A Dynamic Multi-Application Dialog Engine for Task-Oriented Voice User Interfaces

Dissertation

zur Erlangung des Grades
Doktor der Ingenieurwissenschaften (Dr.-Ing.)
der Technischen Fakultät
der Universität des Saarlandes

vorgelegt von
Jürgen te Vrugt

Saarbrücken
2006

Dekan:
Prof. Dr.-Ing. Thorsten Herfet

Vorsitzender des Prüfungsausschusses:
Prof. Dr.-Ing. Thorsten Herfet

Gutachter:
Prof. Dr. Dr. h.c. mult. Wolfgang Wahlster
Prof. Dr. Dietrich Klakow

Promovierter akademischer Mitarbeiter der Fakultät:
Dr.-Ing. Jörg Baus

Tag und Ort des öffentlichen Kolloquiums:
12. Juni 2006, 14^{15} Uhr
Universität des Saarlandes, Saarbrücken

Dedicated to my wife Inka
and my son Janus.

Their love, support, and patience
made this thesis possible.

Contents

Abstracts

Short Abstract

This thesis introduces the DymaLog framework for spoken language dialog systems, which separates the applications from the actual dialog system. It facilitates the control of a plurality of applications through a single dialog system, changeable during run time. This is achieved by application-independent knowledge processing inside the dialog system, based on a hierarchical representation of obtained information (o^2I-Trees). The approach enables the realization of generic dialog functionalities.
DymaLog is composed of a collection of components; each serves mainly a single purpose. It fosters the generation of competing hypotheses during the processing of the user input in order to derive an optimal interpretation at a certain stage in the processing.

The *Marvin* dialog system puts DymaLog into practice. We discuss selected interactions with various applications enabled for the operation through the system. The parameterized hypothesis selection process is considered in detail, especially the parameter estimation algorithm Grail, and the same holds for the development process in the generation of competing hypotheses for the user input.

Kurzzusammenfassung

Die Arbeit stellt die Grundlagen zur Realisierung des sprachbasierten Dialogsystems *Marvin* für die Interaktion eines Benutzers mit verschiedenen Applikationen vor: DymaLog. Es erlaubt die Kontrolle unterschiedlicher Applikationen durch ein einziges System und ermöglicht u.a. dynamische Änderungen der verfügbaren Applikationen zur Laufzeit. Dies wird durch applikationsunabhängige Wissensverarbeitung erreicht, basierend auf modularen ontologischen Beschreibungen der Anwendungsfreiheitsgrade (o^2I-Trees). Die Trennung von Dialogsystem und Applikationen ermöglicht die Realisierung generischer Dialogfunktionalitäten.
DymaLog besteht aus einer Reihe von separaten Einheiten, jede beinhaltet im Wesentlichen ein Modell zur Verarbeitung der Benutzereingabe. Um die optimale Interpretation der Benutzereingabe zu erlangen wird die Generation alternativer Interpretationen gefördert.

Das *Marvin* Dialogsystem realisiert die Konzepte aus DymaLog. Ausgewählte Interaktionen mit verschiedenen Applikationen werden diskutiert. Ferner wird der parameterisierte Auswahlprozeß der 'besten' Interpretation beleuchtet, insbesondere der Parameter-Schätzalgorithmus Grail, und die Erzeugung alternativer Hypothesen durch ausgewählte Einheiten diskutiert.

Extended Abstract

Spoken language dialog systems allow users to control applications by voice. The vast number of present systems tightly integrate the applications to control them, even though knowledge sources of the building blocks are often configurable. Some dialog systems in the research labs controlling multiple applications start to loosen the coupling.
This thesis introduces DymaLog (dynamic multi-application dialog framework), an innovative and powerful framework for voice user interfaces. DymaLog decouples the applications from the dialog system. On top level, it is divided into input processing, dialog engine, output creation, and application environment. Our main interest concerns the dialog engine Aide (application-independent dialog engine for dynamic environments). It forms the central entity of our application-blind approach: in contrast to systems that include the application directly or an appropriate application model, DymaLog limits itself to application operation parameters (AOPs).

We will now further elaborate the concepts underlying DymaLog by means of five central questions that describe the issues tackled by this framework:

***Q1**: Which methods allow the reduction, or even elimination, of the close interdependency between a dialog engine and its hosted applications?*

The appropriate representation of AOPs, in order to communicate with applications and thus trigger operations by these applications, is a fundamental means of the separation of the dialog system and the applications. The object-oriented interpretation trees (o^2I-Trees) used in DymaLog, derived from ontologies, turned out to facilitate the disentanglement from the applications. They serve multiple tasks, among these are the representation of the interpretations derived from the user input and of the applications' processing results, and the exchange of the state of knowledge between the components. o^2I-Trees also define the interface between the dialog system and the applications.
The basic units of the o^2I-Trees are complex nodes, the dialog objects (DOs), that have an inherent non trivial structure. Relations between the DOs lead to the tree-like o^2I-Trees. Generally speaking, Aide performs operations on the o^2I-Trees, with major support from the DOs, especially during the integration of the current user input with the previous discourse. For an application-independent integration, we present a novel three-step procedure, consisting of the identification of relevant entities for the integration, establishing the relation of outstanding DOs, and the actual integration step.
In Aide, we were able to concentrate the dependency on the applications in the domain model. Thus, DymaLog is application-blind. There is no explicit application modeling performed in the framework, e.g. DymaLog does not depend on a rule set that describes the application.

***Q2**: Can we generalize interaction concepts required for a variety of applications or related to meta-communication?*

DymaLog approaches two types of generalization: generic functionalities and building blocks. The focus of generic functionalities is the support of the user in the interaction with the dialog system, while building blocks intend to support the realization of applications or application interfaces for use in conjunction with DymaLog. The transition between these two classes is smooth.
Generic functionalities realized within DymaLog, which exemplify the methods to manipulate the dialog state, are the navigation in the discourse, allowing for easy recovering of previous states, and the access to the viewport of the visual output. An interesting tool that supports the interaction with applications is the access to lists, realized as a generic functionality.

Building blocks include a set of knowledge sources, especially ontologies to define the AOPs, and, optionally, already implementations to process the entities in the building blocks. An application designer may use building blocks as they are or modify them for use. Presently available building blocks relate to descriptions of a person, place, time, etc.
An innovative generic concept, heavily related to Q1, is sharing of knowledge between applications. DYMALOG automatically shares 'compatible' content between applications, without the need to explicitly formulate compatibility of entities between the different applications.

Q3: *On which principles can multi-application dialog systems base the handling of dynamic changes in the set of applications?*

Naturally, online changes of the set of accessible applications require the adaptation of the dialog system at run-time. For DYMALOG, that implies changes in the knowledge sources and their adaptations in the components on the fly. Due to our application-blind approach, the update of the knowledge sources does not require any application description to be integrated. Inside AIDE, only the domain model is affected and requires the updated set of AOP descriptions. Application-independent processing in DYMALOG only needs updated information on *how to speak about*, but not on *how to operate* an application.
Thus, dynamic changes are strongly supported by the o^2I-Trees and the generic operations performed on these trees, especially by merging the current input with the previous discourse.
The dynamic multi-application processing is further facilitated by the application management layer between the dialog system and the applications. The layer administrates the set of applications on behalf of the dialog system, including handling and distribution of an updated set of knowledge sources.

Q4: *How does a dialog system determine the best interpretation hypothesis for the user input, given the uncertainty and ambiguity in the underlying models and knowledge sources?*

The input into a dialog system can be interpreted in a variety of different ways, mainly due to ambiguities and uncertainties. DYMALOG faces this challenge by allowing multiple hypotheses and introducing an explicit hypothesis selection process. Each component up to the hypothesis selection component may introduce additional hypotheses.
In order to select the 'best' hypothesis, we rely on a parameterized rating motivated by a stochastic measure. The optimization of the parameterized rating is performed by the novel estimation algorithm GRAIL (global-rank optimization algorithm). It is based on rank considerations of the 'true' hypothesis compared to a set of alternatives.
The behavior of the system depends on the parameterization of the selection measure. Therefore, the annotation guidelines of the training corpora for GRAIL influence the overall system behavior. Due to the centralized decision, we observe a consistent behavior with respect to the selection of the 'best' hypothesis. Presently, the system is trained to integrate the current input with certain content from the discourse.

Depending on the selected configurations of the DYMALOG components, on average 1.9 to 155.8 hypotheses are generated by the system.

Q5: *How can the proposed approaches be combined into a multi-application dialog platform?*

First of all, the *Marvin* dialog system, which implements DYMALOG, proves that the framework is able to combine all the different concepts in a single dialog framework.
We observe major dependencies in some of the concepts. The separation of dialog system and applications, especially the application-blind approach, strongly facilitates dynamic changes in the underlying application setup. We could clearly identify the touch points of the applications with the dialog system.
The multi-hypotheses approach required appropriate design decisions, especially in the AIDE

components. The use of a central hypothesis selection component completed the multi-hypotheses handling.
The application management layer to administrate the applications leads to a single entry point to the different applications. It includes the handling of the various knowledge sources. With this component, the *Marvin* dialog system is able to deal with a collection of applications in the field of consumer electronics.

The DYMALOG with its innovative concepts was successfully realized in the *Marvin* dialog system. By means of this system, we are able to experiment with this collection of approaches and to refine the concepts iteratively.

Erweiterte Zusammenfassung

Sprachbasierte Dialogsysteme erlauben die Kontrolle von Applikationen mittels Sprache. Dialogsystem und Anwendung sind zumeist eng miteinander verwoben, die zugehörigen Wissensquellen der Komponenten eines solchen Systems sind jedoch oft konfigurierbar. Die enge Anbindung der Anwendungen wird bereits durch einige Dialogsysteme aus der Forschung, die gleichzeitig auf mehrere Anwendungen zugreifen können, gelockert.
Die vorliegende Arbeit stellt das innovative und umfassende Framework DYMALOG (dynamic multi-application dialog framework) für sprachbasierte Benutzerschnittstellen vor, in dem die Applikationen vom Dialogsystem separiert werden. Auf höchster Ebene lässt sich DYMALOG in Eingabeverarbeitung, Ausgabeverarbeitung, Dialog-Engine und applikations-relatierte Elemente aufteilen. Das Hauptinteresse gilt der Dialog-Engine AIDE (application-independent dialog engine for dynamic environments). AIDE stellt das zentrale Element des ‘applikationsblinden’ Ansatzes dar: DYMALOG beschränkt die Abhängigkeit bezüglich der Applikationen auf die so genannten ‘application operation parameter (AOP)’, wohingegen heutige Systeme die Applikationen üblicherweise direkt oder mittels eines geeigneten Modells intern nachbilden.

Die weiteren Ausführungen zu den Konzepten von DYMALOG werden anhand von fünf zentralen Fragen gegliedert:

***F1**: Welche Methoden ermöglichen die Verringerung oder sogar die Beseitigung der engen Verbindung eines Dialogsystems und der assoziierten Applikationen?*

Eine wesentliche Grundlage zur Separation von Dialogsystem und Applikationen ist die geeignete Darstellung der AOP, welche den Anwendungen zu Verfügung gestellt werden um entsprechende Aktionen auszulösen. Die objekt-orientierten Interpretationsbäume (o^2I-Trees) bilden die Grundlage dieser Separation in DYMALOG. Den Bäumen liegen Ontologien zu Grunde. Die o^2I-Trees dienen verschiedenen Aufgaben, insbesondere der Darstellung der Interpretationen der Benutzereingabe und des Ergebnisses der Applikations-Verarbeitung. Ferner tauschen die Komponenten in DYMALOG, wie auch das Dialogsystem und die Applikationen, Daten mittels der o^2I-Trees aus.
Dialog-Objekte (DOs) bilden die Grundlage der o^2I-Trees und besitzten selbst eine komplexe Struktur. Beziehungen zwischen diesen DOs führen zu den baumartigen o^2I-Trees.
Die DOs bilden die Basis für Operationen in AIDE, insbesondere während der Integration der aktuellen Benutzereingabe und der bisherigen Diskurshistorie. Die Integration wird anhand einer neuartigen applikations-unabhängigen Integrationsstrategie durchgeführt: zunächst werden die relevanten Elemente zur Integration identifiziert, anschließend zentrale Objekte in Beziehung gesetzt und schließlich die eigentliche Integration durchgeführt.
Innerhalb von AIDE ist die Abhängigkeit in Bezug auf Applikationen im Domänen-Modell konzentriert. Somit ist DYMALOG applikationsblind; es findet keine explizite Modellierung der Applikationen im Dialogsystem statt.

***F2**: Ist die Generalisierung von Interaktionskonzepten möglich, insbesondere im Hinblick auf verschiedene Applikationen und Meta-Kommunikation?*

Derzeit befasst sich DYMALOG mit zwei Arten von Generalisierung: den generischen Funktionalitäten sowie den (Applikations-)Bausteinen. Während die generischen Funktionalitäten direkt auf die Interaktion von Benutzer und System abzielen, richten sich die Bausteine an den Applikationsentwickler. Eine strikte Trennung ist jedoch nicht möglich.
Zur Klasse der generischen Funktionalitäten gehören Methoden um den Status des Dialogsystems zu manipulieren, etwa die Navigation in der Diskurshistorie, um z.B. vorherige Zustände einfach wieder herzustellen, oder die Kontrolle des sichtbaren Ausgabebereichs. Als mächtiges

Hilfsmittel hat sich die generische Behandlung von Listen erwiesen.
Die Bausteine unterstützen die Entwicklung von Applikationen mit Schnittstellen zu DYMALOG und umfassen Wissensquellen, insbesondere Ontologien, sowie optionale Implementierungen von zugehörigen Funktionalitäten. Ein Applikationsentwickler kann einen Baustein als solchen oder in modifizierter Form verwenden. Derzeit sind u. a. Bausteine mit Bezug auf Personen, Orte und Zeit verfügbar.
Als herausstechende generische Funktionalität steht die Bereitstellung von Wissen über Applikationsgrenzen hinweg zur Verfügung, siehe auch F1. DYMALOG kann 'kompatibles' Wissen automatisch anderen Applikationen zur Verfügung stellen. Dabei ist es nicht nötig, die Kompatibilität zwischen den Anwendungen explizit zu formulieren.

***F3**: Wie kann ein Multi-Applikationssystem dynamische Änderungen in der Menge der verfügbaren Anwendungen ermöglichen?*

Eine Änderung der verfügbaren Anwendungen zur Laufzeit bedingt natürlich die Anpassung des Dialogsystems zur Laufzeit. Im speziellen Fall von DYMALOG bedeutet dies die Anpassung der Wissensquellen in den Komponenten zur Laufzeit. Der applikationsblinde Ansatz sorgt in diesem Fall dafür, dass keinerlei interne Applikationsmodellierung angepasst werden muss. In AIDE ist nur das Domänen-Modell betroffen und benötigt die aktuellen Beschreibungen der AOP. Die applikations-unabhängige Verarbeitung benötigt das Wissen wie über die Elemente einer Anwendung *gesprochen* wird und nicht wie die Applikation *kontrolliert* wird.
Die o^2I-Trees und die generischen Operationen auf diesen Bäumen unterstützen somit dynamische Änderungen der verfügbaren Anwendungen, insbesondere durch die Integration von Benutzereingabe und Diskurs.
Ein weiteres zentrales Element ist die Applikations-Management Komponente, die zwischen Dialogsystem und Applikationen vermittelt. Diese Komponente verwaltet die verfügbaren Applikationen im Auftrag des Dialogsystems, dies umfasst im Besonderen die Verwaltung der Wissensquellen und die Bereitstellung aktualisierter integrierter Wissensquellen.

***F4**: Wie kann ein Dialogsystem die beste Interpretations-Hypothese für die Benutzereingabe bestimmen, insbesondere unter Berücksichtigung der Unsicherheiten und Ambiguitäten der zugrundeliegenden Modelle und Wissensquellen?*

Unsicherheiten und Ambiguitäten können zu verschiedenen Interpretationen der Benutzereingabe führen. DYMALOG begegnet dieser Herausforderung mit der Verwendung alternativer Hypothesen und der Einführung eines zentralen Prozesses zur Hypothesen-Auswahl. Verschiedenste Komponenten können Hypothesen generieren.
Der Hypothesen-Auswahl liegt eine parametrisierte Bewertung zugrunde, abgeleitet von einem stochastischen Maß. Um die Parametrisierung zu optimieren wird der neuartige offline Algorithmus GRAIL (global-rank optimization algorithm) verwendet. Die Grundidee von GRAIL basiert auf dem Vergleich des Ranges der 'besten' Hypothese im Vergleich mit den Konkurrenten.
Das Verhalten des System hängt stark von der gewählten Parametrisierung ab. Somit beeinflussen die Annotationsrichtlinien für die gesammelten Korpora auch indirekt das Verhalten des Dialogsystems. Aufgrund der zentralisierten Auswahl zeigt das System ein konsistentes Verhalten im Bezug auf die Auswahl der besten Hypothese. Zur Zeit wurde die Parametrisierung so trainiert, dass das System die aktuelle Benutzereingabe mit bestimmten Informationen aus der Diskurshistorie anreichert.

Abhängig von der gewählten Konfiguration der DYMALOG Komponenten wurden im Mittel zwischen 1.9 und 155.8 Hypothesen beobachtet.

***F5**: Wie können die vorgestellten Ansätze in einer Multi-Applikations-Dialog Plattform kombiniert werden?*

Zunächst lässt sich feststellen, dass die verschiedenen hier vorgestellten Konzepte in einem System realisiert werden können. Den Nachweis liefert das *Marvin* Dialogsystem.
Einige der Konzepte weisen starke Abhängigkeiten auf. Die Spaltung von Dialogsystem und Applikationen, hier besonders der applikationsblinde Ansatz, ermöglicht dynamische Änderungen der verfügbaren Applikationen. Die Berührungspunkte von Applikationen und Dialogsystem lassen sich in diesem Ansatz klar identifizieren.
Der multi-Hypothesen Ansatz benötigt entsprechende Design-Entscheidungen, insbesondere in den AIDE Komponenten. Die Komponente zur Hypothesen-Auswahl komplettiert die Multi-Hypothesen Verarbeitung.
Das Applikationsmanagement stellt zentral den Kontakt zu den Applikationen her und verwaltet diese. Dies umfasst die Verwaltung und Bereitstellung der Wissensquellen. Im *Marvin* Dialogsystem erlaubt dieses Vorgehen die Kontrolle von unterschiedlichen Applikationen der Unterhaltungselektronik.

Die innovativen Konzepte in DYMALOG konnten erfolgreich im *Marvin* Dialogsystem umgesetzt werden. Anhand dieses Systems konnte mit den verschiedenen Ansätzen interagiert und iterativ Verbesserungen erzielt werden.

Acknowledgements

During the last years when I worked at this thesis, I found overwhelming support by my supervisor, colleagues, friends, and family. A few people took over an outstanding role, and I want to express my appreciation.

First of all, it is my heart's desire to very much thank my supervisor *Prof. Dr. Dr. h.c. Wolfgang Wahlster*. When I approached Prof. Wahlster with my ideas for a doctoral thesis, he encouraged me and took over the role as supervisor. Initially, we shaped the ideas to define the topic of the thesis. As things developed, Prof. Wahlster provided inspiration and guidance likewise. While the concepts underlying DymaLog evolved, we discussed details as well as more global issues, which pointed me to alternative directions, where necessary, and avoided some pitfalls. The broad experience of Prof. Wahlster was also valuable during the realization of the *Marvin* dialog system, which implements the concepts.
I also would like to thank *Prof. Dr. Dietrich Klakow* and *Prof. Dr. Th. Herfet*. When I started working in the area of dialog interfaces, I was able to work in close cooperation with Prof. Klakow. During this time, I was able to benefit from his broad expertise in the field of (spoken) language technology. He was also very open for discussions when my wish to start a doctoral thesis became more and more concrete, and accompanied me during the last years. Prof. Klakow was very open to act as reviewer when I contacted him. When I approached Prof. Herfet to take over the chair of the examining board, he directly agreed. Since Prof. Herfet led the Embassi consortium, which had a close relationship to the SmartKom project, I was glad to get his commitment for the chairmanship.
At the DFKI, I especially would like to thank *Sabine Burgard* and *Marie-Luise Fackler*. They supported me wherever possible in organizational affairs.

Since this thesis was realized partly at the Philips *Technologie GmbH Forschungslaboratorien* (Philips Research Laboratories, Aachen, Germany), I got lots of support from my colleagues. I was able to use our collection of tools, equipment, and the environment provided by the *'Man-Machine Interfaces (MI)' group* and its successor, the 'Medical Signal Processing (MSP)' group, headed by *Eric Thelen*.
In particular, I thank *Dr. Thomas Portele* who took over a special role at the Philips site. Together, we worked on the AddVoice project, which also covered Philips´ contribution to the SmartKom project. Jointly with Prof. Klakow, Dr. Portele established the contact to Prof. Wahlster. Moreover, Dr. Portele supported me wherever possible during the last years; we had numerous discussions on the framework, he implemented parts of the framework, and he gave me lots of feedback on this text.
In addition, I would like to thank my *colleagues of the MI-group*, especially *Eric Thelen*, *Andreas Kellner*, *Dr. Barbertje Streefkerk*, *Dr. Henrik Botterweck* (especially for his great work on Philus), *Dr. Jan Kneissler*, *Dr. Alexander Fischer*, *Dr. Matthew Harris*, *Dr. Anne-Katrin Kienappel*, and *Dr. Gerd Lanfermann*. Their support was manifold, including uncountable discussions,

tips, critical considerations, reviews, and motivation.

Furthermore, I wish to thank my *parents*, *sister*, *family-'in-law'*, and *friends* who encouraged and supported me during my studies and this thesis wherever possible.
Finally and in particular, I thank my wife *Inka* and my son *Janus* with all my heart; their love, their backing and motivation, and their infinite patience allowed me to realize my dream of this doctoral thesis.

Many thanks to all who made this thesis possible!

The people I mentioned here played an outstanding role, but I found lots of support, motivation, and positive feedback in my environment; too much to list all these people by name here. Again, thanks to all who helped me in realizing this thesis.

Introduction

‘Ambient intelligence’ (Aarts and Marzano [2003], Aarts et al. [2001]) and ‘connected environments’ (Brumitt et al. [2000]) start to pervade our daily lives more and more. The living room of the future may connect media players (e.g. DVD-player and harddisk-recorder), devices to access services (e.g. through the Internet), communication equipment (e.g. telephone and e-mail), and other devices and services. The TV can take over the role of an output modality (audible and visual output). A remote control and/or keyboard could be utilized to interact with the devices and services. A microphone would enable speech input, and thus spoken language interactions of a user with the devices and services in the connected environment. Figure 1 illustrates the setting: a user interacts with devices and services in the living room by spoken language input.

