a leaf in the semantics *Timetable*(*Train*), but K_{train} itself is not a leaf in the DDM in Fig. 4.1). The benefit of constraining to paths to leaves is in maximizing the compactness of the outcoming task and data fragments (see also Fig. 4.11).



(a) original input semantics (unfragmented)



(b) task-related component in the input semantics (task fragment)



(c) data-related component in the input semantics (data fragment)

Fig. 4.11 Input semantics fragmentation process motivational example.

In the following sections, we will concern ourselves with two major questions: *when* to initiate the fragmentation process and *how* to determine the optimal fragmentation.

4.6.5.1 Initiating the Fragmentation

To illustrate the problem, let us suppose a single path to leaf (to simplify the terminology and explanations, we from now on will say only "path", unless a different kind of path is meant). This path may be part of: (1) the task fragment only, (2) the data fragment only, or (3) both of the fragments. Thus, the complexity of the fragmentation problem is in general $O(3^N)$, where N is the total number of paths in the input semantics. To lessen the computational demands, a heuristics is used to mark those paths whose membership in either of the fragments is certain. The heuristics can be easily described by introducing the *cardinality of information* carried by the leaf, as follows:

- atomic information (e.g., a single time point "14:30") has a *zero cardinality* since it is always certain (i.e. it involves a single option),
- non-atomic information (e.g., a time interval "around 14 o'clock") has a *non-zero cardinality* since it tends to be uncertain (i.e. it involves more options), and finally,
- undefined information (e.g., an unknown time point hidden behind the word "when") has an *infinite cardinality* since it is uncertain.

Although exemplified for different time point values, we do not provide any formal general recipe on how exactly to compute the cardinality. The reason is that any computation of the cardinality is information-dependent matter. For example, each $C_{discount}$ object (Fig. 4.1) might take on one of three values: *Child*, *Premium Customer*, or *Senior*. The cardinality returning function for this object would be computed as follows: if the value is defined, then return zero, otherwise return infinity. If we want to combine discounts (i.e. create a collection of multiple $C_{discount}$ instances to catch, for instance, that a senior may also be a premium customer), the cardinality returning function would need to remain the same – unlike with the time point, "more options" in this case do not indicate that the value is uncertain.

Given the information cardinality, we can spot two important points. First, atomic information cannot contribute to an intentional shift as there is nothing to discuss about it – it is therefore *always* a part of the data fragment *only*. Second, an empty information never brings data to the dialogue and is thus *guaranteed* to be added to the task fragment. This, however, does not regard an exclusive membership. For instance, consider the following dialogue snippet:

S There is a train going to Karlovy Vary at 9:30, and a bus at 9:40.
U When does the train arrive?

```
_interrogative_(
    Timetable( _ref_( Train( Arrival( TimePoint( Hour:— ) ) ) ) )
)
```

The single leaf in user's utterance contains undefined information. The infinite cardinality naturally grants the corresponding path to the task layer. However, the utterance also refers to the previously mentioned train object – it therefore needs to be considered as a candidate for the data layer as well.

For completeness sake, let us note that the membership for an non-atomic information cannot be determined any other way but using an exhaustive fragmentation. However, we furthermore can reduce the exponential costs by constraining to cases that potentially contain the expected intentional shift. One of the obvious cases is to directly recognize the *Request* or *Request*-suspected dialogue acts in user's utterance (as in the earlier example in Section 4.6.3). The second case occurs when agent's last dialogue act has been a *Request* (e.g., recall "*How may I help you?*") or *Request-yn*. In such cases, user's declarative response should be interpreted as an indirect *Request* act. The following two rules summarize the initiation of the fragmentation process.

Rule 1. If the user currently performed either a *Request* or *Request*-suspected dialogue acts, initiate the fragmentation process.

Rule 2. If the agent lastly performed a *Request* or *Request-yn* dialogue acts, initiate the fragmentation process.

If neither of the two rules fires, user's utterance is assumed to contribute to the data layer only, and the fragmentation is bypassed by setting the task fragment empty and the data fragment equivalent to the input semantics, i.e. $\mathbf{F}_T = \varnothing$ and $\mathbf{F}_D = \mathit{Semantics}$.

In this section, we responded the first of the two questions: *when* to fragment. We also have described ways to minimize the computational expenses of the process. Interested reader is suggested to trace the `UrciOptimalniFragmentaciSemantiky` procedure in `Semantika.pas` to see additional, although merely minor, facilitating constraints put on the fragmentation process.

The second question, *how* to approach the optimal fragmentation, is a complex one and as such needs to be split into two related subproblems: analyzing a given fragment against it respective layer content, and evaluating the result of the analysis. The following two sections cover these subproblems.

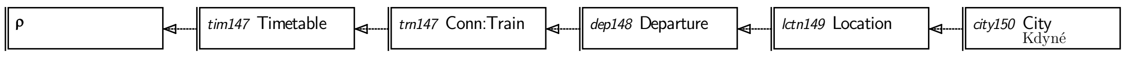
4.6.5.2 Analyzing Fragment Against Its Layer

The analysis can be simply described as matching a given fragment against the layer content in order to *qualify* the fragment from three distinct points of view:

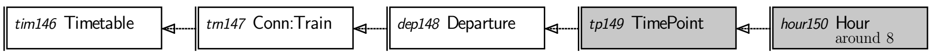
- what is the information that the fragment brings into the layer,
- how can underspefied information be "explained" using the current content of the layer, and
- which existing objects the fragment refers to.

Naturally, there are slight differences in analyzing the data and task fragments (which exceed the most apparent dereferencing, which reasonably cannot be carried out on the task layer). Nonetheless, with not taking these distinctions into account, we will be able to describe the basic analysis more clearly.

Let us start with the first of the three points – spotting new pieces of information conveyed by the fragment. The essence is in fact very similar to the one we saw earlier in Section 4.5.1: starting from the roots of the fragment and the layer, a current concept instance (element of the layer) is tested for existence of a particular subcollection (element of the fragment). The process then recurrently continues up to fragment leaf concepts. However, unlike with the algorithm in Fig. 4.2, no instantiation nor inferrention occur. Intuitively, given the below layer content:



the process finds the following new information in the below fragment (in gray):



The second aspect of the analysis, how can underspefied information be explained, does not refer only to the SIL-like anchoring. In fact, it is a wider conception of which anchoring is merely one special case. The process of information explanation aims to match underspecified information against some known structure. This structure may be either the current content of the layer, or the underlying DDM itself ("basic view" in SIL terms), should the current content fail to explain it completely. For instance, given an empty layer, user's information *City*:"Kdyně" will be explained as a location. The reason is that the empty layer fails to explain it, however, in the underlying DDM from Fig. 4.1, the only parent of C_{city} is $C_{location}$. The remaining parents cannot be resolved at this moment: it is not clear if the location regards a departure or arrival, whether it should concern train or bus, and finally, if the user speaks about a timetable or a ticketing service.

city140 City
Kdyné
→
lctn142 Location
city140 City
Kdyné

City:"Kdyně" *Location*(*City*:"Kdyně")

The final aspect of the analysis is to dereference known objects. In order for the dereferention to account for both current and historical objects (no longer in the data layer but in the dialogue history), these objects are ordered descendant by their salience (a similar approach may be found in [Zah03, Nes09]). Moreover, the framework supports two ways of dereferencing objects, explicit and implicit. We have already encountered the *explicit* dereferention earlier in Section 4.4. To recall, hints for explicit dereferences are indicated with the *_ref_* directive; for instance, the sentence "*I will buy a ticket for the train*" can be represented as *Ticket*(*_ref_*(*Train*)), causing the framework to attempt to match such expression against the list of salient objects.

In contrast, the *implicit* dereferention does not rely on being supplied correctly parsed references. Instead, it assumes that references remained unrecognized or unexpressed (which may be the case in some languages, for instance Czech), and that each object in the semantics is a potential reference. The analysis therefore first attempts for the dereferention, and no earlier than once failed continues with spotting new information (as already described above). To illustrate, the sentence "*I will buy a ticket for the train*" would be now represented as *Ticket*(*Train*). Given that C_{ticket} is a topic, the only relevant candidate for dereferention is the I_{train}. Therefore it will be first subjected the implicit dereferention, and if unsuccessful, declared as new information.

To put all of these three elemental processes into perspective, the following example roughly demonstrates the overall principle of the fragment analysis.

Example 4.7 (Fragment analysis)
Let us suppose the following dialogue:

U_1	I want to go from Brno to Hodonín.
S_1	There is a local express train going from Brno to Hodonín at 7 o'clock, and an intercity train at 9 and 10 o'clock. Can I help you with anything else?
U_2	When do they arrive there?
S_2	The express train arrives in Hodonín at 9 o'clock, the intercity train at 9 arrives at 10, and the one at 10 arrives at 11 o'clock.

After the dialogue, the corresponding data layer content looks like shown in Fig. 4.12. Let us suppose the dialogue continues with:

U_3 I will buy a ticket for the intercity train at ten o'clock.
Ticket(*Price*:— , *Type*:"intercity", *Hour*:"10")

Notice the presence of C_{price} enclosed within C_{ticket} as an explicit indicator of "purchasing something". This instance can be introduced by the parser to

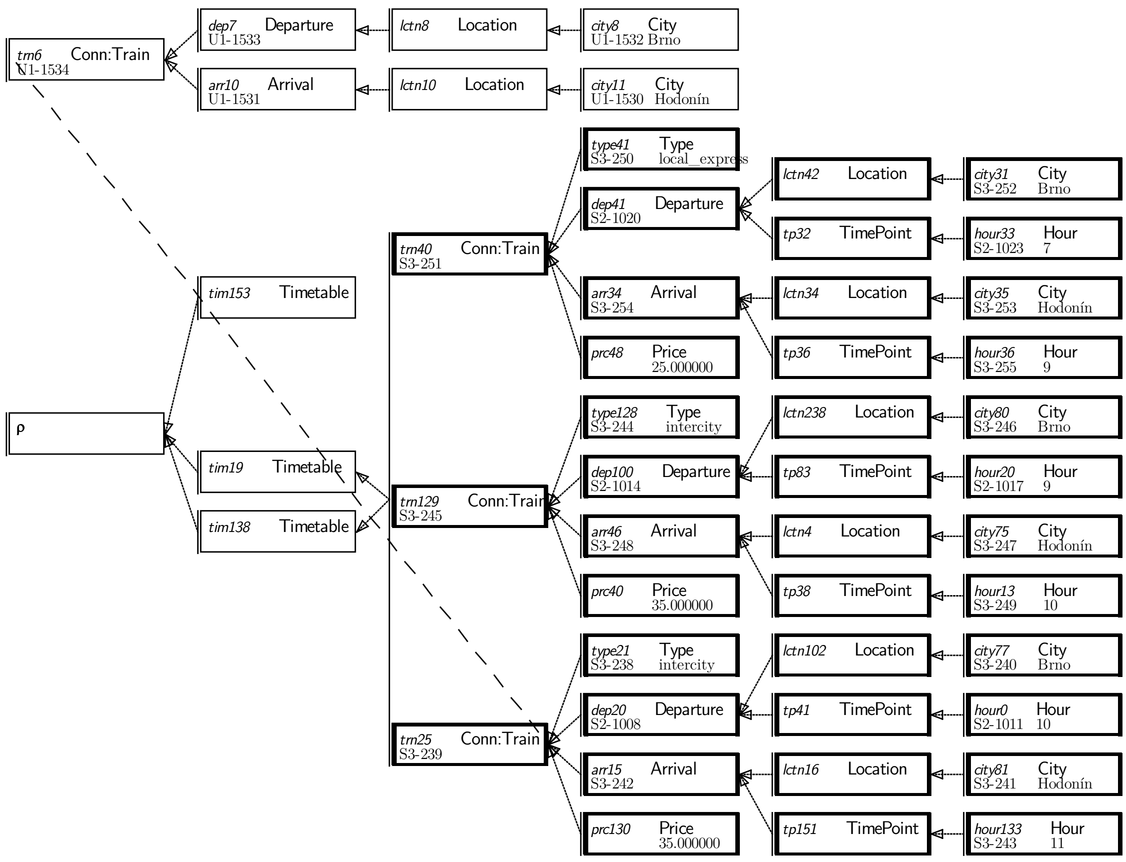


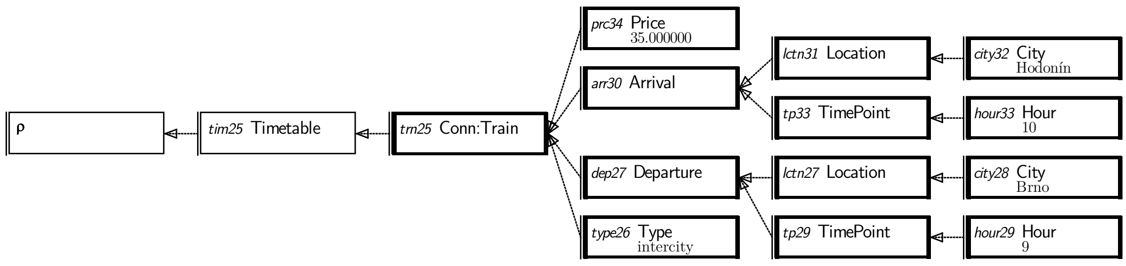
Fig. 4.12 Data layer content after speaking utterance S_2 in Example 4.7; instances annotated with *utterance–salience* pairs.

overcome the lack of indirect queries detection by recognizing the "shape" of the sentence,[8] the so called "task token", see Section 4.4. The C_{price} instance is assigned infinite cardinality, while both the C_{type} and C_{hour} zero cardinality, hence the whole semantics equals the data fragment. Let us show how its analysis is conducted with respect to the data layer in Fig. 4.12 (notice instances are annotated differently than in Fig. 4.5; also, although Brno and Hodonín have been said merely once in S_1, each appearance is individually salient – the reason relates to utterance production, not covered here).

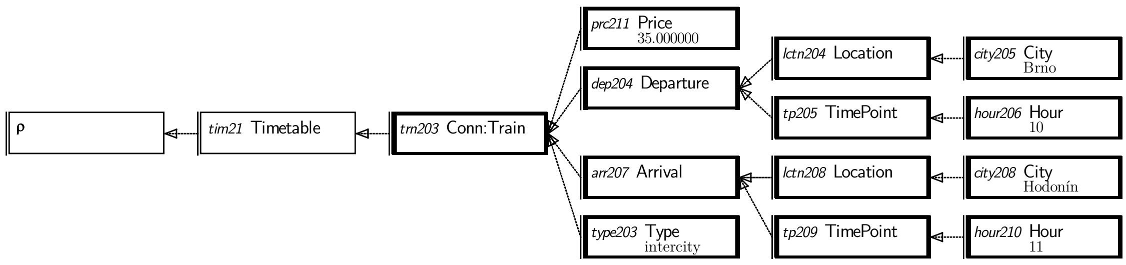
- The analysis begins with the layer and the fragment roots. The attempt to find an instance of C_{ticket} in the layer fails, hence the analysis turns to the underlying DDM (Fig. 4.1) to find that C_{ticket} is directly accessible from within the root. Hence, semantics I_{ticket} is recognized as new information.
- The framework attempts to trigger implicit dereferention using children of I_{ticket}. Given that the dialogue history is empty, only the ordered list of data layer salient objects is considered. In this list, the candidates are *trn40*, *trn129*, and *trn25* as only they can be the immediate children of the ticket instance.

8 As buying a ticket cannot be formulated as a direct interrogative question.

- The first cadidate, *trn40*, is tested. The analysis attempts to match the *Price*:—, *Type*:"intercity", and *Hour*:"10" specifications. The train satisfies the price specification with *prc48*; the attempt to match types fails, however. The analysis is therefore abandoned and *trn40* refused.
- The next candidate, *trn129*, already satisfies the type demand. The C_{hour} specification is explained by traversing instances *arr46* and *tp38*. This train therefore fully matches all requirements, and one of the analysis results, called *unification*, therefore is:



- Finally, the last candidate, *trn25*, satisfies the demands with the C_{hour} specification explained using instances *dep20* and *tp41*. Therefore, the second unification is:



Thus, as it can be seen, the analysis is a non-trivial operation that purposely switches between the three intrinsic processes to meet its objective of analyzing a fragment against a respective layer. Nevertheless, this example demonstrated the analysis merely from a rough top-level point of view. Interested reader is therefore advised to trace the `spodproc` procedure in `Semantika.pas` to see the analysis at work in full detail. Note in this regard, that, due to historical reasons, the three elemental processes are called there `akceEmulace`, `akceDomysleni`, and `akceDereference`, respectively. To be able to interact together, they comprise a great state-based automaton that is the core of the analysis, with additional supportive routines called on its behalf. □

4.6.5.3 Evaluating Analysis Unification Pairs

At this moment, we suppose that the input semantics has been split, resulting in the task and data fragments, and both of the fragments have been analyzed against their respective layers. We call the task and data fragments a *pair*. Each pair must now be *evaluated* with respect to the current state of the dialogue context. In other words, it is necessary to *quantify* the match with the layer content from three distinct points of view (compare with points in the previous section):

- how many new pieces of information are brought in by the fragment,
- how well can underspefied information be "explained" using the current content of the respective layer, and
- how well are existing objects referred by the fragment.

The evaluation conceives of a set of rules concerning with different context situations. Each rule is to *penalize* a fragment if it does not fit the particular situation. The final sum of the penalties, denoted as P, then indicates how well the pair fits the layers (for instance, how well system expectation is met, discussed later). The pair that yields the lowest compound penalty, P^*, is considered optimal and integrated into the layers using the algorithm presented in Section 4.5.

The cornerstone in the rules is salience. Let us recall that we defined it as a number with the following semantics: the *higher* the number the *older* the information, and vice versa. The rules involved in the evaluation are as follows:

Rule 3. (extending the set of Rules 1 and 2 from Section 4.6.5.1) describes the most obvious situation – a user referring to an object. We want to address the most salient object that matches user's description, therefore we add each object's salience to the penalty sum (recall that the higher the salience, the lower the penalty). **Formally**: Let there be a path from root ρ to leaf information L in a *Fragment* (to spare space, we will abbreviate as $\langle \rho \leftarrow L \rangle \in$ *Fragment*) that is completely unifiable[9] with the layer content. Then for each object on the path add its salience to the total penalty P.

Rule 4. Describes a situation in which the user introduces new information (e.g., when no object matches user's reference). In this case, we add the minimal penalty for the user changing the layer content. **Formally**: Let $\langle \rho \leftarrow L \rangle \in$ *Fragment*. Let $\langle \rho \leftarrow E \rangle \subset \langle \rho \leftarrow L \rangle$ be the maximum length subpath unifiable with the layer content (we say $\langle \rho \leftarrow L \rangle$ is *partially unifiable*). Then for each object whose distance is greater than E add minimal penalty P_m to P. (This rule can be considered a special case of Rule 3.)

Rule 5. Dictates that an addressed object must fully match a given reference, otherwise it cannot be considered resolving it. **Formally**: Let $\langle \rho \leftarrow L \rangle \in$ *Fragment* be completely unifiable with a layer. Let $E \in \langle \rho \leftarrow L \rangle$ be an object for which Rule 4 applies. Then for each object on the path add its salience to P.

Rule 6. Demands objects to be maximally described by the semantics (e.g., it is wrong to not consider all information from the semantics that matches an addressed object during reference resolving). **Formally**:

9 Object X is said to be *unifiable* with object Y if parents of X are subset of parents of Y and one of the following holds (Prolog-like unification): (1) values of both objects are equal, or (2) at least one of the objects has undefined value.

Let $\langle \rho \leftarrow L \rangle \notin \mathit{Fragment}$ be completely unifiable with a layer content. Then for each object on the path add twice its salience to P.

Rule 7. Requires objects that the user disagrees with to exist. **Formally**: Let $\langle \rho \leftarrow L \rangle \in \mathit{Fragment}$ be partially unifiable with a layer content. Let $E \in \langle \rho \leftarrow L \rangle$ be an object marked as disagreed. Then for each object on the path add thrice its salience to P.

Rule 8. Defines that infinite cardinality objects are more "valuable" for task detection than non-zero cardinality objects. **Formally**: If the task fragment contains at least one leaf with infinite cardinality, then all paths from the fragment root to leaves with non-zero cardinality must be unifiable with the task layer content, otherwise assign P infinite penalty.

Rule 9. Forbids information that most probably regards task detection to be integrated into the data layer. **Formally**: In a data fragment, all paths from the root to leaves with infinite cardinality must be unifiable with the data layer content, otherwise assign P infinite penalty.

Rule 10. Forces the task fragment to always exist if the semantics content indicates a possible intentional shift. **Formally**: Let the semantics contain a non-zero or infinite cardinality piece of information. If the task fragment is empty, assign P infinite penalty.

Rule 11. Favours objects currently in the system focus over those that are not; that is, this rule accounts for an implicit arbitration for cases in which analysis of the semantics would be ambiguous. **Formally**: Let $\langle \rho \leftarrow L \rangle \in \mathit{Fragment}$. Let $\langle \rho \leftarrow E \rangle \subset \langle \rho \leftarrow L \rangle$ be the maximum length subpath unifiable with system focus, $\langle \rho \leftarrow F \rangle$. Then for each object with distance greater than E add maximum penalty P_M to P.

Rule 12. Favours objects either expected by the agent (e.g., required to solve a task) or used by the agent (e.g., in some of planned steps) over objects that are useless in the scope of the given task. This supports Grosz and Sidner's term satisfaction-precedence (covered later). **Formally**: Let $\langle \rho \leftarrow L \rangle \in \mathit{Fragment}$ be not completely unifiable with any system expectation $\langle \rho \leftarrow X_i \rangle$. Then for each object on the path add maximum object penalty P_M to P.

With having mentioned several different penalty increments in the above rules (constant or salience-based), let us put them in their mutual relationships to symbolically indicate their influential character on the compound penalty P:

$$0 \leq P_m \leq S(\mathit{object}) \leq P_M < \infty \, .$$

Recall that the rules aim to describe common situations that may arise over the course of a dialogue. The benefit of encoding these situations into rules is the scalability of the approach. Nonetheless, the downside is the "proper" assignment of corresponding penalties. It requires a decent amount of effort to equilibrate the whole system of rules to evaluate the fragmentation pairs as desired. A good clue for assigning reasonable penalties may be to prioritize the situations that the rules express. Not surprisingly, the multipliers of penalties in Rules 6 and 7 intend to approach such priorities (yielding Rule 6 is preferred over Rule 7 by its penalty being less multiplied). The similar notion then also holds for the two constant penalties, P_m and P_M, used in Rules 4, 11, and 12. Finally, to overcome the issue of the penalization being dependent on the current path number (the more paths there are in a fragment the higher penalty would be gathered), the sum of penalties P is divided by this number, producing the "normalized" pair penalty P_{norm}. Interested reader may further trace the `ohodnotFragment` function in `Semantika.pas`, returning the normalized pair penalty P_{norm}.

4.6.6 A Complete Example

Before moving on, let us demonstrate the working cycle of our dialogue context approach in its entirety. Ponder the following dialogue snippet:

S_1 How may I help you?

U_1 I need to go to Ostrava.

```
_declarative_(
    Timetable(
        Departure( Hour:— ) ,
        Arrival( City:"Ostrava" )
    )
)
```

S_2 Which transportation means would you prefer?

U_2 A cheap one, please

```
_declarative_( Criterion:"cheap" )
```

S_3 I understood Ostrava. I can offer you a train, bus, or airplane connection. Which transportation means would you prefer?

U_3 A cheap one, so bus or train?

```
_interrogative_( Criterion:"cheap" , Bus , Train )
```

S_4 What time approximately would you like to depart?

U_4 About eleven.

```
_declarative_( Hour:"about 11" )
```

S_5 I understood you want from Brno to Ostrava by bus or train, departing at eleven. Please confirm this by saying Yes, or say your correct request.

U_5 Yes.

```
_declarative_( _agree_ )
```

S_6 The next cheapest bus from Brno to Ostrava leaves at 15:30, train at 13 o'clock, and bus at 13:30. Do you want to order a ticket now?

U_6 What are the prices?

```
_interrogative_( Timetable( Price:— ) )
```

S_7 The ticket for the bus departing at 15:30 costs 30 coins, train 35 coins, and bus at 13:30 40 coins. Can I help you any further?

U_7 When does the train arrive?

```
_interrogative_( Train( Arrival( TimePoint( Hour:— ) ) ) )
```

S_8 The train arrives in Ostrava at 18 o'clock. Do you want to order a ticket now?

U_8 Yes, please.

declarative(_agree_)

S_9 Ticketing service. …

As usually, let us suppose that both of the layers are empty at the beginning. The system performs a *Request* act in S_1, responded to by the user with a declaration of wanting to get to Ostrava. By speaking U_1, the user performs a *Request*-suspected dialogue act. The utterance semantics itself consists of an undefined *Time* object (to indicate the queried information) and an atomic *City* object. Rule 2 forces semantics to be split. Rule 10 then requires the task fragment to exist (as it contains an undefined value object). The task fragment therefore consists of the $\langle \rho \leftarrow \mathit{Time} \rangle$ path, as the atomic *City* object cannot update the task layer. The data fragment consists of the $\langle \rho \leftarrow \mathit{City} \rangle$ path only, as, according to Rule 9, the undefined *Time* object cannot be introduced into the data layer. This splitting also equals the optimal fragmentation. Given that the optimal task fragment is not empty, implying an intentional shift has been made by the user, the *Request*-suspected dialogue act is further narrowed as a regular *Request* act. Thus, after incorporating U_1 into the dialogue context, the task and data layers contain the $\langle \rho \leftarrow \mathit{Time} \rangle$ and $\langle \rho \leftarrow \mathit{City} \rangle$ information, respectively.

Utterance U_2 is anchored into the data layer only as there is only a single object $\langle \rho \leftarrow \mathit{Criterion} \rangle$ with zero cardinality recognized, causing neither of Rules 1 and 2 be triggered. In U_3, the interrogative type of the utterance triggers Rule 1. The task fragment then consists of both of the transportation means only (our heuristics prevents the atomic *Criterion* be part of the task fragment). However, the data fragment has two possibilities: (1) complete semantics with penalty $P = 2 \cdot P_m + 2 \cdot S(\mathit{Criterion})$ (because Rule 4 yields minimal penalty P_m for I_{bus} and I_{train}, and Rule 3 yields the penalty of the $\langle \rho \leftarrow \mathit{Criterion} \rangle$ path salience), or (2) $\langle \rho \leftarrow \mathit{Criterion} \rangle$ object only with penalty $P = 2 \cdot P_m + 4 \cdot S(\mathit{Criterion})$ (as Rule 4 yields minimal penalty P_m for for I_{bus} and I_{train}, and Rule 6 twice penalizes $\langle \rho \leftarrow \mathit{Criterion} \rangle$ for being not part of the fragment). The latter option does not beat the former one, hence the non-atomic I_{bus} and I_{train} objects are contained in both of the fragments. The following utterance U_4 is of a trivial nature similarly as U_2, i.e. no fragmentation is required and the semantics updates the data layer only. Rule 11 interprets the information as a departure time due to the system being focused on a departure in S_4. Utterance U_5 does not contain any instantiable objects, i.e. again no fragmentation is necessary. Once the system has said S_6, the data layer looks like in Fig. 4.13.