Figure 1: Interaction of a user with devices and services in a connected living room. The sophisticated remote control 'PHILIPS iPronto' presents the graphical user interface of the dialog system on its touch sensitive screen. In the left part of the picture, parts of HiFi devices – like amplifier, tuner, and DVD player – are visible. A central device (e.g. a PC, not shown in the picture) may host the compute intensive parts of the dialog system and serve as gateway to the devices.

In order to realize these ambitious visions of the future, significant amounts of computing power as well as network infrastructure are nowadays already available to research and development, and more and more penetrates our daily lives. This includes near- and far-field inter-device communication technologies (like Zigbee, Bluetooth, Wireless-LAN) and pervasive computing. Increasing computing power and upcoming network infrastructures allow for new applications and extend the range of accessible functions, the Internet being herald of this development. On the user interaction side, the Internet is mainly accessed from the personal computer (PC).

The vision of accessing a potentially large set of functionalities 'anytime and anywhere' requires an interface between the user and a system that permits the user

- to handle this complexity (in other words: hides the complexity to the user),
- to access the provided functionality to her advantage,
- to be in control of a variety of devices and applications, and
- to operate complex processes without being a technological expert.

Spoken language is the main interaction modality for human beings. It allows to efficiently exchanging information between humans. Therefore, a spoken language dialog system is a promising (and natural) approach to enable the interaction of a user with such intelligent environments. Wahlster and Wasinger [2006] show how a user can benefit from a tangible multi-modal interface in ambient environments, illustrated by means of an in-store setting.
The dialog framework presented in this thesis solves the challenge of providing a user interface in connected environments with dynamically changing sets of applications. Thereto, it clearly separates between knowledge and processing that is related to (i) the interaction of a user and the dialog system on the one hand and (ii) the applications that can be controlled via the system on the other hand.

Nowadays, speech as an input modality starts to penetrate out daily life. In commercial applications, three major areas that utilize speech input are:

Telephony and Terminal Based Services. Many companies apply speech-enabled telephony and terminal based services. This includes banking services, train or flight timetables, weather information, and service portals. The deployment of such systems usually requires expert knowledge to obtain a reasonable performance. However, in real life systems human operators are often integrated to handle problematic interactions.

Command-&-Control. A presently popular class of devices enables for example controlling (a subset of) the operations of a device by speech: mobile phones. The main application is name dialing. Currently, command and control applications are emerging in upper class, highly priced cars. Again, controlling a phone or operating a navigation system is appealing. Especially in the case of navigation systems, the input of the destination could strongly benefit from speech input.

Dictation. The transcription of spoken language into written language is available to the end user as PC software for quite some time now. Adopted systems with specialized knowledge sources are available to lawyers and medical experts. Speech recognition made major improvements, yet dictating *and* controlling the system by voice is still a challenging research topic. For example, should the utterance *"increase the size"* be executed or transcribed by the dictation software?

The first two application areas are concerned with the operation of devices and services. Speech is a means that could make the operation more efficient, or enable functionality that is hardly accessible, e.g. on mobile devices. Utterances of a user trigger reactions of the device or the service, like executing an action or initiating a request for further information. The user expects a reaction on her input. Further input might be required to actually execute the intended request.

Therefore, the user performs a (possibly trivial) dialog with the machine to make the machine fulfill her wishes.
In its pure version, dictation software 'just' needs to transcribe the spoken words. However, the border between dictation and the previously discussed areas is fuzzy: as soon as a dictation program is able to also execute actions on the basis of speech input, the user enters into an interaction with a 'service', similar to e.g. the telephony services described before. In an advanced setup, the dictation software may not only literally transcribe the input but also interpret the data such that a report of a predefined format is generated.

Spoken language and multi-modal dialog systems enable a machine to perform dialogs with a user. Between the commercially available systems and the vision of science fiction, the dialog systems in research laboratories address a variety of aspects. This includes the study of spoken language generation and rendering, spoken language recognition and analysis, the structure of discourse, the interpretation of utterances in the context of discourse and environment, and the interpretation of certain linguistic clues to mention some of the areas.
All these areas are still challenging. Research is performed to understand the phenomena underlying spoken language interaction, but also to enable (commercial) real world applications. The interactions of between humans differ from human-machine interactions. Machines provide only a restricted set of functionalities, and their conversational capabilities are still limited. E.g., identification and dealing with irony are open questions.

We restrict ourselves to the broad field of dialog systems for the control of devices and services, including and beyond command and control applications. The major concerns are

- enabling human-computer interaction in a setting with dynamically changing application setups,
- abstracting generic functionalities from the applications, and
- maintaining consequently alternative interpretations during the analysis of the user input.

The DYMALOG framework (dynamic multi-application dialog framework), introduced in this thesis, provides the foundation for a dialog system according to these items. The framework is realized as a PC based dialog system, being capable of operating several applications from the consumer electronics domain as a showcase.
In one of the classical science fiction books, the 'Hitchhikers Guide to the Galaxy', a robot called 'Marvin' interacts by voice in a natural way with persons. He even shows emotions. This robot heavily influenced us in our choice of a name for the dialog system†.

Allowing dynamic changes in the application setup, i.e. applications become accessible through the dialog system or get out of scope during runtime, has severe impact on the design of a dialog system. In our framework, major parts of the analysis as well as the underlying representation of information reflect this. Applications need to provide knowledge sources such that the dialog system can adapt in order to allow the user to control these applications through the dialog system. The ontologies representing the information related to an application are based on principles similar to object-oriented program design. The tree-based representation, which can be derived from such a representation, allows us to formulate application-independent methods for the integration of newly received knowledge, e.g. from the present user input, with existing knowledge from the previous discourse. The dialog system has to provide the application related

†In the BBC TV series, the marketing division of the company that produces robots like 'Marvin' defines these robots as 'your plastic pal who's fun to be with' (Wikipedia [2005]). Furthermore, in one of the early versions of the dialog system presented later, the synthesized voice sounded monotonic; 'Marvin (the Paranoid Android)' is often in a depressive mood with a monotone voice. This led to the name of the system.

processing with a consistent view on the present knowledge state. This enables the execution of operations by the applications.
Parts of knowledge sources and functionalities from different applications may interfere, e.g. recording a broadcast when a VCR and a DVD-recorder are available. Parts of some applications and the related knowledge sources may have relevance for various applications, e.g. dealing with times and dates.
This leads us to the abstraction of functionalities from the applications. Two layers of abstraction are addressed by DYMALOG. On side of the applications, it is worthwhile to separate out generic units. These units can include means to represent information and the logic to deal with this information. Such building blocks could then be used by lots of applications, like in the case of time and date mentioned above, or by a limited number of applications, e.g. the representation of a broadcast on TV. In addition, the framework enables the communication such that dialog system inherent functionality, instead of an application-dependent realization, supports the user in operating an application. We exemplify these types of generic functionalities by means of sample functionalities, like navigation in the discourse or the access to list items.
Several automatic speech recognizers, and also some natural language understanding/analysis packages, generate a result structure that contains a set of alternative processing results given their input. In general, these results are ranked according to their assessment during the recognition and analysis process. We foster the use and generation of reasonable interpretations of the user input inside the dialog system up to a certain decision point. When the selection of a hypothesis takes place, properties of the hypotheses, as well as evaluations of preceding processing steps, can be taken into account to find the optimal representation of the user's input.

In chapter I, we discuss the effects of application-independent knowledge processing for the separation of applications from the dialog system on the design of a dialog framework. The approach presented here emphasizes the need for an appropriate representation of knowledge gained through the interaction, and the processing that closely links to the selected representation. The work is connected to the literature, and the terminology as used in this text is clarified. Also, the possibilities facilitated by the clear distinction of dialog system and applications are highlighted. At the end of the chapter, we develop a criteria catalogue as a compendium for the design of a framework, which enables the separation, and introduce five key questions . An overview of the sections in chapter I is shown in figure 2.

Chapter II implements the findings of chapter I in DYMALOG. The framework is structured into components. Each component mainly serves a specialized task in the framework, the corresponding relation to the criteria in chapter I are given. After the introduction of the framework structure, chapter II comprises two major threads. Figure 3 depicts the major dependencies of the sections in the second chapter.
The first thread relates to multi-hypotheses processing. Hypotheses are generated in several places. The hypothesis selection process requires an underlying measure to assess each single hypothesis. We use a rank-based criterion. It combines ratings from preceding components, which already processed a hypothesis, with ratings based on the structure of the hypothesis together with enhanced content in a single node. In order to combine the single ratings optimally, a corpus based estimation method of the parameters required in this criterion is developed. The division of the framework allows for parallel processing of competing hypotheses of a single user input. The details on the generation of alternative hypothesis are outlined whereever the methods applied in the framework are discussed.
For multi-hypotheses processing, the major topics of interest are related to the parameterization of the selection measure and the number of generated hypotheses. We discuss the choice of an optimized parameter set for the selection measure, especially the results from the automatic

Chapter I *(p.9)*

Sec.1: Automatic Dialog Systems *(p.10)*

↓

Sec.2: Effects of Dynamic Environments on Dialog Systems *(p.13)*

↓

Sec.3: Generic Knowledge Representation and Processing *(p.25)*

↓

Sec.4: Criteria for Flexible Dialog Processing in Dynamic Application Setups *(p.34)*

↓

Sec.5: Key Questions *(p.38)*

Figure 2: Major dependencies between the sections of this thesis for chapter I. The dependency graphs for the chapters II and III are shown in the figures 3 and 4.

computation of an 'optimal' parameter set. The dialog system spreads the generation of competing hypotheses over different components. We analyze the generation distribution for varying parameterizations of the most relevant components.
The second thread relates to the handling of discourse and content. As already indicated, the representation of information collected during the discourse is based on hierarchical structures of non-trivial 'atomic' objects. The flexible approach presented here is based on ontologies. Besides other knowledge sources, each application describes its parameters required for operation in such structures. The approach strongly supports the design of new applications for operation through the dialog system. As a major benefit, the selected representation facilitates the enrichment of the present user input with previous discourse related knowledge, without bothering the applications. Actually, only the structures from the ontologies must be known to relate atomic units. Since DYMALOG splits the applications from the actual dialog engine, interfacing with applications and the design of applications comes into focus. In addition, dealing with multiple applications at the same time requires awareness by the dialog engine. The basic dialog strategy presently applied on top of the framework still contains application specific elements[†]. Yet, the strategy includes the means to access the showcases of the generic functionalities already included in DYMALOG.

As an embodiment for the framework, chapter III presents the *Marvin* dialog system. The system enables interactions of a user with the dialog system, and thus with the applications connected to the dialog system. The DVD bundled with this thesis contains video recordings showing interactions of a user with the system. An overview of chapter III is given in figure 4.
The selection of a broad range of applications allows us to observe the previously introduced concepts in interactions with the realized dialog system. The phenomena are discussed in detail on the basis of selected fragments from interactions of users with the system.

[†]However, the design of the dialog strategy based on the information structures provides hooks for the investigation of a more powerful dialog management, even more detached from the applications.

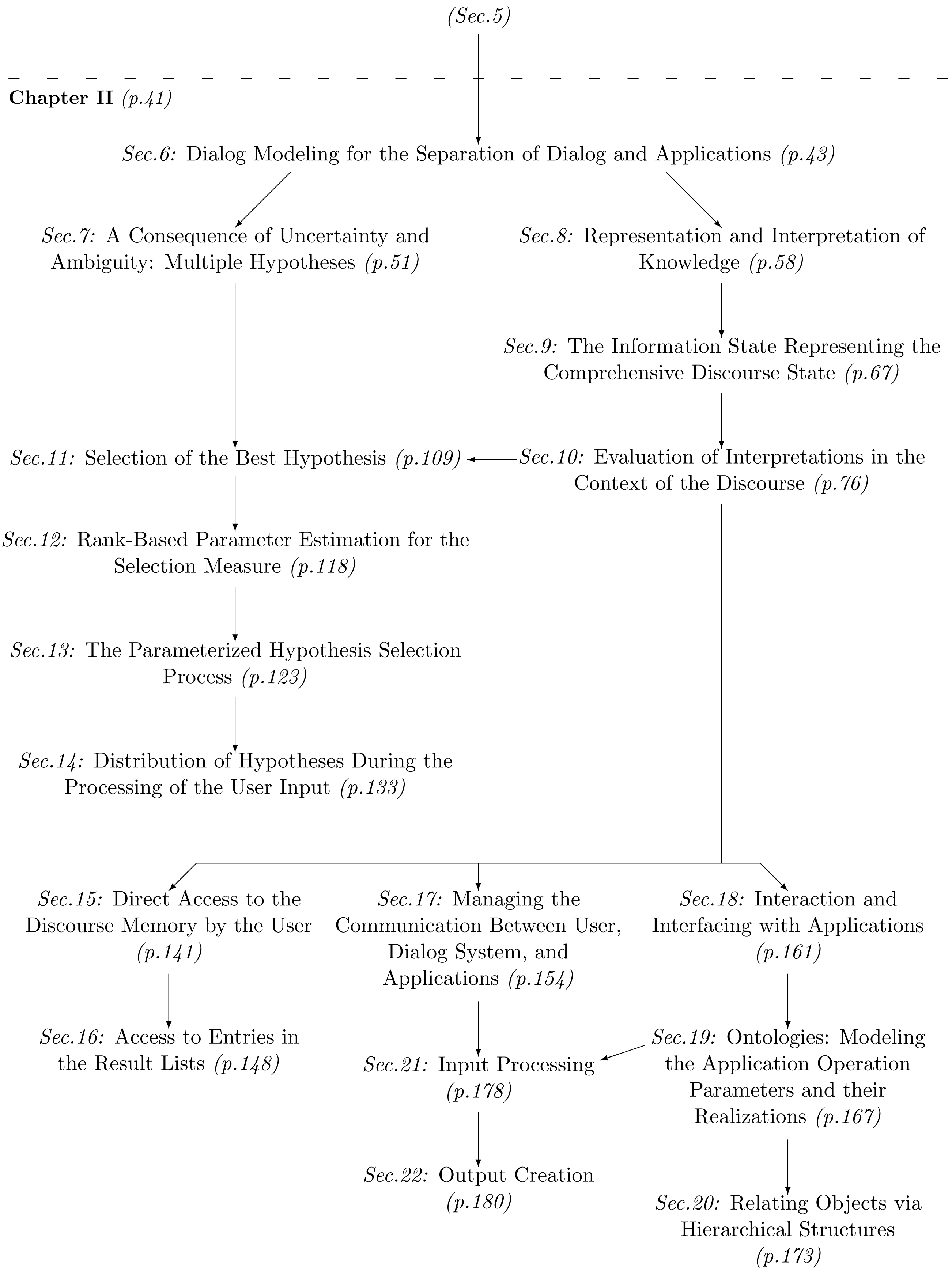

Figure 3: Figure 2, dependencies of sections, continued for chapter II. The bracketed section generates the anchor to the dependency graph in figure 2.

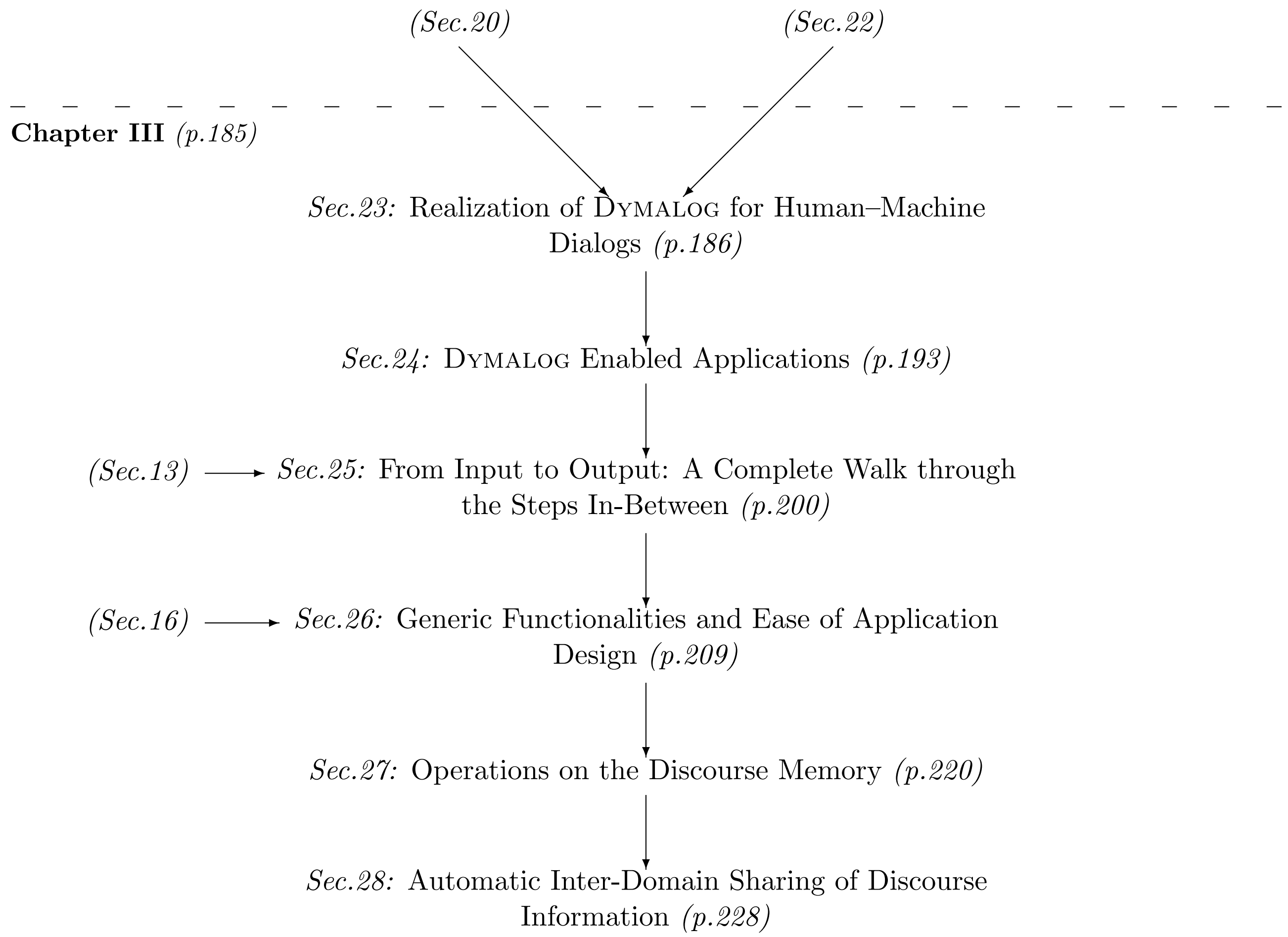

Figure 4: Dependency graph between the sections of chapter III. This completes the dependency overview for the chapters I to III.

Working Environment

The thesis was supervised by Prof. Dr. mult. Wolfgang Wahlster, Saarland University (Saarbrücken, Germany). At numerous occasions, I was able to discuss my ideas and progress with Prof. Wahlster: during the definition and derivation the framework as well as during the realization in the *Marvin* dialog system. I was able to profit from the huge expertise of Prof. Wahlster, which inspired me to find an optimal approach and let me avoid several pitfalls.

During the work on this thesis and at present, I am employed by the PHILIPS Research Laboratories Germany located in Aachen. Major parts of the work have been carried out in conjunction with my work at the PHILIPS Research Labs. During my first years at the Research Labs, I worked in the 'Man-Machine Interfaces (MI)' group. The group has been active in the in the field of speech processing for more than 25 years; initially focused on the field of automatic speech recognition and then also active in the area of automatic dialog processing. In 2004, I joined the 'Medical Signal Processing (MSP)' group, which concentrates on applications in the Personal Health Care domain and succeeded the MI-group.

The* AddVoice *and* SmartKom *Projects. Until the end of the SmartKom project† in September 2003, I was able to work directly on spoken language dialog systems. The Philips' SmartKom participation was embedded into the AddVoice project. AddVoice intended to make the results and work from the SmartKom project utilizable at Philips Research.

DyMaLog presented in chapter II forms the basis of a spoken language dialog system (chapter III). The dialog system was meant as the incarnation and extension of the result transfers from SmartKom into AddVoice. The *Marvin* dialog system can look back on a series of preceding system in the MI-group. It was developed within the AddVoice project.

Supporting Contributions to* DyMaLog *and the* Marvin *Dialog System. During the first $1\frac{1}{2}$ years of my thesis, the work on DyMaLog and the related dialog system was carried out in close co-operation with Dr. Thomas Portele. After discussing the requirements on the degrees of freedom by an application-independent dialog framework, especially the need for an integration algorithm of basic ontological units, Dr. Portele took over major parts in the realization of the domain model (described in section 20) on the basis of ontologies (section 19, especially 19.1–19.4). He further designed the application management layer with the creation of integrated knowledge sources from various applications together with a first example for an application implementation (section 18.1–18.3).

The demonstration system is based on the Philus-platform derived from MultiPlatform. The Philus-platform was developed at the MI-group as the basis of modularized demonstration systems. The dialog system realizing DyMaLog is split up into a set of modules, which communicate through Philus. Philus basically provides an adaptation of MultiPlatform for internal use by the MI-group. The ideas and concepts from MultiPlatform remain valid and also characterize the Philus-platform. Therefore, we will not make a distinction between these platforms and stick to the term MultiPlatform for clarity.
Furthermore, the input processing components utilize the Phicos speech recognizer and Susi natural language understanding, also being developed at the MI-group and used in various systems.

Communication Management. The AddVoice-project originally included a separate working area to manage the communication between a user and the dialog system on the basis of speech acts. In addition, the communication management should have organized the interaction of dialog system and applications.
Due to the withdrawal of Philips Research Aachen from spoken language technology, the communication management work package was abandoned. Thus, DyMaLog relies on a simplified communication management, section 17. It integrates basic needs from the framework and, as a result, the interaction between user, dialog system, and applications is partly formulated more explicitly than required in a more advanced communication management.

†Parts of this work were performed in the context of the SmartKom project, funded by the German Ministry of Research and Technology under grant 01 IL 905. The responsibility for the content is with the author.

Chapter I.