Utterances U_6 and U_7 are spoken under similar dialogue conditions (the agent performs a *Request* act and the user responds with an interrogation), hence let us proceed to U_7. Rule 1 makes U_7 a subject of the fragmentation process. The task fragment is created by Rule 10 and consists of the $\langle \rho \leftarrow \mathit{Time} \rangle$ path. However, there are two options for the data fragment: (1) $\langle \rho \leftarrow \mathit{Time} \rangle$ which is treated by Rule 3 as a reference to the *trn130* object introduced by the agent in S_6 (and further used in S_7); this unification gains the penalty $P = S(\mathit{Train})$; (2) empty which is penalized by Rule 6 (I_{train} object could be used to resolve a reference, however, the unification does not account for it), and thus gains the penalty $P = 2 \cdot S(\mathit{Train})$. For completeness sake, let us note that the agent passes the referred

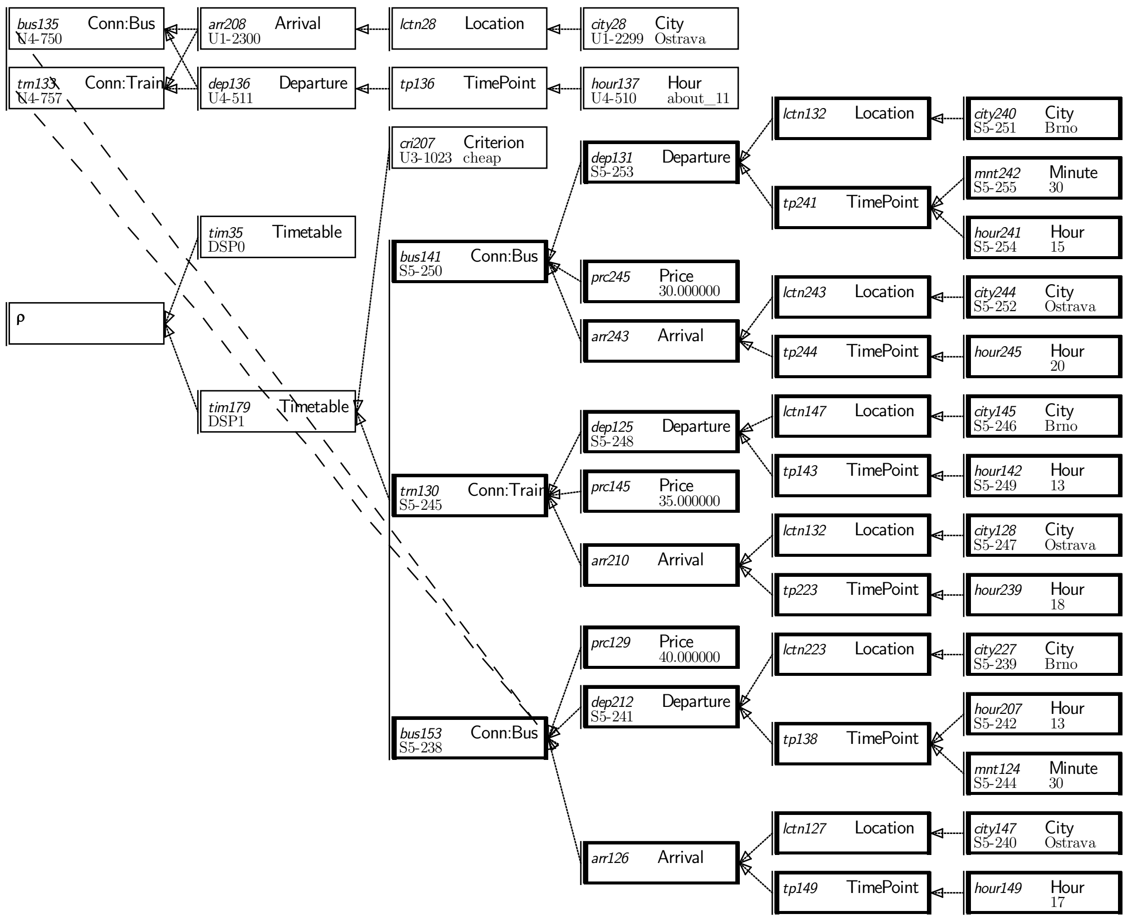


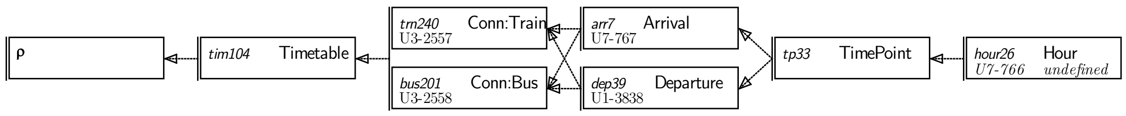
Fig. 4.13 Data layer content after S_6 has been uttered.

trn130 over to the “arrival time query” submissive for further handling. At the moment of uttering S_8, the layers looks like in . The rest of the dialogue is then processed analogously.

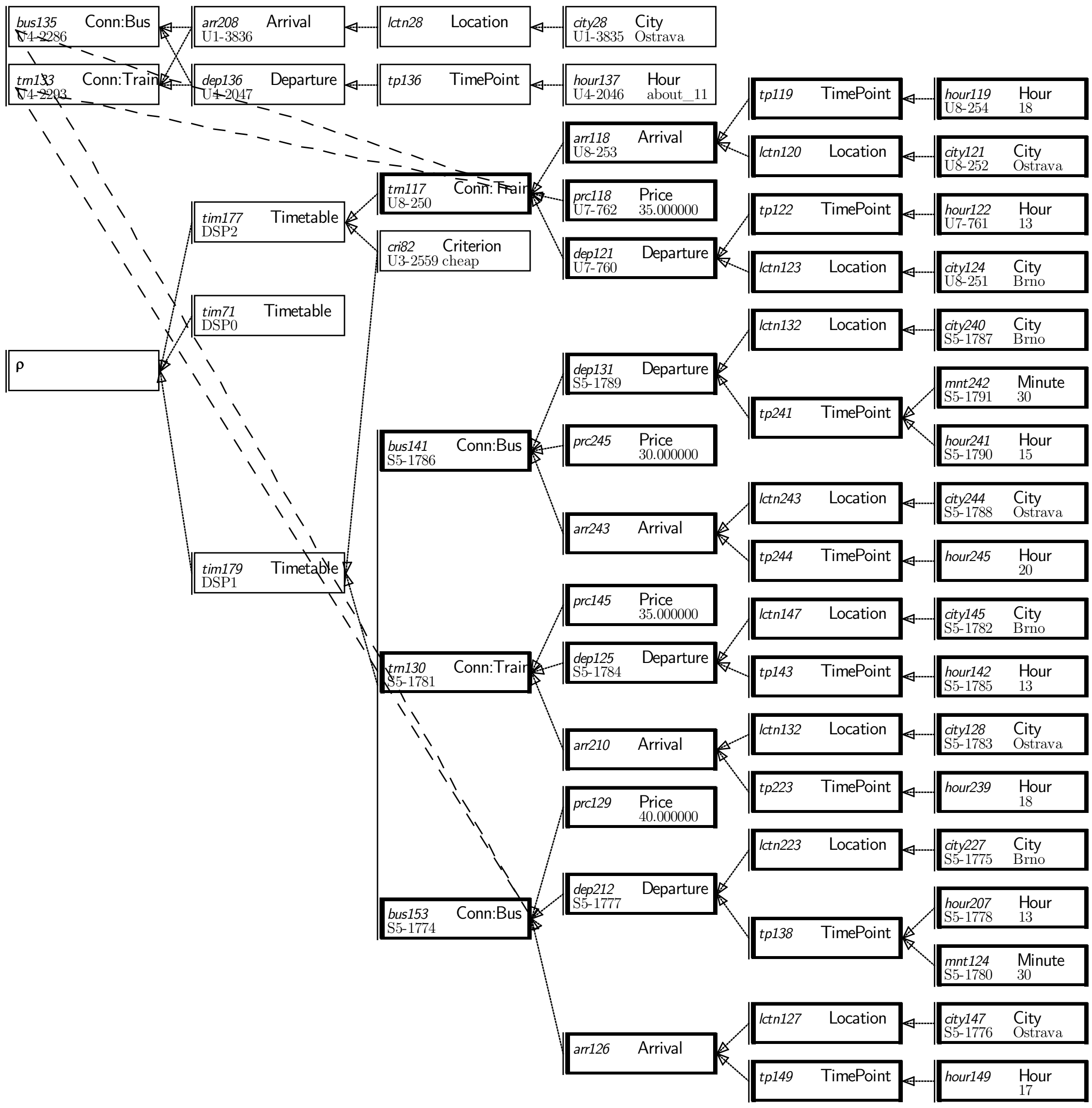
4.7 Task Recognition and Dialogue Stack Management

So far, we intuitively recognized tasks in user's utterances. For instance, we were able to find in "*When does the train arrive?*" a reference to the "*arrival time request*" task that should operate with a train object, known to the user. To maintain a coherent dialogue with the user, the agent needs to perform the recognition too. However, not all user's utterances are *Request* acts – some are to clarify ambiguities or provide more information.

In task-oriented systems, the recognition is usually approached using a simple solution. For instance, in [Eck95], the task is recognized from the overall structure of the utterance by passing the duty over to the parser that eventually produces a token to identify the task (e.g., the tripple $(request, sourcetime, wh\_asked)$

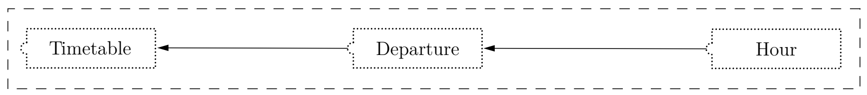


(a) Task layer content after uttering S_8

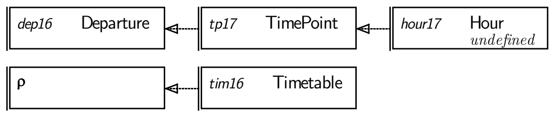


(b) Task layer content after uttering S_8

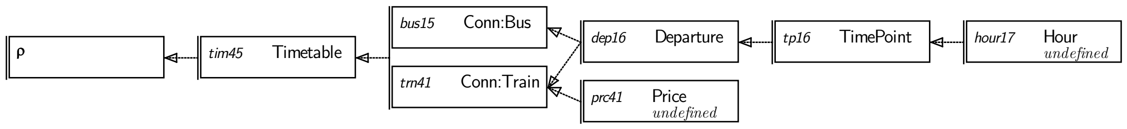
Fig. 4.14 Dialogue context after uttering S_8.



(a) Task recognition template



(b) Task layer that contains the recognition template



(c) More complex task layer that contains the recognition template

Fig. 4.15 "*Departure time request*" template with example task layers that contain it.

indicates a request on departure times). A similar approach is also adopted in [Boh09, Tur05]. Contrarily, in [Ngu06b], tasks are recognized merely on the basis of keywords. This limitation is justified by the interactively complex nature of the speech application and the intended aim "to reduce the effects of speech recognition errors and to avoid restricting the user's vocabulary".

The task recognition is practically identical to the identification of discourse segment boundaries – we need to recognize its beginning and its end. Recall that we identify two essential components in the dialogue, tasks and task-related information, and store them within two distinct layers. Each of these layers is updated by incorporating a corresponding fragment of the input semantics. The initial problem, when the user initiates a new task, can be easily resolved by identifying the user's dialogue act as a *Request*. As known from Section 4.6.5, this guarantees a non-empty task fragment, describing the shift of actions.

The second problem, how to recognize the newly initiated task, can be solved by making use of a plain template matching. Obviously, given that the task layer aggregates all relevant information, this is a reasonable way – a general, robust, and straightforward one, taking advantage of the overall nature of the two-layered information model to recognize tasks that may span multiple user's utterances. For instance, Fig. 4.15a schematically shows a template to recognize a "*departure time request*" task (dashed around); it misses a K_{conn} definition, as the particular transportation means is irrelevant for the proper identification of the task, as well is the presence of K_{time_point} (according to Fig. 4.1, K_{hour} can only be parented by K_{time_point}, hence there is no ambiguity). The rest of Fig. 4.15 then shows two sample task layers that contain this template (of course, Fig. 4.14a contains it too).

Templates comprise a set in which each two patterns are mutually non-interchangeable (although not necessarily disjunctive) in order for the tasks to be uniquely identifiable. As described above, the recognition is triggered once user's *Request* act has been observed. It accounts for trying each pattern in the set if it entirely matches against the task layer content. If it does, its score of match is computed as the sum of saliences of object involved. Apparently, as the task layer accommodates all task-related information in the dialogue, more than one pattern may match. In such situation, the template with the highest score is received as user's current intention. Apart of that, provided that the score of match may be used as a metric of actuality, the scoring plays another important role – it implicitly organizes the dialogue. This way, the most recent task with the highest score is pushed onto the top of the stack, whereas the most dominant task (usually the initial "How-may-I-help-you" task) always occupies the bottom.

However, before pushing any new task T onto the stack, and thus declaring the beginning of a new discourse segment, we check if the currently topped task T_{Top} dominates it,

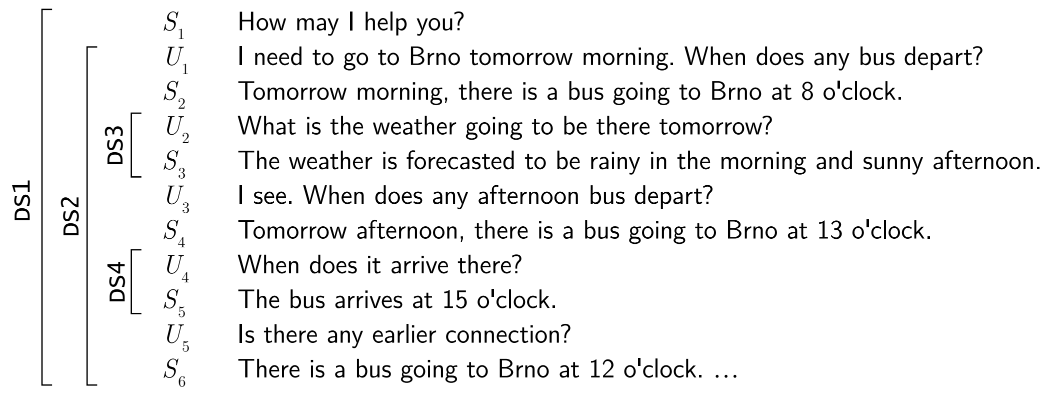
$$(\; T_{Top} \text{ DOM } T \;) \quad \rightarrow \quad \text{push } (\; T) \; .$$

From a purely technical standpoint, note that our conception of the dominance relationship is influenced by the inner organization of the framework. Therefore, the dominance relationship is realized as "T can be performed within the context of T_{Top}". In other words, unlike with Grosz and Sidner's work, T is not a priori supposed to support the solution of T_{Top}. This implies from the fact that both of the tasks need to be self-contained entities in order for the agent to deliberate on them the optimal way. Nevertheless, the *Domain Editor* (see attached CD) even so allows for a decomposition of a task into particular subproblems using the conception of generic macros. During the "compilation procedure" of the domain, the Domain Editor, among other things, expands these macros at places where they are referred to produce a self-contained task solution plan. We will revisit this essence once again in the next section on agent's deliberation. However, despite the slight deviation in meaning, we will stick to Grosz and Sidner's terms.

If the domination relationship is not met (i.e. "T does not support the solution of T_{Top}", or in our terms, "T can *not* be performed within the context of T_{Top}"), it indicates a permanent change in user's intentional focus, and thus the end of a discourse segment. The result is the top-positioned intention be popped out of the stack and the domination relationship re-tested,

$$(\; T_{Top} \neg\text{DOM } T \;) \quad \rightarrow \quad \text{pop } (\; T_{Top}) \;\; \wedge \;\; \text{re-test dominance} \; .$$

The intentional shift is yet detected in two additional, less explicit ways, both of which also express clues for the detection of the end of a discourse segment. First, the user provides information that contributes to the satisfaction of a dominant task. Intuitively, this rule is well applicable in a multi-domain system and indicates the return to a previously discussed topic. Obviously, it has the limitation of no two tasks be allowed to share the same topic concept. The second rule therefore fills in the gap by detecting the return to a dominant task by observing that user's optimal data fragment suggests to override system information in the data layer (see Section 4.5.2). To illustrate both of the situations, ponder the dialogue snippet in Fig. 4.16. The task in DS2 is intentionally unoptimized to demonstrate the applicability of the above two detection rules (the preferred way would be to inform the user about the arrival without her or him having to ask explicitly). As it can be seen, both DS3 and



S_1 How may I help you?
U_1 I need to go to Brno tomorrow morning. When does any bus depart?
S_2 Tomorrow morning, there is a bus going to Brno at 8 o'clock.
U_2 What is the weather going to be there tomorrow?
S_3 The weather is forecasted to be rainy in the morning and sunny afternoon.
U_3 I see. When does any afternoon bus depart?
S_4 Tomorrow afternoon, there is a bus going to Brno at 13 o'clock.
U_4 When does it arrive there?
S_5 The bus arrives at 15 o'clock.
U_5 Is there any earlier connection?
S_6 There is a bus going to Brno at 12 o'clock. ...

Fig. 4.16 Segmented multi-domain dialogue example.

DS4 are dominated by DS2, implying their tasks can be requested in the context of DS2. In U_3, the user returns to DS2 by changing the departure time of the bus to afternoon. Obviously, the $C_{departure}$ and C_{bus} concepts do not contribute to the task in DS3 on weather, but support the solution of the task in DS2 on departure times. In U_4, the user initiates DS4 into which the bus object used by the system in S_4 is passed. In U_5, the C_{hour} concept behind the value "earlier" can contribute to both DS4 and DS2 – the user either refuses the arrival at 15 o'clock proposed by the system, or intends to change the departure from "*afternoon*" to "*around noon*". In either case, the user is overriding the system (see the $U(.)$ function in Definition 4.10) and the control therefore needs to be returned to the discourse segment in which the system has overridden the user, which is DS2.

To summarize our approach, we recognize the beginning of a new discourse segment by matching a set of templates against the content of the task layer. We recognize the end of a discourse segment as either (1) the new segment not dominating the current one, (2) user's information contributing to a dominant segment, (3) user overriding system information, or (4) user not re-opening a segment considered by the system closed (this case has not been discussed here, however is adopted from [Ric01]). Apparently, the template-based approach is more general than the approaches overviewed at the beginning of this section by, first, offloading the parser the duty to recognize a task from the shape of an utterance, and second, allowing a task identification to be evolved over time.

4.8 Dialogue Planning

Once the task has been recognized, a related plan is adopted by the agent to get the task solved. A plan is commonly represented as a structure resembling a tree [Boh09, Ric01, Jok10, Bui06]. Such representation usually accounts for the real activities be stored in leaf nodes (e.g., utterances to say or back-end interaction to carry out), while the remaining non-leaf nodes capture the relationships among particular activities. The decomposition sometimes also accounts for one extra axis, namely dynamic building of plans using a library of subplans, each solving an elemental problem [Ing92]. The natural advantage is the scalability.

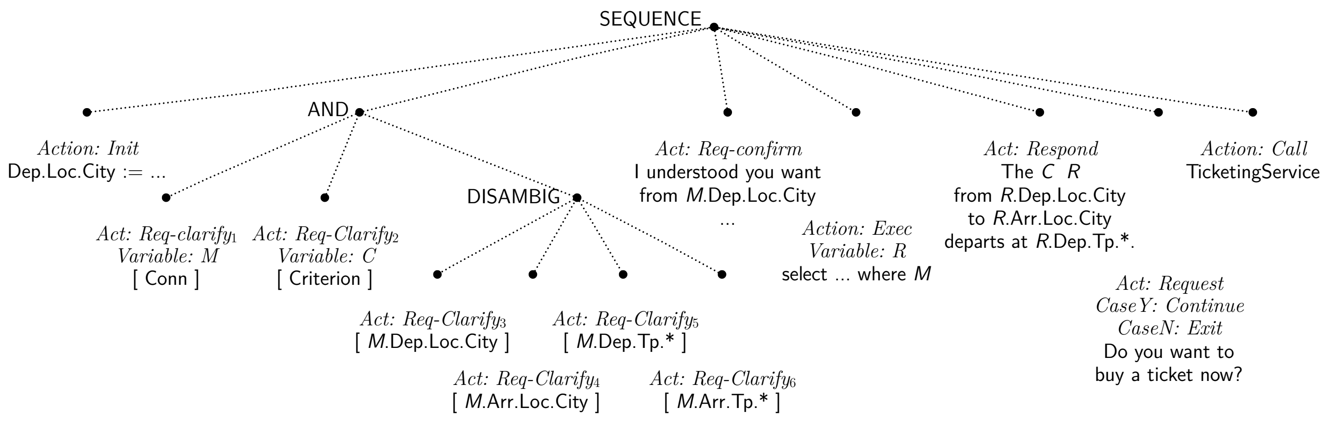
4.8.1 The Role of User's Initiative

In our case, plans follow the tree paradigm as well. Nevertheless, we currently do not consider a library of subplans. Instead, we constrain the design to a one-to-one relationship between a task and a plan. In other words, each task has a single accompanying static plan in which activities are spread over leaves, as suggested above. Although this may be argued restrictive, the variability of the single plan is guaranteed by, first, the possibility to create alternative solution branches for a given problem, and second, by the variability of utterances that may be parametrized with different predefined conditions. Thus, the tight coupling between a task and its plan is merely a minor constraint that may be easily overcome by a proper design of the solution (branches) and interactions (back-end communication and foremost utterances).

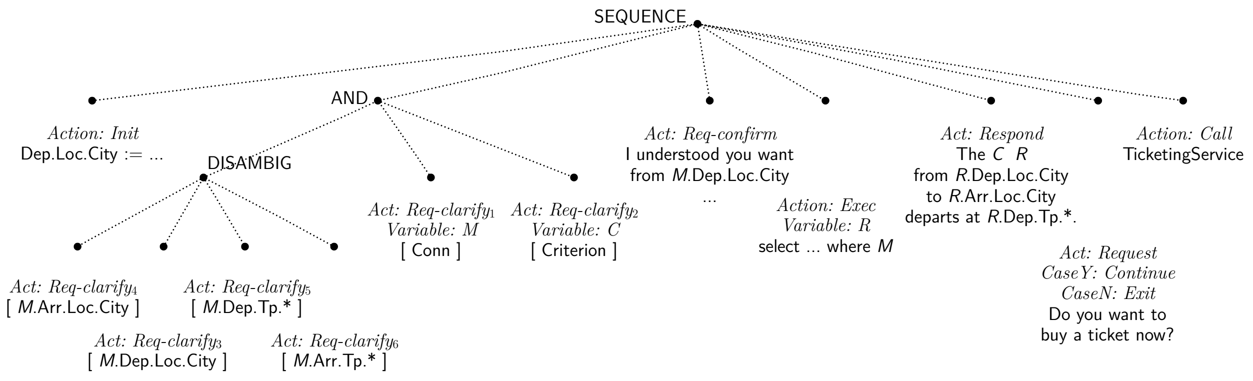
An example of a plan that solves the "*departure time request*" task may be seen in Fig. 4.17a. Obviously, this plan is merely a demonstrative plot and is not intended to provide a fully functional solution. Nonetheless, we will use it throughout this section to demonstrate different aspects of agent's dialogue planning capabilities. As it can be seen in the plan, first a background interaction is carried out to establish the default departure city at the time of plan triggering, if not yet specified by the user (action node labeled *Init*). Then, the user is asked to specify the transportation means to find the departure time for ($\mathit{Req\text{-}clarify}_1$) and the criterion to apply during the search ($\mathit{Req\text{-}clarify}_2$). The results of her or his answers are bound to the respective variables M and C to facilitate further addressing in the data layer. Next, if there are more transportation means allowed (three, $E(C_{timetable}, 3)$ in Fig. 4.1), their parameters are constrained by posing (some of) the disambiguation questions ($\mathit{Req\text{-}clarify}_{3\ldots6}$). Finally, gathered data may be subjected to confirmation, depending on the current strategy (*Req-confirm*), database queried (*Exec*), and results presented to the user (*Respond*). No earlier than now, the agent considers user's intention be satisfied.

The plan in Fig. 4.17a is rather a simple one, however, it is sufficient enough to demonstrate the first level of agent's adaptability. It accounts for simply swapping plan tree branches according to the data layer object saliences. In other words, the agent adopts the results of user's initiative – if she or he prefers to discuss certain part of the task prior to discussing the rest, the agent adopts the decision. For example, consider user's elliptical utterance "*by train to Pardubice*" has been misrecognized by not understanding the transportation means. The city of arrival (Pardubice) is now assigned the highest salience in the data layer. From the agent's point of view, the user wants to first discuss the city of arrival ($\mathit{Req\text{-}clarify}_4$) and then continue with the rest of the task. The agent therefore adjusts the plan tree structure by moving the corresponding branch towards the beginning (Fig. 4.17b).[10] However, this time the mandatory M variable remains unbound due to the misrecognition error. The agent is therefore forced to traverse through the tree structure, searching for how a value can be reached. Once having found the $\mathit{Req\text{-}clarify}_1$ node, it puts the corresponding branch to the beginning of the plan again (Fig. 4.17c).

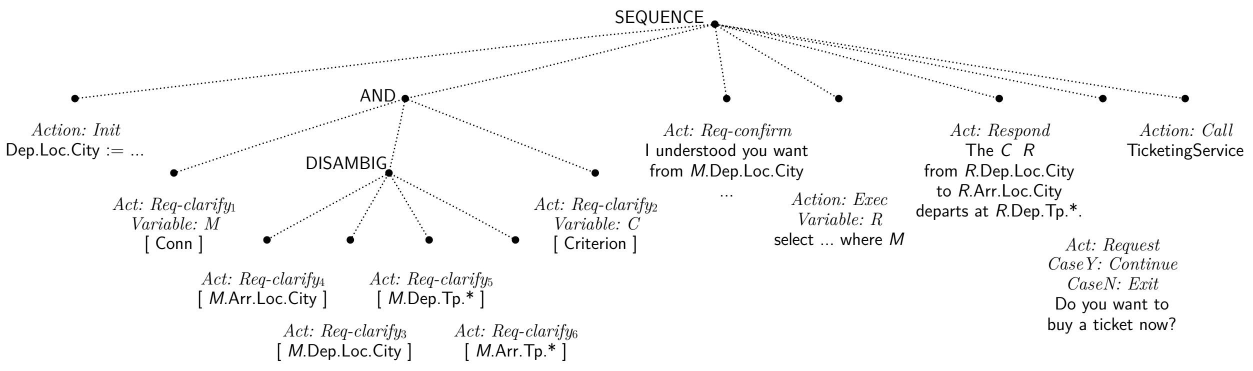
10 Naturally, parent nodes are respected; for instance, it is forbidden to change the order of nodes within a *Sequence* parent node.



(a) initial order of actions to solve the "*departure time request*" task



(b) agent's adaptation to user's initiative; the *Request-clarify*$_4$ act is moved to the beginning of the plan



(c) agent reorganizes the plan back to a feasible state

Fig. 4.17 Evolution of the "*departure time query*" task plan; abbreviations used: Dep = Departure, Loc = Location, Arr = Arrival, Tp = TimePoint.

Table 4.8 Daisy framework plan node types; asterisk denotes acts to which the *Rogerian-psychologist* may also apply if corresponding strategy is allowed.