Conversational Human-Computer Interaction in Dynamic Application Environments

User interfaces are omnipresent in modern societies, ranging from a simple button, like a light switch, to computer-based applications and/or professional equipment, like computer tomography devices or computerized numerical control machines in industry. The number of 'available functionalities' in a person's surrounding increases, and thus the need to enable a sufficiently easy to use interface. In this broad field of user interfaces, we will focus on interfaces that stress spoken language as input modality.
The first chapter provides the context for the DyMALog framework (dynamic multi-application dialog framework) in the field of spoken language dialog systems, being part of the broad area of user interfaces. As already stated in the introduction, spoken language user interfaces start pervading our daily lives. Telephone services based on spoken language dialog systems are in the market for quite some time now. Voice dialing in mobile phones and dictation software is also available, and car manufacturer start to integrate speech interfaces into premium cars, e.g. to control and ease advanced functionalities like navigation. Research in speech technology had, and still has, its stake in the development of commercial systems. For example, Aust [1998] describes components that form the basis for several commercial dialog systems. However, Pieraccini and Huerta [2005] notice differences in the major goals of research and industry: major concerns in research systems are the naturalness and freedom of communication, while commercial systems prove that these features are not necessarily a prerequisite for effective interaction and usable systems, given the constraints defined by the present technology. They even demand a closer dialog between research and industry to foster further maturation of spoken language dialog systems.

The modularized approach for a dialog framework presented in this thesis, which clearly separates between the dialog engine and the applications accessible by this engine, honors the need for a consistent interface towards multiple applications. In addition, it faces the challenge of connected 'plug and play' environments, where applications may be dynamically added to the system. Such environments are currently emerging in the consumer electronic (CE) domain. Therefore, the framework can be considered as one of the first contributions to the dialog between research and industry, as requested in Pieraccini and Huerta [2005].

First, we introduce the fundamental notions related to dialog systems as it will be used throughout this text (section 1). Applications, which the dialog system can control, play a central role in this text. These applications are considered in the context of spoken language dialog systems.

Section 2 deals with the separation of the dialog system from the applications controlled by such a system. In this context, so-called dynamic application setups are considered, and a link to graphical user interfaces is created. This section discusses the application-dependence of knowledge sources.

The separation of the dialog system and the applications requires an appropriate representation of knowledge, e.g. as obtained from the user input. A dialog system requires a consistent view on the discourse, i.e. the present user input must be bought into the context of the previous interaction steps. In section 3, also possibilities which arise from a flexible representation are shown. This includes sharing of knowledge between applications, and generic functionality like the navigation in the discourse history. Multiple hypotheses are used to allow alternative interpretations of the user input up to a certain point.

The chapter is concluded by a catalog that lists criteria for a dialog system and its related applications to actually perform the separation, section 4. It summarizes the previous sections and structures the features being expected from a dialog system for the separation from applications in this text. The criteria are condensed into five key questions in section 5.

1. Automatic Dialog Systems

In order to arrive at a sensible definition of the term 'automatic dialog system', we start with the definition of 'conversation' by Webster's:

> "... a conversation between two or more persons;
> also: a similar exchange between a person and something else (as a computer) ..."

This leads us to the definition of such a system, as it will be used throughout this text.

1.1 Definition (Automatic Dialog System)**.** An *(automatic) dialog system* is a software artifact executed on a machine that enables a conversation with one or more human users about one or more domains of discourse. ◇

Due to constraints in the artifact, the set of topics, which can be addressed, may be restricted compared to the interactions between humans.

Usually, a device with significant computing power and a collection of programs represent the dialog system: in fact, many research dialog systems are actually software packages running on a PC.

1.2 Definition (Dialog System Enabled Application)**.** A *(dialog system enabled) application* is a software artifact, possibly connected to a hardware device and/or consisting of a set of single software artifacts. Furthermore, the notion *application* includes knowledge sources required to address functionality of the application by the dialog system. ◇

Operating an application may cause a state-change in the real world, e.g. switching a TV channel. Another common operation is the access to databases, e.g. for train timetable information. An application can be distributed over a set of software packages, e.g. a client accessing server components. Examples for knowledge sources belonging to an application are the lexicon for automatic speech recognition or the grammar for natural language understanding (NLU).

The only task of the software artifact could be to mediate between the dialog system and an existing application or device. E.g., some modern TV-sets and DVD-recorders can be operated remotely by a software program. An application for one of these devices would contain a software interface component to drive the device, the device itself, and the required knowledge sources.

The notion *knowledge sources* used throughout the text refers to knowledge sources relevant to the dialog processing. Examples are the automatic speech recognition (ASR) lexicon or NLU grammar, as already mentioned before. Not included are solely application relevant knowledge sources like databases, e.g. the database of an EPG application that contains the broadcasts for the next two weeks.

Figure 1-1 illustrates definition 1.1. Even though a dialog system might have to interact

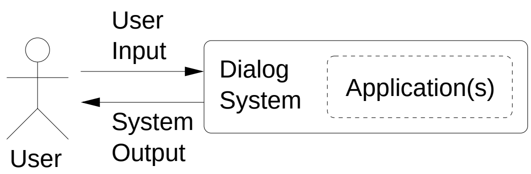

Figure 1-1: Automatic dialog system providing a user interface towards one or more applications.

with various users simultaneously in certain situations, e.g. in a living room environment, we assume that a *single user* at a time interacts with the dialog system. Besides the challenges for the core dialog processing, simultaneous input of two or more users at the same time can also be a hard task for recognition and analysis of the input. Consider e.g. the ASR in a distant microphone setup for several speakers at the same time.

1.1. Input and Output Modalities

The input by the user and system output shown in figure 1-1 can each be mono- or multi-modal, whereat these terms refer to the opportunity to generate multiple separate input or output streams (Rudnicky [2005]). Examples for common audible and visual input modalities are

- touch and gestures,
- written language,
- spoken language, and
- facial expressions.

Many systems use loudspeakers and/or a screen to provide feedback and present results of the processing. The output modalities for these systems are similar to the input modalities and include

- spoken language (prerecorded or synthesized),
- audible notifications,
- written language,
- pictures and movies, and

- gestures and facial expressions transmitted through a virtual character like an avatar.

The list of input and output means is certainly not complete: Brain-reading could be help for handicapped people with a particular disability. On the output side, robots – potentially capable of showing facial expressions – might provide new capabilities to communicate towards a user. However, we will not elaborate on these possibilities and focus on more 'traditional' modalities: spoken language will be our main input modality and, combined with visualizations on a screen, also form the output means. The framework presented is not restricted to such a setup of modalities for input and output. It might also be used e.g. for phone-based services, Bickmore and Cassell [2006].

1.2. Spoken Language Dialog Systems

In this thesis, we will focus on speech as the main input modality.

1.3 Definition (Automatic Dialog System; continued)**.** A *spoken (language) dialog system* is a mono- or multi-modal automatic dialog system using speech as input modality. Such a system will also be denoted *conversational dialog system.* ◇

The system that realizes the presented framework is a spoken language dialog system. We will focus on these types of dialog systems even though the framework presented here is in principal independent of the chosen input and output modalities. Thus, the term 'dialog system' is used as an abbreviation for 'spoken language dialog system'.

A spoken language dialog system should honor universal principles underlying the interaction of user and dialog system. E.g., the 24 principles of a cooperative dialog between human and machine derived from Wizard-of-Oz experiments in Bernsen et al. [1996]. These include the classic maxims of Grice (Grice [1975]): quantity, quality, relation, and manner. Alternatively, one might consider the Q and I principles deduced from Grice's maxims (van Rooy [2001]). A dialog is often regarded as a joint activity of the partners in the dialog, and a dialog system should heed this, Cohen and Levesque [1994] and Rich et al. [2001]. Compliance to these principles may either define a requirement for the designer of an application or be inherently available by the dialog system if possible. For the design of a spoken language dialog system one has to take into account that remarkable differences between human-human and human-machine interactions can be observed, e.g. Doran et al. [2001] notice that the machine is dominant with respect to several aspects like initiative, amount of speech, and number of dialog acts. In addition, clear differences in the dialog act patterns are visual.

The *applications*, which can be controlled by a dialog system, play a central role in our framework. An application in our terminology is an entity that provides functionality that can be controlled by a user, either directly via the entity's user interface or indirectly using another entity. Such an entity is usually a device or a service. One can think about a CEs device (e.g. TV-set, DVD-recorder), a service available through the Internet (e.g. train timetable information system, hotel reservation, book shop), or the control of a remote autonomously operating device (e.g. service robot).
For spoken language based dialog systems, Allen [1995] lists a set of potential applications areas:

- 'question-answering' systems to access databases,
- automated services via the telephone line,
- tutoring systems,

- 'command and control'-applications, and
- systems for cooperative problem solving.

The spoken language dialog system provides the technical means of a user interface, thus the term *spoken (language) dialog interface* is used to refer to such a user interface. Analogical, we will speak about *voice interfaces.*

1.3. Task-Oriented Dialogs

For DymaLog, which will be introduced in this text, we concentrate on task-oriented dialogs. Therefore, interactions of a user with the dialog system serve a certain task, e.g. executing some action or providing information to prepare an operation by an application. In Allen et al. [2001], the task-oriented dialogs are considered a subclass of so called 'practical dialogues'. For these types of dialogs, the following assumption is formulated (Allen et al. [2000]):

> *The Practical Dialogue Hypothesis:* The conversational competence required for practical dialogues, although still complex, is significantly simpler to achieve than general human conversational competence.

As we will see later, DymaLog does not focus on the realization of an interaction style where the conversation of human and machine is as close as possible to human-human interactions. A strategy that is consistently applied in the interaction with different applications constitutes the basis for predictable responses of the dialog system on input by a user.

2. Effects of Dynamic Environments on Dialog Systems

Improvements in public communication networks and inter-device communication capabilities together with increasing processing power, especially in small devices, are the first steps towards the vision of ambient intelligence and connected environments. Improvements in ASR enable the deployment of dialog systems in new contexts. Deploying spoken dialog systems as user interfaces in these intelligent environments[1] thus poses additional specific demands on spoken language dialog interfaces.

To elaborate on these specific needs, we will now outline two application scenarios of intelligent environments.

Example: Living Room. Connected CE devices and services demand for a user interface being able to control a plurality of functionalities. Together with techniques facilitating more robust ASR, e.g. beamforming and echo cancellation, dialog systems show the potential to be a powerful means to handle this complexity as central user interface. The application bandwidth comprises classical command-and-control applications, like switching the TV channel, and access to content from databases, e.g. in an electronic program guide. I.e. devices and services are candidates to be controlled by a conversational voice user interface in the living room.

[1] The notion *intelligent environment* will be used for an environment equipped with connectivity technology to (i) communicate to the 'outside' world, (ii) allow devices and services to communicate among themselves, and (iii) allow remote control of devices and services being connected by other entities in the environment. In addition, a 'sufficient amount' of computational processing power is available.

Connecting CE devices in the living room allows for new interfaces to these devices. Evolving near- and far-field communication technologies also allow sophisticated voice interfaces for mobile devices, e.g. see Pieraccini et al. [2005].

Example: Mobile Devices. Current and upcoming mobile telephony networks, like the 3rd generation Universal Mobile Telecommunications System (UMTS) networks, provide high data communication bandwidth. Mobile devices are equipped with more and more processing power. Therefore, these devices can become the basis of an advanced user interface to

1. control the built-in functionality of such a mobile device, and
2. operate services that become available through a network.

The latter includes location-based services enabled by mobile telephony networks, network infrastructures based on Wireless Local Area Network (W-LAN) technology, or peer-to-peer networks.
Even if the computing power of such devices is not sufficient to perform all required computations locally, high-speed data connections allow spinning out the processing by usage of remote resources.

2.1. Dynamic Application Environments

Both application areas of dialog systems sketched in the above examples require capabilities to adapt to changing setups of the controlled applications. The period and frequency of changes in the application setup can significantly differ depending on the applications and environmental conditions.
A TV-set, a DVD-recorder, or an EPG in the 'living room' scenario are replaced rarely. However, devices like a portable music player, a personal digital assistant (PDA), or mobile phone – if connected to a living room network – can also provide functionality and data. Moreover, the availability of such devices in the living room environment may change frequently. The user interface must reflect this.
A mobile device in the 'mobile devices' example could integrate means to access location based services. Once the mobile device is connected and some location-based services become available, the mobile device serves as interface between the user and the service. The service, which is controlled by the mobile device, might already be installed on the device and just be activated when a certain location is approached, or the complete application or parts of it are dynamically uploaded onto the device.

In both situations, the system controls a set of applications that cannot be anticipated and will therefore be called 'dynamic'.

2.1 Definition (Dynamic Application Setup). An *application setup* $(\mathcal{A}, \mathbf{h})$ *(in connected environments)* consists of a set of applications $\mathcal{A}$ and a central hub $\mathbf{h}$, each application $a \in \mathcal{A}$ being capable of connecting to the hub $\mathbf{h}$ such that

- the hub can bidirectionally communicate and interact with a, and
- the application might provide knowledge sources to the hub defining specifics of the application.

A *dynamic application setup* $(\mathcal{A}_{\text{conn}}, \mathcal{A}_{\text{non-conn}}, \mathbf{h})$ is an application setup that furthermore distinguishes between applications being connected to the hub $\mathcal{A}_{\text{conn}}$ and not being connected to the

hub $\mathcal{A}_{\text{not-conn}}$. The status of an application can change over time which is reflected by changes of $\mathcal{A}_{\text{conn}}$ and $\mathcal{A}_{\text{not-conn}}$.
The setups will also be denoted *(changing) application environment.* ◇

The set of all applications relevant in the context of the dialog system $\mathcal{A}_{\text{conn}} \cup_{\text{disjoint}} \mathcal{A}_{\text{non-conn}}$ is not static, i.e. as new applications become available over time, these sets may be extended (e.g. when new applications are realized).
For a dynamic application setup $(\mathcal{A}_{\text{conn}}, \mathcal{A}_{\text{non-conn}}, \mathbf{h})$, the related application setup is given by $(\mathcal{A}_{\text{conn}} \cup_{\text{disjoint}} \mathcal{A}_{\text{non-conn}}, \mathbf{h})$.

Potential realizations of the hub as introduced in definition 2.1 are a dedicated hardware device or a virtual entity, i.e. a software component integrated in one device or spread over two or more connected devices (Maes [2000]). Figure 2-1 illustrates the dynamic application setup for the 'living room' example.

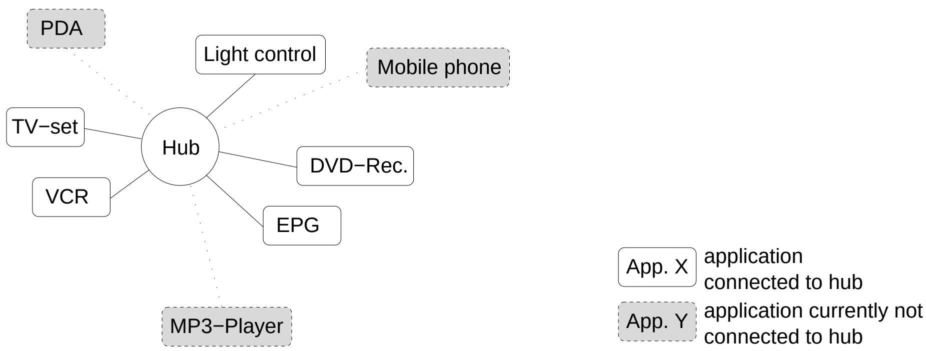

Figure 2-1: Dynamic application setup for a living room scenario (see also page 13). Five applications are connected to the hub, three are currently not connected.

The case of a hub operating the connected applications will be of particular interest in the remainder of this chapter:

> Dynamic application setups pose special requirements on a dialog system. Connecting or disconnecting an application from the dialog system implies appropriate adaptations of the system. For example, ASR and NLU need to reflect changes in the application setup. Formulations to address functionalities of the affected applications are integrated or removed after a change. The internal processing of the user input to operate the applications also has to allow changes in the application setup. Either the internal processing is reconfigured after a change or the processing is formulated independent of applications. The dialog system cannot restrict itself to a single application, but must enable access to one application out of set of connected applications.

In Pakucs [2002], 'Plug and Play' dialog management (for mobile environments) is addressed. Applications are described using VoiceXML. However, the 'SesaME' dialog manager presented in these papers derives its dialog strategies from the VoiceXML descriptions, but is not a VoiceXML interpreter. The VoiceXML of a newly added application is used to automatically generate program code (`Java`), which is added into the dialog system. During the interaction, the dialog system tries to identify the dialog description of the addressed application, activate the chosen description, and then perform the dialog management according to this description.

Yet, 'SesaME' (Pakucs [2003b]) tries to 'coordinate and control multiple concurrent speech interfaces' in mobile environments, i.e. each application has its own interface. Application specific data includes dialog management capabilities, domain knowledge, etc. and is encoded in service descriptions stored at service provider side. It cannot be avoided that a single user utterance triggers several different applications.

2.2. Accessibility of Knowledge Sources

Dynamic application setups pose challenging requirements on the components of a dialog system. Consider the ASR and NLU in the case of a system using speech as the main input modality. The performance of speech recognition and language understanding depends on the sizes of related knowledge sources like lexica, language models, and grammars. If the coverage of user utterances, which can be recognized and processed, is increased, this leads to interactions that are more flexible. However, the risk of misrecognitions and misunderstandings through confusion and ambiguity also increases.
In a dialog system, which dynamically adapts to changing application setups, the state of the knowledge sources, associated with an application, change according to the state of the application. The relation of the knowledge sources of the application and the knowledge sources present in the dialog system can be divided into three states:

- *Knowledge sources of a disconnected application.*
  The knowledge sources of a disconnected application do not contribute to the knowledge sources utilized in the dialog system. The dialog system operates independently of such a disconnected application. Application app.#1 in figure 2-2 illustrates the relationship between the knowledge sources of the dialog system and the application.

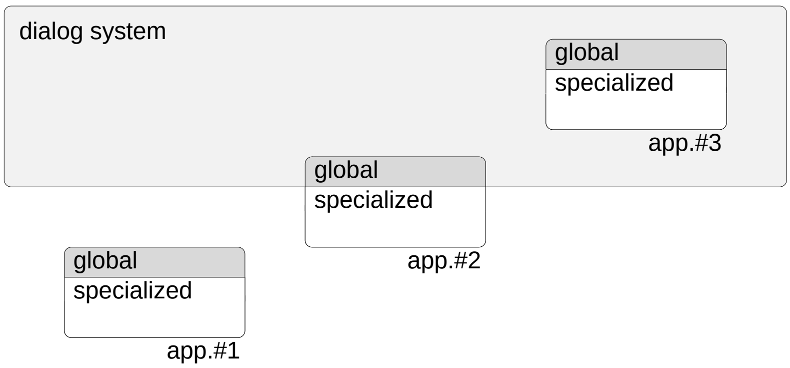

Figure 2-2: Illustration of different states in the relationship of the knowledge sources from an application and the knowledge sources of a dialog system. The application's knowledge sources may either be independent of the dialog system's knowledge sources (app.#1, disconnected), partly contribute to the overall knowledge sources (app.#2, connected & deactivated), or fully add to the system's knowledge sources (app.#3, connected & activated).

- *Knowledge sources of a connected application.*
  Knowledge sources of an application connected to the dialog system (partly) contribute to the overall knowledge sources available to the dialog system.

– *Global parts.* As soon as an application connects to the dialog system, parts of the knowledge sources are published to the dialog system. The sections of the knowledge sources that become available directly after connecting to the dialog system are called *global*, see application app.#2 in figure 2-2.

– *Specialized parts.* When a user addresses functionality of the global parts of an application, a state change of the application may be implied. As a consequence, additional elements of the knowledge sources are delivered to the dialog system and integrated, application app.#3 in figure 2-2.
  As an example, utterances like *"go to the <application name>"* or *"enter the <application name>"* trigger the application to publish its specialized knowledge. On the other hand, *"leave the <application name>"* may be used to remove the specialized knowledge from the dialog system's knowledge. I.e. the application is (explicitly[2]) activated or deactivated.

The specialized knowledge sources of an application may be further partitioned. Thus, depending on the application's internal state the knowledge sources published to the dialog system are adjusted, e.g. the normal operational mode of an application generally differs from a selection of an item out of a list of n items.
In addition, the knowledge sources of an application may not contain specialized parts at all. Instead, only the global part is present. Applications with a 'manageable' size of their knowledge sources are candidates for such non-partitioned knowledge sources.
Besides the explicit deactivation of an application, a time-out could be used to deactivate an application that was not accessed for a certain period of time.

The dialog system can benefit from the division of knowledge sources in case of applications that require complex user interfaces. In order to improve the (technical) performance of components, the applications declare very limited parts of their knowledge sources globally, while the major functionality becomes available only if the application is explicitly entered.

2.3. Separation of Dialog and Application

As will be shown later, applications often closely integrate with the (dialog) system that provides the user interface. For devices, this connection is very common and meaningful: devices deliver the user interface to control and access their functionality. Consider e.g. a mobile phone, a DVD-player, or a car:
Current mobile phones provide a keypad and a small screen to access the phone's functionality like dialing a number or sending, receiving, and reading short messages. Most DVD-players are equipped with a remote control and a small display on the device. However, DVD-players often make use of the TV-screen to provide feedback and show information to the user: thus 'embed' parts of the user interface in another device. A common car operates completely autonomously; the user interface comprising the steering wheel, pedals, and gearshift is completely adapted and integrated into the car.

As an increasing number of sophisticated operating systems find their way into devices, these devices are extensible in their original functionality. We can identify three types of functionalities:

1. built-in functionality of a device,

[2] An application may also be implicitly activated. E.g. an EPG application may include means to process utterances like *"what is on TV"* in the global knowledge sources. If the application is not activated yet, this input may trigger both, an operation by the application and the publication of some specialized parts from the knowledge sources.

2. functionality available through an upgrade of the device, e.g. a program that is installed on a mobile phone, and

3. remotely operated functionality, thus the device only provides the means for the user interface.

The latter includes internet services (type 3) running in a browser (type 1 or 2)[3].

2.3.1 Graphical User Interfaces. This division finds its counterpart in state of the art operating systems together with graphical user interfaces (GUIs) for the PC, e.g. Microsoft Windows XP, which closely integrates the GUI into the operating system, Apple MacOS X or Unix/Linux desktops like the K-desktop-environment or Gnome that build upon the X-server architecture. Applications controlled through a web browser are out of scope by the GUIs considered here; i.e. these applications usually provide their own look-and-feel and do not adhere to interface guidelines of a certain GUI, e.g. its controls (like buttons) and layout. Though, applications that are embedded into the environment of a graphical interface should adhere to interface guidelines of the particular desktop, usually supported by application programming environments (APIs). These aim at a common look-and-feel such that a user can orient herself easily in various applications. This includes the interaction with windows shown on the screen (e.g. `minimize`, `maximize`, `close`), unified menu structures (e.g. `file`→`save`, `edit`→`copy`), common controls (e.g. buttons, list views) and their placement, and standard components shared between applications (e.g. dialog to open a file, help browser). An approach in this direction for spoken language dialog systems is given in Rosenfeld et al. [2000].

The GUIs currently do not have an analog in the field of spoken language interfaces. For instance, GUIs provide means to

- create applications with a unified look-and-feel,
- manage the interaction between the user and a variety of applications, and
- allow installing, executing, and accessing new applications during run time.

A program, which makes use of a GUI, usually can be divided into two parts: the user interface related part and functionality decoupled from the user interface. Consider a program to visualize fractals. The algorithms to compute the fractals are independent of the user interface, while the presentation towards the user, maybe enriched with the option to modify some parameters, directly relates to the underlying user interface realization. Often, algorithms and user interface are realized in the same programming language, e.g. `C`, `C++`, `C#`, or `Java`, and thus closely integrated. Modern integrated development environments (IDEs) strongly support the connection, they often include means to design GUIs.

The side-by-side coexistence of applications as perceived by the user is manifested by a set of windows, organized by the GUI (with support from the underlying operating system). The user explicitly activates the application she intends to interact with while, at the same time, all other applications run in the background. In a multitasking operating system, the applications in the background can still perform computations.

Maes [2000] postulates the 'conversational user interface' (adopting the 'Model, View, Controller' design paradigm) as the next step in interaction of user and applications after the GUI.

[3]To be precise, type 3 itself must be divided into applications for which the device only renders the output, e.g. Internet services, and 'real' client-server applications where some proprietary user interface runs on the device, but the functionality provided by the application is offered remotely.

While some of his requirements also meet for a dialog system enabling the interaction with applications in a dynamic application setup, Maes [2000] especially faces the challenge of interacting with an application from different interfaces with varying modalities, e.g. a mobile phone on the move or via a web browser or legacy application on a PC.

2.3.2 Relationship of Applications and Dialog System. The bonding of applications and dialog system may differ strongly. An extreme integration is achieved when applications and dialog system form a monolithic unit. E.g. a designer of an automatic dialog system programs the entire system in a standard programming language, including rules to analyze the user input or strategies for the interaction with the user. Such a solution may be practical for setups with low computing resources. 'Porting' such a system to another application would require a complete redesign of the application (Glass [1999]).

In the next advanced stage, the dialog system and applications may still be closely connected. However, the system is divided into knowledge sources and processing units. The processing units implement strategies that can be adapted to certain setups by the knowledge sources. A NLU unit e.g. may implement the basic means to parse some text input and return a semantic representation of some kind. To define the mapping between the input entities and semantic entities for the output, the NLU could rely on a set of rules provided by the related knowledge source. Alternatively, a dialog management unit may receive a set of rules that define the reactions on certain user input. Generally, the differentiation between processing units and knowledge sources is the least commonly applied for most building blocks of today's dialog systems. The applications and the dialog system are still closely integrated. The processing units implement functionalities to operate the application (or even implement the application itself), and the target application may heavily influence the knowledge sources. This holds e.g. for the knowledge sources utilized during recognition and analysis, but usually also for knowledge sources that contain the dialog strategy. These might also comprise directives how the knowledge collected in previous turns merges with knowledge from the current user input. Enabling new application functionalities or porting the system to another application usually requires adaptations of all knowledge sources. Often, also certain processing units need modifications to enable additional actions. Advanced (commercial) design tools are available to support professionals in creating and maintaining an application, like the 'DialogDesigner' for rapid application development (Dybkjær and Dybkjær [2005]).

Up to now, the designer of a speech interface had to ensure consistency of the user interface. For the future, it is desirable that the dialog system provides means to design a consistent user interface, a step in the direction of even more separating application and dialog system. Consistency does not necessarily imply that the dialog system is capable of understanding human-like natural speech input. Pieraccini and Huerta [2005] state that commercial dialog system implement efficient interfaces, while mostly not aiming at natural, free speech input. Many research dialog systems however aim at powerful input processing, allowing as natural as possible speech input. The 'Universal Speech Interface' project (Shriver et al. [2001]) targets a standardized speech interface for different applications. It is based on 'basic design principles'. However, porting the interface to different applications still requires major adaptations, though the effort seems to be reduced compared to most present dialog systems. A user who is accustomed to interact in accordance to these principles should easily be able to control another application developed for this interface. Hagen and Popowich [2000] describe a dialog manager based on so-called 'primitives'. An appropriate description of the application should allow the user to control the application through the dialog system. Pieraccini et al. [2001] aim at an (partly) automated design and implementation of a dialog strategy. The selected approach is based on the identification of com-

mon building blocks between applications. The dialog designer may use these blocks to build her application. In Alshawi and Douglas [2001], strategies are presented to ease the design of a dialog system by deriving interfaces for new domains from examples and background knowledge. The system developer does not deal with intermediate semantic representations. The communicator system described in Rudnicky et al. [1999] still closely integrates the target application into the system, but already starts separating between the dialog manager and functional components inside the dialog system related to the application.
Phone based dialog systems to enable automated services with speech input and speech output are already widespread. Besides a variety of proprietary solutions by various companies and research laboratories, standards as VoiceXML and SALT define a common basis to develop phone or web based speech interfaces (Kellner [2005], Ragget [2002]).

An advanced approach to profitably combine multiple applications has been developed in Embassi (The Embassi Consortium [1999–2003]). Herfet et al. [2001] describe the approach to profitably combine a variety of applications to execute goals in a system that provides 'polymodal, conversational dialog structures'. The user who interacts with the Embassi system specifies a goal, which is intelligently translated by the system into 'a sequence of actions that will produce a desired effect ("goal")'. Embassi applies a goal-oriented approach to dialog modeling – in contrast to the commonly used function-oriented interaction. The 'pragmatics-first view on rational interaction' underlying the goal-oriented approach relies on description logic as representational framework to describe dialog and application related logic, thus favoring a plan-based approach for dialog modeling. This approach can be used to draw inferences on the present state to facilitate the interaction of the system with the user. The pragmatics-first view is motivated and described in more detail in Görz [2002].

Yet, these approaches still integrate characteristics on the knowledge processing for certain applications into the dialog system, and even the dialog engine. Consequently, the next stage is to separate the application at least from the dialog engine.

2.3.3 Voice User Interface. The task of a dialog system is to some extent similar to that of a GUI on a PC, even though interacting by speech poses some peculiarities on a system. E.g. in a *voice user interface (VUI)*[4] a user can in principle address functionality of an arbitrary application, no matter if this application is currently 'in focus' or not, and without explicitly switching to that application (Papineni et al. [1999], Ramaswamy et al. [1999]).
In order to enable the counterpart of the GUI in the field of voice user interfaces, we address the tasks of

- providing a framework that can be utilized by a broad class of applications to realize a voice interface without realizing a single interface per application,
- allowing addressing a variety of applications through one interface,
- serving as interface in a dynamic application setup, and
- enabling a consistent behavior, i.e. having a similar 'look-and-feel' in the interaction,

(Maes [2000]). From these tasks, we can conclude that such a voice interface must include capabilities to switch between applications being under control. Furthermore, the framework

[4]The term *voice user interface* is often used in literature analog to the term *dialog user interface*, e.g. see Pieraccini and Huerta [2005].

should be able to integrate existing applications. Usability and user interface issues in multi-domain dialog management for mobile environments (Pakucs [2003b]) should be considered for whole environments of such an interface, but not for isolated services and appliances.

Allen et al. [2001] state that a dialog system 'has to know much about the task being implemented by the back end'. However, first steps in the direction of a dialog engine that explicitly defines an application description for the separation of the application from the dialog engine can be found in Denecke [1997, 2002].

2.2 Definition (Separation of Dialog System and Applications)**.** A dialog system **d** is *separated* from a set of applications $\mathcal{A}$ if, in order to control an application $a \in \mathcal{A}$ through the dialog system **d**, the adjacent holds:

1. **d** defines a fixed set of knowledge sources that have to be supplied by a to let **d** provide a dialog interface to a. It is not required to modify **d** – apart from updates through changing knowledge sources – in order to provide such an interface.
2. The knowledge sources from a comprise means to
   - analyze the user input, including the extraction of semantic meaningful entities,
   - establish relationships between semantic entities, including the entities from the analysis of the user input, and
   - transform content obtained from a into a representation that is suitable for rendering by one or more of the output channels.
3. **d** does not include processing logic that is specific for performing actions in a.
4. The interpretation of semantic entities required to perform actions by a with respect to the given user input takes only place inside the application, but not within **d**.

The applications $\mathcal{A}$ are then *decoupled* from the dialog system **d**. ◇

Section 2.3.4 will illustrate how the existence of application-specific knowledge inside the dialog system according to items 1 and 2 is necessary to allow a user to interact through the system with an application.

Definition 2.2 encompasses dialog systems being restricted to control only one application at a time without being able to switch between applications. However, we will not elaborate on this but concentrate on multi-application setups.

One can think of trivial examples in consensus with the definition. An architecture for such a trivial dialog system is depicted in figure 2-3 (Pieraccini and Huerta [2005], Kellner [2005]). The system in this example only serves the purpose of transforming spoken language into semantic entities and transforming back the results of the processing by the application into audio feedback, e.g. spoken language. An intermediate layer in front of the application contains logic on how to operate on the semantic entities delivered by the dialog system and which results to return.
However, the dialog system in figure 2-3 leaves the strategies on how to interact with the user completely to the dialog logic of the application. We target at a formulation where the dialog system integrates the basic interaction schemes.

Since we want to be able to access a variety of applications – like devices, existing services, or applications developed from scratch – the communication of the system and the application is managed through a *dialog interface layer* (short: *interface layer*) on the application side, figure 2-4. The dialog interface layer represents the need for an application to

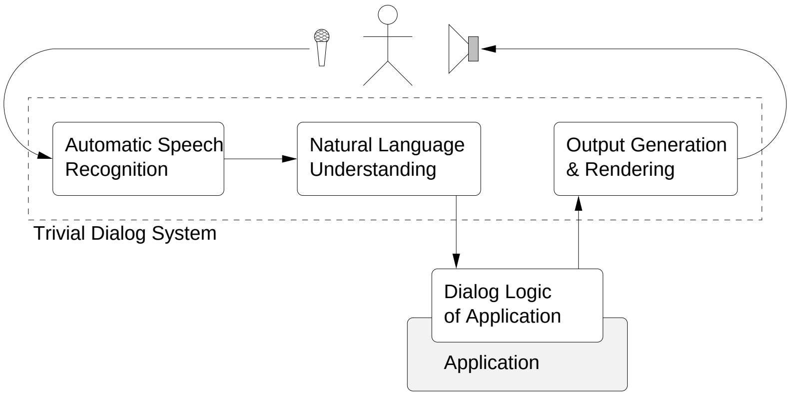

Figure 2-3: Exemplary architecture of a trivial dialog system according to definition 2.2 that separates between the dialog system and an application.

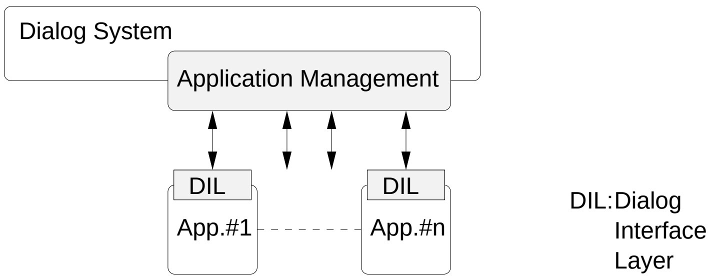

Figure 2-4: Dialog system separated from applications being controlled by the system. The figure explicitly shows the interface layers.

- adhere to a given interface between dialog system and application,
- process the data delivered by the dialog system in order to perform one or more operations, and
- respond to the given request.

Thus, the interface layer is required to enable the operation through the dialog system. Different realizations of the interface layer are possible. It might be a separate piece of software that drives a device or an existing application, i.e. controlling existing entities. It could also be an integral part of the application, especially for applications being explicitly developed for the control through the dialog system.

In the remainder, we implicitly assume that each relevant application possesses such an interface towards the dialog system. To emphasize the existence of the interface layer wherever necessary, the dialog interface layer is highlighted separately as an essential part of an application.

2.3.4 Dependence on Application-Specific Knowledge. A spoken language dialog system cannot completely decouple from applications, even if its applications are separated from the dialog system. Some components have to rely on specialized knowledge.

In the case of a *spoken language* dialog system, we can clearly illustrate why decoupled applications still have to supply specific knowledge sources to the system.

Applications can be implemented before the system is put into operation by the user or after this point in time. In the latter case, the system would have to anticipate all potential applications

that might connect to it. Of course, it is a hard task to imagine all possible applications probably controlled through the voice interface in the future. Furthermore, a complete coverage puts a heavy burden especially on the recognition and analysis components.
Roughly speaking, the performance of ASR degrades with an increasing number of application specific lexicon entries, which further may lead to ambiguities in the NLU. Complete knowledge sources prepared for all eventualities put an unnecessary load on the components, mainly because of superfluous constituents.

As a handy side effect of the separation, we formulate the core components *language independent* and restrict dependencies on the user's language in the recognition/analysis and generation/rendering pairs dealing with input from and output to the user (Xu et al. [2001]). However, this also means that knowledge sources provided by an application used in one of these pairs contains language dependent content, e.g. words in a certain language needed by an application.

2.4. Structure of a Framework for the Separation of Dialog and Applications

A major result of 'the First User Interface Software Tools Workshop' (Pfaff [1985]), which took place in Seeheim (Germany), was the logical model of a user interface management system: the so-called *Seeheim Model* depicted in figure 2-5.

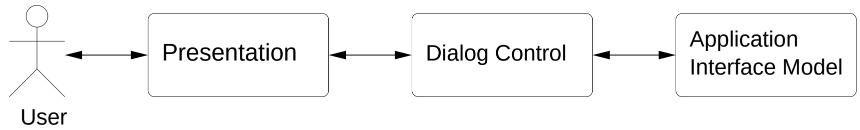

Figure 2-5: The Seeheim Model: a logical model of a user interface management system.

The presentation component is responsible for the generation of output directed towards the user, and it receives the user input. The dialog control analyzes the input and embeds the result into the present state of the dialog control, defined by the previous turns and the environment. The application interface model defines the interface between the user interface management system and the 'rest of the software'.
It is stated that all user interface software must support these components. Yet, it is not predefined if and how the components are separated. If a system consists of several components, it is not clear how these components actually interface with each other.

The general structure of a variety of mono-modal or multi-modal dialog systems is similar to the structure pictured in figure 2-6. The figure presents the basic structure and relationship of basic terms, as used in the context of the dialog framework presented in this thesis. However, there is a major difference in present systems and the architecture sketched in this figure: the application interface and the applications with their interface layers are closely integrated into the dialog engine. Thus, the applications controlled by a current dialog system are not a separate unit but an integral part of the dialog system.

The model underlying many present dialog systems can be traced back to the general Seeheim Model. A much more detailed insight into a certain class of multi-modal systems than the coarse Seeheim Model is provided by the Dagstuhl architecture. At the 'Dagstuhl Seminar on Coordination and Fusion in Multimodal Interaction' (Schloss Dagstuhl, Germany, October 2001), the participants agreed on an abstract architecture for 'mobile, human-centered intelligent

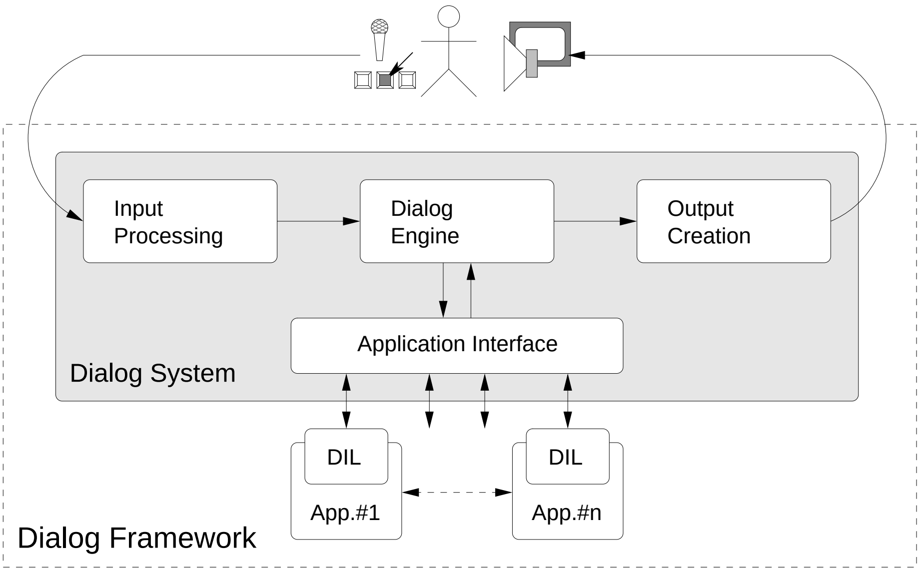

Figure 2-6: Basic building blocks and terminology for a framework separating between dialog system and applications.

multimodal interfaces' as described in Bunt et al. [2005]. It is intended to serve as 'a common reference architecture to consolidate current understanding, facilitate systems description, and to help formulate future research' and refines the architecture presented by Maybury and Wahlster [1998]. The Dagstuhl architecture provides an outlook on the future of multi-modal interfaces, including means for input processing and output creation for various modalities, fusion and fission components, sophisticated interaction management related components, and the interface towards applications. A variety of models accompany the components, among these are the domain model and the application models. Some state of the art systems already partly realize a subset of the building blocks and models, including DymaLog, cf. the overview in figure 6-1. An advanced system in conformance with major parts of the Dagstuhl architecture is given by the SmartKom system (Wahlster [2006], The SmartKom Consortium [1999–2003]).

The DymaLog framework presented in chapter II divides the interaction of the dialog system and applications into two units as indicated in figure 2-6. The application interface manages the application handling from the dialog system's side, e.g. transforming the knowledge sources from the applications for use inside the dialog system and organizing the interaction with the application addressed by the user. The second element in the interaction is the dialog interface layer on the application side. As stated before, the interface layer ensures that the application adheres to the required interface towards the dialog system. In addition, the interface layer comprises purely application-specific logic to deal with the input, e.g. in a train timetable application the logic should include that a departure station, arrival station, and the departure (or arrival) point in time need to be specified to request detailed information on connections between these stations.

State of the art dialog systems use a variety of architectures (Kellner [2005]). The design tool 'Speechbuilder' (Glass and Weinstein [2001]) is based on a central hub architecture and allows for easy implementation of dialog systems in strictly restricted domains. The 'WITAS' system is also hub based. 'WITAS' targets at the asynchronous and multi-modal control of an autonomous robot (Lemon et al. [2001]). The general open agent architecture (OAA) forms the basis of 'WITAS'. Other dialog systems also use OAA, like Cheng et al. [2000]. The basis of 'WITAS' is e.g. also utilized in Clark et al. [2002]. Another hub based system is the 'CU Communicator'

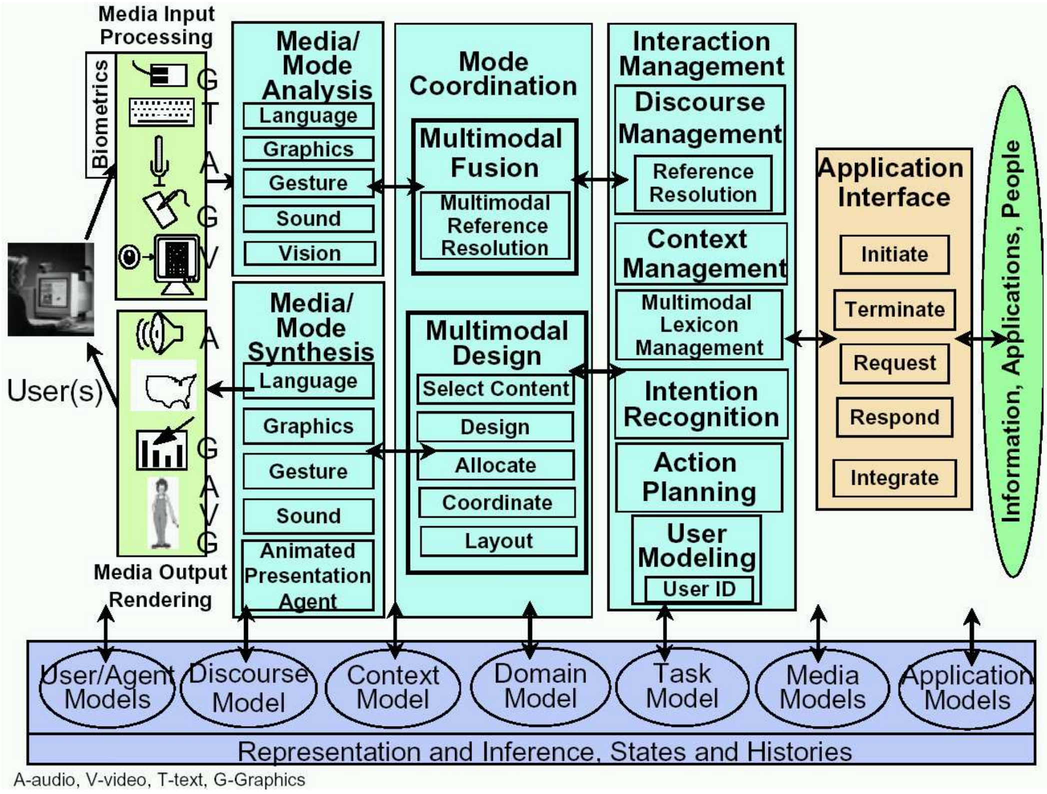

Figure 2-7: Refinement of the architecture for multi-modal dialog systems introduced by Maybury and Wahlster [1998], as adapted during the Dagstuhl seminar. The architecture shown here was published in Bunt et al. [2005].

(Pellom et al. [2000])), an implementation for the 'DARPA Communicator' task. Bagga et al. [2000] shows a pipeline architecture to access a database through a dialog system. SmartKom (Wahlster et al. [2001]) is an example of a system based on a multi-blackboard architecture to combine a variety of modules. The Trindi project uses a set of centrally organized rules to organize the interplay of the single components (Larsson et al. [2000]). In Dahlbäck et al. [1999] the 'dialog manager' operates as central coordinating instance.
All these systems basically share the same (natural) order of processing steps as shown in figure 2-6, yet the applications are much closer integrated into the dialog system than shown in that figure.
Since many of the groups working in the field of spoken language dialog system base their demonstrators on propriety architectures, a complete overview cannot be given here. We refer to a more complete overview, e.g. given in McTear [2002].

3. Generic Knowledge Representation and Processing

Decoupling can be used to realize a voice interface for dynamic application setups. According to definition 2.2, knowledge sources are provided by the applications, but the dialog system does not include procedures specific to some applications on how to process data internally.