Type (abbr.)	Dialogue act	Description
AND	—	Conjunction of subactions, order is variant
CAL	—	Subplan or macro call
DIA	—	Groups disambiguation actions (constraints+relaxation)
EXE	—	External function call; a way in for system semantics
IFE	—	Branching if-else; condition queries existence of objects
INI	—	Task initialization
QUE	*Request-clarify**	Missing information; can be skipped
REQ	*Request*	Subtask elicitation; may be answered with yes-no
SEL	*Request-clarify**	Selection of objects proposed by the system
SEQ	—	Conjunction of subactions, order is fixed
STA	*Respond*	Statement; can be marked satisfying the intention
TES	—	Forced termination of the current session
VAL	*Request-confirm*	Information confirmation
YNQ	*Request-yn*	Yes-no question

In the above example, the mutual relationship between the city of arrival in *Req-clarify*$_4$ and the transportation means in *Req-clarify*$_1$ is obvious: one informatively contributes to the other. Apparently, such behaviour is in coherence with Grosz and Sidner's term *satisfaction-precedence* [Gro86]. The distinction is that the two pieces of information are not discussed in separate discourse segments, but instead, are part of a single segment.[11] Similar structuring of information within segments may be found in [Ngu06b, Boh09]. The overall principle of the agent adopting user's initiative in the dialogue is then adopted from [Boh09, Boh07].

Finally, Table 4.8 overviews plan node types available for task modeling in the framework, along with corresponding dialogue acts, if any. In this section example, we used the terms *Init*, *Req-clarify*, *Req-confirm*, *Exec*, and *Respond* to intuitively identify the *INI*, *QUE*, *VAL*, *EXE*, and *STA* node types, respectively.

4.8.2 Deliberation

One of agent's key characteristics is autonomous deliberation about the perceived surrounding environment in order to meet its objectives [Woo00, Zbo04]. In case of a conversational agent, this (among other things) means to simultaneously optimize the dialogue flow in a certain way to successfully satisfy the current task in focus. A common approach to this is to employ a greedy algorithm [Ngu06b, Eck95, Tur05, Boh09]. Depending on the overall design of the conversational agent, this algorithm can successfully draw a task to a satisfaction state in no

11 Which can, however, be just a matter of segmentation.

more than $O(n^2)$ explorations, where n is the maximum number of possible interactions in any task at any time point. Despite its quickness and easy applicability, the downside is its local manner with respect to optimization.

Before explaining our approach to the dialogue optimization, let us first briefly recall particular entities that describe the mental state of the agent:

- *task and data layers* – represent agent's fundamental beliefs in information exchanged during a dialogue,
- *task instances* – recognized in the task layer, they represent agent's desires to take part in solving a given problem with a user,
- *plans* – do **not** serve as the source of agent's intentions; they are merely to represent recipes for joint activity with the user.

Hence, as it can be seen, these entities serve as *heterogeneous* containers of different kinds of information. To bridge these containers, we have adopted and further evolved a system of *events* (McGlashan uses a similar approach [McG96], however, applies event-like principles to "dialogue data" only). In our case, events of different types represent elemental *intentions* the agent can have. We distinguish among the following ones:

- *generalization event* (further denoted as *GEN*) – triggered if an object misses its parent (i.e. the parent could not have been resolved during the fragmentation process, see Section 4.6); this event may relate to any DDM collection instance in either of the layers,
- *confirmation event* (*CON*) – an object has been recognized with a confidence score lower than a custom threshold and might need an explicit confirmation; this event may relate to any DDM collection instance, DDM edge instance, or task instance,
- *value-missing event* (*DEF*) – an object has undefined value; this event relates to DDM collection instances in the data layer only,
- *plan-interpretation event* (*PLN*) – the current plan contains leaf nodes that can be processed by the plan interpreter; this value applies to plan instances only.

As events relate to specific *entities* (collection instance, task instance, etc.), they may be understood as an abstraction of contents in the heterogeneous containers. Each event represents one option the agent can take at a given point in a dialogue. Hence, event allow the agent to make decisions when planning its behaviour. For instance, consider the simple confirmation scheme in Fig. 4.18. There are three pending confirmation events, $CON_{1...3}$, each proposing a different way to interact with the user in order to confirm the time information.[12] The agent may therefore want to first confirm the *hour110* instance by handling the CON_2 event ("*I understood you said 16 hours. Is that correct?*"), and

12 The particular utterances are an embodied part of the DDM definition. For simplicity reasons, this feature has not been discussed in Section 4.2. In the Domain Editor (see attached CD), collection-related confirmation utterances may be defined by clicking the *Data Model* → *Show* menu option.

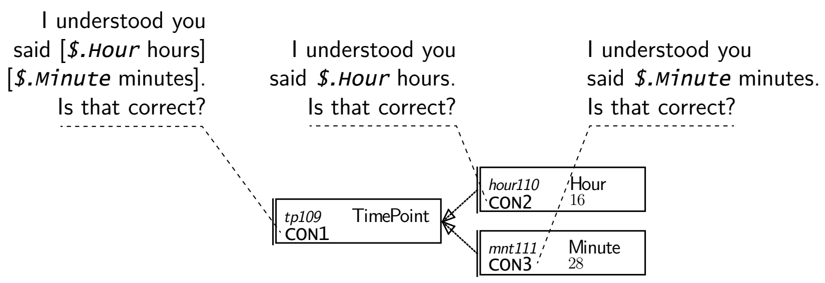

I understood you said [$.Hour hours] [$.Minute minutes]. Is that correct?
I understood you said $.Hour hours. Is that correct?
I understood you said $.Minute minutes. Is that correct?
tp109
TimePoint
CON1
hour110
Hour
16
CON2
mnt111
Minute
28
CON3

Fig. 4.18 Confirmation scheme to validate the time point 16:28.

then confirm the *min111* instance by handling the CON_3 event ("*I understood you said 28 minutes. Is that correct?*"). Once confirmed, the CON_1 event becomes satisfied as well, as it no longer refers to any unconfirmed information. However, the agent may also want to begin with confirming the *tp109* instance by handling the CON_1 event ("*I understood you said 16 hours 28 minutes. Is that correct?*"). If successful, both the CON_2 and CON_3 events would be satisfied as well. Which one of these two plotted courses will be preferred depends on the deliberation mechanism.

An event may be at different *phases* of processing. For most of the events, the agent must (1) utter to the user (*initial phase*), (2) wait for the user's response (*expectation phase*), and finally (3) check the event satisfaction (*satisfaction phase*). All events follow this cascade model. Additionally, for each event we track its recovery state. An event is said to be *recovered* if it reached the satisfaction phase but user's interaction has turned it back to the initial phase (by making changes to the dialogue context). With this final parametrization, we can formally describe events as follows.

Definition 4.13 (Event)
An event is an ordered quadruple $V = (Entity, Intention, Phase, Recovered)$, where *Entity* is an instance (DDM, task, or plan), $Intention \in \{GEN, CON, DEF, PLN\}$, $Phase \in \{initial, expectation, satisfaction\}$, and $Recovered \in \{no, yes\}$. □

Events constitute a foundation for agent's deliberation, constrained to the current task. This is not a limiting characteristic, provided that the task plan must be a self-contained entity. This way, Grosz and Sidner's *satisfaction precedence* is easily accounted for by optimizing plans based on their final outcomes, cutting off unrelated (unnecessary) interactions with the user.[13] In addition, this also simplifies the deliberation: the task layer content is fixed and only the data layer is evolved. Nonetheless, this is not to say events do not fire on the task layer – in contrast, the result of deliberation may involve actions regarding both layers.

13 If a decomposition into subproblems is suitable (e.g. timetable and ticketing services may have a common part of requesting the user to specify the transportation means), these parts can be defined as *macros* in the Domain Editor (see attached CD). When "compiling" the domain, the Domain Editor expands these macros at places where referred, this way complying with the framework notion of Grosz and Sidner's satisfaction precedence.

The principle of the deliberation is as follows. At the beginning, all pending events are gathered and put into an *event queue*, denoted as Q. The initial contents of the task and data layers comprise one *possible world* [Woo95], denoted as W_0, from which we can move to another by satisfying one or more of pending events in the queue. We search through a space of possible worlds until we have found one in which the objective of the task is met and the task is thus satisfied. We denote such world as W_S. Before explaining further, let us formally define this notion of a possible world.

Definition 4.14 (Possible world)
A *possible world* is an ordered triple $W = (Q, \mathbf{Y}_T, \mathbf{Y}_D, W_p)$, where $Q = \{ V_i = (E_i, I_i, initial, R_i) \}$ is an event queue associated with W, $\mathbf{Y}_T$ and $\mathbf{Y}_D$ are the task and data layer information pools, and W_p is the "parent" possible world that W infers from. □

The above outlined principle naturally fits the principle of the A* algorithm. Given a possible world W_i, the A* algorithm evaluates it as

$$f^*(W_i) = g(W_i) + h^*(W_i)$$

where $g(.)$ is a function evaluating the "path" from the initial world W_0 to W_i, and $h^*(.)$ is a function ideally estimating the value of the remaining path from W_i to the final world W_S, with the word "ideally" referring to its ideal behaviour of $f^*(W_i) \cong g(W_i)$, i.e. the estimated value always tightly approaching the real value. Hence, with using the A* algorithm, the problem of optimizing a dialogue reduces to finding such sequence of pending events whose satisfaction transforms the initial world W_0 to the final state W_S the optimal way,

$$W_0 \xrightarrow{V_0 \in Q_0} W_1 \xrightarrow{V_1 \in Q_1} \;\ldots\; \xrightarrow{V_{i-1} \in Q_{i-1}} W_i \xrightarrow{V_i \in Q_i} W_S$$

We currently consider only one optimization criterion: *the length of the dialogue in terms of dialogue turns*. (Therefore, in the confirmation scheme in Fig. 4.18, confirming the *tp109* instance by handling the CON_1 event would be the preferred way, as it yields the shortest dialogue.) There naturally may be more optimization criteria, for instance the length of the dialogue in terms of real time (elapsed and remaining[14]). However, the current implementation of the framework currently does not account for any multi-criteria optimization.

The $g(.)$ and $h^*(.)$ functions are realized as a set of heuristic rules, each capturing one independent dialogue optimization criterion (Table 4.9). To guarantee the monotony of $g(.)$, the notion of each rule is to *penalize* the world for "being not optimal". For better comprehensibility, the contributive penalty of each rule is hidden – the precise computation would require knowledge of different backgrounds we currently do not have, or we do have but not to the extent required. Nevertheless, given the intuitive nature of the rules, showing their precise computation would be redundant, and interested reader is suggested to trace the `usuzujNadObsahemVrstvy` procedure in `plan.pas` to see the evolution

14 The estimation of the remaining time should cover merely system utterances as the only source of well predictable data is agent's Prompt Planner module. Estimation of user's utterance times would require user modeling.

Table 4.9 Dialogue optimization criteria.

Penalization criterion and description	
Utterance is generated incompletely (Agent prefers fully generable utterances)	$g(W_i)$
Utterance uses already valid objects (Agent prefers implicit validation)	$g(W_i)$
Event is not recovered (Agent first processes corrections supplied by the user)	$g(W_i)$
Event priority penalty (Agent prefers processing events in a certain order)	$g(W_i)$
Average salience of objects involved in satisfying an event (Agent prefers sticking to the current course in a dialogue)	$g(W_i)$
Interaction with user penalty (Agent prefers satisfying events without involving the user)	$g(W_i)$
Number of remaining pending events (Agent estimates the difficulty of satisfying an intention)	$h^*(W_i)$

of the total penalty $f^*(.)$, encoded in `gSuma` and `hHvezda` that correspond to $g(.)$ and $h^*(.)$, respectively.

By using the rules in Table 4.9, the $f^*(.)$ function can be rewritten as

$$f^*(W_i) = \sum_{r=1}^{R=7} RULE_r(W_i)$$

The A* algorithm is then to find such sequence of pending events, V_0, V_1, ..., V_N, that gains the minimum total penalty, i.e.,

$$[V_0, V_1, ..., V_N] = \underset{[X_0, X_1, ..., X_N]}{argmin}(\, f^*(W_S)\,) : W_0 \xrightarrow{X_0 \in Q_0} W_1 \; . \; . \; . \; W_N \xrightarrow{X_N \in Q_N} W_S$$

Due to the structural complexity of the search space, the A* algorithm accounts merely for the *OPEN* list to accommodate yet unexplored worlds in a penalty-ascending order. The *CLOSED* list is not used, as the determination if a world has already been explored would be computationally inefficient, given the unpredictable variability of utterances the agent may speak to the user. Hence, the working cycle of the A* algorithm may be put as follows.

1. Assuming the task in focus must be confirmed, the event V_{task} = (*Task*, *CON*, *initial*, *no*) is to be created. In addition, for the plan associated with the task to be interpreted, a plan-interpretation event V_{plan} = (*Plan*, *PLN*, *initial*, *no*) must be created as well. The *OPEN* list is then initialized by the world $W_0 = (\{ V_{task}, V_{plan} \}, \mathbf{Y}_{T0}, \mathbf{Y}_{D0}, \varnothing)$, where $\mathbf{Y}_{T0}$ and $\mathbf{Y}_{D0}$ are the current task and data layers, respectively.

2. The least penalized world $W_i = (\, Q_i, \mathbf{Y}_{Ti}, \mathbf{Y}_{Di}, W_{i\text{-}1})$ is popped out of the *OPEN* list. If the transition $W_{i\text{-}1} \rightarrow W_i$ involved the system performing a *Response* act, W_i is considered the satisfactory world, $W_S \equiv W_i$, and the

optimal sequence of events [V_0, V_1, ..., $V_{i\text{-}2}$, $V_{i\text{-}1}$] (between worlds W_0, W_1, ..., $W_{i\text{-}1}$, W_i) is backtracked, i.e.,

$$[V_0,V_1,\ldots,V_{i-2},V_{i-1}],\ \forall j \in \{0,\ldots,i-1\}:\ W_j \xrightarrow{V_j \in Q_j} W_{j+1}$$

3. The queue Q_i is extended with pending events not present in $W_{i\text{-}1}$ (these regard objects newly added to W_i in response to the parent transition $W_{i\text{-}1} \to W_i$).
4. The world W_i is explored by examining each of its queued events $V_j \in Q_i$, and inferring a new world W_j from W_i, i.e. $W_j = (\, Q_i, \mathbf{Y}_{Ti}, \mathbf{Y}_{Di}, W_i)$, currently fully adopting its layers. In this new world, effects of V_j on the adopted layers are examined and adopted. For instance, objects referred in a *Request-confirm* act (wrapped in a *CON* event) are confirmed, or expectation of a *Request-clarify* act (wrapped in a *PLN* event) is satisfied by incorporating the corresponding *Clarify* act into the data layer (thus predicting user's upcoming behaviour). Finally, the updated $W_j = (\, Q_j, \mathbf{Y}_{Tj}, \mathbf{Y}_{Dj}, W_i)$ is inserted into the *OPEN* list.
5. The deliberation continues by revisiting Step 2.

Before continuing, let us make three remarks. First, not all events are to interact with the user. In fact, the agent prefers to not involve the user in the conversation at all, attempting to solve the task only on its own. This is in coherence with agent's pro-activity. Consequently, the optimal sequence returned by the deliberation process needs to be interpreted as long as no interaction with the user has been planed. In other words, the agent passes the turn to the user only all if possibilities how to satisfy the task have been exhausted.

Second, the plan-interpretation event exists merely in a single instance whose state is continuously renewed. A dedicated function returns the first unvisited interpretable node in the plan. This is a conception preferred over each plan node having its own pending event. Obviously, the reason is the computational tractability. The latter case would in fact involve a random access to the plan, causing an explosion of possibilities. However, a plan is usually a fixed sequence of actions (e.g., initialization, user interaction, database querying, etc.). The overall order of actions is therefore explicitly given and there is no need to engage with combinatorial overhead (satisfaction-precedence between individual subactions would cause actions to be implicitly re-ordered after all). Let us note that this does not mean the agent follows a plan in a rigid way "from left to right". The plan can be re-structured based on user's initiative (Section 4.8.1), or disambiguation requirements (constraints and relaxation, not covered in this text). Interested can trace the high-level *`cyklusInterakce`* procedure in *`Session.pas`* to gain a complex view at the agent's deliberation, execution, and impacts on future behaviour.

Third, the rules in Table 4.9 call for being modular. That is, the current penalization scheme represents merely one of agent's possible optimization strategies (denoted as $\boldsymbol{O}_i$), and another one may easily be created by modifying this scheme. We naturally take advantage of this approach, hence the agent has

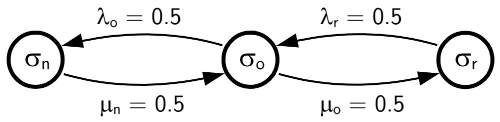


Fig. 4.19 Markov network is used to model transitions between strategies; 0.5 indicates it takes two attempts in average to transit between the neighbouring states.

some penalties when optimizing dialogue using the user-initiative strategy and some different penalties when using the system-initiative strategy.

4.9 Dialogue Strategies

Recall from Section 2.1 that the acceptability of a system depends primarily on its correct outcomes [McG96, Eck95, Chu00]. However, the user satisfaction is also influenced by the style of interaction. There are generally two opposite styles, menu-based and free conversation. It is usually the agent's duty to choose the proper style when interacting with a user.

4.9.1 Choices and Their Arbitration

As also already seen in Section 2.3, the most common parametrization covers the following two aspects of interaction [Sin02, Wal97, vZa99, Lit02, Boh09, Jok10]:

- *initiative* – who of the participants is expected to be "pro-active", i.e. who sets the course of actions (system or user),
- *confirmation* – how the confirmation of user-provided information is to be carried out (explicitly or implicitly).

The Daisy framework adopts the commonly opted combinations, "*system-initiative with explicit confirmation*" and "*user-initiative with implicit confirmation*". For simplicity reasons, we further will refer to them as the *narrow strategy* (denoted as σ_n) and *open strategy* (σ_o) respectively, as these are more appropriate names, provided the compound definition of agent's utterances and their production. In addition, we introduce an new strategy, called the *Rogerian psychologist strategy* (σ_r) discussed below.

To determine the appropriate strategy we adopt the approach from the Jaspis architecture [Tur05, Tur03] – with having three strategies to decide among, it is selected the one whose evaluation score reaches the highest value. The scoring is based on four independent evaluation criteria:

- *Quality of the dialogue using the current strategy*, σ_c. Quality is a wide term borrowed from the PARADISE framework on dialogue system evaluation [Wal98]. We measure quality merely in terms of the number of unsuccessful and successful applications of σ_c, denoted as λ and μ, respectively. A simple Markov network is then used to model the transitions between neighbouring strategies (Fig. 4.19) [Nes07, Nes10a]. In this respect, λ and μ are understood as the so called *failure* and *repair* in the process of strategy arbitration.

- *Task is* **not** *known*, T_{known}. This criterion is used to support the open and Rogerian strategies. The idea is to take advantage of system open-ended prompts at the beginning of a dialogue to motivate the user to freely say her or his request before transiting towards the narrow strategy.
- *The number of missing pieces of eigen information*, n_{ψ}. To shorten a dialogue, this criterion supports the open-ended and Rogerian strategies if at least two pieces of information are missing (inspired from [Cen04]). Naturally, this is an alternative to the previous Rogerian criterion for cases in which the desired task is already known.[15]
- *The change in the number of pieces of eigen information after user's last utterance*, Δ_{ψ}. This number may be either negative (user has supplied information), positive (user has retracted some information), or zero (ambiguous). This criterion in fact determines if the user knew what to say, i.e. $\Delta_{\psi} \neq 0$. This criterion is used to support the Rogerian strategy only.

To facilitate further description, let us put these criteria into a vector $\boldsymbol{P}$ to capture the current state of the *dialogue progress*,

$$\boldsymbol{P} = \begin{bmatrix} \sigma_c \text{ is } \begin{Bmatrix} \textit{unsuccessfull} & \rightarrow & \textit{random} < 1 - e^{-\lambda_c \cdot \, nAttempts} \\ \textit{successfull} & \rightarrow & \textit{random} < 1 - e^{-\mu_c \cdot \, nAttempts} \end{Bmatrix} \xrightarrow{\text{integer}} \{0,1\} \\ \neg T_{known} \xrightarrow{\text{integer}} \{0,1\} \\ n_{\lambda} > 2 \xrightarrow{\text{integer}} \{0,1\} \\ \Delta_{\lambda} \neq 0 \xrightarrow{\text{integer}} \{0,1\} \end{bmatrix}$$

In addition, each strategy σ_i is accompanied by *scoring scheme* $\boldsymbol{S}_i$, corresponding to the above four criteria. These scoring schemes column-wise comprise the scoring matrix $\boldsymbol{S}$, taking on the following form,

$$\boldsymbol{S} = \begin{bmatrix} \boldsymbol{S}_{\text{n}} & \boldsymbol{S}_{\text{o}} & \boldsymbol{S}_{\text{r}} \end{bmatrix} = \begin{bmatrix} 5 & 5 & 5 \\ 0 & 2 & 2 \\ 0 & 1 & 2 \\ 0 & 0 & 1 \end{bmatrix}$$

Presumably, we gain the *score* for each of the strategies by multiplying the vector $\boldsymbol{P}$ with the scoring matrix $\boldsymbol{S}$,

15 In [Cen04], other features are proposed as well; for instance, agent's experience in similar dialogue situations (user modeling) or the range of elicited information. Due to time reasons, these features have currently left unimplemented in the Daisy framework.

$$\begin{bmatrix} SCORE_n \\ SCORE_o \\ SCORE_r \end{bmatrix} = \boldsymbol{P}^T \cdot \boldsymbol{S}$$

The final arbitration is then made by normalizing each score, $SCORE_i$, by the corresponding scoring scheme, $\boldsymbol{S}_i$, to gain the proportional rate of suitability. The most suitable strategy, σ_{opt}, is then selected,

$$\sigma_{opt} = \underset{i \in \{\text{n,o,r}\}}{argmin}(\ SCORE_i \cdot \ |\boldsymbol{S}_i| \ )$$

Naturally, the selected strategy has not only a direct impact on the way the agent speaks to the user (utterance production) but also prescribes particular dialogue optimization scheme, $\boldsymbol{O}_{opt}$ (deliberation),

$$\boldsymbol{O}_{opt} = \boldsymbol{O}_{\underset{i \in \{\text{n,o,r}\}}{argmin}(\ SCORE_i \cdot \ |\boldsymbol{S}_i| \ )}$$

Hence, the optimization scheme must be in coherence with the nature of the strategy. For instance, assuming the narrow strategy has been selected, the optimization scheme must prefer explicit confirmation over implicit confirmation, if available. It also needs to prefer working with smaller pieces of information (ideally DDM leaf nodes) over more "abstract" concepts (closer to the DDM root). Thus, for the confirmation situation in Fig. 4.18, the preferred way would be satisfying events CON_2 and CON_3 (in any order) over CON_1.[16]

4.9.2 Rogerian Psychologist Strategy

The narrow and open strategies model the common adaptability habits in which the system formulates an open-ended prompt (using the open strategy), and if user's response does not provide enough information, asks for it one piece at a time (narrow strategy) [Boh09, Tur05, Mel05, Cen04]. However, as users generally adopt the interaction style suggested by the system [Gus03], then once applying the system-initiative guidance, the task is constrained to a linear progress as the system dictates it. Hence, we focused on suppressing this side-effect by modifying the outlined interaction pattern: *let the user keep the initiative as long as she or he knows what to say.* More particularly, we involve the Rogerian psychologist into our model whose goal is to gain more information by “encouraging the user to keep on talking”. In terms of the so called Rogerian therapy, clients are better helped if they are encouraged to focus on their current subjective understanding rather than on some unconscious motive or someone else's interpretation of a situation [Rog51]. Therefore, by applying the Rogerian psychologist in a dialogue management, one of the anticipated implications is a less forced dialogue as the conversational agent gives users more time (turns) to formulate their intentions before taking over the initiative.

16 Note that a better design style would be to define a C_{time_point} concept with an external T_{time_point} data type to represent the time information as a single value. The confirmation would then be as natural as with the CON_1 event. This once again shows the limiting character of the built-in intrinsic types, as well as demonstrates the benefits gained by declaring external domain-specific types.

Nonetheless, the idea of initiative handed back is not completely new in the realm of dialogue systems. One of the well known implementations is Weizenbaum's Elisa [Wei66], a general chatting robot for leading a conversation an uninformed way, i.e. without properly modeling or even understanding the information being discussed. This behaviour is accomplished using a quite simple procedure: the text is read and inspected for the presence of keywords. If such word is found, the sentence is transformed in accordance with a rule associated with the keyword; if it is not, a context free remark or an earlier transformation is used instead [Bui06]. Thus, for instance, user's sentence "*I visited my friend yesterday*" could be continued by the system with "*That sounds interesting, tell me more about your friend*".

Another example is a dialogue system called CONVERSE, embodying a persona of a young female New York-based journalist [Lev97]. Given the nature of the system, it covers about 80 conversational topics, represented as complex scripts that can be interrupted and reentered later. It also makes use of different informational resources, for instance Collins dictionary or personal information elicited from the user that can be at any point involved into utterance generation. Its control structure is a simple blackboard system in which the scripts compete to take control over the generation and thus the upcoming course of the dialogue. These decisions are made numerically based on weights assigned by the closeness of fit of the input to their expected input [Wil06]. Naturally, the system has merely a limited error recovery mechanism for cases in which it is unable to find a relevant topic for the input to continue in the dialogue. As such, it therefore mainly relies on handing the initiative back to the user whenever possible.

Thus, both Elisa and CONVERSE are systems to lead only a "plausible" conversation with resulting dialogues having no particular goals. However, the Rogerian psychologist is also one of the strategies humans use in a co-operative conversation. For instance, Wallis *et al.* [Wal01] observed a phone call agent to use it when attempting to gain more information from a client during a car booking. After they have analyzed dialogue transcripts, they found that instead of pushing them to say what she needed to know about their bookings, she sometimes preferred either to keep silent or respond with a short sentences accounting for a grounding ("*Yep*") or repeat the lastly gained bits of information. Thus, without intervening the dialogue by taking over the initiative, she supported clients to say more.

Hence, the Rogerian psychologist is for humans apparently a familiar approach whose purpose they can recognize and properly respond to. The problem we can spot now is to find conditions under which it can be used in an automated dialogue management.

In general, the most obvious usability is in cases where the user has provided some information that, however, is useless in the scope of a given task or domain (that is, if we have extracted some useless semantics, or simply know the user was not silent). Of course, this strategy cannot be overused – a conversational agent must always make a tradeoff between being reactive and pro-active, i.e. giving users a chance to provide relevant information, and taking over the initiative after judging about the dialogue qualities. Therefore, the Rogerian psychologist strategy can be applied in dialogues that are evaluated as progressing well. In

```
     Procedure SelectBestStrategy ( ) {
1        Let $R$ denote user's response when discussing intention $I$.
2        Evaluate suitability of each strategy and choose the optimal strategy $\sigma_{opt}$.
3        If $\sigma_{opt}$ is the Rogerian psychologist strategy {
4            If $\neg A \wedge B \wedge \neg C \wedge D \wedge E$ (see figure legend) {
5                If dialogue stagnates (system generates the same prompt as in its previous turn) {
6                    If $R$ supplied some information that supports getting $I$ solved {
7                        User has satisfied one of future expectations, ground with "Uhu.", "Ok.", or "I see."
8                    } else {
9                        Response $R$ did not bring any information into $I$, remain silent.
                     }
10               } else {
11                   System is about to generate a different utterance than in its previous turn
                     because the expectation has been met – randomly choose one of sentences
                     available to the Rogerian psychologist (e.g. "Please say me more.")
                 }
12           } else {
13               Rogerian psychologist strategy cannot be applied. Do not drop it, just override it
                 by another strategy in this turn (thus temporarily assign $\sigma_{opt}$ a different value).
             }
14       }
15       Generate response in accordance with the strategy $\sigma_{opt}$.
     }
```

Fig. 4.20 Agent's utterance generating procedure with the Rogerian psychologist approach at Lines 3–14; A = agent's focus has changed, B = user's intention is known, C = user's first turn response expected, D = agent produces a dialogue move that the Rogerian psychologist can be applied to (e.g., it cannot be applied to *Request* act), E = two or more pieces of information expected (missed).

our case, we approach this by checking if user's last response contributed to the current task satisfaction ($\Delta_\psi \neq 0$). In other words, if the user lastly knew what to say, there is a chance that she or he will know what to say now as well. Another contributive criterion for the Rogerian psychologist may be a high recognition score: if user's responses are less certain, it is more safe to pose direct questions and receive direct answers than later having to recover from errors.

Another application of the strategy may be in cases where the agent expects multiple pieces of information. Naturally, if only one piece is missing, it is always more efficient to ask for it directly. Finally, this strategy can be applied in cases in which user's intentions are unknown (i.e. after the open-ended initial prompt) or ambiguous (later in the dialogue when recognizing user's subintentions).

Finally, let us note that the current implementation of the strategy accounts for both general context-free sentences as well as custom domain-specific alternatives. The context-free sentences are the default; they comprise a set of three general-purpose prompts asking the user to provide additional information: "*Please say me more*", "*Please be more specific*", and grounding-like "*Uhu*". This initial set may be further extended or modified using the Domain Editor (see attached CD), for instance with sentences "*I see*" or a *system silence* token. As already revealed above, apart of the general context-free sentences, the model also allows for domain-specific alternatives, analogous with Elisa's rule-driven ones, for instance "*Please try to detail the train* [*to find departure of*; *further*; *a bit more*;...]"). However, in either case, we put one restriction on their design – the utterances should not contain any cue phrases that could indicate a possible change in the dialogue course [Gro86], i.e., they should not make the user think the system wants to take over the initiative (e.g., "*and now*", etc.).

Fig. 4.20 shows the Rogerian strategy in detail along with the surrounding background logic of agent's utterance production (Lines 3–14). Note that the strategy can be to an extent argued comparable with the open strategy. However, the main difference between them is that while the agent utters at least some question in the open strategy (e.g., open-ended), it either keeps silent using the Rogerian strategy (thus handing the initiative back to the user) or encourages the user to say more by uttering one of context-free or domain-specific sentences.

4.10 Discussion

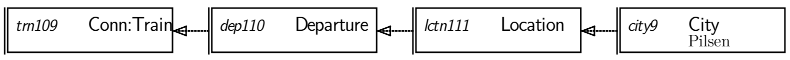
4.10.1 Comparing DDM with SIL

As already discussed in Section 4.4 on semantics, the most significant distinction is the lack of taxonomy in DDM. While objects in SIL are to an extent "self-explanatory" thanks to the presence of their semantics and predefined taxonomy, objects in DDM merely describe the static, neutral data which gain their meaning and taxonomy no earlier than during a particular plan processing. Another significant difference is the degree of freedom among concepts. While SIL in fact allows for a random nesting (as long as nested concepts fit the semantical role), DDM captures a strongly structured and finite hierarchy of objects. The implication is that while SIL in fact acts as a universal tool for describing any kind of information, DDM along with all its restrictions allows merely for a subset of it, as discussed in Example 4.5. The less important restrictions regard the linearity of structures (i.e. absence of recurrent patterns) and unambiguous addressability (i.e. without transitive subpaths, see Definition 4.5). The more important restrictions regard the information layout – any information is to be stored in leaves, and parents then disambiguate similar pieces of eigen information (e.g., the city of arrival from the city of departure). Arguably, such highly organized way puts additional requirements on the input semantics pre-processing. In other words, the expected input is incompatible with the traditional vector-like "flat" semantics, and must therefore be "converted" to the expected hierarchical form (see Appendix A.1). Contrarily, a sequence of UFOs resembles the traditional flat semantics (Fig. 4.21).

The SIL and DDM expressions, naturally differ as well. First of all, the local closure (Definition 2.6) that holds for SIL expressions does not apply to DDM. Unrelated expressions may therefore have common parts (e.g., as demonstrated in Fig. 4.14b in which *tim177* and *tim179* topics share the same part *cri82*). Hence, there hold strict rules for comparing two expressions: they must be recurrently absolutely "compatible" in order for one to merge with the other (e.g., exact validity, exact set of fired events, etc.). On the other hand, splitting a common DDM expression apart is not directly possible. If merging rules are not met, the

$$\begin{bmatrix} id: trn109, \; type: train & \\ thesource: \begin{bmatrix} id: loc111, \; type: location & \\ thecity: \begin{bmatrix} id: city9, \; type: city \\ value: \text{Pilsen} \end{bmatrix} \end{bmatrix} \end{bmatrix}$$

(a) SIL representation



(b) DDM representation

Fig. 4.21 Comparison of structural representation of "*train departing from Pilsen*".

framework may only infer a new expression from the common one, and modify a local portion of it.

The SIL formalism has also the potential of representing (although not directly detecting) requested tasks in user's *statements* (recall "*I want to know the nearest train to Erlangen*") that are usually pre-processed yet before the dialogue manager by observing a particular utterance "shape" [Eck95, Boh09, Ngu06b]. In this respect, the SIL formalism demonstrates why information is best not to be decoupled into task and data sub-information – in general, simply because some soft-grained aspects of user's semantics might get lost (e.g., that the "*nearest train to Erlangen*" [data] is "*wanted to be known*" [mental state, not a direct task]). On the other hand, SIL expressions on their own are a too weak means to capture the organization of a dialogue. The projecting into views is far from accounting for the Grosz & Sidner's work on discourse segmentation [Gro86]. The effort needed to incorporate their framework would very probably result in an approach comparable with ours. In addition, some singular cases of object unification would have to be cleared or even dropped to avoid the overhead of rules required to deal with them. This is a step we had to make as well (discussed in Section 4.4).

All spotted differences can be justified by the two approaches being devised for distinct cases of use. While SIL has been designed as a means for describing any kind of knowledge, and is therefore strongly semantics-oriented, DDM puts most stress on being capable to represent and further handle different situations that arise during a dialogue. The apparent implication is naturally the different representation of the dialogue context – while held as the history of raw semantics in SIL, the same information is pre-processed and more effectively stored in the case of DDM. This then leads to easier and more elaborated approaches to solve common issues in a dialogue:

- *Corrections.* By relaxing the demand on information monotony, it is possible to perform different kinds of corrections, yet benefiting from being able to keep track of the information causality – changes to the data layer force the agent to re-deliberate on modified pieces. On the other hand, the SIL representation allows for old overriden information be recovered once new information has been retracted (e.g., "*to Erlangen*" + "*to Erfurt*" + "*not to Erfurt*" → "*to Erlangen*").

- *References.* Their resolving is to an extent very similar. In our case, we have added the support for gender-specific references (*ten*, *ta*, *to* in Czech, or *der*, *die*, *das* in German), unexpressed nominative (recall "*I buy it*" with *it* referring to a known ticket object), and nested references (recall "*How much is a ticket for the train with the first class coach?*").
- *Disambiguation.* In DDM, disambiguation can be made by a reference (that resolves to a set of candidate objects) or by providing new information (e.g., parent for an underspecified information). Additionally, there is a complex analysis of the input semantics against the dialogue context to discover its relationship to existing objects. We believe that its elemental parts cannot constitute isolated units as they are in SIL where the "anchoring" stands apart from the remainder of processing, being called only under specific conditions (Fig. 3.12).

Hence, in DDM optimization has been made in favour of the dialogue context processing. However, this is not to say that none of the above presented features would be possible to incorporate into SIL. With respect to our previous discussion, this would very probably mean to reduce (or eliminate altogether) a number of built-in semantic concepts, i.e. the cornerstones of SIL (see example at the beginning of Section 4.5.1). Eventually, another possibility would be to construct a set of exception rules to work around the soft differences that prevent otherwise compatible objects from being unifiable. Whichever way would be taken, SIL could be extended with mentioned DDM features no earlier than after this point.

4.10.2 Extending Agent's Planning

Our approach to agent's planning makes use of the A* algorithm. This is considerably a different course of planning from the commonly adopted greedy algorithm. In this section, we will discuss two additional extensions to the current deliberation procedure, that due to time reasons remained unimplemented.

The first extension regards dialogue games [Man88]. Briefly, dialogue games describe certain re-occurring patterns of utterance types commonly observed in dialogues. For instance, it is usually the case that a question is responded by an answer; a proposal is accepted or rejected; or a greeting is returned. However, as Hulstijn points out [Hul00], dialogue planning and dialogue games are usually considered two competitive approaches. As he further explains, they may be merged together to create a robust and better organized dialogue management. Thus, instead of structuring the whole dialogue as one stack of running dialogue games, he proposes to identify dialogue games separately at the level of each discourse segment ("stacks inside stacks").

Naturally, the opening of each dialogue game must be explicit: either the user or the system needs to perform a corresponding *move*. Each move relates to a type of utterance. So what we called the communicative function of a dialogue act (Section 4.6.2) is now defined as a game move. In contrast, the closing of a dialogue game may occur multiple ways. One of them is that the original initiative (e.g., a question) is followed by an appropriate reaction (e.g., a response). Under such condition, the dialogue game is considered *closed* and popped out of the stack. Another way may be to observe a newly opened game breaching the satisfaction-precedence; in other words, it does not contribute to the solution, nor is it possibly related to the current course of actions in a task. To accommodate dialogue games in our framework, the set of recognized dialogue acts (Table 4.5) would need to be further revised and extended in order to properly model the different conversational situations that particular dialogue games describe. Also, the set of penalization rules (Table 4.9) would have to be extended. The new rule(s) would need to capture agent's goal of *closing the currently running game* as its primary operation, i.e. with the lowest or no penalty. Apparently, this is an alternative formulation of agent's pro-activity (compare with its current primary goal of satisfying the open task in focus).

Let us now concern with the second possible extension to the deliberation – a multi-agent environment. The overall framework currently behaves as a monolithic entity, with each demand on any external functionality being of a *blocking* nature (e.g., database requests, system-related component operations, etc.). However, in a highly interactive and/or time-critical environment (and possibly for the reason of facilitated maintenance), this may not be the preferred way. The proper solution would therefore be a multi-agent approach. In such environment, agents communicate among each other to exchange their current statuses, demands, results of operations, etc. Therefore, each of the external functionality calls is a priori non-blocking and the requesting agent may continue working on its objectives until a response is received from the requested agent.

Apparently, the most straightforward way of preparing the current dialogue agent for a multi-agent environment is to replace the event queue with a message queue (as events may be understood as "internal messages"). The A^* algorithm can then be applied to optimize the dialogue agent's behaviour with taking into account the messaged states of other agents in the system. Intuitively, one of the significant modification to the A^* algorithm might be to accommodate reasoning about time. This can be possibly an important aspect to allow for agent's efficient, time-aware planning. For instance, by knowing that the requested operation result will be ready no earlier than after a certain period of time (by the requested agent), the dialogue agent might continue working on other things in the meantime (possibly other communications). However, the question of proposing a suitable inter-agent communication has to be left unanswered at this moment as available communication protocols have not been the main subject in this thesis. The inter-agent communication and protocols to consider are overviewed, for instance, in [Zbo04].

4.11 Summary

This chapter has aimed to provide a comprehensive and detailed description of the Daisy framework. In its entirety, the framework has been designed to provide a multi-user environment for multi-domain, task-oriented dialogues. The framework has been implemented in accordance with the specifications found in this chapter, unless clearly stated otherwise. To further facilitate its understanding, we several times referred to particular routines in source codes where eventual ambiguities may be closer consulted against the underlying documented code.

We furthermore would want to highlight the following achievements, each of which has been thoroughly designed and implemented[17] from scratch during the study, targeting all major topics in the realm of dialogue systems:

- soft-grained data representation and their advanced management,
- intention detection and management,
- dialogue length optimization (unique),
- extended set of dialogue strategies (unique),
- robust and parametrizable utterance production,
- framework Domain Editor.[18]

17 Using "unmanaged" Delphi, i.e. without making use of any kind of managed components (e.g., String, ArrayList, etc.) that could eventually impair the framework efficiency. Delphi has been prefered over C++ as it is a block-structured language (i.e., allows functions to be "nested" within other functions) which soon turned out to be very beneficial; Delphi has also been preferred over .NET due to performance reasons (as at the beginning of the development it was not clear how many exponential algorithms will be there and how time-consuming they were going to be) and portability (while there exist numerous Object Pascal compilers on Linux and other operating systems, there is only the Mono Project as for .NET whose compatibility with Microsoft implementation was not convincing at the start of working on the Daisy framework).

18 Native Win32 build unstable due to Delphi's poor treatment of custom interfaces (which is a well known and common issue of custom interfaces in the Delphi compiler version 7.2 or earlier). Interested user is therefore advised to use the .NET 4.0 improved re-implementation instead, provided on the attached CD as well.

This page intentionally left blank.

Chapter 5

Experiment and Results

In the previous chapter, we used a timetable application to demonstrate different dialogue situations and work out their solutions. The application was a non-public concern that was later abandoned as a new project was started, and maintenance of two parallel projects became inefficient. The new project was launched with the kind support of Sympalog Voice Solutions, GmbH.[1] Unlike with the timetable project, this one was a real spoken application tested publicly with volunteers, and it is this application that we will be concerning with in this chapter. We first will describe the application, showing its structure and technical background (Section 5.1); then, the experiment prepared to test the framework will be described (Section 5.2); finally, we will present the results and provide a thorough discussion both on them and further improvements to the overall system (Section 5.3).

5.1 The DORA Dialogue System Overview

The new project started with Sympalog was a banking domain dialogue system. This was in fact a reimplementation of a system created earlier for one of their customers (Fig. 5.1). The functionality of this original system covered bank branch information (opening hours, addresses, etc.), and individual account basic information (balance, recent activity overview, etc.). While we kept all branch-related services, the account information was in our case reduced to mere balance status. The reason was a lack of interaction featured in these account-related tasks – they simply were too straightforward to be used for testing of the Rogerian strategy.

Before showing the usage of the framework in implementing the plotted functionality, let us mention the technical background that we will refer to as needed throughout this chapter. One of the main concerns in creating the system

1 *http://sympalog.de*

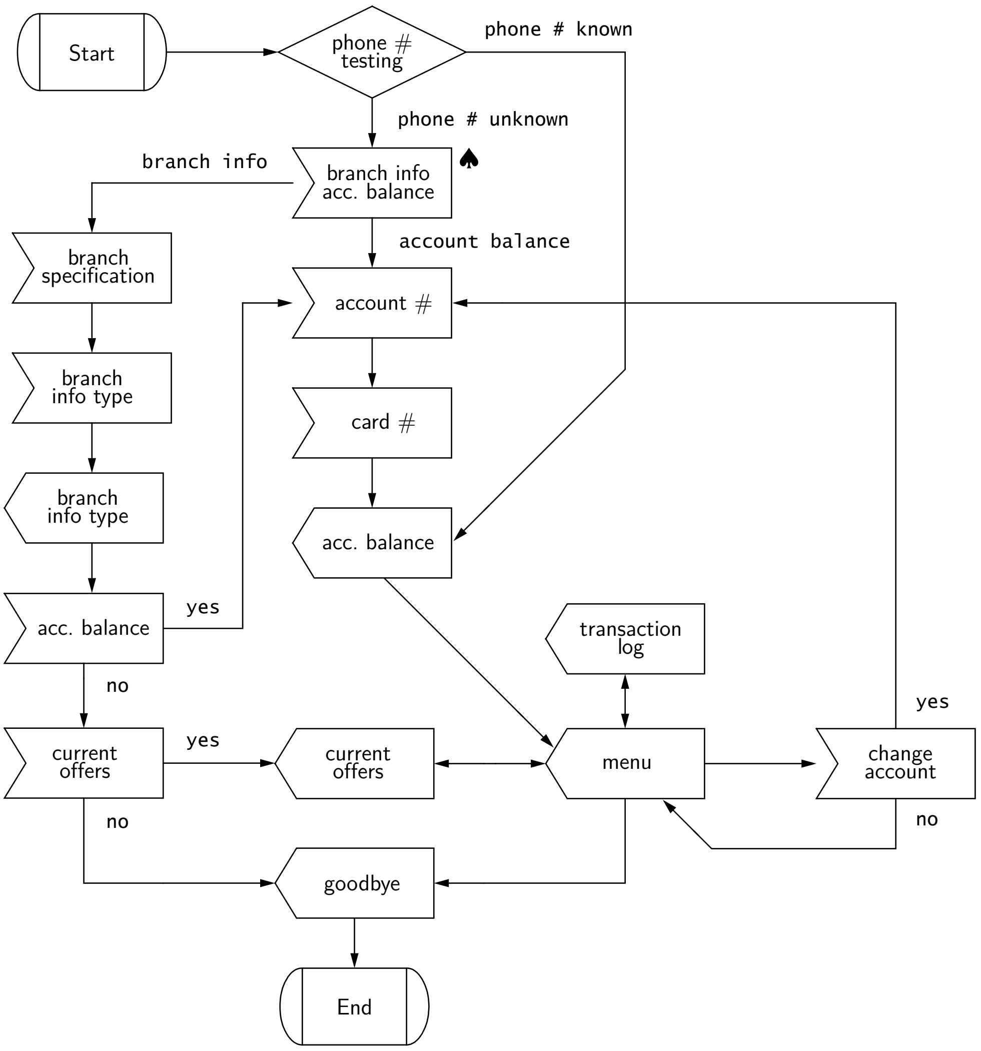


Fig. 5.1 Original dialogue system functionality outline by Sympalog, see Appendix A.6 for SDL notation explanation [Bel89]; node marked with ♠ referred from the text.

was to bridge two incompatible frameworks: Sympalog's SymBase and Daisy. The SymBase framework (see leaflet attached on the CD) constitutes a complex and scalable platform for creating mixed-initiative dialogue agents. These in turn can be hosted on desktop servers as well as mobile devices, with limited capabilities. The platform as such consists of various modules each of which contributes with unique services to the resulting dialogue system functionality, including speech recognition, semantics processing, dialogue management, utterance production, dialogue testing, etc. The modularity of course enables the platform to be split apart at virtually any point and replaced with equivalent custom functionality. We made use of this aspect to replace the dialogue management and utterance production services. The result was a distributed dialogue system with modules running in Erlangen, Nürnberg, and Pilsen. Fig. 5.2 shows the overall structure.

To properly model the banking system interaction, three components needed to be prepared: data, plans, and dominance models. The first of them, the

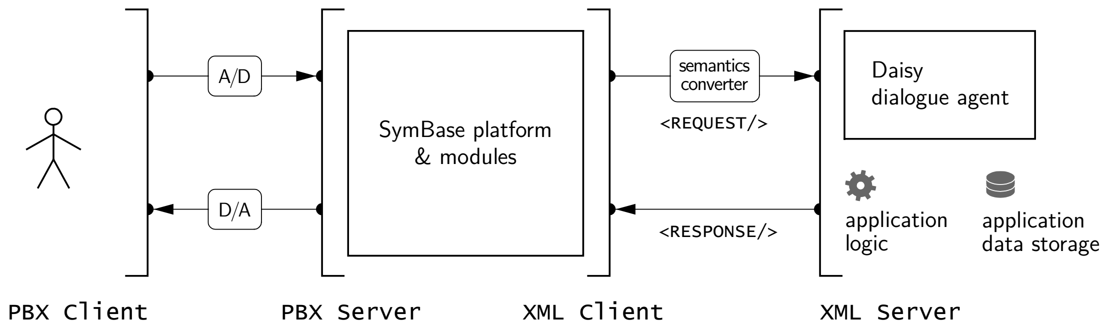


Fig. 5.2 Dialogue system technical background.

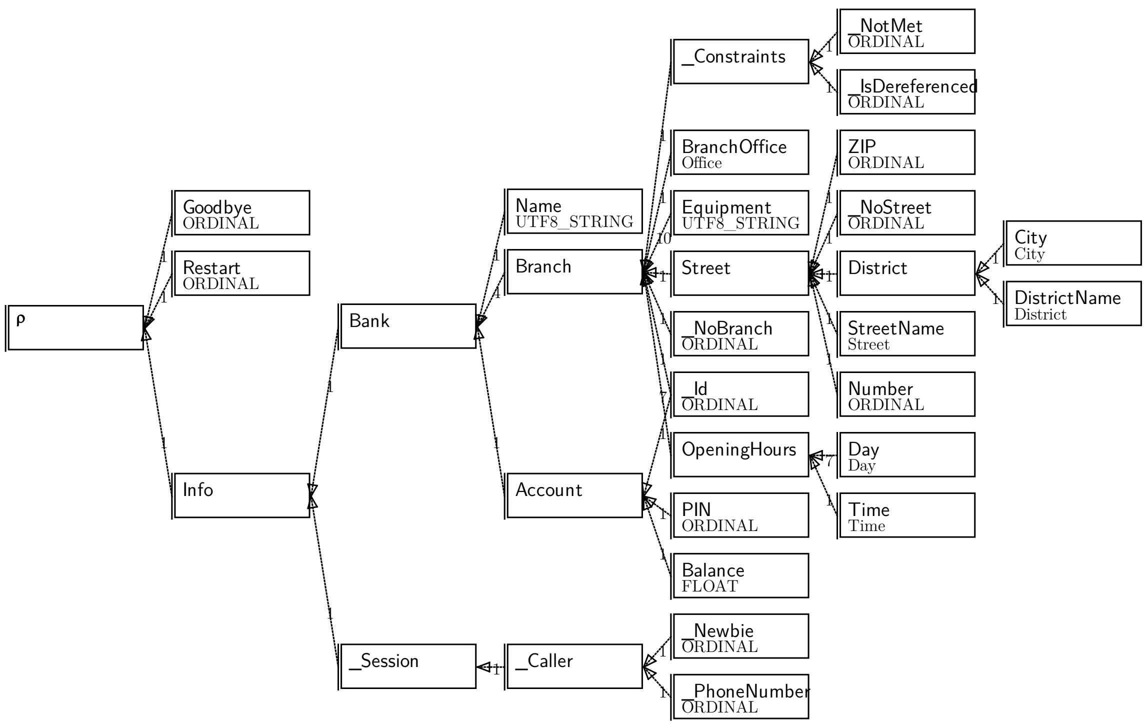


Fig. 5.3 Banking system DDM; concepts with underscore prefixed names are for system internal purposes only; camel-cased data types are external.

data model, is shown in Fig. 5.3. As it can be seen, there are three topics, τ_{info} (providing information of different kinds), $\tau_{restart}$ (restarting the current task; more specifically, restarting the task *below* the Restart-task in the stack), and $\tau_{goodbye}$ (terminating the session after user's explicit confirmation). The last two mentioned topics, $\tau_{restart}$ and $\tau_{goodbye}$, were introduced due to the lack of equivalent "out of the box" functionality in the Daisy framework;[2] the τ_{info} topic is already a "regular" one. As indicated by the DDM, information in τ_{info} always concerns a bank. This bank has a name, several branches, and several accounts.

2 Because they also are leaves in the model, they must carry a dummy value, in this case of the $T_{ordinary}$ built-in type.

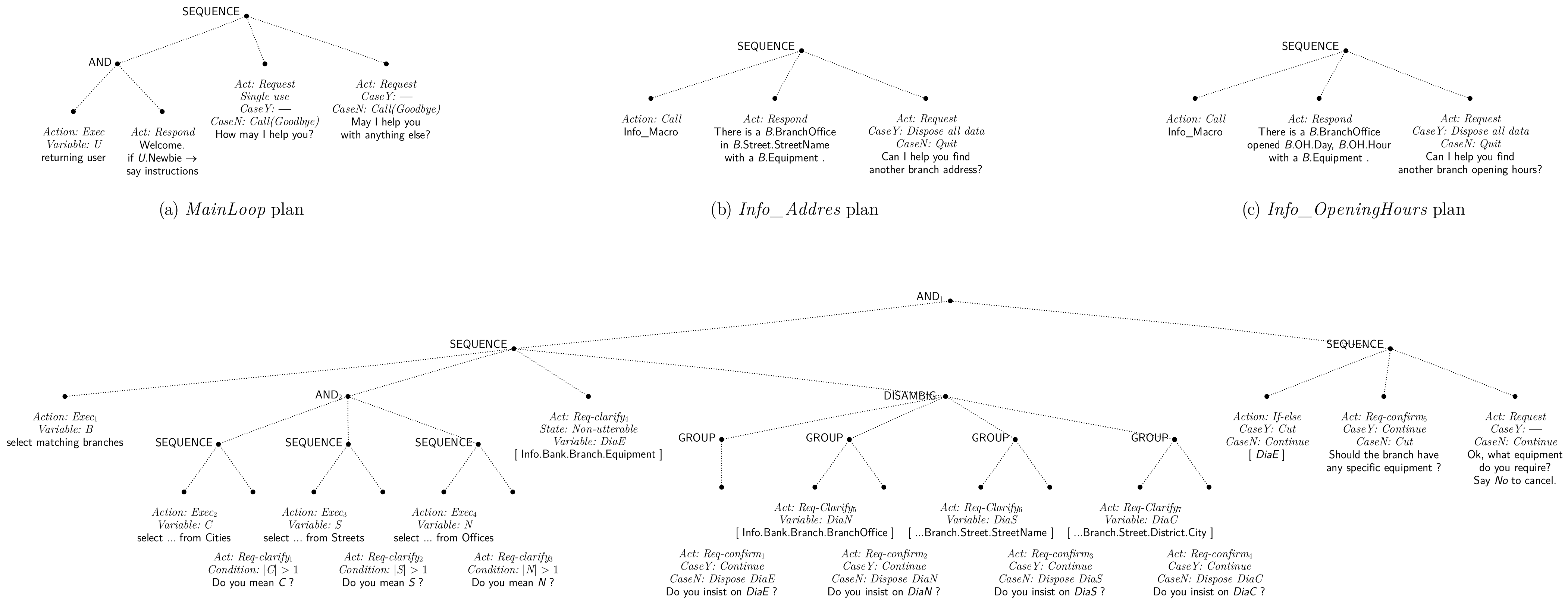
Furthermore, each branch has a name (K_{branch_office}), equipment,[3] opening hours, and is located in a street (which is located in a particular district, which in turn is located in a particular city). Thus, the DDM captures a hierarchical structure of underlying data in the domain. However, as already discussed in Section 4.4 on semantics, such representation is not used for raw semantics, for which a key-value pair vector is more common. To suppress any incompatibilities (including different dialogue control acts representation, like agreement, ask for help, etc.), a one-way semantics converter was developed.[4] For completeness sake, let us also note that some concepts in the DDM are underscore prefixed, e.g. $C_{\_session}$. These were introduced for internal purposes only (database query result, etc.) and we will ignore them in this chapter. Also in the DDM, external data type names are camel-cased, while the built-in ones are fully capitalized. The T_{city}, $T_{district}$, and T_{street} external types have a common combinatorial behaviour – their values must stay in boundaries of a predefined set. Thus, when deciding about the replacement or extension of an existing *old* value by an incoming *new* value, the result of their hypothetical combination is matched against the set of known names. If a match is found, extension is committed, otherwise replacement chosen. From the technical point of view, this base behaviour is provided by the common external type T_{base} which the three mentioned types infer from and override some of the inherited functionality (the Domain Editor shows the dependency more clearly). Finally, the T_{day} and T_{time} types are each of an individual behaviour. Particular implementations of any of the five external types may be consulted with the files `typ*.pas` on the attached CD.