This leads us to *generic* entities inside the dialog framework. The prefix '*generic dialog*' will refer to constituents inside a dialog system that are available without relying on one of the

applications. It comprises technical necessities like the technical means to exchange data between components or more discourse related functionality, see section 3.3 below.
We approach the challenge of the separation by formulating processing methods based upon a generic data representation independent of the connected applications.

3.1. Distributed Processing Logic

Collecting semantic entities to operate an application requires awareness of the dependency between operations, which can be performed, and the semantic entities needed for the execution. In case of underspecified input – the user did not provide sufficient information to perform an operation – the need for additional input arises. One could think of alternative ways the user can complete her request. Thus the applications require logic to connect the processing result of the user input handled by the dialog system and the operations belonging to these applications. The *application logic* contains the rules to process the input from the dialog system to perform an operation, update the input for the application's response, and/or generate an appropriate result as outcome of the operation.
Input from the user can be regarded as an event for the dialog system. Further, an application may initiate events that trigger the system. The adequate steps to deal with such events are part of the dialog system.
This *dialog logic* includes means to drive the applications[5]. Further pieces of the dialog logic deal with e.g. the extraction of semantic entities from the user input, the integration of present user input with the previous discourse, and the generation of output from the system internal processing and processing by the application.

In order to achieve the division in sense of definition 2.2, the application logic cannot be an integral part of the dialog logic. If this were the case, we would again have to know and integrate the logic of all applications worth considering for controlling into the dialog logic. Consequently, the logic belonging to an application has to be bundled with the applications.
However, this does not predetermine where the application logic has to be located in the separated case. The two extreme cases are:

1. *Application centric.* The dialog system does not incorporate logic from the applications. The applications take over the task of relating operations and knowledge. We will call this approach *application-blind*, see section 8.

2. *Dialog system centric.* The logic specific to an application is handed over to the dialog system, which is 'reprogrammed' to support this application. The application itself focuses on the actual execution of operations. Thus, the application logic is part of the knowledge sources provided to the dialog system.

The second approach requires some definition language to formulate the application logic in such a way that the dialog system can interpret and integrate it. I.e. a scripting or programming language to allow the exchange of the logic would be needed.
In the first case, the application logic remains at the applications, which seems to be more appealing since the interface is less complex. Yet, the conversation of a user and the dialog system is about the controlled applications. The parameters required to operate an application must therefore be accessible inside the dialog system as soon as an application connects; of course, the dialog system centric case also requires the availability of the application parameters.
The *application operation parameters* (short: *operation parameters*) describe parameters of an

[5]But the dialog logic might not be responsible for actually operating the functionality of an application.

application that can be accessed by the user in order to operate the application. An example for such parameters is the channel of a TV device. Or, the departure station and departure point in time together with the arrival station and arrival point in time of a train timetable application. The parameters of an application could be simply organized as a set of 'slots' (also known as 'feature-value pairs') as done in many present dialog systems (Chu-Carroll [2000]). One could also think of relating single parameters using hierarchical structures ('trees'). A dialog system may also apply other organizations of the parameters.

We will favor the clear partitioning of logic to bundle with the dialog system and applications respectively (case 1). This avoids the need for a detailed description of the applications' capabilities inside the dialog system. For figure 2-4, this means that the application logic is part of the dialog interface layer: the logic required to process events, like input by the user, is distributed over system and applications.

This approach requires an appropriate data representation inside the non-application components (section 3.2). Furthermore, it allows the realization of generic dialog functionality (section 3.3).

3.2. Knowledge Representation

We can formulate four major requirements on a generic representation of knowledge:

Easy adaptation to (new) controlled applications. A major motivation for the separation is the easy adaptation towards new application setups. As an essential part, the internal knowledge representation has to adapt towards this requirement.

Relational structured representation of the application operation parameters. Building up relation structures between atomic semantic entities allows us to formulate a measure on the proximity of these entities which e.g. can be used to evaluate the result of the input processing or calculate the next system reaction. It further allows reusing substructures at different places, which is especially useful for some common structures like time, place, or person.

Handling of multi-application setups. Once an interface, which is to be used in dynamic application setups, establishes the separation, the system should be able to make a number of active applications available at the same time. Then the user has the flexibility to switch between the applications she currently controls during the interaction. I.e. we do not want to restrict the user to reconfigure the dialog system in a separate setup step in order to control another application.

Sharing discourse information between applications. Applications connected simultaneously to the dialog system can benefit from the mutual use of knowledge acquired during the interaction, i.e. from carrying over knowledge from one application to another. E.g. assume that a user looks up the broadcasting time for a movie in an EPG. When she decides to record this movie, the required information should be handed over to a recording device in order to create an appropriate timer entry.

The framework will be based on hierarchical structures to represent the application operation parameters with their relations from the various applications. It incorporates mechanisms to reuse existing structures, handle multiple applications, and allow for dynamic application setups. As we will see, the structures are derived from ontological descriptions. Macherey and Ney [2002, 2005] describe the structure of a dialog system using a tree, the nodes defining the parameters of applications. From the root node to the leaf nodes, the application descriptions are getting more and more specific. Paths inside the tree reflect the state of the discourse and determine the next action to take.

3.3. Generic Dialog Functionality

As already shortly hinted above, separating the logic of dialog system and applications together with an application-independent representation inside the dialog system enables us to introduce generic dialog functionality. Remember the term *generic* that we introduced before:

3.1 Definition (Generic Dialog Functionality)**.** Let **d** be a dialog system which is separated from applications. *Generic dialog functionality*, or short *generic functionality*, comprises all functionality offered by **d** that can be addressed by a user during the discourse independently of the applications. ◇

Basically, we can distinguish between two types of generic dialog functionalities. It can be used

1. to control means supplied by the dialog system itself, or

2. to provide building blocks utilizable by the applications.

We will now supply showcases of basic generic functionality to take a closer look onto both types. The functionality supports both, the interaction of user and system as well as the development of the dialog interaction layer to enable applications for the control through the dialog interface.

3.3.1 Central Comprehensive Discourse State. Again, we can distinguish between the system itself and the applications: both may maintain their own state at some point in time. However, external events, being honored in a certain way, influence the state as outlined next. Consider e.g. the case of a recording device like a DVD-recorder (continue the 'living room' example). When programming a recording, the date, the start time, the length, and the channel completely determine a broadcast. No matter how this information was supplied to the recording device, as long as the required information is available, it records the specified broadcast. The user may use a dialog interface or an interface shown on a screen together with the remote control to enter the data.

The state of the dialog system is based on input by the user and the applications[6]. If we observe human-machine interaction, the input by humans is currently serialized, i.e. one semantic entity (or a small number of semantic entities) is provided one after the other to the machine[7]. Nontrivial operations often require more than one or two semantic entities. This leads to the task of integrating the knowledge that is collected during the interaction of a human and a machine. Consequently, a consistent snapshot of the current knowledge state is needed. Of course, we could leave the task of integrating knowledge collected over several turns to the applications. This however would require the separate integration for each application. Realizing the integration of knowledge at a central place inside the dialog system prevents to 'reinvent the

[6] The 'environment', e.g. the current time or place, might also influence the state of dialog system or applications. We will focus on events that originally descend from a user or an application. E.g. you might have scheduled a phone conference and five minutes before (time event, environment) the conference begins the application requests the dialog system (application event) to ask if the phone conference should be set up.

[7] In a face-to-face meeting, human-human communication is not limited to a sequence of words. Other modalities like prosody, which is a part of speech, and facial expressions can enhance the communication, thus broadening the communication channel.

wheel' for each application. Furthermore, it enables the dialog system to operate on knowledge from the discourse and make generic functionality available. Holding a central discourse state permits us further to communicate the complete snapshot of the discourse that is relevant for the current operation towards the applications.

Information collected during a discourse may originate from a user, some application, or the environment. A central place that integrates information collected during the discourse, which is able to derive a consistent view on the current knowledge state of the dialog system, strongly supports the separation from the applications. In principle, applications can then concentrate on the internal state for their operations and do not have to care about the discourse between user and system (Lin et al. [1999]).

The single application spoken dialog system Higgins described in Edlund et al. [2004] is targeted at research on error handling for dialog systems on several levels. It focuses on pedestrian navigation and guiding. Similar to our approach, the domain knowledge is represented in tree structures, however these structures also explicitly represent the related communicative act for each utterance, and the goal of the interaction in appropriate sub-structures (Skantze [2005]). In order to merge the present user input with the previous discourse, the system e.g. resolves ellipses on the basis of 'domain dependent context rules'. The approach adapts concepts presented by Carbonell [1983], who defines a collection of rules for the merger of user input and discourse. The applicable rule needs to be identified, mainly on the basis of certain premises and relations of the user input and the discourse elements in focus, and is then applied on the current user input and the previous discourse.

3.3.2 Reuse of Structures. Consider the example depicted in figure 3-1. It illustrates relational structures of an entry in a train timetable as it could be modeled to cover the departure and arrival of train connections. The reuse of structures is possible in three places: departure and arrival share the same substructures ***`station`***, ***`date`***, and ***`time`***.

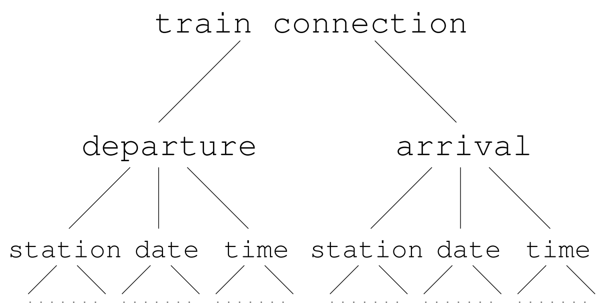

Figure 3-1: Illustration of relational structures modeling the departure and arrival for train connections in a train timetable (substructures are not further detailed). The lines indicate parent-child relations, the children are components of the parent (the relation depicted here corresponds to the 'has-a' relation introduced in the context of ontologies in section 19.1).

More general, reusing existing structures facilitates the development of applications. As we will see later, during the realization of the dialog interface layer one can benefit from the reuse of structures. Especially the frequently used structures representing a date or time, a place, or a person are very useful. The example showed the reuse of the substructures in one application, yet the reuse is not restricted within a single application.

3.3.3 Sharing Discourse Information between Applications. Another opportunity for reuse is sharing of discourse information between applications. Information collected during the interaction of a user and a system to operate application a_1 can potentially be of interest for a second application a_2.
To illustrate this, assume that a user has already selected a train connection from her home place to an airport in a first interaction sequence using the voice interface. The left side of figure 3-2 illustrates such a connection for a train. Assume further that the input *"I want to take a flight*

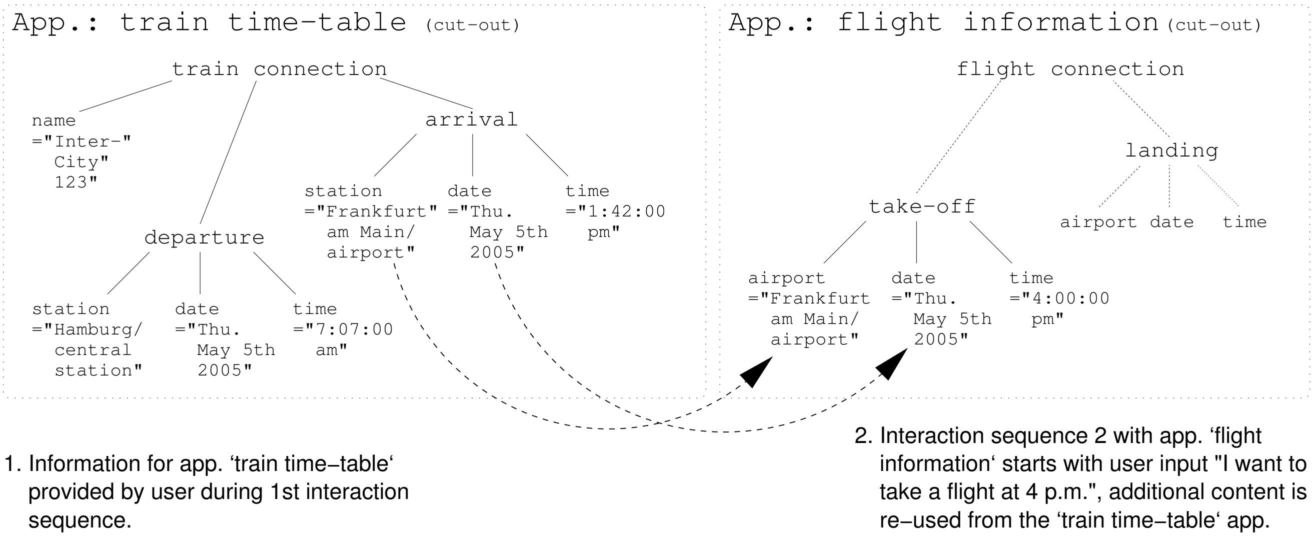

Figure 3-2: Illustration of relational structures for an example of the reuse of information between two applications (substructures for places, date, and time are collapsed for readability).

at 4 p.m." initiates the second interaction sequence to select a flight. Enriching this input with parts of the first interaction sequence provides information beyond the user input for the 'flight information' as illustrated on the right side of figure 3-2.

The procedures to share discourse information between different applications are part of the generic dialog functionalities in our framework. The mechanisms for sharing discourse information are based on the reuse of structures introduced in section 3.3.2.
Of course, the applications could directly negotiate such a recycling without involving the dialog system. However, then the applications would have to know in advance which applications might exchange knowledge. Otherwise, a separate mechanism must be defined to allow this exchange. More advanced cooperation between applications and their knowledge respectively might e.g. be modeled as collaboration of agents based on the knowledge query and manipulation language (KQML), Finin et al. [1994].

3.3.4 Navigation in the Discourse. Consider a dialog system with a central discourse state, section 3.3.1[8]. Memorizing the temporal development of the discourse in the discourse state facilitates navigation in the discourse history.

Analyzing and interpreting the user input could currently not guarantee that the result

[8]Remember that a central discourse state does not prevent the applications from having their own state.

of the input processing is a correct representation of the user's intention[9]. In such a case, the system and the applications process information that does not reflect the intent. Navigating in the course of the interaction enables the user to discard the last input: restoring the state before the last input cancels out this input and resets the voice interface to the previous conditions, thus supporting an easy repair of misrecognitions and misinterpretations.
Enhancing the dialog state will also allow to realize forward and backward navigation in the discourse comparable to navigating in a web browser.
The recovery of previous discourse states (canceling the last command, backward navigation) and the reexecution of input (forward navigation) allow the user to easily correct wrong input as demanded in Lamel et al. [2000] – either because the user changes her mind or due to faults during the input processing.

In contrast to the utilization of navigation for the correction of errors, spoken language dialog systems could integrate repair strategies. I.e. either the user notices problems in the interaction and tries to correct these problems, or the system itself detects errors and interacts with the user to overcome problems that might arise from these errors (Hirschberg et al. [2001], Kirchhoff [2001]).

3.3.5 Dynamic Adaptation of Knowledge Sources. The dynamic adaptation of the knowledge sources comprises two major reasons:

1. changes in application setup, i.e. applications are connected or disconnected, and
2. updates triggered through changes in the status of one or more applications, e.g. activation or deactivation of a connected application.

After adding or removing applications from the set of connected applications, the voice interface needs to adapt its capabilities to reflect these changes, see section 2.
Problems inside the input processing due to large knowledge sources have already been addressed in section 2.2. Allowing applications to dynamically update their published knowledge sources depending on the internal state can help to limit the sizes of the knowledge sources.

The processes to update the knowledge sources applied inside the dialog system are directly dependent on changes in the application setup and the states inside the applications. Thus, the dialog system has to provide appropriate interfaces towards the applications.

3.3.6 Analysis and References to Lists. Lists are a common means to represent a series of entities. On a screen, lists are often presented as tables, like a train timetable, an electronic programming guide, or the inventory of a warehouse. In spoken language, the output of lists is restricted to sequentially reading, which is only acceptable for short collections. Yet, lists are often a good way to present alternatives to the user for selecting items. Two obvious possibilities of accessing an item in a list by spoken language are:

1. directly addressing an item via its content, and
2. indirectly addressing an item via its (relative) position in the list.

To elaborate on that, consider the typical EPG in table 3-1. Addressing of an item via its content can be done in various ways, e.g. using the title, the time, the channel, the genre, or combinations of these. Accessing a table via its content heavily depends on the structure and – of course –

[9] Even between human beings, misunderstandings are quite common.

	title	channel	start	length	genre
1.	News of the Day	CNN	8:00 p.m.	0:30	news
2.	Fox News	Fox	8:00 p.m.	0:15	news
3.	NBC at Eight	NBC	8:00 p.m.	0:15	news
⋮	…				
8.	James Bond 007 – The World is Not Enough	Fox	8:15 p.m.	1:50	movie
9.	Apollo 13	ABC	8:15 p.m.	2:30	movie

Table 3-1: Excerpt from an EPG (fictive).

the content itself. In the example, allowing the user to address an item by the title may lead to numerous alternative approaches by the user, e.g. *"give me details on The world is not enough"* or *"show info on James Bond"*.

A much more general (with respect to adaptation towards applications) and content independent approach is the usage of positions of items in the list. It does not require knowledge on the structure of the list nor the content of items. A prerequisite to address items in a list by their positions is the identification of list structures. Ideally, the list structures are automatically derived from the representation of results from the applications. Examples for numbered addressing are *"the eighth item"* or *"item number eight"*, other absolute positions are *"the top element"* or *"the last element"*.
A stronger relation of the positional referencing of list elements and the applications can be achieved by replacing generic formulations like *"item"*, *"element"*, and *"entry"* with more specific terms. In case of table 3-1, such terms might be *"broadcast"* or *"show"*.
If the visual modality is available together with touch input, selecting list items by touching the related screen area can be compared with the positional referencing using speech.

3.3.7 Control of the Viewport of the Visual Output Modality. Content represented outside the visible area of the screen, e.g. when long lists or large maps are shown, should be accessible by the user. Moving the viewport of the screen can be realized independent of the applications and can therefore be provided as generic functionality by the dialog system.

Application dependent movements of the viewport can also be realized, but will not be addressed in this text. E.g. a part of a map showing the *west*-coast of the US is displayed on the screen, and the user requests *"show the New York area"* instead of moving n screens to the right.

3.3.8 Relaxation. The comprehensive discourse state, section 3.3.1, provides a consistent view on the discourse as seen by the dialog system. Depending on the selection strategies of competing hypotheses, see section 11, the dialog state might favor the integration of the processing result of the current input and the previous discourse. The user could decide that she wants to dismiss some of the earlier introduced restrictions. E.g. consider for an electronic programming guide the sequence:

USER#1 (U1): *"what is on CNN"*
SYSTEM#1 (S1): *≪shows the program of channel CNN, assuming that the user wants to know what is on CNN starting now≫*
U2: *"what is on TV at eight p.m."*

S2: *≪shows program of channel CNN, starting at 8 p.m.≫*

U3: *"show all channels"*

(the user wants the program of all available channels beginning 8 p.m. or later)

The dialog system relies on the applications to address operation parameters that should be explicitly relaxed.

Automatic relaxation strongly depends on the applications and will therefore not be part of the generic dialog functionalities. E.g., the electronic programming guide cannot find any shows that match the restrictions posed by the user input *"which movie starts at eight p.m. on CNN"* and decides to look for movies that might start on channel CNN *after* 8 p.m.. Yet, general strategies for automatic relaxation may be provided elsewhere such that they can be used during the implementation of the applications.

Bühler and Minker [2005a] present a logic-based reasoning approach based on a problem assistant to derive a consistently satisfiable and unique model, i.e. the mutually accepted constraints are not over-constrained or under-constrained. The dialog manager aims at relaxation or restriction instructions to achieve such a state.

3.4. Multiple Hypotheses

Stochastic models are heavily used in processing input from the user, e.g. hidden Markov models are widespread in ASR. The quality of competitive hypotheses representing the input is evaluated on basis of such models. Restricting the outcome of the input processing to one hypothesis selects the hypothesis with the best rating according to some underlying measure. Yet, after putting the hypotheses in the context of additional knowledge, e.g. the previous discourse, the selection mechanism might prefer another hypothesis that was discarded before.

If, for example, the question *"where do you want to go"* is answered by *"to Düsseldorf airport"* by the user, the best hypothesis obtained from ASR and NLU may contain the representation of the airport 'Düsseldorf' as a flight destination. Assume that the user was previously busy looking for a train connection with departure Aachen, and she did not define the destination so far. Then a hypothesis that contains the *train station* of the 'Düsseldorf' airport as the destination of a train travel seems to be more valid. Thus, in this case the context would have influenced the decision for the best hypothesis.

Maintaining alternatives demands more computing power to process the additional hypotheses and needs a selection mechanism that honors the additional knowledge integrated into the hypotheses. The integration itself may also lead to additional hypotheses. The components of a system that have to deal with competing hypotheses must also take care to present the same state towards each hypothesis of the same turn. Thus, the components must not update their states before one of these hypotheses is selected. Otherwise, one hypothesis might influence the processing of a competing hypothesis and therefore falsify the result for this hypothesis.

Figure 3-3 illustrates the usage of multiple hypotheses by continuing the example of figure 3-2. The input processing delivers a number of alternative interpretations of the user input due to uncertainties (e.g. stochastic models in ASR) or ambiguities (e.g. ambiguous rules in NLU) in the underlying models and knowledge sources. Knowledge from the previous discourse enriches these interpretations. This enrichment may be ambiguous, which again increases the number of alternative views on the user input. The hypothesis selection has to evaluate the different interpretations to choose the best one according to an underlying measure. This measure might be based on features inherent in the hypothesis and/or ratings from previous processing stages.

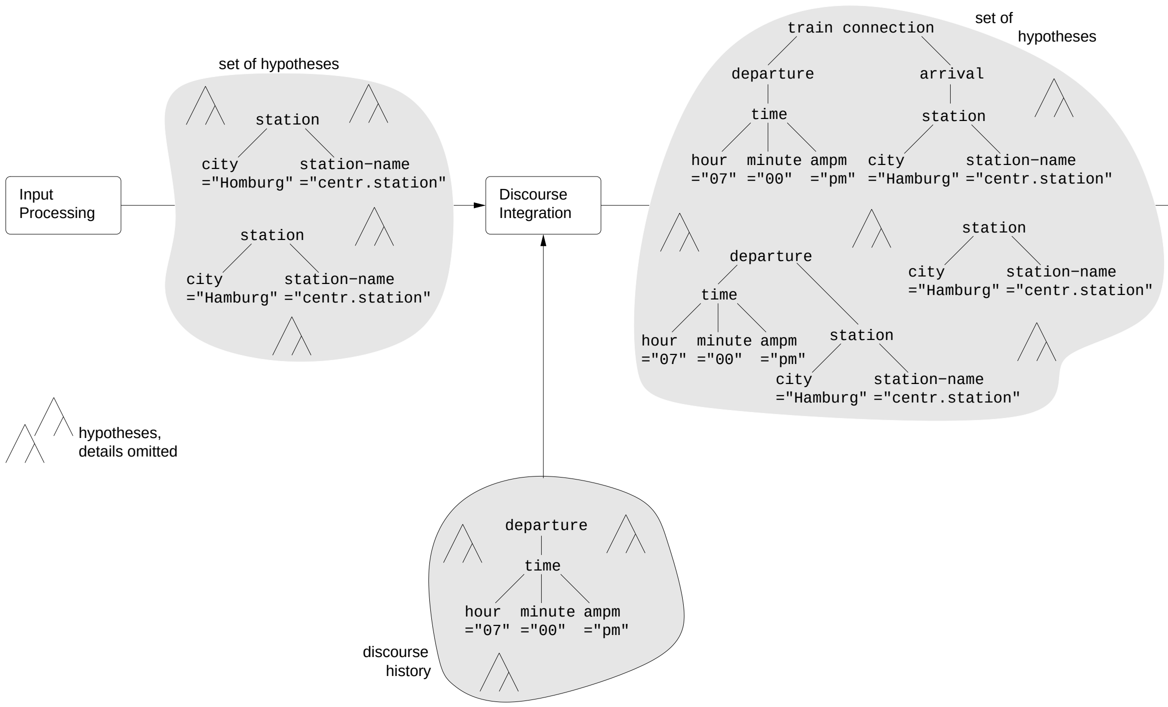

Figure 3-3: Illustration of the processing of multiple hypotheses for the interaction sequence (i) *"I want to leave at seven p.m."*, system reaction *"please specify a station"*, and (ii) *"Hamburg, central station"*, see also figure 3-2. The discourse integration combines the hypotheses with the discourse history. For the structure carrying a ***station*** with ***city***="Hamburg", three outcomes of the discourse integration are shown: one result shows that ***station*** may not be integrated with the previous discourse at all, two results integrated ***station*** with the ***departure*** from the previous discourse.

As a side effect, extending the usage of alternative hypotheses beyond the input processing enables us to change the behavior of the system centrally by modifying the selection criteria.

4. Criteria for Flexible Dialog Processing in Dynamic Application Setups

The previous sections presented characteristics for a dialog framework that separates between a dialog system and applications by application-independent processing inside the dialog engine. In this section, we utilize these characteristics to formulate a catalog of requirements. The criteria contained in this catalog formulate the requirement specification for the DymaLog framework introduced in chapter II.
The criteria will be referenced as ‘criterion <section>’ or ‘criterion <section>(<item number>)’, e.g. criterion 4.1(1) for the first criterion in section 4.1.

4.1. Application-Independent Formulation of the Dialog Core

The application-independent formulation of the dialog core prepares the dialog system for dynamic application setups. Most important aspects for the application-independent formulation of the dialog core are:

1. *Application-independent knowledge integration into a comprehensive discourse state.*
   The analyzed user input is brought into the context of the previous discourse of the interaction between user and dialog system. The integration process does not depend on the applications. (See also sections 4.2.1 and 4.2.2.)

2. *Structured representation of the application operation parameters, extensible during run time.*
   A structured representation of operation parameters may support the application-independent knowledge integration postulated in the previous claim. Changing the application setup has to be reflected by these representations.

3. *Separation of dialog logic and application logic.*
   In order to exchange the set of controlled applications during run time, the logic related to these applications must be disjoint from each other, and in addition from the logic inside the dialog system.

4. *Automatic sharing of discourse information between applications.*
   Applications controlled through a dialog system may for logical reasons share discourse information acquired during the discourse, thus avoiding respecifications of information already available inside the system.

5. *Allow applications to connect and disconnect during run time.*
   Formulating the applications as dynamic components of the dialog framework requires mechanisms to detect and/or announce changes in the set of controlled applications.

6. *Role of the interface between dialog system and applications for decoupling.*
   Since the applications form entities separated from the dialog system inside the dialog framework, the interface is of major importance. It should be as 'minimal' as possible to keep the interface simple. However, the interface must allow to fully enabling the operation of an application that is connected to the dialog system. At least the application operation parameters (and their relations) derived from the user input should be transferred to the applications.

4.2. Generic Dialog Functionality Supporting Human-Computer Interaction

In section 4.1, we summarized the criteria for an application-independent dialog core. One step further, such a dialog core allows to formulate (semi-)generic functionalities to support the interaction of a user and the machine with some applications.

4.2.1 Dialog Phenomena. The evaluation of the current user input in the context of the previous discourse and the access to the application response is crucial for non-trivial interactions of a human with a dialog system.

1. *Resolution of ellipsis.*
   For non-trivial interactions, the user input needs to be enriched with context information from the previous discourse. The resolution of ellipsis integrates meaningful content from the previous discourse with the present user input.

2. *Resolution of anaphoric references.* To perform actions on the result of the last application operation, demonstratives and definite noun phrases can be used to refer to the most recent output, e.g. *"remove* this *recording"* or *"remove* the *recording"*. Pronoun resolution is presently not available, it would require some sort of salience analysis[10].

3. *Analysis of lists and references to their elements.*
   Lists are widely used to present information to a user, especially by displaying them on a screen. To access items in a list, the automatic analysis of the content to be presented together with means to access the elements of a list are needed.

4. *Serial multi-tasking with task resumption.*
   During the interaction of a user with a certain application, a second application may be accessed in-between. To illustrate this, assume that a user browses the EPG while listening to music in the background. The user may decide to reduce the volume of the music and then continue to browse the TV program. The context of the ongoing interaction with the EPG should be maintained after the interaction with the music player; the interaction of the user with the EPG is interrupted by an interaction with the music player and later resumed again. I.e. the application in the dialog focus shortly switches and the focus comes back to the previous application.

4.2.2 Meta-Communication in the Interaction. Mostly, the user intents to operate an application by her input. However, from time to time also functionality of the dialog system itself needs to be addressed. In the context of a dialog system for dynamic application setups, the following meta-communication phenomena are of special interest:

1. *Navigation in the discourse.*
   After the system dealt with the last user input, the user may decide that the effect did not match her expectations or discover that the application actually accessed by the system did not match the application originally wanted by the user. This might e.g. be due to failures during the analysis or changes in her mind. Means for navigation allow for easy corrections.

2. *Control of output modalities.*
   During the interaction, the need to change the properties of output modalities could arise. E.g. adjusting the volume might become necessary to successfully proceed with the interaction. As an example, the framework presented in chapter II realizes the control of the viewport of the visual modality.

3. *Activating and deactivating applications.*
   Presently, the performance of components like ASR and NLU in general worsens with an

[10]Assume that a user presently defines a recording in a DVD-recorder application and the last item specified by the user is a time. The utterance *"remove* it*"* may refer to the *time* defined in the last utterance. However, it could also refer to the *recording* item. Such a decision requires an evaluation of the salience of the different candidates. Such an analysis could e.g. be based on the 'age' of the objects involved and/or the grounding state of these objects.

increasing complexity of the underlying knowledge sources applied in these components. To reduce this effect, complex applications may need to be explicitly activated to access the complete scope of functionality. The dialog system can provide means to (de-)activate applications.

4.2.3 Building Blocks that Support the Design of Applications. To ease the design of applications for use in a dialog system, components to plug into an application can be used. Prominent examples are modules to deal with time, date, place, or person. A building block for one of these entities may include the representation of the related application operation parameters, means for recognition and analysis, means to generate appropriate output, or even algorithms operating on such an entity.
Also referencing list elements as listed before can be regarded as a building block. E.g. generic formulations to reference a list item (like *"the third item"*) may be enhanced and adapted by an application (like *"the third movie"*), yet resolving the reference is still done by the dialog system and not by the application. Similar, means to trigger a certain state of an application can be made accessible by all applications, e.g. 'entering' and 'leaving' an application.

We will concentrate on the generic functionalities presented in the last three subsections. However, additional dialog system inherent functionalities may be identified and added in the future.

4.3. Parallel Multi-Hypotheses Processing

Human statements, especially in spoken language, often permit different interpretations, depending on the situation, discourse history, or perhaps the present participants on a conversation. For human-machine interaction, the limited capabilities, e.g. in perception and context knowledge, together with methods that allow for non-unique solutions generally lead to various interpretations of the same input. Allowing one single voice interface to address multiple applications even intensifies this diversity.
To equip a dialog system with capabilities to handle competing interpretations of the same input, it must be able to perform *multi-hypotheses processing*. I.e. alternative interpretations must be dealt with at various stages. In addition, whenever ambiguities or uncertainties arise new interpretations may come up. At a certain stage however, a decision must be taken which of the competing hypotheses best represents the user input in the current context of previous discourse and situation in the end. The multi-hypotheses processing allows non-synchronous handling of alternative hypotheses and has to face challenges similar to the ones described in Blaylock et al. [2002].

4.4. Applying the Criteria

The criteria summarized in this section served as basis for the development of the framework presented in chapter II. Thus they also serve as requirement list for a dialog system realized on top of this framework, chapter III.

The application-independence of the dialog core will be the basis and starting point for the generic dialog functionalities. The multi-hypotheses processing is not a direct requirement for a dialog system that should handle dynamic application setups. Yet it allows for a better separation of single steps into autonomous units, each unit handles different contexts and is free to generate different results for the same input. Another unit can again put these results into its particular context.

The system realizing DymaLog can be positioned between frame-based and agent based systems according to the classification stated in Block et al. [2004]. Knowledge is represented in hierarchically organized 'frames'. Autonomous entities may contribute to the discourse, e.g. the generic functionalities defined in the framework. Applications can utilize building blocks to ease their design, and the applications are actively involved in the discourse as they influence the response and further strategy of the system in the interaction with a user.

5. Key Questions

An alternative view on the characteristics of a dialog framework, as described in the preceding part of the chapter, – compared to the criteria catalog in section 4 – is given by a collection of key questions. These questions reflect open issues in present research that need to be addressed in the remainder of this text, in order to enable a voice user interface in dynamic multi-application environments.
As we will see next, the select questions target different aspects. Yet, for all of these, the relevant context is given by multi-application spoken language dialog systems.

Recent dialog systems, research systems as well as commercially available systems, show strong connections of the dialog engine with the applications. Reusable components are already widespread in such systems. These components are adapted for use in different applications when an existing system is ported to a new application.
As a fundamental objective, DymaLog aims at outsourcing the applications, and therefore creating a system that, once realized, can be applied to different applications:

> *Q1:* Which methods allow the reduction, or even elimination, of the close interdependency between a dialog engine and its hosted applications?

One step further, we will not restrict DymaLog to a single application, which can be controlled via the dialog system. Instead, the abstraction from the applications should enable a user to interact with various applications through the dialog system. Thus, the dialog system is a gateway between the user and a variety of applications.

Predefined units start to ease the development of dialog systems. For instance, a 'subdialog' that requests a confirmation from a user can be used in different stages of a dialog with an application, and might be reused in different applications. The separation of the dialog engine from the applications even widens the possibilities to provide capabilities detached from a certain application:

> *Q2:* Can we generalize interaction concepts required for a variety of applications or related to meta-communication?

The abstraction of capabilities required in the interaction of a user and a dialog system can prevent the reinvention of the wheel, and thus eases the realization of new applications. Furthermore, it ensures that the user perceives a consistent behaviour of the interface. Otherwise, if such functionalities would be implemented for each application separately, each single application designer has to ensure that her realization respects some given guidelines.

As an extension of a dialog system that decouples the application from the dialog system, such a system can aim at supplying an interface to a dynamic set of applications. The dynamics relates to attaching and detaching an application for control through the dialog system. The extension into the direction of a dynamic multi-application system leads to the question:

> *Q3:* On which principles can multi-application dialog systems base the handling of dynamic changes in the set of applications?

Problems, which naturally occur for such a 'plug-and-play' multi-application system, include the assignment of elements from the user input to a certain application, handling of changes in the required knowledge sources, and the flexible processing of new content in the context of the previous discourse. As we will see later, applications can benefit from each other when information can be transported from the interaction of a user with an application to the interaction of this user with another application (see section 28 for an example). The question arises, how it can be assured that a newly added application may also profit from the sharing of knowledge between applications.

Ambiguity and uncertainty are generally inherent in the interpretation process of the user input by a dialog system. Different contexts can change the meaning of an utterance, or the models underlying a processing step may not be able to derive a unique result. This leads to the question:

> *Q4:* How does a dialog system determine the best interpretation hypothesis for the user input, given the uncertainty and ambiguity in the underlying models and knowledge sources?

The modularization of a dialog system even complicates the selection of the 'best' interpretation. E.g., a processing step early in the processing chain may throw away an interpretation too early. I.e. the actually forwarded interpretations are 'worse' than the discarded interpretation in one or more subsequent processing steps later in the chain.

Spoken language as an input modality already reached the commercial world some years ago, most commonly in form of voice control applications (like name dialing in mobile phones), telephony applications (like directory assistance), or dictation products (e.g. applications in law and medicine). In the light of convergency of research and industry – in order to utilize the advanced concepts from research in robust real world applications – sophisticated approaches in various areas from research are expected to help realizing advanced interfaces, being usable by anyone. The concepts presented in this thesis concern a variety of aspects in spoken language dialog systems. Yet, a priori it is not clear if the various approaches can be combined into a single system.

> *Q5:* How can the proposed approaches be combined into a multi-application dialog platform?

The combination of the various concepts into a single framework would ensure the compatibility of and interplay between these concepts.
One step further, a prototype enables us to validate, and iteratively improve, the concepts in DymaLog. Moreover, a prototype can be used to generate the corpora required for some data-driven approaches, potentially in an iterative process. For instance, such data is needed to estimate parameters of the selection algorithm in the *Marvin* dialog system.

Chapter II.

Dialog Framework for Application-Independent Dialog Processing

In this chapter, we introduce a flexible spoken language dialog framework with respect to the criteria summarized in section 4 that enables the interaction of a user with one or more applications. The framework permits changes in the application setup during run time. A clearly defined interface between the dialog engine and the applications together with minimal dependencies of the dialog engine on application specific knowledge provide the required flexibility. The framework further serves as the basis of a number of generic functionalities in the dialog engine.

The framework makes use of the TRINDI approach to dialog modeling. It is formulated around an information state, which holds the state of the discourse between user and machine as perceived by the system. Sets of rules interact with the information state to update the information state and decide on an appropriate reaction with respect to the user input. While ideas from TRINDI had major influence of the approach to formulate an application-independent dialog core, the SMARTKOM project also heavily influenced the partitioning of the framework into functional units. Section 6 outlines the anthology and relates the framework presented in this text with TRINDI. Some of the ideas applied in this framework already appeared in the architecture described in LuperFoy et al. [1998]: the discourse processing is already separated into dialog management, context tracking, and pragmatic adaptation. Moreover, the architecture uses blackboards for the communication between the modules.

The treatment of alternative interpretations of input by the user is addressed in the sections 7, 11, and 12. It includes an overview on the sources of competing hypotheses (section 7) and the criterion for the evaluation of the quality of the various interpretations in order to prepare the selection of the best hypothesis according to the underlying quality measure (section 11). The parametric nature of the selection measure needs methods to estimate these parameters (section 12).
Allowing alternative interpretations of the user input must be paid attention to by all effected components in the dialog framework.

The dependence of the hypothesis selection process on the parameterization of the underlying measure is considered in section 13. We use a parameter study to investigate the influence of single ratings on the overall measure. It is shown that the parameterization of the selection measure automatically obtained from a training corpus using the estimation algorithm GRAIL outperforms the best ratings derived during the parameter study.

Some of the components in the *Marvin* dialog system are configurable in such a way that the configuration influences the number of hypotheses generated during the processing of the user input. Section 14 shows the evolution of the number of hypotheses at several components depending on the underlying configurations for these components.

Due to the application-independence of the framework postulated in this thesis, the possibility to explicitly integrate the application operation parameters and the related methods as part of the framework gets lost. Yet, the user must be able to communicate her wishes towards the applications. Section 8 introduces an approach to represent the operation parameters from the applications decoupled from the dialog framework. The representation is derived from ontological representations provided by the applications, section 19. Atomic entities are connected by hierarchical structures, indicating the relations between the entities. Tasks are related to the representations of the operation parameters to indicate which action is to be taken with these operation parameters.

The counterpart of the Trindi information state with respect to the interaction between user and dialog system contains the system's view on the state of the discourse, section 9. It supplies a consistent view on the present discourse situation. Furthermore, the information state includes the means to conserve previous states, prepared to be available on demand.

Nowadays, the integration of newly acquired knowledge from the user input and existing knowledge from the previous discourse is closely related to the operated applications. Detaching the data representation of the application operation parameters from the dialog core and allowing dynamic application setups needs an application-independent formulation of the knowledge consolidation, section 10. It requires the extraction of relevant entities from the previous discourse and the means for merger. The integration process is highly ambiguous. It becomes even more complex when the integration of knowledge between different applications is permitted.

The sections 15 and 16 show generic functionalities which can be realized on top of the application-independent processing in the dialog framework due to the design of the information state and representation of the operation parameters.
The powerful means to navigate in the discourse are explained. The user is supported in acting, and even experimenting, with the dialog system since she can move backward and – under certain conditions forward – in the discourse.
Lists are commonly used to present the results of an operation performed by an application. To allow the access on all kinds of lists by various applications, methods to analyze the result for identifying lists and means to reference items in the list are given.

Decoupling dialog system and applications added an additional communication layer into the framework. As a framework for a classic dialog system, DymaLog has to organize the communication between user and system. In addition, however, the applications are no longer an integral part and thus the communication with the applications has to be modeled. Section 17 shows how the communication towards the user and the applications is monitored, organized, and reacted on by the dialog engine.

Interfacing and interacting with the applications as well as remarks on the application design with respect to the dialog system is covered in section 18. It includes considerations how the knowledge sources are affected by the various states of applications, especially for 'larger' applications. Yet the separation allows to encapsulate the applications, therefore the dialog system does not care about the internal structure and the means to realize an application.

Section 19 introduces the ontological model to allow an application to specify its operation parameters. The basic ontological objects are associated through two different relationships. From the ontological relations, the representation applied inside the dialog system, section 8, can be derived. The construction of ontologies supports the reuse of already defined entities. It also provides a mapping of compatibility to share discourse information between different domains, section 9.

The domain model described in section 20 applies the ontologies to create a relation between atomic objects. These objects can be either single objects or root-objects of hierarchical structures. It is utilized to create a relationship between a set of atomic objects during the integration of an interpretation of the current user input with the previous discourse.

As the input processing, section 21, is at present based on 'standard' components for ASR and NLU, we take only a glimpse on these units. In order to adapt the input processing on varying application setups, the knowledge sources are updated (Flycht-Eriksson and Jönsson [2000]).

The inverse of the input processing, the output creation discussed in section 22, is based on a handy combination of standard components to create visual output and to create text sequences that can be synthesized. An active component provides direct feedback to the user on the actual state inside the system.

6. Dialog Modeling for the Separation of Dialog and Applications

As we have seen before, various dialog frameworks developed in research and companies allow the interaction of a user with a system in order to operate a fixed set of functionalities. The applications realized by these systems cover a broad spectrum, ranging from name dialing in a mobile phone to call routing (Chu-Carroll and Carpenter [1998]), control of an unmanned autonomous helicopter (Lemon et al. [2001]) or robot (Burke et al. [2002]), or the *Trains* planning task (Ferguson et al. [1996]). Even though the components are often designed to allow a reuse in dialog systems addressing some other applications, in most cases the dialog system is designed to interface with a static configuration with respect to one or more applications. Allen et al. [2000] propose the 'Generic Dialogue Shell', an architecture that aims at components that 'should either be usable in any domain as is, or be easily adaptable to work in new domains'.

Taking as starting point the demand for a dialog system that decouples dialog processing from processing related to applications in order to allow flexible changes in the application functionality, we developed a collection of criteria for such a system in the previous chapter, summarized in section 4.

We will elaborate on an approach that completely takes the criteria of chapter I into account (te Vrugt and Portele [2004]). We refer to it as 'DymaLog' *(dynamic multi-application dialog framework)*. The knowledge processing of the dialog engine is inspired by Trindi (task oriented instructional dialogue; The Trindi Consortium [2001a]). We will look at Trindi in section 6.1. The subdivision of the framework into specialized functional units is heavily influenced by SmartKom (Wahlster et al. [2001]).

Furthermore, DymaLog partly shares ideas with recent dialog systems developed at the Spoken Language Systems group at the MIT on the basis of the Galaxy architecture. Filisko and Seneff [2003] presents a domain-independent 'context resolution server' for Galaxy-based systems being configured for different domains through appropriate knowledge sources provided by 'external files'. A separate dialog manager facilitates each domain addressable by one of the incarnations of the Galaxy-based systems. The context resolution server is composed of units to share common procedures between different domains; domain-dependence is configured through the 'external files'. Future research also targets at a more generic approach of a dialog manager

for the user in different domains. Yet, presently the systems concentrate on one single (fixed) application each.
Further inspiring work is presented in Denecke [2002]. It targets at the rapid and easy developing of dialog systems for single applications by reducing the burden of explicitly formulating the dialog strategy by introducing a description language for applications. The definition of data is based on typed feature structures.

6.1. The TRINDI Information State Approach to Dialog Modeling

First, we recap the units that build up the TRINDI view of an information state theory of dialog modeling as listed in The TRINDI Consortium [2001a] chapter 2:

1. "a description of the informational components of the information state (...)
2. formal representations of the above components (...)
3. a set of dialog moves that will trigger the update of the information state. These will generally also be correlated with externally performed actions, such as particular natural language utterances. A complete theory of dialogue behavior will also require rules for recognizing and realizing the performance of these moves (...).
4. a set of update rules, that govern the updating of the information state, given various conditions of the current information state and performed dialogue moves. Some of these rules will also *select* particular dialogue moves for the system to perform (in the case of a system participating in a dialogue rather than just monitoring one).
5. a control strategy for deciding which rule(s) to select at a given point, from the set of applicable ones. This strategy can range from something as simple as 'pick the first rule that applies' to more sophisticated arbitration mechanisms, based on game theory, utility theory, or statistical methods."

As indicated above, the TRINDI approach is not restricted towards human-machine interaction, e.g. it can be used to monitor human-human interaction. Let us now address the question how DYMALOG implements parts of these TRINDI elements.