The second component, a set of *plans*, constitutes the domain-specific, managed logic. There are three essential tasks that are detailed in Fig. 5.4: *MainLoop*, *Info_OpeningHours*, and *Info_Address*. The goal of the *MainLoop* plan is to welcome the user an individual way (newbie user versus returning user), ask for the first request, then ask for any subsequent request (the difference is merely in the formulation), and farewell. A different approach has been chosen for the two *Info* plans. As they differ merely in the kind of information that satisfies user's request, they both first call a common macro to subsequently make a task-satisfying statement about branches found by that macro. Presumably, the macro covers all necessary interaction with the user and the back-end to filter out a reasonably low number of branches that meet user's demands (four, according to the DDM in Fig. 5.3). The macro itself unexpectedly starts with querying the database for branches that match user-provided criteria, whatever they are at this moment, eventually returning the whole data set ($Exec_1$).[5] Next, the macro checks if each of the criteria resolves to a unique value (node And_2 and its subnodes). If this is not the case, the system offers possibilities for user's ambiguous specification (e.g., user's "*Frankfurt*" is ambiguous with Frankfurt am Main and Frankfurt an der Oder). Then, there is a "non-utterable"

3 Credit goes to Sympalog for retraining their original ASR to incorporate an ATM as the equipment a branch can have.

4 By Václav Struhár.

5 The overhead is to be handled a domain-specific way. One possibility might be to introduce a DDM concept whose instantiation would indicate a "too many results to return" answer. As our full data set consisted merely of 38 records, we left this case unaddressed. However, we recognize the "no results to return" case by the presence of the $\Pi(K_{branch}, K_{\_no\_branch})$ path in the result.



(a) *MainLoop* plan

(b) *Info_Addres* plan

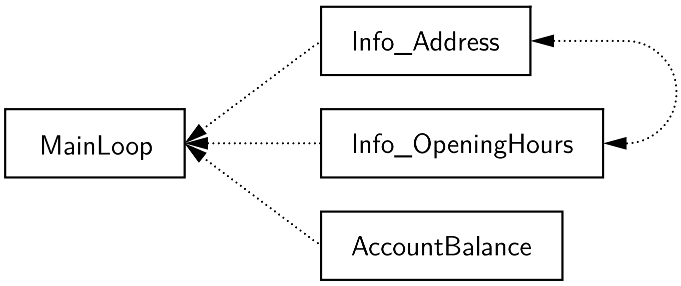
(c) *Info_OpeningHours* plan

(d) *FindBranches* common macro

Fig. 5.4 Dialogue system main plans to model interaction with the user.

query (*Req-clarify*$_4$) to record demands on branch equipment, if any. Two facts accompany queries of the non-utterable kind: first, their information is not mandatory for the task satisfaction, i.e., the agent will not query the user if it cannot find a corresponding value in the data layer; and second, information may be elicited from the user at another, possibly more suitable point in the plan (see below). Next, constraints gathered so far from the user are further restricted or relaxed if the number of matching branches is greater or lower than the given thresholds (*Disambig*). Restrictions are represented by regular queries (*Req-clarify*$_{5\ldots7}$), whereas relaxations by yes-no queries (*Req-confirm*$_{1\ldots4}$). User's updates to the data layer return the agent to some previous point in the plan. Finally, the system asks the user if she or he demands a specific equipment (*Req-confirm*$_5$), expecting *yes*, *no*, or the equipment directly. From the technical point of view, in conjunction with *Req-clarify*$_4$, this may seem an unnecessarily complicated solution, however, there are historical reasons for that.[6] Once processed, the agent has gathered enough information to satisfy the given task by responding with information requested.

The third component, *dominance model*, defines relations among tasks. As already mentioned earlier, the functionality of the original application by Sympalog was slightly reduced to eliminate some transitions and allow this way for more straightforward decomposition into tasks evaluated as important for testing. (Important tasks were those with possibly rich interactivity where the Rogerian strategy could be invoked.) Therefore, for instance, the query node marked with ♠ in Fig. 5.1, asking for the type of information an unknown user wants to know about a branch, was included in the extension of the initial prompt and its help ("*I can offer you the following services: branch address, branch opening hours, account balance,…*"), leading to the following set of tasks and dominance relationships among them:



Presumably, some transitions do not have a matching counterpart in the original design, however, the important matter is the new and original designs intersect reasonably closely. To model the transitions exactly, some single-purpose,

6 One may argue that a regular query node (QUE) meets the desired objective as its implementation may combine both *Request-clarify* and *Request-confirm* acts at the same time. However, the domain proposal originally counted with time information constraint as well, allowing the user to filter branches also by their opening hours. Thus, two additional non-utterable queries were present in a similar fashion like *Req-clarify*$_4$ to store eventual hours and minutes, respectively. The prompt in *Req-confirm*$_5$ then read "*Do you need the branch to have any specific equipment or be opened at a certain time?*", suggesting either information was accepted, allowing for mixed-initiative interaction. Adopting this approach, the user could have collectively refused to provide any of this information by simply responding with *no*. Contrarily, if the regular query node-based approach was taken, the user would have to refuse it individually, one piece at a time, leading to three subsequent *no*'s before the dialogue could proceed.

mostly non-interactive "proxy tasks" would have to be introduced.[7] This was, however, evaluated as unnecessary. Also notice the mentioned tasks *Restart* and *Goodbye* are not included in the model. This is caused by them being defined as "ambient" plans, i.e., a behaviour that may interrupt the dialogue at any time. Finally, the *Transaction Log*, *Current Offers*, and *Change Account* functionality was not included in our reimplementation as it was evaluated unimportant.

5.2 Tested Objectives and Preparation

The system utterances were designed to test different dialogue strategies. The experiment as such required every user to complete a set of application tasks, ideally in the ascending order, although this was not demanded. Instructions to the users were given on a set of web pages (see Appendix A.3), one task per page. Each web page consisted of a brief description of agent's functionality, a list of hints for talking to the agent, a task description, and information on how to call the agent. Each page also contained a form for specifying information acquired from the agent during the dialogue. Users read the instructions before calling the agent from their personal phones. Usage of public phones or shared office phones was explicitly forbidden as the phone number was used as each user's unique identifier. After completing the tasks, each user was given a survey designed to probe her or his satisfaction with the system.

5.2.1 Dialogue Strategies Tested

The types of strategies differed in terms of agent's initiative. More particularly, we tested the usage of the narrow strategy (NS), open strategy (OS), and Rogerian strategy (RS). To briefly recall, the NS strategy uses directive prompts and controls the dialogue by explicitly querying the user for particular information. To demonstrate, the following example is a real dialogue snippet (to comply with the European personal privacy policy, we were obliged to obtain each user's explicit permission prior to recording their voice; as in many cases the permission was not granted, we only could log the internal representation of conducted dialogues – software communications, utterance semantics, etc.; for consistency reason, we use only these logs to abstract from user's literal expressions in all dialogues; as the semantics conveys usually just a small amount of information, it is easy to induce user's underlying intention that resolved to that semantics).

S	Hello, thank you for your call (...). How may I help you? (Hallo, danke für Ihren Anruf (...). Wie kann ich Ihnen helfen?)
U	*_timeout_*
S	I can offer you the following services: account balance, branch addresses, and branch opening hours. (Ich kann Ihnen die folgenden Dienste anbieten: Kontostand, Filialauskunft und Öffnungszeiten.)
U	*_declarative_(Branch(OpeningHours(Time:—)))*

7 This cumbersome approach is currently the only way how to imitate the "goto" feature of SymBase frame-based management by using merely a dialogue stack. In the upcoming versions of the Daisy framework, it is counted with adding a native support for this feature.

S	There are too many branches. Please say which city the branch should be located in. (Es existieren zu viele Filialen. Bitte sagen Sie, in welcher Stadt sich die Filiale befinden soll.)
U	*_declarative_*(*City*:"Gelsenkirchen")
S	Ok, you are looking for a branch in Gelsenkirchen. In case you know it, please say which street the branch should be located in. (Ok, Sie suchen eine Filiale in Gelsenkirchen. Bitte sagen Sie, falls bekannt, die Straße, in der sich die Filiale befindet.)

In contrast with that, the open strategy does not propose directly what possibilities there are at a given time. Instead, it assumes that users know what to do and therefore it lets them control the dialogue as in the following snippet where coincidentally the same amount of information has been exchanged.

S	Hello, thank you for your call and welcome back. How may I help you? (Hallo, danke für Ihren Anruf und willkommen zurück. Wie kann ich Ihnen helfen?)
U	*_declarative_*(*City*:"Gelsenkirchen" , *Branch*(*OpeningHours*(*Time*:—)))
S	There are too many branches in Gelsenkirchen. In case you know it, please say which street the branch should be located in.

Finally, the Rogerian strategy is an experimental one whose goal is to motivate users to say more by handing the initiative back to them, using context-free or context-aware sentences. Hence, the system may or may not suggest the topic of discussion (see Section 3.9.2 for details). The following snippet shows a dialogue with a successful application of the context-aware variant in S_5–U_5 turns. For clarity, each system prompt is now also labeled with the strategy in use at the moment of particular prompt production.

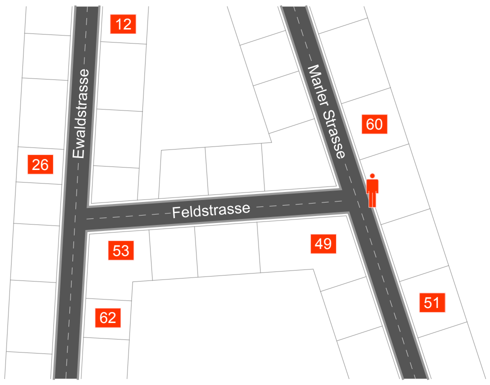
S_1	RS	Hello, thank you for your call and welcome back. How may I help you?
U_1		*_declarative_*(*Equipment*:"Geldautomat" , *City*:"Gladbeck")
S_2	RS	Ok.
U_2		*_timeout_*
S_3	NS	I can offer you the following services: account balance, branch addresses, branch opening hours.
U_3		*_declarative_*(*Branch*(*StreetName*:—))
S_4	NS	There are too many branches in Gelsenkirchen. In case you know it, please say which street the branch should be located in. —*Pause*— If you don't wish a particular street, continue with saying No.
U_4		*_declarative_*(*_disagree_*)
S_5	RS	Please specify the searched branch further. (Bitte spezifizieren Sie weiter die Filiale, die Sie suchen.)
U_5		*_declarative_*(*BranchOffice*:"Filiale_Bertlich")

We did not focus on investigating the suitability of the NS and OS strategies (as previous work has shown best suitability for novice and experienced users, respectively [Lit02, Wal97, vZa99]). We instead focused on the Rogerian strategy by intending to proof two hypotheses: (1) users can respond to it, despite they hear it from a computer, and (2) the strategy gradually improves the information exchange rate by decreasing the number of turns needed to solve a task with a dialogue agent. Thus we hypothesized that a system *without* the Rogerian strategy might be superior for the first task, but that a system *with* the Rogerian strategy would have better performance by the last task.

5.2.2 Experiment Description and Preparation

Each user performed three tasks of different levels of difficulty in a sequence. Each task was represented by a scenario where the user had to find a branch satisfying certain constraints, by using the agent to retrieve and process online branch information. The tree tasks were as follows:

- *Task 1.* Try to find all **Bertlich** branches with an **ATM** in **Gladbeck**. Please write down their exact **addresses** that you were told by the system.
- *Task 2.* Given the below map of **Herten**, try to find the exact **address** of the nearest **Hassel** branch that has an **ATM** (based on your current position depicted in the map, it is up to you to determine the shortest route!). Once known, check its **opening hours** and make sure you can visit it even after 6 pm. Please spot the nearest branch you have found, and also write down its exact address.



- *Task 3.* Try to find the **opening hours** of all **Horst** branches in **Gelsenkirchen**. Check their **addresses** and make sure you can visit any of them in Dorfstrasse even after 6pm. What is the exact location of such a branch?

Thus, the first task consisted merely of a single goal of finding a certain branch address. The second task was already a compound one, consisting of finding a branch address and afterwards checking its opening hours. Finally, the third task was a compound one as well, but composed in the opposite order.

However, as the name of each branch (Bertlich, Hassel, and Horst, in particular) basically identifies that branch, the original data set by Sympalog was modified so that there were multiple branches with the same name in each of the selected cities (two Bertlich branches in Gladbeck, four Hassel branches in Herten, and three Horst branches in Gelsenkirchen, among other branches). Although maybe confusing at first glance, this was the only way how to generate enough parameters a branch can have and thus allowing for the Rogerian strategy to eventually take place during a dialogue. In this respect a classical timetable domain would have provided a much better (and more natural) parametrization of objects the user and the system were to exchange (e.g., different transportation means, tickets, locations specified by names, points of interests, etc.). Nonetheless, such kind of domain was unfortunately not available at Sympalog at the time of creating the banking system. Thus, the reason of introducing identically named branches was to degrade the name as an identifier and make it a common input parameter that merely contributes to the final selection of branches to present. The resulting set of parameters then accounted for the branch name, street name, city name, and equipment. Opening hours were not included in the input parametrization as the amount of ways a time information can be supplied spans a large number of possibilities that have to be recognized by the ASR; for instance, the following expressions may be considered equal:[8] in the evening, after 6pm, between 6 and midnight, around 21 o'clock, etc.

Finally, to convey the system to the public, an advertisement leaflet was composed and published (Fig. A.1). To gather potentially as much data as possible, the experiment was accompanied by a winning draw to stimulate user's motivation in giving the system a call. Copies of the advertisement leaflet were then hanging at the Friedrich-Alexander Universität Erlangen-Nürnberg along with the Regionales Rechenzentrum Erlangen (thanks to Dr. Tino Haderlein), University of Regensburg (thanks to Prof. Václav Matoušek), and Vienna University of Technology (thanks to Leszek Chmielewsky).

5.2.3 Collection and Extraction of Measures

In our study, experimental results were collected from individual users and extracted from underlying dialogue logs. All metrics we mention in this section are given in **boldface**. First, we logged the total time of each interaction (the variable named **Elapsed Time**). In this respect, it could have been interesting to have a more detailed layout of pauses in the interaction (especially as far as reactions to the Rogerian strategy are concerned); however, the SymBase platform unfortunately does not feature such measurement possibility.

Second, the agent's dialogue behaviour was logged in terms of particular message, strategy used to produce that message, and time stamp of the production. Among these, we were particularly interested in the number of times the Rogerian strategy was used (**Rogerian Turns**). In addition, it was also logged the number of timeouts (**Timeouts**) and the number of times that the user accessed the task state-specific help message (**Help Requests**). The number of **System Turns** and **User Turns** were also calculated on the basis of these logs, as well as the average number of pieces of eigen information in user's turns (**Average Turn Length**). On a related note, the SymBase platform features merely weak possibilities to measure user turn-related ASR parameters (recognition score, etc.). Therefore, no speech recognition measures were gathered, which to an extent was a limitation in our investigation. Let us also note that we are only interested in the *relevant part* of each dialogue. We take as the

8 Personal communication with Dr. Martin Schröder.

relevant part anything between the beginning of a dialogue and the first point where the user is provided the correct answer by the system; if the user then interacted further with the system (e.g., asked for a repetition of the answer, or checked the account balance), we do not take such interaction into account for our evaluation.

Third, users were to specify information that they had acquired from the agent (e.g., the address of the nearest branch in Task 2). This was then used in conjunction with data that we had logged during the interaction to compute **Kappa** (more on it below). Finally, users responded to a survey on their subjective evaluation of their satisfaction with the agent's performance. The basic form of the survey followed a common questionnaire on the standard Likert-scale [Lik32]:

- *Question 1.* Do you think it was easy to obtain the information we requested you?
- *Question 2.* Was the pace of interaction appropriate? Were you able to follow the information conveyed by the system?
- *Question 3.* Did you know what you could say at each point of the dialogue?
- *Question 4.* Do you reckon the speed of system's reactions was appropriate?
- *Question 5.* Did the system work the way you expected it to during the dialogue?
- *Question 6.* Based on your current experience, do you think you would use Dora regularly to access information on branches?

All of the responses ranged over predefined values from {*I don't agree*, *I rather don't agree*, *I don't know*, *I rather agree*, *I agree*}. Each of these values was mapped respectively to an integer in 1…5. Each question emphasized the user's experience with the system, with the hope that satisfaction measures would indicate an overall evaluation of the system over the three tasks. We calculated the **User Satisfaction** score (cumulative satisfaction) for each dialogue by summing the scores of the multiple choice questions in the survey.

For interactions where the agent applied the Rogerian strategy, the questionnaire continued with additional questions shown below. Their aim was to determine the position of the RS strategy in the context of its applicability *informally.* The questions were to imitate an interview with the user in the sense of the *cognitive task analysis* (CTA) [Hof98]. Therefore, instead of aiming at knowing how users experienced the system, we wanted to reveal what was going on in their minds once they were exposed to one of the RS strategy utterances.

- *Question 7.* What was the first thing that came across your mind when you first heard the prompt <—*RS prompt transcript*—>?
- *Question 8.* Did you feel like being pushed to say more? Did you feel this utterance was motivating you to say more?
- *Question 9.* Were you aware of being expected to say more information on the branch to find?

- *Question 10.* Do such prompts like <—*RS prompt transcript*—> sound acceptably to you?

The set of "CTA-like" questions evolved during a preparation phase with fictional *timetable* dialogues. Because we needed to blend two mutually contradictory procedures, particularly a face-to-face CTA interview with a hidden "Wizard of Oz-like" (WoZ) simulation, we approached this phase with two pre-composed dialogues printed on a paper to satisfy demands on both of the procedures. Thus, the interviewer was the uncovered wizard as well – for our purposes, however, the CTA interview had a higher priority over a regular WoZ simulation. Afterwards, there were N=6 persons from the IT realm with no previous experience in human-computer interaction. They all were given the two pre-composed dialogues, one with the RS strategy involved and one without it (in this order). The evolution of the dialogues proceeded in a turn-by-turn manner, that is the interviewer read up the system utterance, one at a time, and then waited for the person's reaction. The interviewer's main goal was to ask the persons what they think they are asked about and what they can say at each moment in the dialogue. The stress was, of course, put on the system turns with the RS strategy applied. (The most interesting response was that the user would have assumed the system was joking and would immediately hang up.) Once the interviewer has gathered enough information, the expected reaction in the dialogue was revealed to the person and the interviewer eventually posed additional questions. No earlier than by now the interviewer continued in the dialogue by reading up the next system utterance. The results of these individual interviews constitute the basis of predefined answers to Questions 7, 8, and 10 (see Appendix A.4; in case of DORA, Question 9 then ranged over the five predefined values used already for Questions 1…6).

Hence, to summarize the set of measures, our investigation leans mainly on parameters that the system could log automatically, without the need of hand labeling them. Table 5.1 illustrates the type of information that was accumulated at the end of each experiment. Each row in the table represents a user for whom the interaction with the agent was logged along with values for the measures discussed in this section. However, at this stage, it is impossible to prejudge which measures contribute to the user satisfaction most. Below, we therefore use two dialogue system evaluation frameworks to tell us which measures have merit, and to quantify their relative importance.

5.3 Results and Evaluation

The experimental data consists of N=9 users who produced in total M=19 relevant dialogues.[9] We in advance say that this number is a huge disappointment, and therefore, analyze the reasons that could have led to it in the next section. Afterwards, in Sections 5.3.2 and 5.3.3, we process the results formally, making them a subject of the ANOVA and PARADISE frameworks. We conclude with

9 We do not consider dialogues that were either abandoned or crashed. If a user solved one task more than once, only the latest interaction is considered as relevant.

Table 5.1 Overview of logged measures about dialogue sessions; US = User Satisfaction, ET = Elapsed Time, STs = System Turns, TOs = Timeouts.

User	**Task**	**Dialogue turns**	**US**	**κ**	**ET**	**STs**	**TOs**	**…**
1	1	S_1–U_1, S_2–U_2, S_3–U_3, …	*us*	*k*	*et*	*st*	*to*	…
1	2							
1	3							
…								…

Section 5.4 that gives an informal view at the results and discusses interesting aspects observed in individual dialogues.

5.3.1 Reasoning about the Low Attendance

Before engaging with the system evaluations, let us try to find reasons for the low number of volunteers willing to participate in the project. We have pinpointed three main reasons which serve us as lessons learned for further experiments of similar kind.

First, our experimentation lacked any form of *warm-up session* [Smi92] that could have helped users with getting them more familiar with the system. In many cases, these sessions are conducted with an instructor who is to explain the essence of the system, present basic keywords, and help subjects do their first steps in the system which usually involves accomplishing a simple task [Ngu06a, Sti01, Ste07, Smi92, Sto12]. Such conducted warm-up session mainly targets more complicated or highly specialized systems. For less sophisticated systems like ours, a simple static dialogue transcript usually suffices [Wal98]. As our system was accessed by remote users by phone, we could not have taken the human instructor way. Neither did we, however, consider the latter way as we wanted to imitate a queuing system for which users were not required to read through illustrative dialogues (instead, such system was to adapt based on the current qualities of a dialogue). Another reason is that we did not want to influence user's way of interaction by showing them an example. On the other hand, this assumption might have been in collision with a HCI-related factor that people do not trust unknown and attempt to avoid it [Yan95]. This factor could have therefore discouraged potential users by the fact that they did not know what was expected from them (our websites were accessed by more than 150 visitors). Hence, if we were to repeat the experiment, we would include a warm-up session by opting for a dialogue transcript as an example.

The second aspect that could have impaired the experiment attendance were technical difficulties that the system suffered from during the first weeks of its running. More specifically, we dealt with typos in string literals ("*#Gladbeck*" instead of "*Gladbeck*", and "*strasse*" instead of "*Strasse*"), malformed communication messages between the XML Client and XML Server (see Fig. 5.2), and most importantly database communication failures – a hidden problem

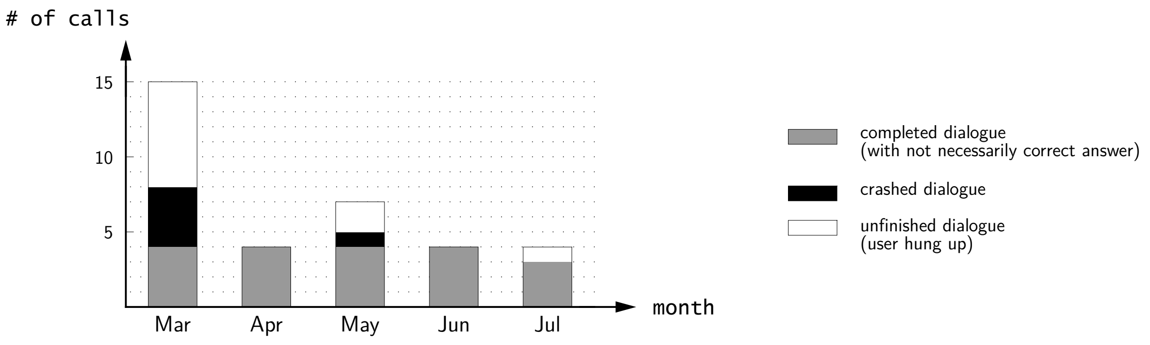


Fig. 5.5 Number of calls distribution over the testing period between March and July 2014.

that was discovered no earlier than after a month the system was running. The database difficulties prevented users from submitting their answers acquired from the system along with the final questionnaire as well. Looking at the dialogues distribution over the testing period (Fig. 5.5), we believe this was the most significant factor that contributed to the low attendance. Thus, after a promising start in March, we suddenly observed a decline in the number of dialogues (originally ascribed to the holiday season). In parallel to the decline, we also observed an increased number of users who had finished one task but did not continue in the experiment.[10] We believe the database problems have deteriorated the system initial reputation to the extent which it was unfortunately unable to recover from over the remaining months.

There is one more aspect that does not regard the influence on the low attendance in the experiment directly, however, has impacts on how we now can evaluate the results – the final questionnaire itself. Let us recall that the experiment was designed to show the final questionnaire to users who had responded to all three tasks. Nonetheless, as some users did not continue in the experiment after they had submitted the answer to the first task,[11] we unfortunately cannot evaluate their partial accomplishments *formally* (Sections 5.3.2 and 5.3.3); we, however, include these dialogues in our *informal* discussion (Section 5.4). The motivation for leaving the final questionnaire to the end was to evaluate the system as a whole. We found inspiration for this decision in [Ngu06b] where ten users were to cooperate with the system in solving twelve different tasks. Despite some of them did not finish them all, it was the human factor who then immediately decided whether the dialogues observed are worth an evaluation. For our case, one way to work around the computer factor would be to let users submit the questionnaire and check for fulfillment of all tasks manually. A better way would be to make users submit the questionnaire for every task [Wal98, Lit02]. That way, each user's response to the questionnaire would

10 As we later tracked down, the problem was in the way the XML Server was connecting to the database. Once connected on its start, it was supposed to remain connected until the end of July. However, the database had set a disconnection timeout that fired after approximately three idle days (i.e., if no call was received by then). For us, the server was always working as expected when checking its functionality minutes after re-starting it. We then eventually received a complaint from a user calling few days later.

11 We believe the reason might have been the overcomplicated formulation of Tasks 2 and 3.

reflect the most recent dialogue she or he took part in; this could be combined to evaluate their overall experience with the system or the questionnaire extended with additional questions.

5.3.2 Evaluation Using ANOVA

To begin with our formal evaluation, let us first briefly describe the notion of using ANOVA (ANalysis Of VAriance) in the dialogue systems realm [Coh95]. In general, ANOVA is a statistical method to compare differences of means among several measures. It does so by looking at variation in the data and where that variation is found. More specifically, it compares the amount of variation *between* groups with the amount of variation *within* groups. When we take samples from a measure, we expect each sample mean to differ simply because we are taking a sample rather than measuring the whole population. Thus, we always expect there to be some differences in means among different groups. Like other statistical tests, we use ANOVA to calculate a test statistic (the F-ratio, or F-test) using which we can obtain the probability (the p-value) of obtaining the data assuming the null hypothesis. The null hypothesis states that all population means are equal, so there is no difference among groups. A significant p-value (for which $p < \alpha$, where usually $\alpha = 0.05$) suggests that at least one group mean is significantly different from the others. In such case, we say that we reject the null hypothesis.

To gather at least two groups of measures, we developed two versions of the dialogue agent: *Agent A*, which was equipped with the "common" set of strategies (NS+OS combination), and *Agent B*, which employed the Rogerian strategy where applicable (NS+RS combination). Each calling subject was assigned one of the agents on the basis of the phone number: even numbers were assigned Agent A whereas odd numbers Agent B.

Our experimental design then consisted of two factors: strategy (with nominal conditions *Agent A* and *Agent B*) and task (with nominal conditions *Task 1*, *Task 2*, and *Task 3*). Each of our measures was analyzed using a two-way, repeated-measure ANOVA for these two factors. For each result we report F and p values indicating the statistical significance of the results for $\alpha = 0.05$. Apparently, effects that are significant as a function of strategy indicate differences between the two strategies, whereas effects that are significant as a function of task are potential indicators of learning [Wal97].