The counterpart of the information state for DYMALOG is covered in section 8. Foundation for the informational components is the structured representation of the application operation parameters, particularly enhanced by e.g. the source of an item, its age, and a rating. The information state pictures the systems belief on the state of the conversation between a user and the operated applications, including the intentions by the user.

The update of the information state is connected to events initiated by the user and/or one or more applications. Examples for events that lead to a modification of the information state are speech input by the user or a state change of an application due to changes in the environment. The latter could e.g. be an 'external' communication request (like an incoming phone call) or triggered by a certain point in time (like a wake-up time).

Maintaining alternative interpretations of the user input inside parts of the dialog system after the input processing took place, section 7, requires a two-tier approach for updating the information state:

1. putting the new user input into the context stored in the information state while retaining the original status of the information state, and

2. integrating the hypothesis that survived the selection process into the information state afterwards,

section 8. We further can distinguish between input that adds to the information state, e.g. semantic entities which define operation parameters, or that operates on the information state without adding operation parameters, e.g. navigation in the discourse history.
In DymaLog, the information state is not directly responsible for the response of the system, i.e. the dialog moves of the system. However it provides the decision basis of the next system reaction.

The decision strategy on the selection of rules to apply for the update of the information state or selection of the next output to be presented to the user differs from Trindi.
For the update of the information state, a broad collection of potential interpretations of the user input integrated with the current information state is evaluated, and finally the best one according to an appropriate measure is selected. Thus the decision which rules best apply is delayed up to the selection process, sections 7 and 8.
For the creation of the system output as reaction to the user input and/or the changes in one or more applications, DymaLog relies on sets of rules provided by the system and each of the connected applications, section 22.

In the Trindi environment, application specific databases and operations are closely connected and directly addressed in the information state ('resource interfaces', The Trindi Consortium [2001a]). At this point, DymaLog clearly breaks with the Trindi approach.

The Trindi project provides an environment that can be used to realize 'task oriented instructional dialogs': the software package TrindiKit (The Trindi Consortium [2001b]). DymaLog however is based on the MultiPlatform architecture by the SmartKom project, see section 23.6.

6.2. *Multi-Modal Dialog Processing for Multi-Application Control in the* SmartKom *Dialog System*

DymaLog has a close connection to the dialog system developed within the SmartKom project founded by the German government (Bundesministerium für Bildung und Forschung, BmBF). The SmartKom demonstration system is an innovative multi-modal dialog system that allows the user to control a variety of services and devices in a flexible and intuitive way.

6.2.1 The* SmartKom *Scenarios. Apart from minor variations due to environmental and application specific needs, the same system core covers three scenarios (Wahlster et al. [2001], Blocher et al. [2003]). These scenarios are:

- SmartKom-*Home.*
  The SmartKom-Home dialog system allows the user to control devices and services in the area of consumer electronics (Portele et al. [2003]). Devices included for the control are a TV-set and a VCR, and an EPG gives an example for a service that can be accessed through the SmartKom dialog system. For the interaction, audio and video is used. A small and light dedicated device with a touch-screen enables visual output, and touch and gesture input respectively. The screen has a size close to an A4 paper.

- SmartKom-*Mobile.*
  The mobile scenario supports people on the move (Bühler and Minker [2005b]). E.g. it

allows to interact with an integrated navigation solution for car drivers and pedestrians. The user specifies a target and, if appropriate, she is guided to drive to a suitable place by the SmartKom unit inside the car. Once she arrived at this place, a handheld device seamlessly takes over the navigation and directs her to the requested target during the walk. While walking, the user might be provided with further information on highlights on her route, e.g. background information on a historical building.

- SmartKom-*Public.*
  SmartKom-Public provides an interface to a communication center, realized as a modern version of a telephone booth. It integrates different communication channels like phone, fax, and e-mail while ensuring the safety of personal data, e.g. the personal address list. The authentication might be performed via hand contour recognition or a person's signature.

Even though the varying incarnations of the same system differ in some parts, e.g. the home scenario uses a TV-set not available to mobile and public, and the car is unique to SmartKom-mobile, sharing the underlying basis allows mutually accessible applications. For example, a cinema information and reservation system is accessible in all scenarios and, also, access to the phone is useful in all scenarios.

The SmartKom project covered a wide range of aspects, yet we will briefly discuss areas of particular interest with respect to DyMaLog.

6.2.2 Multi-Modality. The SmartKom dialog system makes use of a variety of input and output modalities, varying for the three scenarios. The recognizer and analyzer for dealing with input from the user comprise spoken language, speech prosody, pointing, gestures, and facial expressions. While spoken language is an essential input modality for (spoken language) dialog systems and therefore covered by many historic and present dialog systems, multi-modal interactions often focus on the addition of pointing or gestures. Johnston et al. [2002] give an example for an advanced dialog system that combines different modalities. The 'MATCH' dialog system is an interface to a restaurant- and subway-information system. It can be controlled via speech, gestures on a screen, handwriting, and combinations of those. Thus, one challenge for SmartKom was the integrated view on a variety of input channels.
On the output side towards the user, SmartKom makes use of a persona. It mediates the reaction and state of the dialog system to the user through spoken language, gestures, and facial expressions. The persona called 'Smartakus' works with a set of tools to convey its state, e.g. during a relatively long computation 'Smartakus' pulls out a pocket calculator and starts calculating.

6.2.3 The Dialog Engine and Applications. The set of accessible devices and services changes between the scenarios. A functional model serves as intermediate layer between the dialog related components in the SmartKom dialog system and the applications. However, accessing certain functionalities during the interaction requires the explicit formulation of these operations in the plans of the plan-based dialog model, thus closely binds the dialog engine to the applications through the dialog model.
The application side in the demonstration system realized access to a set of services and physical devices, e.g. to send a fax, browse the current TV program, or spot the present location to enable location based information.

6.2.4 The Data Model Based on Ontologies. SmartKom applies an ontological model to represent the application related knowledge, Gurevych et al. [2003]. The structures and the semantics given by the model are published for use by each component that might take advantage of these structures and semantics. Some components that build up the SmartKom system strongly make use of this knowledge, e.g. the discourse model which puts the interpretation of the current user input into the context of the previous discourse. The ontological model allows deriving specialized descriptions of different entities from more general descriptions. Entities of different domains derived from the same ancestor may share discourse information during the interaction. Also, knowledge sources, e.g. for recognition and analysis, are presently handled separately from the ontologies – yet due to the variety of modalities the problem of connecting ontological entities and knowledge sources is complex.
Changes in the set of available application functionalities therefore imply significant changes in the dialog system, including the adaptation of components and update of knowledge sources like NLU grammars or the dialog plans.

6.2.5 Architecture. As already hinted, SmartKom pursues a modularized approach. More than ten universities and companies contributed components to the common system. The German Research Center for Artificial Intelligence[1] took the lead in coordinating the integration on the basis of the MultiPlatform architecture.

6.3. The Dialog Framework 'DymaLog'

The dialog framework 'DymaLog' *(dynamic multi-application dialog framework)* is a collection of individual units, each serving a specialized task. The DymaLog framework can be partitioned into four parts: input processing, dialog engine, output creation, and application related elements. Figure 6-1 sketches this division. Our main focus is on the dialog engine Aide *(application-independent dialog engine for dynamic environments)* and its separation from the applications.
Each single component in DymaLog can subscribe to one or more blackboards provided by MultiPlatform. The subscription can be 'read'-only, 'write'-only, or 'read' and 'write'. The number of components, which connect to a single blackboard, is not limited. Together with the blackboards, the subscriptions determine the order how the constituents consider data. Therefore, no central distribution component is required to mediate data between the components. The connections define 1-to-n relations ($n \geq 1$) between the components, with n typically being 1 or 2.

The components and connections between these components for DymaLog are shown in figure 6-1. The next sections of this chapter cover details on the constituents. We will now sketch the main task of each component to provide the overall picture on the composition of the framework.
In conjunction with figure 2-6, the applications are part of the dialog framework but not elements of the dialog system due to their changing nature. Only the application management is contained in the system itself.

6.3.1 In- and Output Devices. The physical devices that interface with the user are not part of the dialog framework itself. Common input devices are a microphone and a touchscreen; output devices comprise loudspeakers and displays.

[1]Deutsches Forschungszentrum für künstliche Intelligenz (DFKI)

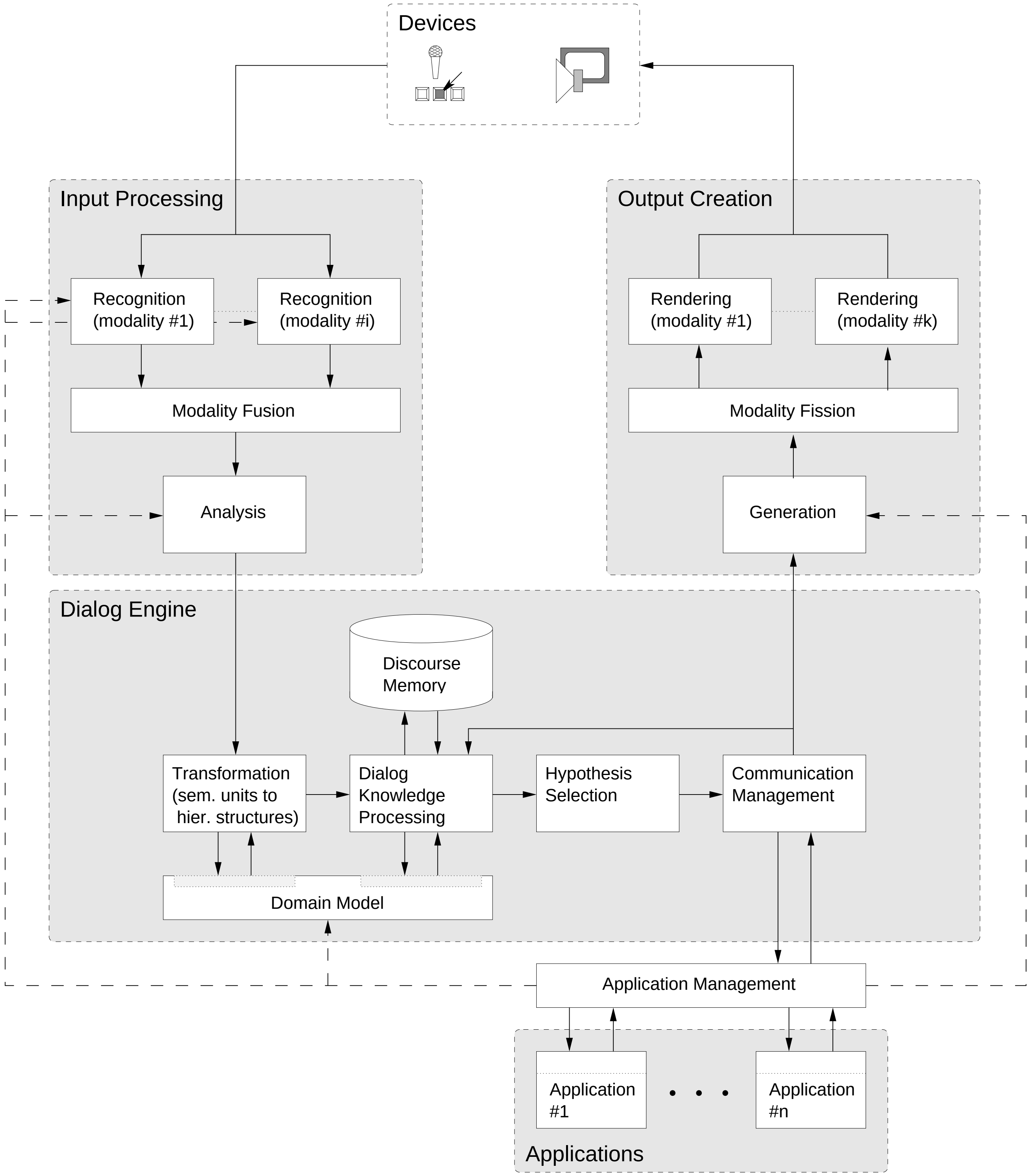

Figure 6-1: Outline of DYMALOG together with the input and output devices. The directed connections indicate the data flow (dashed: flow of knowledge sources). The framework consists of components that can be grouped into three parts directly related to the interaction with a user: input processing, dialog engine AIDE, and output creation. A fourth part with a strong link to the dialog engine is the mediator between the dialog system and the applications.

6.3.2 Dialog System. The components inside the dialog system are clustered into four areas. DymaLog is characterized by an 'application-blind' approach in its dialog engine, see section 8. The core dialog system renounces an explicit, inherent application model.

Input Processing. The input processing takes over the input from the user as recorded by the devices. The overall goal is the extraction of semantic entities. A sequence of two components performs the extraction.

- *Recognition.* The `recognition` components take the raw input and extract an annotation that represents this input. E.g. an ASR uses the acoustic information from the spoken input as recorded by the microphone to compute a representation for one or more hypotheses of the words likely to be uttered by the user.
- *Analysis.* Starting from the recognition result, the `analysis` obtains a semantic representation that is further processed by Aide. The result of the semantic analysis could e.g. be a set of so-called feature-value pairs. In such a case, the features might be directly associated with operation parameters of some application. Its value determines the value that should be used by the application.

The *Marvin* dialog system is currently limited to speech input. However, the input processing is prepared to handle input by additional modalities. It is realized by integrating the results of the modality specific recognitions (*Modality Fusion*) before the `analysis` is carried out.

Similar, in Trindi two units perform the processing: one to transcribe the user input ('input') and an analysis ('interpretation').

Dialog Engine Aide. Basically, Aide evaluates input by the user in the context of the discourse to operate an application and initiate an appropriate output to the user. Alternatively, the dialog engine might have to react on input by the application, which was generated due to some application internal or external event.

- *Transformation of Semantic Entities into Hierarchical Structures.* The input processing extracts a flat structure of semantic entities from the user input, which basically corresponds to a sequence of enhanced feature-value pairs. However, Aide utilizes hierarchical structures, which represent the relations between application operation parameters. Therefore, a `transformation` that (i) connects the semantic entities to operation parameters and (ii) builds up the structures is needed. The `domain model` (see item below) supports the `transformation`.
- *Discourse Memory.* The `discourse memory` captures the progress of the discourse and of the interaction with the applications as perceived by the dialog system. It closely connects to the dialog knowledge processing (see next item). The content of the memory is made available so that newly obtained knowledge can be treated in the context of the interaction with user and applications.

  The `discourse memory` corresponds to major parts of the Trindi information state. Yet, parts of the information state are located in the applications.
- *Dialog Knowledge Processing.* Serving as the driver of the `discourse memory`, the knowledge processing is responsible to integrate newly obtained information into the current belief status as contained in the `discourse memory`. This includes the intra- and inter-domain integration, resolution of list references, and handling of instructions towards the dialog engine Aide itself. Among other things, it is further responsible to

initiate updates of the `discourse memory`, either triggered by newly obtained knowledge or by instructions operating on the memory. By this, some of the basic generic functionalities heavily rely on this processor. For the integration task, the knowledge processing makes use of the `domain model` (see next item).

The knowledge processing can be regarded as being the counterpart of the TRINDI dialog move engine.

- *Domain Model.* Formulating an application-independent knowledge processing in the dialog engine leads to the challenge of finding an appropriate representation of the applications' operation parameters. The hierarchical structures used to face this challenge, section 8, provide the basis to formulate the instructions inside the knowledge processing abstracted from the applications. As already noted before, the hierarchical structures are derived from a flat structure of semantic entities. The `domain model` provides means to establish a relationship between semantic entities and/or already existing hierarchical structures, thus serving as the connective link between the two scopes of the knowledge representation.

  In terms of TRINDI, the `domain model` also realizes parts of the dialog move engine. The knowledge on ontologies, however, seems to be coded in TRINDI's information state.

- *Communication Management.* Basically, the `communication management` supervises the interaction of the system with the 'outside world', i.e. with the user and the applications. As such, the manager e.g. controls the activity status of the speech recognition component, or it might decide to continue or abort waiting for a reaction of an application on a request posed by the `communication management`.

  In addition, the communication management implements parts of the TRINDI dialog move engine, but also takes over some functionalities of the control module.

- *Hypothesis Selection.* At various stages in the input processing, a variety of options to deal with the actual content under consideration is available. Instead of directly choosing one of these options – and thus rejecting the possibility that one of the alternatives could lead to a more appropriate result in the end – a set of alternatives is maintained up to the `hypothesis selection`. The selection decides on the basis of a rating computed for each hypothesis on the 'quality' and selects the best hypothesis according to this measure.

  The update of states and modification of hypothesis inside TRINDI relies on a set of rules from the dialog move engine. In a unique process, one sequence of rules is selected and applied. DYMALOG replaces the unequivocally determined selection by multi-hypotheses processing together with the hypothesis selection to allow a wider variety of interpretations of the user input.

The `dialog knowledge processing` and `discourse memory` together build the core of the information state.

Output Creation. The output creation is the inverse operation to the input processing. The basis of the output, which is presented to the user, is the hierarchical structure. A reduction to a flat structure of semantic entities is not enforced. The two main constituents of the output creation are:

- *Generation.* Starting from a hierarchical structure, comprising the application operation parameters relevant for the current interaction step and enriched by the dialog system, the `generation` creates a representation that embeds the information to be presented to the user. It can be used to prepare the output for different modalities. DymaLog uses spoken language and (mainly static) visual output.

- *Rendering.* The `rendering` takes over the presentation of the generated content towards the user, especially through the audio channel and a screen.

Like for the input processing, the output creation has a direct counterpart in Trindi ('generator' and 'output').
Within the present realization of the *Marvin* dialog system, the *generation* already processes its input separately for each of the available output modalities. Therefore, the task of the current *modality fission* component is simple: separate the generated output with regard to the target modalities and deliver the modality specific output to the proper renderer.

Application Management. The application management is realized through a single component, the *application management*. The main task is to mediate between the dialog system and the applications. Furthermore, the management is the central point to collect the individual knowledge sources and to consistently integrate these. The integrated knowledge is made available for the usage in the affected components of the dialog system.

Since the approach of interfacing and interacting with applications is different, no adequate counterpart is present in Trindi (like the resource interfaces).

6.3.3 Applications. As DymaLog is designed for flexible adaption to changing application configurations, the dialog framework does generally not restrict the realization of applications. The applications have in common that they adhere to a certain interface towards the application management, including:

- a description suitable for the operation of an application (realized by a 'request'-'response' pair), and
- means to publish the knowledge sources related to an application.

The operation parameters required to operate functionality inside an application must allow a representation according to the hierarchical descriptions that will be introduced in section 8.
The approach aims at permitting the integration of existing applications and applications that become available in the future. This includes applications being developed especially for the use in the dialog system and applications extended by an interface to enable the operation through the dialog interface.

We will consider the constituents and aspects of the framework in the next sections of this chapter in more detail. This includes the components as outlined in this section. It also enfolds the foundations underlying and connecting the components, like the data representation and interfaces between the components.

7. A Consequence of Uncertainty and Ambiguity: Multiple Hypotheses

Maintaining alternative interpretations of the user input during various stages of the processing allows to decide on the most suitable hypothesis after additional knowledge is available (Souvignier et al. [2000]), e.g. after the interpretations of the user input are brought into the context of the previous discourse. Thus, a more complete picture is given and the finally selected hypothesis is not the result of a sequence of decisions for locally optimal solutions in each component.

The main task of the `hypothesis selection` is to pick out the 'best' representation of the user input integrated into the context of the previous discourse from a collection of possible representations. Let us first introduce those terms that will be used in conjunction with the processing that allows for non-unique interpretations of input to the dialog system.

7.1 Definition (Hypotheses, Hypothesis, and Alternatives)**.** Let s be a signal processed by some processing chain p. Assume further that the result of the processing chain is in general influenced by ambiguity and uncertainty, i.e. various different outcomes exist that represent valid results of the processing by the chain.
The set of valid results $R = \{r_1, \ldots, r_n\}$, $n \geq 1$, processed by p is denoted *hypotheses*. If one element $r \in R$ is considered, this is called *hypothesis*, the remaining elements $R \setminus \{r\}$ are the *alternative hypotheses*, or short *alternatives*.

The hypothesis $h \in R$ is denoted *best* or *true hypothesis* of all hypotheses in R with respect to a measure $\mathtt{m}$, if $\mathtt{m}(h) \geq \mathtt{m}(h')$ for all $h' \in R$. The short form *best* or *true hypothesis* h will be used throughout this text, omitting the underlying measure. ◇

Note that definition 7.1 does not depend on DymaLog.
The term *ambiguity* is used according to Merriam-Webster's online dictionary (Merriam-Webster OnLine [2006]): ambiguity denotes 'the capability of being understood in two or more possible senses or ways'. Adapted to DymaLog, ambiguity means that some input to the system can be interpreted in two or more possible senses or ways. Throughout this text, the ambiguity of interpretations of input into a dialog system is in general related to the dialog system and not the applications. The question how an application should handle an application related ambiguity is presently left to the application designer[2]. Each single interpretation is represented by a separate hypothesis.
Merriam-Webster's dictionary explains *uncertainty* as 'not known beyond doubt', 'not certain to occur', or 'not clearly identified or defined', among other meanings. The use of uncertainty in this text is oriented towards 'not known beyond doubt' and 'not clearly identified'. E.g., 'their' and 'there' sound similar. An ASR based on stochastic models may deliver 'nearly equal' probabilities for these two words given certain user input. If there is no need to decide on one of these words, the recognizer may provide both variants due its uncertainty.

Input into the dialog system is processed and modified many times before an application is contacted to perform an operation based on the processed input and/or the system reacts on the input. We refer to processing results as *interpretation* or *processed representation of the user input*. The short form *representation* will also be used, the term 'of the user input' might be omitted. These notations are not restricted to completely processed results, but will also address intermediate results.
The approach taken in DymaLog is to combine a variety of processing steps without prior restrictions on the models underlying the single processing steps. The models may e.g. be based on stochastic models and thus produce different results due to uncertainty, or be unique in a sense that there is only a single result of the processing. During each of the processing steps, alternative

[2]For instance, ambiguities in the applications due to under-specification are omitted. To illustrate under-specification in applications, assume that a user utters *"show news"* at 8 p.m. in a TV application that can utilize EPG knowledge. If more than one channel broadcasts news at 8 p.m., an ambiguity with respect to under-specification can be observed: the application does not know which channel to choose. It could be resolved by the user specifying a channel, or by using background knowledge like the user's preference to watch news on channel `ARD`.

interpretations of a single input could arise due to the uncertainty which interpretation is the best one in the context of other processing steps (van Rooy [2001]). However, also approaches using stochastic models for dialog management model the uncertainty in the interaction of user and machine, especially Markov decision processes and the derived partially observable Markov decision processes (Zhang et al. [2001]). Competing interpretations are often a consequence of uncertainty during one or more of the processing steps, Lemon et al. [2002b].