Based on our results, the ANOVA test revealed that the amount of eigen information supplied by the user per turn is a function of strategy ($F(1{,}6) = 6.23$, $p = 0.047$). This is a desired and hoped result, indicating that the RS strategy had a positive impact on the information exchange rate: while the average amount of supplied eigen information was 1.06 pieces per turn for Agent A, it was 1.16 for Agent B (timeouts with zero exchange rate are not counted in).

In tandem with this observation, we also hoped to reveal a significant variance between elapsed time per dialogue and strategy. The F-test nevertheless did not reveal such variance ($F(1{,}6) = 0.02$, $p = 0.886$); the same then applies to the number of turns per dialogue ($F(1{,}6) = 1{,}15$, $p = 0.325$). Despite that, we are not disappointed to fail to reject the null hypothesis, i.e. that the strategies are equivalent in terms of the amount of time or the number of turns, respectively. We believe that the lack of an effect on elapsed time reflects that the dominant time aspect were longer utterances in Tasks 2 and 3, informing about opening hours in Task 2 and using opening hours to refer to known branches in Task 3.[12] On a similar note, we believe that we failed to reject the null hypothesis that the strategies are equivalent in terms of impacts on the number of turns, due to the lack of enough tested subjects during our experimentation.

A similar result is also obtained when computing how the number of RS strategy applications across *all* subjects and tasks in Table A.3 correlates with the number of dialogue turns, which provides us with the number 0.063. Taking again into account the constrained number of subjects participating in the experiment, this cannot be understood as a cardinal answer that the RS strategy has a minor negative impact on the dialogue flow. To make such claim, more participants would be necessary to take part in the experiment (and eventually additional experiments conducted across different domains). With these results, we only can state that Agent B outperformed Agent A in terms of the average number of dialogue turns with $AvgTurns(A) = 12$ and $AvgTurns(B) = 10$, this way being 17% more effective when making use of the RS strategy. By excluding both of the extremes from each of the groups, i.e. the shortest and the longest dialogues, we obtain $AvgTurns(A) = 9.33$ and $AvgTurns(B) = 10.22$ (improvement by 8.70%), thus a less clear result, comparable with those received from ANOVA.

For completeness sake, the ANOVA test also showed two interesting but statistically insignificant covariances. The first of them is a dependence between the number of turns and a task ($F(2,6) = 3{,}68$, $p = 0.090$). Calculating the average number of turns per each task gives us 8.13, 15.17, and 10.00 for Task 1, 2, and 3, respectively. Thus, there is an apparent peak for Task 2 which can be explained as the task being too complex (or fuzzy formulated) for users to solve. We think users were confused by the goal of finding the closest branch using an underlying map. While we hoped this task to be the most enjoyable one as it was no more complex than the other two, i.e. it sufficed to provide the system with all data stated in the formulation and then just check the returned answer against the map, virtually each user chose the longer way of exploring depicted streets individually, which naturally bloated the dialogue length. In addition, despite different complexity, the reason that Tasks 1 and 3 had each approximately the same number of turns can indicate the above mentioned aspect of users learning to use the system.

The second statistically insignificant covariance returned by ANOVA has been observed between the number of timeouts and the strategy used ($F(1,6) = 5.05$, $p = 0.069$). Calculating the average number of timeouts per strategy provides us with 3.98 for Agent A and 1.75 for Agent B. If we exclude both of the extremes from each group (i.e. the minimum and maximum number of timeouts per dialogue), we obtain 3.22 for Agent A and 1.50 for Agent B. The reason for this observation may be that, once heard the RS strategy prompt, users tended

12 E.g., "There is a Horst branch in Gelsenkirchen, opened daily between 9:00 and 17:00 o'clock, another one opened daily 24 hours, and yet another one opened Monday, Wednesday, and Friday between 9:00 and 16:00, and Tuesday and Thursday between 9:00 and 18:30 o'clock."

to repeat information they already provided earlier, instead of letting their turn timeout. The following is one of such cases:

$S_{n\text{-}2}$	NS	…There are too many branches. In case you know it, please say which city the branch should be located in. —*Pause*— If you don't wish a particular city, continue with saying No.
$U_{n\text{-}2}$		*_declarative_*(*City*:"Gladbeck")
$S_{n\text{-}1}$	RS	Please specify the searched branch further.
$U_{n\text{-}1}$		*_timeout_*
S_n	NS	I've so far understood you are looking for a branch in Gladbeck. I need now more information. I suggest you to say the street, but you can say also the place, name, or equipment. (Bisher habe ich verstanden, dass die Filiale in Gladbeck sein soll. Ich brauche jetzt weitere Informationen über die gesuchte Filiale. Ich schlage die Straße vor, aber Sie können auch Ort, Name der Filiale oder Ausstattung ändern und mir die entsprechende Information jetzt geben.)
U_n		*_declarative_*(*Equipment*:"Geldautomat")
S_{n+1}	RS	Ok, you are searching for a branch with an ATM. Please tell me more about the branch. (Ok, Sie suchen eine Filiale mit Geldautomat. Geben Sie mir bitte weitere Informationen.)
U_{n+1}		*_declarative_*(*City*:"Gladbeck")
S_{n+2}	RS	Please specify the searched branch further.
U_{n+2}		*_timeout_*

As the number of unspoken mandatory information for a task was shrinking, a RS repeated application was followed by a repeated known information, and then eventually by a timeout. However, an eventual timeout is necessary to be accepted as an inevitable response in some cases. It is a sign that the user does not know what to say (more), which was well observed even in dialogues conducted by Agent A. In our case, whenever a timeout arose after an application of the RS strategy, Agent B imediatelly switched to the NS strategy whose prompt already gave the user specific information on how to continue in the dialogue.

Thus, we believe that the rejections we made in this section are a matter of utterance design rather than the underlying notion of the RS strategy itself. Let us also recall from Section 3.9.2 that the RS strategy was always used during our experiments only in cases when

- *a dialogue was progressing well*, i.e. no timeouts were observed by the agent, indicating that the user knew what to say; this situation is illustrated by the following snippet:

S	RS	Hello, thank you for your call and welcome back. How may I help you?
U		*_declarative_*(*Branch*(*StreetName*:—))
S	RS	There are too many branches. In case you know it, please say which city the branch should be located in. —*Pause*— If you don't wish a particular city, continue with saying No.
U		*_declarative_*(*City*:"Herten")
S	RS	So you are searching for a branch in Herten. Please tell me more about the branch.

U		_declarative_(*Equipment*:"Geldautomat" , *BranchOffice*:"Hauptstelle")
S	RS	Unfortunately there are no Hauptstelle branches[13] with an ATM. Do you insist on the ATM? (Leider existieren keine Hauptstelle Filialen, die Geldautomat haben. Bestehen Sie auf Geldautomat?)
U		_declarative_(_agree_)

- *the task to discuss was unknown*; however, this application virtually always resulted in a timeout or repetition of already known information:

S	RS	Hello, thank you for your call (...). How may I help you?
U		_declarative_(_agree_ , *City*:"Gladbeck")
S	RS	Ok.
U		_timeout_
S	NS	I can offer you the following services: account balance, branch addresses, and branch opening hours.
U		_timeout_
S	NS	I can offer you the following services: account balance, branch addresses, and branch opening hours.
U		_declarative_(*City*:"Gladbeck")

- *the agent missed two or more pieces of information*; already exemplified in above dialogues.

5.3.3 Evaluation Using PARADISE Framework

The above ANOVA test indicated differences between the two dialogue agents in several aspects but it was unable track down the most important performance features. We therefore make use of the PARADISE framework. The PARADISE framework (PARAdigm for DIalogue System Evaluation) [Wal98] emerged as a response to the lack of complex techniques to evaluate a dialogue system performance. In a nutshell, the framework has been devised to compare different dialogue strategies by providing a task representation that decouples *what* an agent needs to achieve in terms of task requirements from *how* the agent carries out the task via dialogue [Wal98]. The idea proposed is that the system performance can be correlated with a meaningful external criterion such as usability. Because user satisfaction ratings are commonly used as an indicator of the usability, the user satisfaction is the agent's top level objective to be maximized. There are three main assets the framework consists of:

- *Task success*. Each task is represented using an *attribute-value matrix* (*AVM*), representing information that must be exchanged between the user and the system at the end of a dialogue, regardless of the strategy used. The AVM is in turn used to build a *confusion matrix*, counting

13 Although Daisy allows for branching in utterances, we omitted the case of "*Hauptstelle Filiale*" as none of the tasks was on finding the main branch; luckily, such word combinations occurred merely a handful of times in our sessions.

how many times the system was supplied correct and incorrect attribute values in the dialogue corpora. Finally, the *Kappa coefficient*, κ, can be calculated from a confusion matrix, summarizing how well the agent achieves information requirements of a particular task, taking into account that the size of the value set influences the rate of misrecognitions.

- *Dialogue costs.* In general, any user or agent's behaviour that should be minimized is to be considered a dialogue cost. Since it is not clear in advance which cost factors may be the strongest contributors to the user satisfaction, it is important that a wide range of these measures is used during an experimentation [Wal98]. For instance, except for the **Kappa**, **Rogerian Turns**, and **User Satisfaction** measures in Section 5.2.3, all remaining variables can be considered dialogue costs of either *quantitative* nature (**Elapsed Time**, **User Turns**, etc.) or *qualitative* nature (**Timeouts**, **Help Requests**, etc.).
- *User satisfaction.* Users who take part in the experimentation are asked to fill out a survey. They are given a set of questions with predefined answers, each of which maps to a certain value, matching the degree to which they agree with a statement about the system. Cumulating their responses, the overall user satisfaction is approached.

To put all of these parts into perspective, it is proposed a weighted linear combination of the task success and dialogue costs to model (and possibly also predict) the user satisfaction, i.e.

$$Performance \approx UserSatisfaction = \alpha \cdot \mathcal{N}(\kappa) - \sum_{(i)} w_i \cdot \mathcal{N}(c_i)$$

where α is a weight on κ, the cost functions c_i are weighted by w_i, and $\mathcal{N}$ is a Z-score normalization function [Coh95]. The normalization function is used to overcome the problem that the values of c_i are not on the same scale [Wal98] (e.g., elapsed time is measured in seconds whereas timeouts are calculated in terms of number of utterances). If the values are not normalized, the coefficients α and w_i will not reflect the relative contribution of each factor to performance. This is easily solved by normalizing each factor x to its Z-score, i.e.

$$\mathcal{N}(x) = \frac{x - \overline{x}}{\sigma_x}$$

where σ_x is the standard deviation for x.

Hence, in our application of the PARADISE framework, we first calculated the *Kappa coefficient*, indicating the overall task success, as

$$\kappa = \frac{P(A) - P(E)}{1 - P(E)}$$

where $P(A)$ is the proportion of times that the AVMs for the actual set of dialogues agree with the AVMs for the scenario key, and $P(E)$ is the proportion of times that we would expect the AVMs for the dialogues and the keys to agree by chance. Table A.1 and Table A.2 show the two confusion matrices over all subjects and tasks that correspond to Agents A and B. Based on these confusion matrices, we can calculate individual **Kappa** coefficients as $\kappa_1 = 0.588$, and $\kappa_2 = 0.793$. Table A.3 then summarizes measures for relevant parts of dialogues considered for evaluation. We do not consider dialogues that were either abandoned or crashed. If a user solved one task more than once, only the latest interaction is considered. Note that the correctness of results returned by the system as response was irrelevant to us – we only were interested in observing reactions to the RS strategy rather than serving users with correct information.

To determine the unknown coefficients α and w_i, we use the linear regression. However, before that, we first have to exclude the number of turns (**Turns**) from the set of measures as it highly correlates with both **Elapsed Time** ($corr > 0.936$) and **Timeouts** ($corr > 0.81$). We in turn also exclude **Elapsed Time** itself for correlating with **Timeouts** ($corr > 0.66$), **Restarts** for correlation with the number of **Help Requests** ($corr > 0.887$), and finally also **Kappa** for correlation with the **Average Turn Length** ($corr > 0.681$). These exclusions are made due to the fact that correlation above 0.70 can affect the coefficients of the linear regression [Mon80]. In the subsequent linear regression, the **User Satisfaction** is treated as the predicted factor whose variance is accounted for by the remaining loosely correlated measures. For significant contributors to the predicted factor, the linear regression produces coefficients constituting the following performance estimation function (for confidence level of 90%):

$$Performance \approx UserSatisfaction = -0.49 \cdot Timeouts + 0.38 \cdot AvgTurnLength$$

Accounting for 85% of variance in **User Satisfaction**, the formula tells us that users were satisfied the more the less timeouts they experienced and the "longer" utterances they were saying. In other words, the satisfaction linearly depends on whether users knew what to say: the less they knew the less they were satisfied, and the more they knew what to say per turn the more they were satisfied. It is interesting to note that although qualitative measures have typically been assumed to be the most important factors in user satisfaction [Wal98, Lit02], the relative magnitude coefficient for **Timeouts** in the above equation is in absolute value greater than the coefficient for the **Average Turn Length**. This implies that not knowing what to say has stronger importance for users whereas knowing what to say is assumed rather common.

It is also interesting to note that according to the formula, the **Rogerian Turns** measure we most focused on does not seem to affect the **User Satisfaction** ($p > 0.32$). That is, although we did not interview the users of the RS strategy, the formula appears to confirm the informal results of interviewing users during the CTA sessions (see Section 5.2.3 and Appendix A.4), implying that users are not annoyed by the RS strategy exceptional utterances and can rather handle them (the strategy ended with a timeout in 8 out of 26 cases of application; it managed to elicit *new* information in 10 out of the 18 non-timeout cases). Therefore, the factor with the most impact on user satisfaction remains the quality of a dialogue after all. From this standpoint, the RS strategy appears to be an acceptable way of eliciting more information. As already mentioned in the previous section, Agent B outperformed Agent A in terms of the number of timeouts per dialogue (1.50 versus 3.22), the length of a dialogue (9.33 versus 10.22), and also Kappa (0.793 versus 0.588). As no detailed feedback from users was collected, we only can guess that better average Kappa coefficient was obtained due to users being not pushed to say a specific piece of information at a particular point (e.g., name of a branch), but instead, were free to say virtually anything relevant in any order. In contrast, Agent A's insisting on particular information had sometimes the negative effect on user saying "at least something" to satisfy the system demand (e.g., satisfying the branch name with saying "*main branch*", *Hauptstelle*, which was incorrect in any task). Of course, Agent A offered the possibility to skip that information elicitation (e.g., "*If you don't wish a particular city, continue with saying No*"). Adopted by the user, such behaviour prolonged the dialogue.

Finally, plugging our experimental data back into the above performance function shows that the mean performance of Agent A is lower than the mean performance of Agent B, hence Agent B performed better than Agent A. Appendix A.5 extends this formal evaluation by briefly describing an alternative way of evaluating spoken dialogue systems using the three-tiered methodology by Stibler and Denny [Sti01].

5.4 Remarks

Before concluding the evaluation, let us make several notes about the Rogerian strategy. As already mentioned in Section 4.9 on strategies, the Rogerian strategy acts as a "prolonged user-initiative strategy". That is, while the common mixed-initiative approach assumes that user's initiative is always followed by the system initiative, the Rogerian strategy modification suggests that the user can keep the initiative as long as she or he knows what to say. In such cases, the system hands back its turn by speaking with mere hints or context-free sentences. Obviously, this is a risky approach that may result in the opposite effect of making the interaction even longer. We observed users either repeating the already said or letting their turn timeout; examples of such behaviour have already been given above.

On the other hand, not knowing what to say to a Rogerian prompt is a situation comparable to not knowing what to say to an open-ended prompt at the beginning of a dialogue. For instance, one of our non-Rogerian sessions started with:

S_1	OS	Hello, thank you for your call (...). How may I help you?
U_1		*_declarative_*(*City*:"Gladbeck")
S_2	NS	I can offer you the following services: account balance, branch addresses, branch opening hours.
U_2		*_timeout_*
S_3	NS	I can offer you the following services: account balance, branch addresses, branch opening hours.
U_3		*_timeout_*

S_4	NS	I can offer you the following services: account balance, branch addresses, branch opening hours.
U_4		*_declarative_(Branch(StreetName:—))*

It took four system turns to find out what the user wished. In contrast to this, with the Rogerian strategy enabled, the same task was solved by another user as follows:

S_1	RS	Hello, thank you for your call (...). How may I help you?
U_1		*_declarative_(City:*"Gladbeck")
S_2	RS	Ok.
U_2		*_timeout_*
S_3	NS	I can offer you the following services: account balance, branch addresses, branch opening hours.
U_3		*_declarative_(Branch(StreetName:—))*
S_4	RS	There are too many branches in Gladbeck. Please specify the searched branch further.
U_4		*_declarative_(BranchOffice:*"Filiale_Bertlich")
S_5	RS	Ok, you are searching a Bertlich branch. Do you need it to have a specific equipment? (Ok, Sie suchen eine Bertlich Filiale. Soll die Filiale eine bestimmte Ausstattung haben?)
U_5		*_declarative_(_disagree_)*
S_6	RS	There is a Bertlich branch located in Sandstrasse 177 in Gladbeck and another one in Voßstrasse 200. Can I help you find another branch? (Es gibt eine Bertlich Filiale mit der Adresse Sandstrasse 177 in Gladbeck und eine andere mit der Adresse Voßstrasse 200. Kann ich Ihnen helfen, eine weitere Filiale zu finden?)
U_6		*—dialogue continues with user asking for repetition—*

Yet in contrast to this, the Rogerian strategy can be considered an inappropriate way for information elicitation in the following session which was terminated by the user preliminary:

S_1	RS	Hello, thank you for your call (...). How may I help you?
U_1		*_declarative_(City:*"Gladbeck")
S_2	RS	Ok.
U_2		*_timeout_*
S_3	NS	I can offer you the following services: account balance, branch addresses, branch opening hours.
U_3		*_declarative_(Branch(StreetName:—))*
S_4	NS	There are too many branches in Gladbeck. Please specify the street that the branch should be located in. —*Pause*— If you don't wish a particular street, continue with saying No.
U_4		*_declarative_(_disagree_)*
S_5	RS	Please specify the searched branch further.
U_5		*_timeout_*
S_6	NS	I've so far understood you are looking for a branch in Gladbeck. I need now more information. I suggest you to say the street, but you can say also the place, name, or equipment.
U_6		*_declarative_(_agree_)*

S_7	RS	Please give me some more information about the branch.
U_7		*_timeout_*
S_8	NS	I so far understood you are looking for a branch in Gladbeck. You now may say or change the location, name, or equipment of the branch. (Bisher habe ich verstanden, dass die Filiale in Gladbeck sein soll. Sie können Ort, Name der Filiale oder Ausstattung jetzt sagen oder ändern.)
U_8		*_timeout_*
S_9	NS	I so far understood...

After several more turns, the user gave up and hung up. However, what is observable here is that the system repeatedly attempted the Rogerian strategy and narrow strategy to elicit more information from the user. The first timeout in U_2 caused the first switch, after which the system provided a short help. As the subsequent turns U_3 and U_4 supplied required information, the system decided to switch back to the Rogerian style in S_5. However, this attempt was responded to with a timeout in U_5. The turn S_6 therefore summarizes so far known and provides help. User continues with an unexpected agreement; from the Rogerian strategy point of view, this is again evaluated as a sign of progress in the dialogue, which is in conformance with the strategy current specification in Section 4.9.[14] The result is another switch towards the Rogerian strategy in S_7 that is responded to with a timeout, etc.

Thus, the above "good and bad" examples imply that the Rogerian strategy should have even more restricted conditions of use. Obviously, we deal here with a blind adaptability approach without regularly maintaining user's interaction habits. A user model would therefore represent a valuable additional input to the strategy evaluator (and in fact any other strategy). Had that been the case, repeatedly experiencing the Rogerian strategy to fail, the system would eventually stop using it to improve the dialogue progress. However, as we mentioned at the beginning of this chapter, our system essence was to imitate a queuing system rather than serving as a personalized agent. We will briefly discuss user modeling possibilities in the below Section 6.2 on future work.

5.5 Summary

In this chapter, we used the general collaborative dialogue framework Daisy to create a conversational agent DORA providing bank branch information. We made use of the Domain Editor to model the agent's behaviour (DDM, plans, and dominance). With the kind support of Sympalog Voice Solutions, GmbH company, we delivered the system to real users who could called it by using their ordinary cell phones. For each individual call, we automatically logged conversation parameters, like semantics, system prompts, communication messages, etc. Users who completed all experiment scenarios judged the system on the Likert scale, and gave us this way underlying material for evaluating the Rogerian strategy, which is an embedded part of the Daisy framework.[15] The

14 Apparently, this evaluation should be further precised so that the agreements and disagreements have a local logical relevance only.

15 Can be turned off and on using the *SetProperty* API function in Table A.4.

evaluation was to an extent limited with what measures we were supplied from the SymBase platform (for instance, we were not supplied the silence period before the first word in user responses, as the platform does not calculate such measure; for the platform overview see leaflet attached on the CD). Despite that, we were able to evaluate two versions of the banking agent, the so called Agents A and B. We first tested the experimental data against the ANOVA F-test to find out that there was a trend towards the two agents being different. Our next investigation continued with thorough performance analysis using the PARADISE framework, whose results revealed that the user satisfaction linearly depended on whether users knew what to say: the less they knew the less they were satisfied, and the more they said per turn the more they were satisfied. We do not find the results disappointing, provided that we dealt here with a blind adaptability approach with no modeling of user's habits or preferences. We therefore discuss possible extensions to the Daisy framework in the Future Work section in the next chapter.

Chapter 6

Conclusion

This work has concerned with general collaborative dialogue management through a spoken natural language. After specifying and explaining the approach, we demonstrated its usability in a banking domain, developed on the basis of a real dialogue system. Our reimplementation missed some non-interactive parts which were of little use for our purposes. The resulting design featured services on bank branch information and account balance. To finish the development life-cycle, we deployed the system to real users who interacted with it by giving it a call (thanks the kind support of *Sympalog Voice Solutions, GmbH*[1]). To comply with the SymBase platform standards, the dialogue agent was wrapped in a thin XML server. The dialogue agent itself acted as the central component for maintaining a coherent spoken dialogue with the user as well as communicating with the back-end assets (database and application libraries).

We proposed an agent-based approach to dialogue management which meets the requirements for developing multi-domain spoken dialogue systems to host concurrent dialogue sessions. Our decision for the agent-based approach was motivated by disadvantages found in other approaches. While most of the existing methods for dialogue management are suitable for simple and highly constrained tasks, in which complex interactive behaviour has to be solved using an overhead of development, our aim was to create a general framework with most of the common behaviour available "out-of-the-box".

Our approach employs the BDI architecture to dialogue agent decomposition. To model a dialogue, we use existing work in speech act theory and discourse analysis, namely the concepts of conversational acts and discourse segment intentions. To manage a dialogue, we consider it as a rational action, i.e. a product of a goal-directed behaviour. Therefore, the dialogue model is explicitly encoded in agent's plans, each specifying a set of steps to solve a particular task. The dialogue control flow is then derived automatically as the result of the BDI interpretation cycle. This, among other things, includes trading reactiveness

1 *http://sympalog.de*

in order to fulfill user's requests, and proactiveness such as for error recovery. The conversational context along with other domain-specific knowledge are maintained as agent's internal beliefs.

6.1 Contributions to Dialogue Management

There are several contributions to the realm of dialogue management. Overviewing them chronologically, the first contribution has been made to the frame-based management in which nested frames have been extended with a system of journal to automate some commonly repeating routines that otherwise have to be handled manually – they foremost address causality tracking and error recovery. There are proposed several embedded algorithms to manipulate the journals and to override the underlying interpretation mechanism. The system of journals can be applied to any frame(s) whose interpretation mechanism is stateless (i.e., including VoiceXML and its FIA; see specification on the attached CD).

Later on, we abandoned frames and refocused on the more attractive BDI agent-based management. To account for the beliefs component, we turned to the SIL formalism which we have found very interesting as it can capture soft details in information conveyed towards a conversational agent (among other applications). However, its original proposal is quite impractical as for representing information in a cooperative agent as many regular dialogue operations become cumbersome or unnecessarily complicated. The main modifications to SIL therefore address: (1) collectability of objects (e.g., *bus* and *train* are collectable as *transporation means*), (2) suppression of meaning and taxonomy of objects (we argue that objects regain these properties within agent's plans), and (3) more strict organization of objects (otherwise we face an overhead of rules to describe exceptions under which structurally incompatible objects become compatible, see example at the beginning of Section 4.5.1). The strict-organization modification is susspected to be dropped or relaxed, however it has been currently preserved due to time reasons (i.e., to facilitate the proposal of algorithms to deal with dialogue error recovery through user's corrections). Hence, in conjunction with accompanying embedded algorithms, the resulting approach again constitutes a standalone framework, independent on the hosting conversational agent. We subsequently use its two individual instances in the hosting Daisy framework, specifically to create the two-layered approach to dialogue context.

Apart of formally designed approaches, we also demonstrated the potential of an experimental Rogerian strategy. The strategy was inspired by idea of the so called Rogerian therapy, particularly that "clients are better helped if they are encouraged to focus on their current subjective understanding rather than on some unconscious motive or someone else's interpretation of a situation". In other words, we tried our experiment subjects not to push them to say information we wanted to hear from them, but instead, gave them reasonable amount of freedom to express what they wanted. Surprisingly, this way resulted in shorter dialogues compared to cases in which users were to follow system demands. Naturally, we put strong constraints to this strategy. First, the domain in question must be known to the users so that they can describe objects in the domain without aid of the system. Second, the strategy can be used only in special situations during a conversation. Assuming the domain is commonly known, these situations include, for instance: user knows what to say, user's initial intention is unclear, or agent misses more than one information.

6.2 Future Work

Let us continue with the Rogerian strategy and propose modifications to it, and let us then proceed to the Daisy dialogue framework itself.

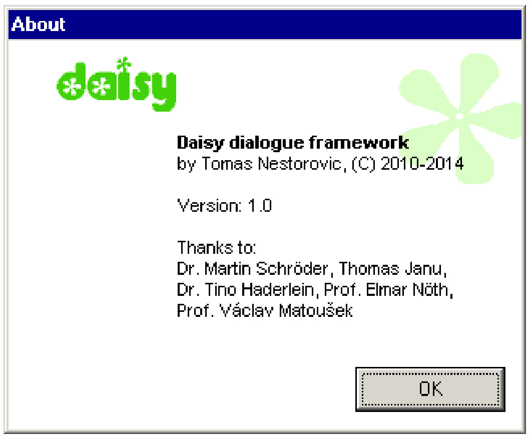
According to our experience with the Rogerian strategy, we think the following two statements fit its current state:

- The banking domain is relatively a simple one and more complex domains with more information to discuss would be valuable.
- The cases of the Rogerian strategy application should be further restricted and supported by user modeling.

As for the first point, we need to search for a domain that meets the condition of being commonly known to potential users. A natural candidate is again an information system which may or may not consist of multiple domains; in fact, if there was such a chance, the latter option would seem to fit the bill better as it naturally bloats the number of objects to discuss. It is assumed that in a more complex domain with a rich information environment the Rogerian strategy will succeed in higher rates due to higher chances that users will know what to say. However, with a more complex domain there is also the ASR performance point of view and one may argue that for complex domains with high variability in input (i.e. utterances which possibly can carry a lot of different information) the ASR may fail to recognize reliably as it is exposed to a too wide portion of a language. A solution to this pitfall is in using context-aware Rogerian prompts. As described, such prompts suggest the next topic to discuss, that way implicitly constraining user's possible responses to a reasonable portion of a language.

With the second point, we propose a user model to be passed over to the Rogerian strategy for evaluation. Lot of user modeling methodologies have been developed over the past years. While some follow a domain-dependent features approach to predict the system utterance that best corresponds with user's behaviour [Chu00, Kom03, Wal00], we would rather like to follow an approach containing domain independent features as Hjalmarsson demonstrates in her work [Hja05] by making an extensive use of the PARADISE framework.

As for the Daisy framework itself, the most challenging feature to incorporate is the representation and management of negative information. This spans far beyond the elemental information combination cases depicted in Table 4.4 in Section 3.5.6. Although we begun to take the first steps in implementing this feature, it has remained unfinished due to time reasons and is therefore blocked in the current version of the framework on the attached CD. Less mentally expensive updates to the framework then account for revising the two-layered approach implementation (as some potential simplification are suspected), and overall optimization as for speed, memory usage, and standard Windows libraries usage.



Daisy dialogue framework
by Tomas Nestorovic, (C) 2010-2014

Version: 1.0

Thanks to:
Dr. Martin Schröder, Thomas Janu,
Dr. Tino Haderlein, Prof. Elmar Nöth,
Prof. Václav Matoušek

Fig. 6.1 The standard built-in "About" dialogue box to incorporate information about the framework into further Daisy-based applications or possibly custom domain editors; invoked using the *ShowAbout* API function, see Table A.4.

Bibliography

[All92] Allwood J., Nivre J., Ahlsén E., "On the Semantics and Pragmatics of Linguistic Feedback." *Journal of Semantics 9*, pp. 1–26. 1992.

[All01] Allen J., Ferguson G., Stent A., "An Architecture for More Realistic Conversational Systems." In *Proc. of Intelligent User Interfaces.* 2001.

[Ara03] Araki M., Kaga A., Nishimoto T., "Comparison of ‚Go back‘ implementations in VoiceXML" In *Proc. of Error Handling in Spoken Dialogue Systems*, pp. 31–34. 2003.

[Bel57] Bellman R., *Dynamic Programming.* Princeton University Press, Princeton, NJ, 1957.

[Bel89] Belina F., Hogrefe D. "The CCITT-Specification and Description Language SDL." *Computer Networks and ISDN Systems 16*, pp. 311–341. 1989.

[Boh07] Bohus D., Raux A., Harris T., Eskenazi M., Rudnicky A., "Olympus: An Open-source Framework for Conversational Spoken Language Interface Research." In *Proc. of Bridging the Gap: Academic and Industrial Research in Dialog Technology workshop at HLT/NAACL*, pp. 32–39. 2007.

[Boh09] Bohus D., Rudnicky A., "The RavenClaw Dialogue Management Framework: Architecture and Systems." *Computer, Speech, and Language 23*, pp. 332–361. Elsevier, 2008.

[Boy99] Boyce S., "Spoken Natural Language Dialogue Systems: User Interface Issues for the Future." *Human Factors and Voice Interactive Systems*, pp. 37–61. Kluwer Academic Publishers, 1999.

[Bra91] Bratman M., Israel D., Pollack M., *Plans and Resource-bounded Practical Reasoning.* MIT Press, Cambridge, 1991.

[Bui06] Bui T, *Multimodal Dialogue Management.* Technical Report, TR-CTIT-06-01, University of Twente, Centre for Telematics and Information Technology. Enschede, 2006.

[Cat02] Catizone R., Setzer A., Wilks Y., *State of the Art in Dialogue Management.* Deliverable D5.1 of COMIC Project. 2002.

[Cen04] Cenek P., *Hybrid Dialogue Management in Frame-Based Dialogue Systems Exploiting VoiceXML.* PhD thesis proposal, VUT, Faculty of Information Technologies. Brno, 2004.

[Chu00] Chu-Carroll J., "MIMIC: An Adaptive Mixed Initiative Spoken Dialogue System for Information Queries." In *Proc. of Conference on Applied Natural Language Processing*, pp. 97–104. 2000.

[Chu05] Chu S., O´Neill I., Hanna P., McTear M., "An Approach to Multi-Strategy Dialogue Management." In *Proc. of INTERSPEECH*, pp. 865–868. 2005.

[Coh95] Cohen P., *Empirical Methods for Artificial Intelligence.* MIT Press, 1995.

[Eck95] Eckert W., *Gesprochener Mensch-Maschine-Dialog.* PhD thesis, FAU, Lehrstuhl für Mustererkennung. Erlangen, 1995.

[Fil05] Filisko E., Seneff S., "Learning Decision Models in Spoken Dialogue Systems via User Simulation." In *Proc. of AAAI Workshop on Statistical and Empirical Approaches to Spoken Dialogue Systems.* 2005.

[Fra93] Fraser N., "The SUNDIAL Speech Understanding and Dialogue Project: Results and Implications for Translation." *Aslib Proceedings 46*, pp. 141–148. 1993.

[Frø00] Frøkjær E., Hertzum M., Hornbæk K., "Measuring Usability: Are Effectiveness, Efficiency, and Satisfaction Really Correlated?" In *Proc. of the SIGCHI Conference on Human Factors in Computing Systems*, pp. 345–352. 2000.

[Ger10] Gervás P., Amores G., Hervás R., Pérez G., Bautista S., Francisco V., Manchón P., "Integrating Aggregation Strategies in a In-Home Domain Dialogue System." In *Proc. of Text, Speech, and Dialogue*, pp. 499–506. Springer Verlag, 2010.

[God96] Goddeau D., Meng H., Polifroni J., Seneff S., Busayapongchai S., "A Form-Based Dialogue Manager for Spoken Language Applications." In *Proc. of ICSLP*, vol. 2, pp. 701–704. Philadelphia, USA, 1996.

[Gri69] Grice H., "Utterer's Meaning and Intentions." *Philosophical Review 68*, pp. 147–177. 1969.

[Gri13] Griol D., Callejas Z., "An Architecture to Develop Multimodal Educative Applications with Chatbots." *Intl. Journal of Advanced Robotic Systems 10*, pp. 1–15. 2013.

[Gro86] Grosz B., Sidner C., "Attention, Intentions, and the Structure of Discourse." *Computational Linguistics 12*, pp. 175–204. 1986.

[Gro96] Grosz B., Kraus S., "Collaborative Plans for Complex Group Action." *Artificial Intelligence 86*, pp. 269–357. 1996.

[Gus02] Gustafson J., *Developing Multimodal Spoken Dialogue Systems – Empirical Studies of Spoken Human-Computer Interaction.* Ph.D. thesis, KTH, Department of Speech, Music and Hearing, 2002.

[Gus03] Gustafson J., Bell L., "Speech Technology on Trial: Experiences from the Augustus System." *Natural Language Engineering: Special Issue on Best Practice in Spoken Dialogue Systems 3*, pp. 273–286. 2003.

[Hem90] Hemphill C., Godfrey J., Doddington G., "The ATIS Spoken Language Systems Pilot Corpus." In *Proc. of ACL Workshop on Speech and Natural Language*, pp. 96–101. 1990.

[Hen08] Henderson J., Lemon O., Georgila K., "Hybrid reinforcement/supervised learning of dialogue policies from fixed data sets." *Computational Linguistics 34*, pp. 487—511. 2008.

[Hja05] Hjalmarsson A., "Towards User Modelling in Conversational Dialogue Systems: A Qualitative Study of the Dynamics of Dialogue Parameters." In *Proc. of Interspeech*, pp. 869–872. Lisbon, Portugal, 2005.

[Hof98] Hoffman R., Crandall B., Shadbolt N., "Use of the Critical Decision Method to Elicit Expert Knowledge: A Case Study in the Methodology of Cognitive Task Analysis." *Human Factors 40*, pp. 254–276. 1998.

[Hul96] Hulstijn H., Steetskamp R., Doest H., Burgt S., Nijholt A., "Topics in SCHISMA Dialogues." In *Proc. of the Twente Workshop on Language Technology*, pp. 89–99. University of Twente, 1996.

[Hul00] Hulstijn J., "Dialogue Games Are Receips for Joint Action." In *Proc. of Workshop on Semantics and Pragmatics of Dialogues*, pp. 1–6. Göthenburg Papers in Computational Linguistics, 2000.

[Hur05] Hurtig T., Jokinen K., "On multimodal route navigation in PDAs." In *Proc. of Baltic Conference on Human Language Technologies*, pp. 261–166. 2005.

[Ing92] Ingrand F., Georgeff M., Rao A., "An Architecture for Real-Time Reasoning and System Control." *Intelligent Systems and Their Applications 7*, pp. 34–44. 1992.

[Jok10] Jokinen K., McTear M., *Spoken Dialogue Systems.* Morgan & Claypool Publishers, 2010.

[Kle00] Klemmer S., Sinha A., Chen J., Landay J., Aboobaker N., Wang A., "SUEDE: A Wizard of Oz Prototyping Tool for Speech User Interfaces.“ In *Proc. of ACM Symposium on User Interface Software and Technology*, vol. 2, pp. 1–10. ACM Press, 2000.

[Kni01] Knight S., Gorell G., Rayner M., Milward D., Koelling D., Lewin I., "Comparing Grammar-Based and Robust Approaches to Speech Understanding: A Case Study." In *Proc of Eurospeech*, pp. 1779–1782. 2001.

[Kom03] Komatani K., Ueno S., Kawahara T., Okun H., "User Modeling in Spoken Dialogue Systems for Fexible Guidance Generation." In *Proc. of Eurospeech*, pp. 745–748. 2003.

[Kon09] Konopík M., *Hybrid Semantic Analysis.* PhD thesis, UWB, Department of Computer Science and Engineering. Pilsen, 2009.

[Lee10] Lee C., Jung S., Kim K., Lee D., Lee G., "Recent Approaches to Dialog Management for Spoken Dialog Systems." *Computing Science and Engineering 4*, pp. 1–22. 2010.

[Les04] Lester J., Branting K., Mott B., "Conversational Agents." *The Practical Handbook of Internet Computing.* Chapman & Hall, 2004.

[Lev97] Levy D., Catizone R., Battacharia B., Krotov A. and Wilks Y., "CONVERSE: A Conversational Companion." In *Proc. of Intl. Workshop on Human-Computer Conversation.* Bellagio, Italy, 1997.

[Lik32] Likert R. "A Technique for the Measurement of Attitudes." *Archives of Psychology 140.* The Science Press, 1932.

[Lit02] Litman D., Pan S., "Designing and Evaluating an Adaptive Spoken Dialogue System." *User Modelling and User-Adapted Interaction 12*, pp. 111–137. ACM Press, 2002.

[Lit06] Litman D., Rosé C., Forbes-Riley K., VanLehn K., Bhembe D., Silliman S., "Spoken Versus Typed Human and Computer Dialogue Tutoring." *Intl. Journal of Artificial Intelligence in Education 16*, pp. 145–170. 2006.

[Man88] Mann W., "Dialogue Games: Conventions of Human Interaction." *Argumentation 2*, pp. 511–532. 1988.

[McG91] McGlashan S., *A Proposal for SIL.* SUNDIAL WP6 (unpublished), 1991.

[McG96] McGlashan S. "Towards multimodal dialogue management." In *Proc. of Twente Workshop on Language Technology*, pp. 1–10. Twente, 1996.

[McT02] McTear M., "Spoken Dialogue Technology: Enabling the Conversational User Interface." *ACM Computing Survey 34*, pp. 90–169. ACM Press, 2002.

[Mel05] Melichar M., *Template Driven Dialogue Management Approach in the Framework of Multimodal Interaction.* PhD thesis proposal, EPFL, Artificial Intelligence Laboratory, Lausanne, 2005.

[Men96] Meng H., Busayapongchai S., Glass J., Goddeau D., Hetherington L., Hurley E., Pao C., Polifroni J., Seneff S., Zue V., "WHEELS: A Conversational System in the Automobile Classifieds Domain." In *Proc. of ICSLP*, pp. 542–545. Philadelphia, USA, 1996.

[Mon80] Monge P., Capella J. *Multivariate Techniques in Human Communication Research.* Academic Press, 1980.

[Nes07] Nestorovič T., *Hlasové ovládání navigačního systému automobilu.* Master thesis, UWB, Department of Computer Science and Engineering. Pilsen, 2007.

[Nes09] Nestorovič T., "Towards Flexible Dialogue Management Using Frames." In *Proc. of Text, Speech, and Dialogue*, pp. 419–426. Springer Verlag, 2009.

[Nes10a] Nestorovič T., "On Managing Collaborative Dialogue Using an Agent-based Architecture.“ In *Proc. of Mexican Intl. Conference on Artificial Intelligence,* pp. 56–68. Springer Verlag, 2010.

[Nes10b] Nestorovič T. "Frame-based Dialogue Management Automated Error Recovery Approach." In *Proc. of Australian Joint Conference on Artificial Intelligence*, pp. 32–41. Springer Verlag, 2010.

[Nes13] Nestorovič T., "Collaborative Dialogue Information Model." In *Proc. of Intl. Conference on Applied Mathematics and Computational Methods*, pp. 62–67. 2013.

[Ngu06a] Nguyen T., Wobcke W., "An Agent-Based Approach to Dialogue Management in Personal Assistants." In *Proc. of IEEE/WIC/ACM*, pp. 367–371. ACM Press, 2006.

[Ngu06b] Nguyen T., *An Agent-based Approach to Dialogue Management in Personal Assistants*. PhD thesis, University of New South Wales, School of Computer Science and Engineering. Australia, 2006.

[Oce98] Ocelíková J., *Zpracování významu spontánních promluv při dialogu člověka s počítačem*. PhD thesis, UWB, Department of Computer Science and Engineering. Pilsen, 1998.

[Ovi02] Oviatt S., "Multimodal Interfaces." *Handbook of Human-Computer Interaction*. Lawrence Earlbaum, New Jersey, 2002.

[Pav09] Pavelka T., *Hybrid Methods of Automatic Speech Recognition*. PhD thesis, UWB, Department of Computer Science and Engineering. Pilsen, 2009.

[Pér06] Pérez G., Amores G., Manchón P., "A Multimodal Architecture for Home Control by Disabled Users." In *Proc. of ACL Workshop on Spoken Language Technology*, pp. 134–137. 2006.

[Pie09] Pieraccini R., Suendermann D., Dayanidhi K., Liscombe J. "Are We There Yet? Research in Commercial Spoken Dialog Systems." In *Proc. of Text, Speech, and Dialogue*, pp. 3–13. Springer Verlag, 2009.

[Qu02] Qu Y., Green N., "A Constraint-based Approach for Cooperative Information-seeking Dialogue." In *Proc. of Intl. Natural Language Generation Conference*, pp. 136–143. 2002.

[Rao95] Rao A., Georgeff M., "BDI Agents: From Theory to Practice." In *Proc. of Intl. Conference on Multi-Agent Systems*, pp. 312–319. 1995.

[Rei81] Reinhart T., "Pragmatics and Linguistics: An Analysis of Sentence Topics." *Philosophica 27*, pp. 53–94. 1981.

[Ric01] Rich C., Sidner C., Lesh N., "COLLAGEN: Applying Collaborative Discourse Theory to Human-Computer Interaction." *AI Magazine 22*, pp. 15–25. 2001.

[Rog51] Rogers C., *Client Centered Therapy: Current Practice, Implications and Theory*. Houghton Mifflin, Boston, 1951.

[Rot07] Rotaru M., Litman D., "The Utility of a Graphical Representation of Discourse Structure in Spoken Dialogue Systems." In *Proc. of COLING*, pp. 360–367. Prague, 2007.

[Sea69] Searle J., *Speech Acts: An Essay in the Philosophy of Language*. Cambridge University Press, Cambridge, 1969.

[Sin02] Singh S., Litman D., Kearns M., Walker M., "Optimizing Dialogue Management with Reinforcement Learning: Experiments with the NJFun System." *Journal of Artificial Intelligence Research 16*, pp. 105–133. 2002.

[Smi92] Smith R., "Integration of Domain Problem Solving with Natural Dialog: The Missing Axiom Theory." *Applications of Artificial Intelligence 10: Knowledge-Based Systems*, pp. 270–278. 1992.

[Son06] Sonntag D., "Towards Combining Finite State, Ontologies and Data-driven Approaches to Dialogue Management for Multimodal Question Answering." In *Proc. of Intl. Language Technologies Conference*. 2006.

[Ste07] Stenchikova S., Mucha B., Hoffman S., Stent A., "RavenCalendar: A Multimodal Dialog System for Managing a Personal Calendar." In *Proc. of the Human Language Technology Conference: Demonstrations*, pp. 15–16. 2007.

[Sti01] Stibler K., Denny J., "A Three-Tiered Evaluation Approach for Interactive Spoken Dialogue Systems." In *Proc. of Intl. Conference on Human Language Technology*, pp. 1–5. 2001.

[Sto12] Stoyanchev S., Stent A., "Concept Type Prediction and Responsive Adaptation in a Dialogue System." *Dialogue and Discourse 3*, pp. 1–31. 2012.

[Sut98] Sutton S., Cole R., de Villers J., Schalkwyk J., Vermuelen P., Macon M., Yan Y., Rundle B., Shobaki K., Hosom P., Kain A., Wouters J., Massaro D., Cohen M., "Universal Speech Tools: The CSLU Toolkit." in *Proc. of ICSLP*, pp. 3221–3224. 1998.

[Tra03] Traum D., Larson S., "The Information State Approach to Dialogue Management." *Current and New Directions in Discourse and Dialogue*, pp. 325–353. Springer Verlag, 2003.

[Tsi12] Tsiakoulis P., Gasic M., Henderson M., Plannels-Lerma J., Prombonas J., Thomson B., Yu K., Young S., "Statistical Methods for Building Robust Spoken Dialogue Systems in an Automobile." *Advances in Human Aspects of Road and Rail Transportation*, pp. 744–753. CRC Press, 2012.

[Tur03] Turunen M., Hakulinen J., "Jaspis^2 – An Architecture for Supporting Distributed Spoken Dialogues." In *Proc of Eurospeech*, pp. 1913–1916. 2003.

[Tur05] Turunen M., Hakulinen J., Häihä K., Salonen E., Kainulainen A., Prusi P., "An Architecture and Applications for Speech-based Accessibility Systems." *IBM Systems Journal 44*, pp. 485–504. 2005.

[vZa99] van Zanten G., "User Modelling in Adaptive Dialogue Management." In *Proc. of Eurospeech*, pp. 1183–1186. 1999.

[Wal97] Walker M., Hindle D., Fromer J., di Fabbrizio J., Mestel C., "Evaluating Competing Agent Strategies for a Voice Email Client." In *Proc. of Eurospeech.* 1997.

[Wal98] Walker M., Litman D., Kamm C., Abella A., "Evaluating Spoken Dialogue Agents with PARADISE: Two Case Studies." *Computer Speech Language 12*, pp. 317–347. 1998.

[Wal00] Walker M., Langkilde I., Wright J., Gorin A., Litman D., "Learning to Predict Problematic Situations in a Spoken Dialogue System: Experiments with How May I Help You?" In *Proc. of the North American Chapter of the ACL Conference*, pp. 210–217. Springer Verlag, 2000.

[Wal01] Wallis P., Mitchard H., Das Y., O'Dea D., "Dialogue Modelling for a Conversational Agent." In *Proc. of Australian Joint Conference on Artificial Intelligence*, pp. 532–544. Springer Verlag, 2001.

[Wal04] Walker M., Whittaker S., Stent A., Maloor P., Moore J., Johnston M., Vasireddy G., "Generation and Evaluation of User Tailored Responses in Multimodal Dialogue." *Cognitive Science 28*, pp. 811–840. Elsevier, 2004.

[Wei66] Weizenbaum J. "ELIZA – A Computer Program for the Study of Natural Language Communication Between Man and Machine." *Communications of the Association for Computing Machinery 9*, pp. 36–45. ACM Press, 1966.

[Wil06] Wilks Y., Catizone R., Turunen M., "Dialogue management: State of the Art Papers." COMPANIONS Consortium: State Of The Art Papers. 2006.

[Win72] Winograd T. *Understanding Natural Langauge.* Academic Press, 1972.

[Woo95] Wooldridge M., Jennings N., "Intelligent Agents: Theory and Practice." *Knowledge Engineering Review 10*, 115–152. 1995.

[Woo00] Wooldridge M., *Reasoning about Rational Agents.* MIT Press, 2000.

[Xu02] Xu W., Xu B., Huang T., Xia H., "Bridging the Gap Between Dialogue Management and Dialogue Models." In *Proc. of the SIGdial Workshop on Discourse and Dialogue*, pp. 201–210. 2002.

[Yam07] Yamaguchi Y., Hayashi K., Ono T., Kato S., Irie Y., Ohno T., Murao H., Matsubara S., Kawaguchi N., Takeda K., "Towards Robust Spoken Dialogue Systems Using Large-Scale In-Car Speech Corpus." In *Proc. of Advances for In-Vehicle and Mobile Systems*, pp. 211–222. Springer Verlag, 2007.

[Yan95] Yankelovich N., Levow G., Marx M., "Designing SpeechActs: Issues in Speech User Interfaces." In *Proc. of the SIGCHI Conference on Human Factors in Computing systems*, pp. 369–376. 1995.

[Yan96] Yankelovich N., "How Do Users Know What to Say?" *Interactions 3*, pp. 32–43. 1996.

[You92] Youd N., McGlashan S., "Generating Utterances in Dialogue Systems." In *Proc. of Intl. Workshop on Natural Language Generation: Aspects of Automated Natural Language Generation*, pp. 135–149. Springer Verlag, 1992.

[Zah03] Zahradil J., Müller L., Jurčíček F., "Model světa hlasového dialogového systému." In *Proc. of Znalosti*, pp. 404–409. 2003.

[Zbo04] Zbořil F., *Plánování a komunikace v multiagentních systémech*. PhD thesis, VUT, Faculty of Information Technologies. Brno, 2004.

[Zue89] Zue V., Glass J., Goodine D., Leung H., Phillips M., Polifroni J., Seneff S., "Preliminary Evaluation of the VOYAGER Spoken Dialogue System." In *DARPA Workshop 89*, pp. 160–167. 1989.

[Zue91] Zue V., Glass J., Goodine D., Leung H., Phillips M., Polifroni, J., Seneff, S.: "Integration of Speech Recognition and Natural Language Processing in the MIT VOYAGER System." In *Proc. of ICASSP*, pp.713–716. 1991.

[Zue00] Zue V., Seneff S., Glass J., Polifroni J., Pao C., Hazen T., Hetherington L., "JUPITER: A Telephone-based Conversational Interface for Weather Information." *IEEE Transactions on Speecb and Audio Processing 8*, pp. 85–96. 2000.

Author's Publications

Relevant

Nestorovič T., Matoušek V., "Collaborative Dialogue Information Model." In *Proc. of Intl. Conference on Applied Mathematics and Computational Methods*, pp. 62–67. 2013.

Nestorovič T., Matoušek V., "Modified Conversational Agent Architecture." In *Proc. of IEEE Intl. Conference on Tools with Artificial Intelligence*, pp. 682–689. Washington, 2013.

Nestorovič T., Matoušek V., "Applying Rogerian Psychologist in Human-Computer Interaction: A Case Study." In *Proc. of Mexican Intl. Conference on Artificial Intelligence*, pp. 286–293. Springer Verlag, 2013.

Nestorovič T., "Creating a General Collaborative Dialogue Agent with Lounge Strategy Feature." *Expert Systems with Applications 39*, pp. 1607–1625. Elsevier, 2012.

Nestorovič T., "Task-oriented Dialogue Agent Architecture." *Journal of Digital Information Management 9*, pp. 1–8. 2011.

Nestorovič T. "Frame-based Dialogue Management Automated Error Recovery Approach." In *Proc. of Australian Joint Conference on Artificial Intelligence*, pp. 32–41. Springer Verlag, 2010.

Nestorovič T., "On Managing Collaborative Dialogue Using an Agent-based Architecture.“ In *Proc. of Mexican Intl. Conference on Artificial Intelligence,* pp. 56–68. Springer Verlag, 2010.

Nestorovič T., "General Agent-based Architecture for Collaborative Dialogue Management." In *Proc. of IEEE Intl. Conference on Software Technology and Engineering*, pp. 207–211. Puerto Rico, 2010.

Nestorovič T., "Agent-Based Dialogue Management Approach." In *Proc. of IEEE Intl. Conference on Application of Digital Information and Web Technologies.* Instanbul, 2010.

Nestorovič T., "Towards Flexible Dialogue Management Using Frames." In *Proc. of Text, Speech, and Dialogue*, pp. 419–426. Springer Verlag, 2009.

Nestorovič T., "A Frame-Based Dialogue Management Approach." In *Proc. of IEEE Intl. Conference on Application of Digital Information and Web Technologies*, pp. 327–332. London, 2009.

Other

Veleba J., Nestorovič T., "On Steady-State Voltage Stability Analysis Performance in MATLAB Environment." In *Proc. of Intl. Conference on Energy, Environment, Ecosystems, and Development*, pp. 141–147. 2013.

Veleba J., Nestorovič T., "Performance of Static Voltage Stability Analysis in MATLAB Environment with Further Applications." *Intl. Journal of Education and Information Technologies 4*, pp. 133–145. 2013.

Vaněk J., Nestorovič T., "Thermal Comfort Determination Approach." In *Proc. of Intl. Conference on Energy and Environment*, pp. 48–51. WSEAS Press, 2012.

Nestorovič T., Struhár V., "Designing a Personal Assistance Application Using Wizard of Oz methodology." In *Proc. of Intl. DAAAM Symposium*, pp. 715–716. Vienna, 2011.

Nestorovič T., "Dog Disease Expert System." In *Proc. of Intl. DAAAM Symposium*, pp. 1021–1022. Vienna, 2010.

Nestorovič T., "Voice and Graphical User Interfaces: Design Issues." In *Proc. of Electronic Speech Signal Processing*, pp. 212–219. TUDpress, 2010.

Nestorovič T., "On Designing an Experimental Navigation System." In *Proc. of Latest Trends on Systems*, pp. 417–421, WSEAS Press, 2010.

Matoušek V., Nestorovič T., "Grammar-based Dialogue Management Techniques." In *Proc. of Electronic Speech Signal Processing*, pp. 64–71. TUDpress, 2009.

Nestorovič T., "Hlasová a grafická uživatelská rozhraní." In *Proc. of Mezinárodní Baťova konference pro doktorandy a mladé vědecké pracovníky*, pp. 1–13. Zlín, 2009.

Nestorovič T., "Navigation System: An Experiment." In *Proc. of NAV/DAGA Intl. Conference on Acoustics*, pp. 1140–1143. Rotterdam, 2009.

Nestorovič T., "Grammar-Based Dialogue Management Techniques." In *Proc. of Intl. PhD Workshop on Systems and Control*, pp. 1–6. Izola, Slovenia, 2008.

Appendix

A.1 Daisy Input Semantics Grammar

semantics	→	{ *expression* ',' } *expression*
expression	→	*identifier* [*subexpression*] | *localcontrol*
identifier	→	*utttype* | *correction* | *reference*
	|	**Name** [':"' **Value** '"'] { *attribute* }
subexpression	→	'(' [*expression*] ')'
attribute	→	'[' *attname* **Integer** ']'
attname	→	'c' | 's' | 'v' | **Char**
utttype	→	*declarative* | *interrogative* | *imperative*
declarative	→	'.'
interrogative	→	'?'
imperative	→	'!'
correction	→	*agreement* | *disagreement*
agreement	→	'+'
disagreement	→	'-'
reference	→	'REF' [':"' *refgendre* '"']
refgendre	→	'm' | 'z' | 's'
localcontrol	→	*timeout* | *help* | *repeat*
timeout	→	'%'
help	→	'*'
repeat	→	'#'

A.2 Information Management Algorithm

A.2.1 Requirement 1

```
    procedure $Incorporate( Y(K_S) = Y_S \in Semantics , I(C_{parent}) = I_{parent} \in \mathbf{Y} )$ {
        // instantiate a new "empty" collection $Y$ (not yet a DDM expression) of the type $K_S$
1       . . .
        // find in and remove from $I_{parent}$ the original collection instance $Y_{orig}$
2       $Y_{orig} := ( K_S, I_{orig}, F_{orig})$: $\exists F_{parent} = ( E_{parent}, I_{parent} ) \in F_{orig}$
3       if $Y_{orig} \neq \varnothing$ {
            // remove $Y_{orig}$ from $I_{parent}$
4           $Y_{orig} := Y_{orig} \setminus F_{parent}$
            // determine the relationship between $Y_S$ and $Y_{orig}$
5               if  $A(Y_S) \in \{ system,\ dereferenced \}$  {
6                   if  $A(Y_{orig}) \in \{ system,\ dereferenced \}$  {
                        // system DDM expression replaces another system expression
7                       $U(Y) := U(Y_{orig})$
8                   } else {
                        // system DDM expression overrides a user expression
9                       $U(Y) := Y_{orig}$
10              } else
11                  if  $A(Y_{orig}) \in \{ system,\ dereferenced \}$  {
                        // user DDM expression overrides a system expression
12                      $Y_{orig} := U(Y_{orig})$
                    } else
                        // user DDM expression replaces another user expression (do nothing)
            }
            // re-instantiate $I(Y_S)$ in $Y$ (promoting $Y$ to a DDM expression)
13          $\forall I_S = ( D_S, C_S, X_S ) \in I(Y_S)$  {
                // re-instantiate $I_S$ as $I$
14              . . .
                // inherit all subcollections from an original $I_0 = ( D_S, C_S, X_S ) \in I_{orig}$ (if any)
15              . . .
                // attempt to spread given user sibling concepts subinformation to $I$
16              if  $A(Y) = user$  {
17                  $\forall K = ( C, E, T ) \in P(C_S)$  {
                        // Rule 1: if no original instance $I_0$ exists, inherit the most salient $K$
18                      if  $I_0 = \varnothing$  { $Incorporate($ most salient $Y_i = ( K, I_i, F_i ) \in P(Y_{orig})$ , $I$ ) }
                        // Rule 2: TODO later (see Section 4.5.5)
                        // Rule 3: if subinformation $K$ is defined on $I_S$, we are done
19                      if  $Y_i = ( K, I_i, F_i ) \in P(I_S)$  { skip all other Rules }
                        // Rule 4: if any remaining $I' \in I(Y_S)$ contains information $K$, apply $K$ to $I_S$
20                      if  $\exists I' = (D', C', X')$: $S(I') > S(I_S) \ \wedge\ Y_i \in P(I')$ { $F_i := F_i \cup ( (C',R_j), I_S )$ }
                        // Rule 5: if instantiated $I' \in I(Y_S)$ contains information $K$, apply $K$ to $I_S$
21                      if  $\exists I' = (D', C', X')$: $S(I') < S(I_S) \ \wedge\ Y_i \in P(I')$ { $F_i := F_i \cup ( (C',R_j), I_S )$ }
                    }
                }
                // recurrently process
22              . . .
                // set instance $I$ meta-information
23              $S(I) := max( S(I_S), S(I_0), 0 )$              // recall 0 = information unspoken
            }
            // rest of the procedure is the same
24          . . .
        }
```

Usage: $Incorporate($ $Y(\rho) \in Semantics$, $I(\rho) \in \mathbf{Y}$)

A.2.2 Requirement 2

```
        procedure $Incorporate($ $Y(K_S) = Y_S \in Semantics$ , $I(C_{parent}) = I_{parent} \in \mathbf{Y}$ ) {
            . . .
            // find in and remove from $I_{parent}$ the original collection instance $Y_{orig}$
            . . .
            // carry out correction (agreed part of semantics)
            if  Semantics contains disagreement $\wedge$ $Y_S$ is not $disagreed$ $\wedge$ $A(Y_S) = user$ $\wedge$ $P(Y_S) = \varnothing$  {
                $\forall Y' = ( K', I', F' ) \in \{ Y = ( K, I, F ) \in \mathbf{Y}: P(K) \neq \varnothing \ \wedge\ A(Y) = user \}$  {
                    // extract incorrect content from $Y'$ into $I_{EXTRACT}$, see Fig. 4.6
                    $Extract( Y' , incorrect)$
                    // attempt to pass extracted content over to $Y_S$
                    $\forall Y_i = ( K_i, I_i, F_i ) \in P(I_{EXTRACT})$, $I_S = I(C_S) \in I(Y_S)$, $E_S = ( C_S, R_S ) \in E(K_i)$  {
```

```
                $F_i := F_i \cup (E_S, I_S)$
            }
            // clean up $I_{EXTRACT}$
            $P(I_{EXTRACT}) := \varnothing$
            // if extracted concent passed to $Y_S$, correction is done
            if $P(Y_S) \neq \varnothing$ { break }
        }
    }
    // rest of the procedure is the same
    ...
}
```

Usage: *Incorporate*($Y(\rho) \in Semantics$, $I(\rho) \in \mathbf{Y}$)

A.2.3 Requirement 3

```
procedure Incorporate( $Y(K_S) = Y_S \in Semantics$ , $I(C_{parent}) = I_{parent} \in \mathbf{Y}$ ) {
    ...
    // re-instantiate $I(Y_S)$ in $Y$ (promoting $Y$ to a DDM expression)
    ...
    // pass objects between different DSPs: for each instantiated topic $\tau_i$,
    // try to transmit its objects to the current topic instance $Y$ if they fit it
    if $K_S$ is a topic {
        $\forall \tau_i,\ Y' = (\tau_i, I', F') \in \mathbf{Y}$ {
            // attempt to pass each most salient object $Y_j \in Y'$ to $Y$
            $\forall K_j \in P(\tau_i) \cap P(K_S),\ Y_k = (K_j, I_k, F_k) \in \mathbf{Y}$ {
                // find original object of type $K_j$ in $Y$ (if any)
                $Y_{origObj} := (K_j, I_{origObj}, F_{origObj}) \in \mathbf{Y}: \exists F_m = (E, I(Y)) \in F_{origObj}$
                // pass $Y_k$
                if $S(Y_k) > S(Y_{origObj})$ {
                    $F_{origObj} := F_{origObj} \setminus F_m$        // remove $Y_{origObj}$ from $Y$
                    $Incorporate(Y_k, I(Y))$       // pass $Y_k$ to the only concept instance in $Y$
                }
            }
        }
    }
    // rest of the procedure is the same
    ...
}
```

Usage: *Incorporate*($Y(\rho) \in Semantics$, $I(\rho) \in \mathbf{Y}$)

A.2.4 Requirement 4

```
procedure Incorporate( $Y(K_S) = Y_S \in Semantics$ , $I(C_{parent}) = I_{parent} \in \mathbf{Y}$ ) {
    // instantiate a new "empty" collection $Y$ (not yet a DDM expression) of the type $K_S$
    ...
    // determine if $K_S$ can be a direct subcollection of $C_{parent}$
    $E_{direct} := (C_{parent}, R_i) \in E(K_S)$
    // find in and remove from $I_{parent}$ the original collection instance $Y_{orig}$
    if $E_{direct} \neq \varnothing$ {
        // $K_S$ is directly accessible from within $C_{parent}$
        ...
    } else {
        // $K_S$ is not directly accessible from within $C_{parent}$
        $Y_{orig} := (K_S, I_{orig}, \varnothing)$: $DSP(Y_{orig}) = CURRENT\_DSP$
    }
    // carry out correction (agreed part of semantics)
    . . .
    // re-instantiate $I(Y_S)$ in $Y$ (promoting $Y$ to a DDM expression)
    . . .
            . . .
            // Rule 2: if an unbound expression of type $K$ exists, $I$ binds it
            if $\exists Y_{unbound} = (K, I_{unbound}, \varnothing)$ { $Incorporate(Y_{unbound}, I)$ }
            . . .
    // rest of the procedure is the same
    . . .
}
```

Usage: $Incorporate(\ Y(\rho) \in Semantics\ ,\ I(\rho) \in \mathbf{Y}\ )$

A.3 DORA Web Instructions

A.3.1 Welcoming Page

Welcome!
DORA is an experimental spoken dialogue system that allows you to access your fictional bank account and find information on branches via a telephone conversation. You will solve with DORA three different tasks. You should try to do each task as efficiently as you can and avoid listening to messages unnecessarily. Please make brief notes about the bank branches when you listen to the information on them. Instructions for calling DORA can be found at each task scenario. Please read through the instructions before calling. At the end of the task (after you hang up the phone), there are a few brief questions for you to answer. Even if DORA aborted before you could complete the task, please finish the survey and continue to the next task. Thanks for participating!

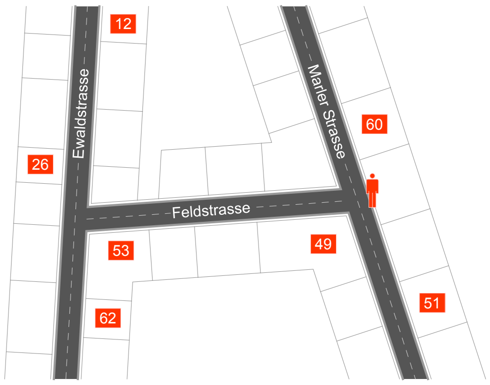
A.3.2 Hints for Using DORA

- Speak naturally and pronounce well.
- If you don't know what to say or don't understand what DORA is doing, say *Help* to hear a help message.
- If you wait too long to tell DORA what to do, DORA will tell you what you can say.
- You can interrupt DORA at any time. For example, if you've heard enough or if you know what you want to do, you don't have to wait for DORA to finish talking. If you don't hear everything when DORA presents the bank information, say *Repeat* to hear the information again.
- If you want to abort your current attempt at the task before finishing, say *I'm done here* to start the task again.

- When you are finished with a task, say *Goodbye* to end the dialogue.

A.3.3 Task Formulations

- *Task 1.* Try to find all **Bertlich** branches with an **ATM** in **Gladbeck**. Please write down their exact **addresses** that you were told by the system.
- *Task 2.* Given the below map of **Herten**, try to find the exact **address** of the nearest **Hassel** branch that has an **ATM** (based on your current position depicted in the map, it is up to you to determine the shortest route!). Once known, check its **opening hours** and make sure you can visit it even after 6 pm. Please spot the nearest branch you have found, and also write down its exact address.



- *Task 3.* Try to find the **opening hours** of all **Horst** branches in **Gelsenkirchen**. Check their **addresses** and make sure you can visit any of them in Dorfstrasse even after 6pm. What is the exact location of such a branch?

A.4 CTA Responses

A.4.1 Question 7

What was the first thing that came across your mind when you first heard the prompt <—*RS prompt transcript*—>?

- It's too general. When I heard it, I though it was great that a machine agreed with me (*note: regarding the "I see" sentence*) but I want to find that connection.
- When and where to go as in one city there may be more stations in one direction.
- To attempt to precise the information on that train.
- Is it joking?

A.4.2 Question 8

Did you feel like being pushed to say more? Did you feel this utterance was motivating you to say more?

- Yes, but in the second sentence I didn't figure out what was missing exactly – I was comparing with an online ticketing service where the only necessary pieces of information regard where to go from and to.
- Yes, I find it reasonable.
- Yes, but specification of what exactly am I supposed to say is missing.
- Yes, but it's a bit weird.
- No, if I heard something like that from an on-line service, I'd reckon it's down or that somebody is joking, and definitely would hang up.

A.4.3 Question 9

Were you aware of being expected to say more information on the transportation means to find?

- Five people out of six confirmed.

A.4.4 Question 10

Do prompts like <—*RS prompt transcript*—> sound acceptably to you?

- It does relatively.
- With some modifications it might sound naturally.
- It sounds weird, but one probably would figure out what to say.
- No, I'd go straight to what is missing, i.e. asked for time, etc.

A.5 Evaluation Using the Three-Tiered Methodology

In Section 5.3.3, we described the system performance as a combination of various system parameters. Linear regression revealed us the most significant contributors to the overall performance, approached by user satisfaction. Although basically sufficient for simple information systems like ours, the PARADISE framework turns out to be hardly usable for more complex systems. More specifically, the notion of blending a set of generally unrelated measures may result in misleading or unprecise conclusions about the system. For instance, it may be assumed that the number of help requests per dialogue has little to do with back-end component speed, and therefore, these two should not be put directly next to each other in a performance formula. In addition, it is not clear whether the system performance can be reliably indicated by a single performance formula as a combination of different evaluation measures [Ngu06b]. As Frøkjær et al. also point out [Frø00], there may be very low correlations between the three usability measures proposed in the PARADISE framework [Wal98]: *effectiveness* (e.g., task completion rate, κ), *efficiency* (e.g., task completion time or help requests), and *user satisfaction*. Therefore, they should be considered independent aspects in a system evaluation. To overcome these pitfalls, the three-tiered methodology was developed [Sti01]. As its name suggests, it models the evaluation process at three levels of abstraction (below listed in a bottom-up manner):

- *System component performance*. Each system component is to be evaluated individually to reveal their influences on other parts of a system, hence preventing potential negative impacts on the task success. The most notable component to measure influences of, is the ASR. For instance, poor recognition may have impact on low scores of task completion, causing low user satisfaction. Metrics to evaluate the ASR performance include: word/utterance accuracy (system confidence score in recognizing a given word/utterance), concept accuracy (semantic understanding of the system), or component speed (time per turn).

- *System support of task success*. Behind this fuzzy name stand metrics to evaluate how capable the system is to meet individual task objectives. In general, for a system to be applicable in a certain domain of problems, it is essential to establish a definition of task success early in the evaluation process. For instance, a task may be considered successfully accomplished if the user signs in the system, starts an intended request, supplies missing information, confirms it, and signs off the system [Sti01]. Metrics to evaluate such interaction include: task completion (success rate of a given task), task complexity (minimal information to fulfil a task), or task pace (time spent conversing with the system).

- *User satisfaction*. Same as with PARADISE, however user responses are further coped with individually, instead of being added to a single value.

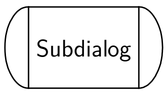
Thus, the three-tiered methodology organizes a dialogue system analysis into a search for dependences among distinct metrics. To reveal the impact in one tier against metrics in another tier, the *principal component analysis* (*PCA*) may be employed.

We did not engage with the PCA, mainly due to the lack of freely available computational toolkits. In fact, the only one we have managed to find is the *OOoStat Statistics Macros*,[1] which however does not feature enough functionality to conduct a dialogue system precise analysis; the crucial missing part is factor rotation to better map as many experiment measures onto as few factors, that way allowing for the analysis of which measures contribute most to the Likert-scale questions.

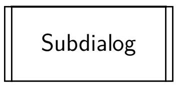
1 *http://sourceforge.net/projects/ooomacros/files/OOo%20Statistics/*

A.6 SDL Notation Overview

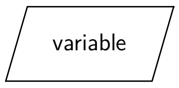
Only necessary minimum of symbols is presented here. For full definition, see [Bel89].



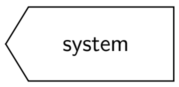
Start symbol



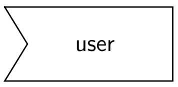
Subdialog invocation



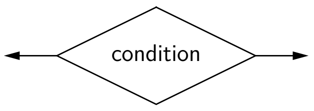
Variable storage



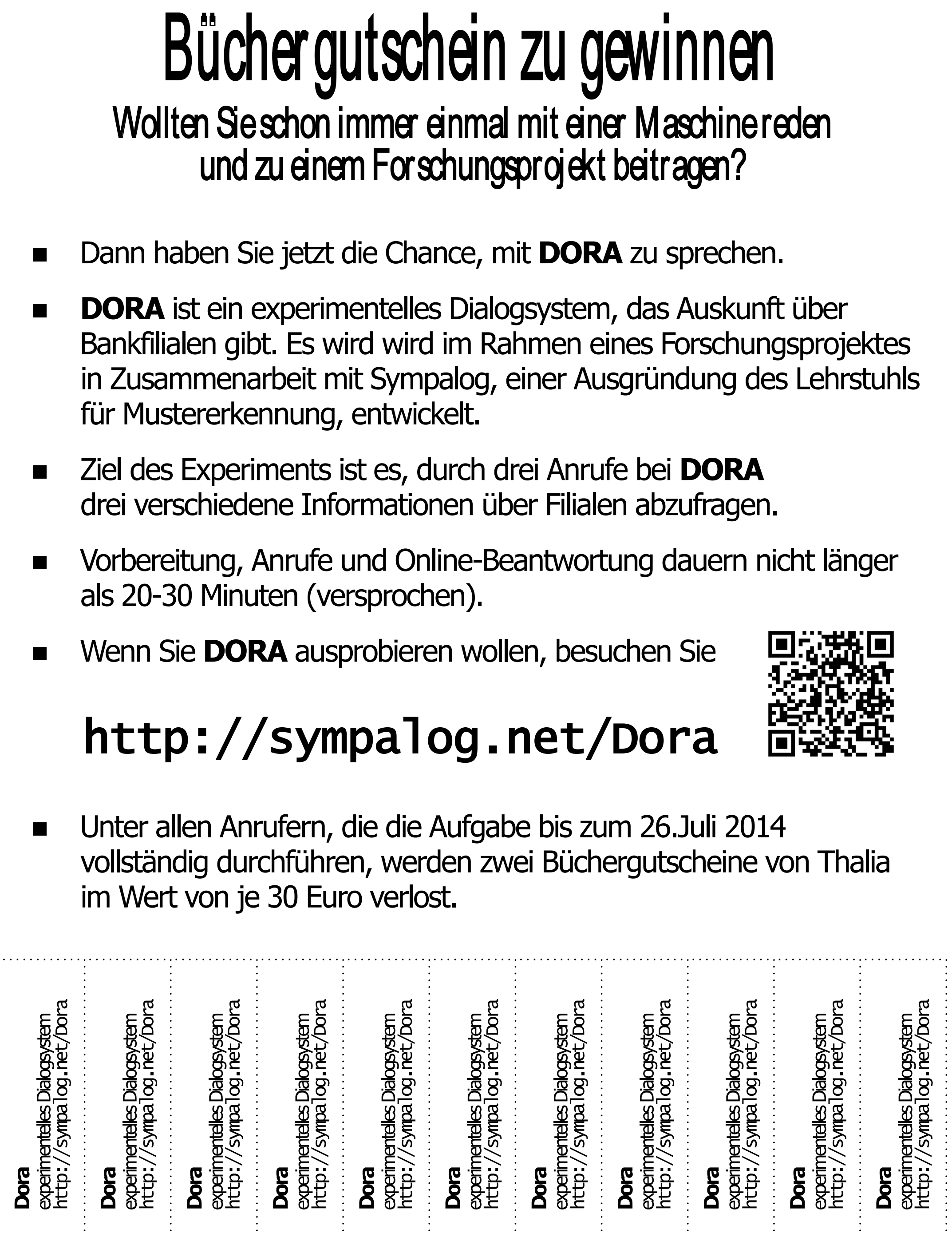
System prompt



User input



Branching logic



Büchergutschein zu gewinnen

Wollten Sie schon immer einmal mit einer Maschine reden und zu einem Forschungsprojekt beitragen?

- Dann haben Sie jetzt die Chance, mit **DORA** zu sprechen.
- **DORA** ist ein experimentelles Dialogsystem, das Auskunft über Bankfilialen gibt. Es wird wird im Rahmen eines Forschungsprojektes in Zusammenarbeit mit Sympalog, einer Ausgründung des Lehrstuhls für Mustererkennung, entwickelt.
- Ziel des Experiments ist es, durch drei Anrufe bei **DORA** drei verschiedene Informationen über Filialen abzufragen.
- Vorbereitung, Anrufe und Online-Beantwortung dauern nicht länger als 20-30 Minuten (versprochen).
- Wenn Sie **DORA** ausprobieren wollen, besuchen Sie

  `http://sympalog.net/Dora`

- Unter allen Anrufern, die die Aufgabe bis zum 26.Juli 2014 vollständig durchführen, werden zwei Büchergutscheine von Thalia im Wert von je 30 Euro verlost.

Dora experimentelles Dialogsystem http://sympalog.net/Dora

Dora experimentelles Dialogsystem http://sympalog.net/Dora

Dora experimentelles Dialogsystem http://sympalog.net/Dora

Dora experimentelles Dialogsystem http://sympalog.net/Dora

Dora experimentelles Dialogsystem http://sympalog.net/Dora

Dora experimentelles Dialogsystem http://sympalog.net/Dora

Dora experimentelles Dialogsystem http://sympalog.net/Dora

Dora experimentelles Dialogsystem http://sympalog.net/Dora

Dora experimentelles Dialogsystem http://sympalog.net/Dora

Dora experimentelles Dialogsystem http://sympalog.net/Dora

Dora experimentelles Dialogsystem http://sympalog.net/Dora

Fig. A.1 DORA advertisement leaflet.

Table A.1 Confusion matrix for Agent A, data attributes matching key given in boldface; see data attribute legend for key values; Addr = branch address, OH = branch opening hours.

		KEY										
		Name			**City**			**Street**	**Equipment**		**Addr**	**OH**
	DATA	**v1**	**v2**	**v3**	**v6**	**v7**	**v8**	**v9**	**v12**	**v13**	**v14**	**v16**
Bertlich	**v1**	**2**										
Hassel	**v2**		**2**									
Horst	**v3**			**2**								
Hauptstelle	**v4**			2								
None	**v5**	1	2									
Gladbeck	**v6**				**3**							
Herten	**v7**					**4**						
Gelsenkirchen	**v8**						**4**					
None	**v9**							**7**				
Marlerstrasse	**v10**							3				
Feldstrasse	**v11**							1				
Geldautomat	**v12**								**7**	1		
None	**v13**									**3**		
Correct	**v14**										**4**	
Incorrect	**v15**										7	
Correct	**v16**											**2**
Incorrect	**v17**											6
	sum	3	4	4	3	4	4	11	7	4	11	8

Table A.2 Confusion matrix for Agent B, data attributes matching key given in boldface; see data attribute legend for key values; Addr = branch address, OH = branch opening hours.

		KEY										
		Name			**City**			**Street**	**Equipment**		**Addr**	**OH**
	DATA	**v1**	**v2**	**v3**	**v4**	**v5**	**v6**	**v7**	**v9**	**v10**	**v11**	**v13**
Bertlich	**v1**	**4**										
Hassel	**v2**		**2**									
Horst	**v3**			**2**								
Gladbeck	**v4**				**4**							
Herten	**v5**					**2**						
Gelsenkirchen	**v6**						**2**					
None	**v7**							**6**				
Marlerstrasse	**v8**							2				
Geldautomat	**v9**								**5**			
None	**v10**								1	**2**		
Correct	**v11**										**5**	
Incorrect	**v12**										3	
Correct	**v13**											**2**
Incorrect	**v14**											2
	sum	4	2	2	4	2	2	8	6	2	8	4

Table A.3 Experiment results; ET = elapsed time, #UT = user utterances, #TO = number of timeouts, #HR = number of help requests, AT = average number of eigen information per turn (excl. timeouts), #RS = number of Rogerian strategy applications, #RR = number of restart requests, Q1...Q6 = value for question on Likert-scale, US = user satisfaction.

User	Task	κ	ET	#UT	#TO	#HR	AT	#RS	#RR	Q1	Q2	Q3	Q4	Q5	Q6	US
1	1	1	2:13	5	1	0	1.25	2	0							
1	2	0.5	6:02	20	3	2	1.15	5	1	4	4	4	5	4	4	25
1	3	1	3:18	8	1	0	1.14	4	0							
2	1	1	1:27	4	1	0	1.25	2	0							
2	2	0.5	7:33	19	2	0	1.05	4	0	3	4	3	4	4	4	22
2	3	1	3:01	7	1	0	1.10	3	0							
3	1	1	2:15	8	3	0	1.13	0	0							
3	2	0.33	1:57	10	1	0	1.10	0	0	3	4	3	4	4	4	22
3	3	1	2:29	8	3	0	1.13	0	0							
4	1	0.5	5:00	14	4	0	1	0	0							
4	2	0.333	8:29	29	13	1	1	0	0	3	3	2	3	3	2	16
4	3	0.4	4:32	18	8	0	1.06	0	0							
5	1	1	2:46	7	1	0	1	0	0							
5	2	0.5	1:57	6	1	0	1	0	0	*database issues*						
5	3	1	3:17	8	4	0	1.11	0	0							
6	1	1	3:20	11	4	0	1	4	0							
7	3	0.2	2:52	11	3	0	1.10	0	0	*user abandoned*						
8	2	0.5	1:21	7	2	0	1	0	0							
9	1	0.5	1:10	5	1	0	1	2	0							

Table A.4 The Daisy framework API; instead of alphabetically, the individial functions are listed in order of their most probable use.

Function C-style synopsis and description
void *LoadDomain(const wchar_t *fileName , const int bufferSize , int *error)   Loads a domain specification constructed using the Domain Editor (see attached CD) and returns a handle of the loaded domain. If errors occur, the error variable indicates them bitwise.
int OpenSession(const void *domain , const char *name , void *reserved)   Provided a domain loaded using LoadDomain, this function creates a named standby session (not yet running), and returns an error code.
int SetProperty(const char *name , const int *property , const int *value)   Sets a property of an existing standby session identified by its name to a specified value. The function returns an error code.
int LaunchSession(const char *name , void *params)   Makes a standby session running; no property changes are allowed to a running session. The function returns an error code.
int WaitForSystemResponse(const char *name , char *bufferUtf8)   Suspends the caller until the system has generated a response for the running session identified by its name. The function returns an error code.
int SignalSemanticsReady(const char *name , const char *bufferUtf8)   Notifies the system that the session identified by its name has a new semantical input. The function returns an error code.
int SignalCompositeSemanticsReady(const char *name , const char *bufferUtf8TaskFragment , const char *bufferUtf8DataFragment)   Notifies the system that the session identified by its name has a new composite semantical input. The function returns an error code.
int AppendLineToLogFile(const char *name , const char *logText)   Appends the specified text as a new line to the log file associated with the named session (logging must be permitted on that session). The function returns an error code.
int CloseSession(const char *name)   Terminates the session identified by its name. The function returns an error code.
void DestroyDomain(const void *domain)   If there are no running sessions with the specified domain previously loaded by LoadDomain, destroys it. The function returns nothing.
int ShowAbout(const HWND hParentWnd , void *reserved)   Shows the framework "About" modal dialog box, as shown in Fig. 6.1. The function returns the button code pressed by the user to close the dialog box.

This page intentionally left blank.