The capabilities introduced in this section enable the simultaneous handling of competing interpretations as formulated in the criteria catalog, section 4.3.

We will outline the creation process of the hypotheses and sketch how DymaLog deals with multiple hypotheses next.

Actually, in DymaLog hypotheses will be created by the recognizers and derived from existing hypotheses at miscellaneous processing stages. This implies that components, which further process interpretations of the user input, have to ensure that each interpretation finds the same condition. I.e. the internal state of a component must not be changed due to a preliminary – not yet selected – hypothesis. Once the choice for one of the hypotheses is made, the components can adopt their state in accordance with the selected hypothesis – this in particular concerns the `discourse memory`.

Figure 7-1 shows the creation process for the hypotheses starting with input by the user up to the selection of a hypothesis, thus concentrates on a part of figure 6-1. From the figure, one can derive the components that add to the number of different hypotheses. Basically, the components can be clustered by types of data they are operating on:

- sequentially ordered (e.g. transcriptions, sequence of semantic entities),
- hierarchically structured, and
- sequentially ordered in the transition to hierarchical structures (mainly inside the `domain model`).

7.1. Sources of Competing Hypotheses

For the recognition process, especially in the ASR, statistical models are the common basis to compute transcriptions representing the input. During the recognition process, a huge variety of possible transcriptions for the input is evaluated according to an underlying measure, in general a probability distribution. Of course, the subsequent considerations could be based purely on the top-rated transcription. However, that would mean to discard the chance of reevaluating the quality of the transcriptions if further knowledge becomes available. In particular, in DymaLog the `analysis` profits from maintaining alternative transcriptions. The recognizers use a lattice to compactly represent a large number of alternative transcriptions (e.g., automatic speech recognizers often represent the recognition result in a so called 'word-lattice'). Furthermore, the recognizers allow restricting the density of the generated lattice, thus allowing controlling the number of transcriptions handed to the subsequent components. The alternative transcriptions represented by a lattice reflect the uncertainty in the models underlying the recognition component.

The fusion of the generated lattices results in a single recognition lattice, which contains the possible and allowed combinations[3].

[3]Actually, the input to the *Marvin* dialog system (chapter III) is currently restricted to spoken language. However, the basis to enable multi-modal input in the future is taken over from the Spice dialog system, Kellner and Portele [2002]

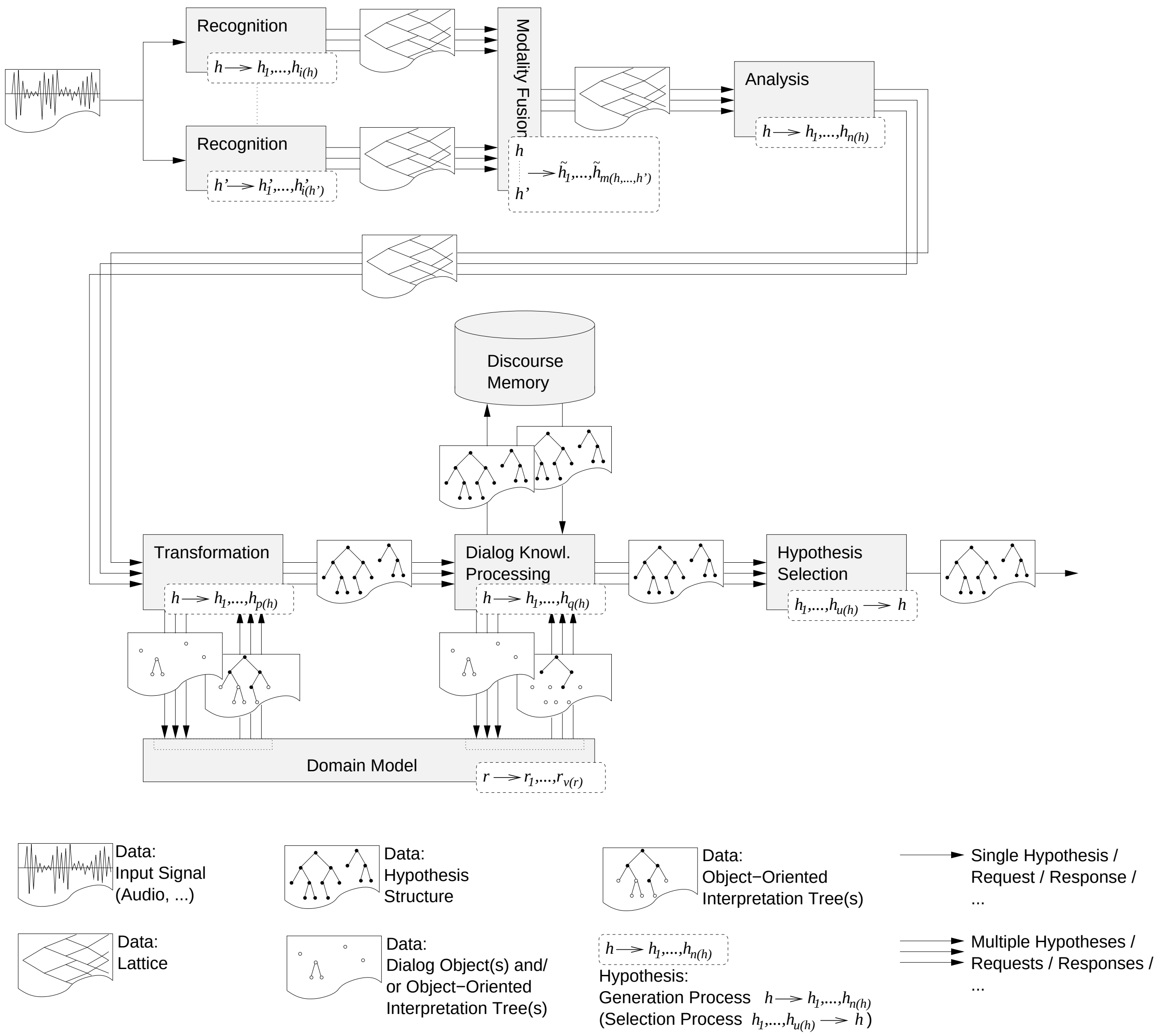

Figure 7-1: Qualitative inspection of the generation process of multiple hypotheses from input to selection of the best hypothesis. Data examples are available in chapter III and section A1.

The models underlying the `analysis` can be pretty different. In the case of our framework, a rule-based NLU extracts semantic entities from the transcriptions. Each transcription leads to one or more hypotheses: as a 'default' trivial hypothesis, the language understanding assumes that no semantic entities can be extracted from the transcription (semantically 'empty' hypothesis). Due to the ambiguity that can be partly inherent in the rules for the extraction of hypotheses, e.g. because of ambiguity of language or the usage of equal phrases for different applications, the computation of hypotheses from a single transcription can lead to more than one hypothesis. Again, a lattice represents the analysis result ('analysis-lattice'). Ammicht et al. [2001] investigate the representation and resolution of 'value ambiguities' and 'position ambiguities' inside a dialog system within a single structure.
Examples of a word-lattice and an analysis-lattice are given in the appendix, section A1. For examples of the data structures inside the dialog engine Aide, i.e. the components following the `analysis`, we refer to chapter III, especially section 25.

The `domain model` supports the transition from the flat structure of semantic entities towards hierarchical structures. A fundamental step carried out by the `domain model` is to build up hierarchical structures covering a set of given single nodes. For a given set of nodes, the coverage also can be ambiguous. Main reasons are the ambiguity inherent in the hierarchical structures, e.g. the time in figure 3-1, or the potential usability of semantic entities in various applications, e.g. a time might be used to define the starting point of a broadcast show or a wake-up time. Therefore, each set of nodes that is connected by the `domain model` might lead to a variety of competing structures.
The number of structures generated for a given set can be limited. This is done by rating each generated structure and then selecting the 'first best' n structures according to the underlying rating (for a given n).

Transforming semantic entities to hierarchical structures at first extracts the n top rated candidates from the lattice representing the analysis result. Starting from one of these n hypotheses consisting of a sequence of semantic entities, together with the `domain model` hierarchical structures are derived. The ambiguity in the `transformation` process is primarily a consequence of the ambiguity in the `domain model`, see before.

The `dialog knowledge processing` also utilizes the `domain model`. In order to integrate an interpretation of the user input and the current state of the discourse as perceived by the dialog system, the knowledge processing consults the `domain model` to obtain relationships between hierarchical structures as generated by the `transformation` and structures contained in the `discourse memory`. Thus, once more the `domain model` serves as source for ambiguity. Based on the relations from the `domain model` between new and existing structures, the second source of ambiguity is the creation of the integrated structures. Figure 7-2 illustrates the ambiguity

1. inherited from the `domain model`,
   see the two exemplary coverages for *`timer-slots`* and *`broadcast`* that can be derived from the structure representing the application operation parameters, and

2. resulting from the integration process,
   see arrow (1) and (2) that point to different integration results for the same covering structure.

When *`timer-slots`* and *`broadcast`* remain independent, the integration result is unique, see arrow (3).
During the integration process, the knowledge processing might vary the scope of turns being considered for the integration with the current user input. This again increases the number of hypotheses generated by the knowledge processing.

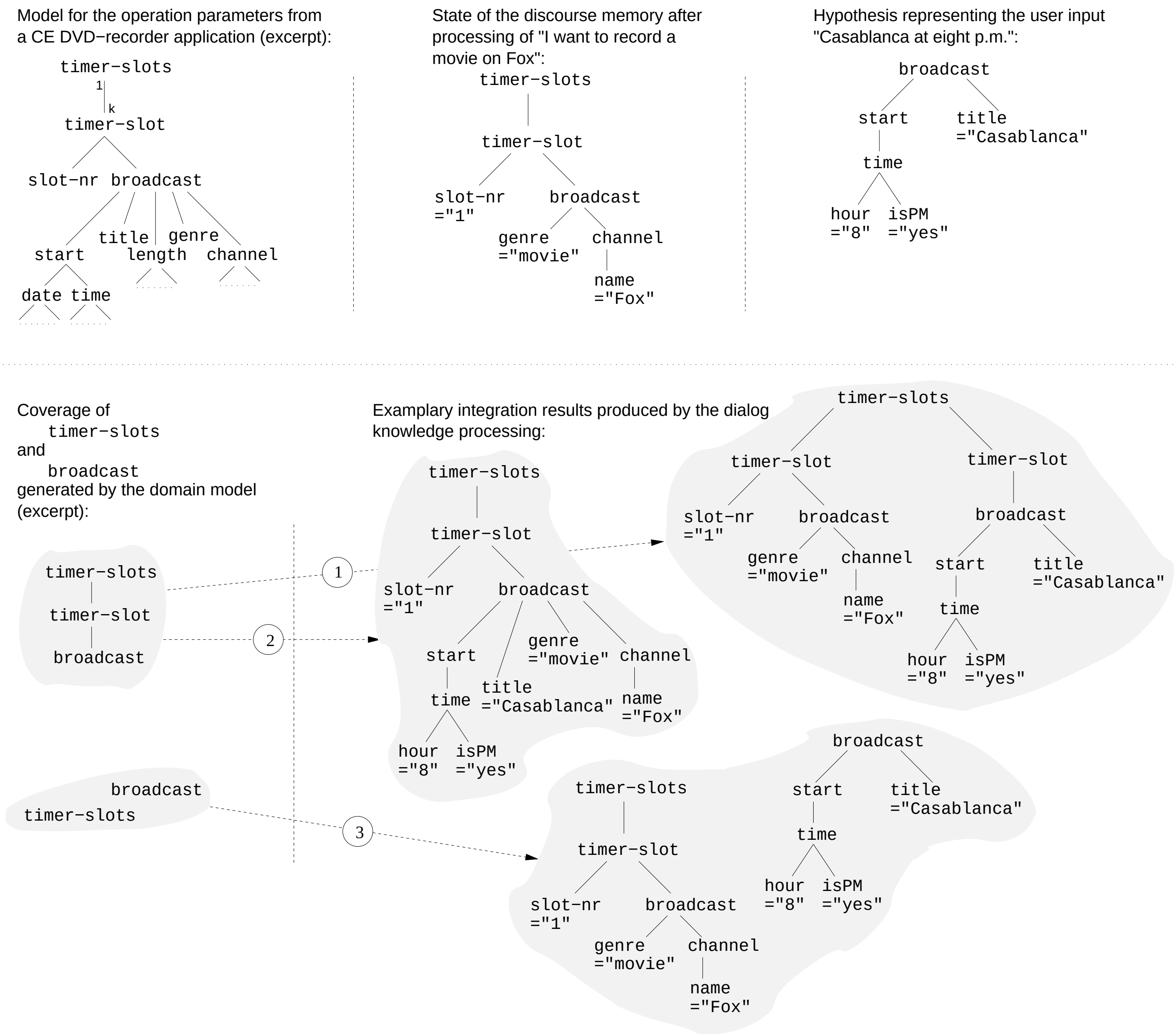

Figure 7-2: Integration of hypothesized user input and discourse memory. In the upper part, the left side shows the model to represent the application operation parameters in this example. The structure in the middle illustrates the possible content of the `discourse memory` after the user uttered *"I want to record a movie on Fox"*, and this input was processed by the system. In this example, only the user input and neither the dialog engine nor an application added to the state of the `discourse memory`. In the upper right, the structure representing the subsequent user input is shown, e.g. as reaction to the question *"when do you want to record a movie"* posed by the system. The left side of the lower part of the picture gives two different ways of combining the root-objects of the structures inside the `discourse memory` and derived from the last user input as provided by the `domain model`. The right lower area shows three possible integration results that can be derived from the given constellation. The hierarchical structures depicted in the figure serve demonstration purposes and do not comply in detail to the exact structures as used in DyMaLog (section 8).

The `discourse memory` itself does not generate additional hypotheses since it does not directly participate in the processing chain for the user input but represent the system's current belief.

The `hypothesis selection` forms the opposite to the components discussed so far since it reduces the set of generated hypotheses for the input to some few – ideally one – hypotheses. We take a closer look on the measure underlying the selection process in section 11.1. Thus starting with the selection process, the adjacent steps can limit themselves to the inspection of one single hypothesis[4].

7.2. Processing Hypotheses in Parallel

Two forms of appearance of a modularized framework with respect to the distribution of the components, e.g. based on the MultiPlatform (Herzog et al. [2004]) or Galaxy (Seneff et al. [1998]), are obvious:

1. The dialog system is realized as a dedicated device, often in combination with one or more applications. In demonstration systems often a single PC or a PC cluster takes over this task.

2. In connected environments[5], e.g. the ambient intelligence vision, a compound of devices can host the dialog system. A device that usually serves a complete different task can include single components required by the dialog system.

Our approach allows us to asynchronously process hypotheses for the same user input, i.e. different components in the processing chain compute different hypotheses simultaneously, so each hypothesis completed a different number of required steps at a certain point in time. Since a single `discourse memory` is maintained, a duplication of some components can be realized if needed, e.g. especially in the second case two or more instances of the `domain model` running on several devices in the network might work in parallel on competing hypotheses (Cheng et al. [2000]).

This also means that the order in which hypotheses are regarded by the processing steps cannot be predetermined up to the `hypothesis selection`: until the best hypothesis is selected, all competing hypotheses have to find the same preconditions. Then the `discourse memory` integrates the best hypothesis. Furthermore, the update of the internal state of an application has to originate from the best hypothesis. These challenges are in line with problems reported for an asynchronous agent-based architecture in Blaylock et al. [2002]. The multi-hypotheses approach presented here also allows a variant of asynchronous processing up to the `hypothesis selection`.

Asynchronous hypotheses processing implies that the hypotheses residing in the system are collected at a central place. Canonically, the `hypothesis selection` takes over this task.
In order to decide when the selection of the best hypothesis should actually be performed, a counter-based indicator triggers the selection process. It heavily relies on the preceding steps: Once the `transformation` constructs the initial hierarchical structures, the `hypothesis selection` is notified on the number of hypotheses generated (as shown in figure 7-1). Each component, which adds to the number of hypotheses, notifies the `hypothesis selection` on changes

[4]Currently the `hypothesis selection` restricts its result to one hypothesis. In the future, the communication management might be extended to deal with hypotheses with a similar or an even equal rating.

[5]We will use the term *connected environment* to refer to environments in which devices and services are connected via some network infrastructure. The term *networked environment* is also widespread to denote such environments.

in the number of hypotheses to expect. E.g. if the `dialog knowledge processing` generates $q(h)$ hypotheses out of a single hypothesis h, $q(h) > 1$, the `hypothesis selection` is informed that $q(h) - 1$ additional interpretations of the user input are now to be collected[6]. Bookkeeping inside the `hypothesis selection` then triggers the selection process as soon as the number of collected hypotheses reaches the number of hypotheses being expected.

For a non-research system, a temporal threshold may serve as additional trigger to actually start the selection process. A certain period after the `hypothesis selection` receives the first hypothesis or input by the user was detected, the `hypothesis selection` chooses the best hypothesis according to the underlying measure, no matter if all hypotheses were received so far. This ensures acceptable response times.

8. Representation of Knowledge and Interpretations on the Basis of Application Operation Parameters

After discussing the aspects of the `hypothesis selection`, we will now turn towards the processing of the content of the interpretations. The structured representation is demanded by 4.1(2) of the criteria catalog.

Up to this point, it was sufficient to note that hierarchical structures are used to represent the information to permit the interaction of a user and the system, and to drive the applications. However, hierarchical structures are also used by other dialog systems, e.g. Filisko and Seneff [2003] use 'semantic frames' to hierarchically represent linguistic knowledge from the user utterance, possibly integrated with the previous discourse. Mori [1999] discusses the representation and utilization of knowledge on different levels in a dialog system. Let us formalize the data underlying DyMaLog, which permits the separation of dialog and application processing.
Our application-independent approach in DyMaLog is *application-blind*: in the separation of dialog engine and application, the application-blind dialog engine makes use of a limited set of knowledge sources to be able to interact with a user about an application. However, we explicitly exclude any representation of processing logic related to the applications from the core dialog system[7] – in contrast to the, possibly abstracted and partial, reproduction of the methods related to an application in the dialog system. Different methods to formulate an application inside a dialog system are state of the art, ranging from the integrated implementation together with the dialog model to the description via an appropriate language or logic framework. DyMaLog's view on an application, and thus also the interface between the dialog engine and the applications, is focused on entities to represent application operation parameters, together with relationships between these entities, as will be outlined in this section.

A priori, the hierarchical structures do not predetermine the initiative in the communication between user and system. In principal, the `communication management` together with the output creation is responsible for enabling user-driven, system-driven, or mixed-initiative interaction (Chu-Carroll and Brown [1998], Novick and Sutton [1997]). The system views the knowledge mainly from two perspectives:

[6] In principle, equal hypotheses might be recombined by the component, thus reducing the number of hypotheses to expect. In the current *Marvin* dialog system, this recombination is postponed until the `hypothesis selection`.

[7] We should mention that a dialog system that is separated from its related applications in the strict meaning of definition 2.2 is application-blind.

- the interaction of the user with the system is supported by carrying the interpretation of the user input integrated with the previous discourse, and
- executing an operation by an application needs the specification of certain application operation parameters.

The representation, which serves those two purposes, is naturally also used to exchange data between the components of the framework (figure 6-1). In particular, it defines the exchange format between dialog engine, application management, and applications.

8.1. Basic Entities of the Knowledge Representation: Dialog Objects

Nodes form the basic building blocks of the hierarchical structures. A dialog object defines the single nodes.

8.1 Definition (Dialog Object)**.** A *dialog object (DO)* o is a tuple

$$(\mathtt{d}, i, \mathtt{c}, [\mathit{children}|\mathit{value}], [\ldots])$$

made out of the core components

- `d`: the *domain* of the dialog object that associates o to applications,
- i: the *instance* specification of the node[8] that is used to establish the relation of a DO to a real-world entity, and
- `c`: the *class* that defines the type of the node; the definition is based on other nodes[9] or atomic types (e.g. integer or string).

A dialog object could include either

- *children*: the set of dialog objects forming the *children* of o such that a directed acyclic graph is built up

or

- *value*: the *value* that is associated with the object o

or none of both. An object that carries children will also be called *inner dialog object*, other objects – especially those with an associated value – are *leaf dialog objects*. Furthermore, a dialog object can comprise a set of optional elements:

- *scores*: a hierarchical substructure made of a *set of feature-value pairs representing different scores*, carrying the ratings associated with this node (generated during the processing by different components),
- *age*: the *age* in terms of number of turns (current turn: $age = 1$, previous turn: $age = 2$, $\ldots$),

[8]For a motivation of the term 'instance' see section 19.

[9]Classes connecting a dialog object to another DO or structures built upon DOs will usually be prefixed by '`C`', e.g. `CChannel` in figure 8-1.

- *isUnique*: a flag that indicates if other children of the object's parent DO with the same domain, instance, and class are permitted to exist, and/or
- *src*: the *source* being of vital influence of the current appearance of the DO. A DO may be created or modified as a direct consequence of the user input, resulting from the processing of the new user input by the dialog system, or created or modified by an application. The source reflects by whom the present form of a DO was primarily influenced.

In order to refer to dialog objects, a set of different (short) notations is used.

> For the core components domain `d`, instance $\mathtt{i}$, and class `c` of an DO $\mathtt{o}$ the notations
>
> $$\mathtt{o} = \mathtt{i}\{\mathtt{c}\}, \qquad \mathtt{o} = \mathtt{d}\colon\mathtt{i}, \qquad \text{and} \qquad \mathtt{o} = \mathtt{d}\colon\mathtt{i}\{\mathtt{c}\}$$
>
> are used. For simplicity, also the instance $\mathtt{i}$ may be used as placeholder for a DO.

Note that these notations omit further constituents of a DO besides the core components.

> To access the value or an optional element e of a DO $\mathtt{o} = \mathtt{d}\colon\mathtt{i}\{\mathtt{c}\}$ the notation
>
> $$\mathtt{o}[e] = \mathtt{d}\colon\mathtt{i}\{\mathtt{c}\}[e]$$
>
> is used. For leaf DOs carrying a value `v` representations of the form
>
> $$\mathtt{d}\colon\mathtt{i}\{\mathtt{c}\}\texttt{="v"}$$
>
> as abbreviation of $\mathtt{d}\colon\mathtt{i}\{\mathtt{c}\}[\mathit{value}]\texttt{="v"}$ is written.

◇

Even though elements of a dialog object are not shown using the above-mentioned notations, keep in mind that these elements a part of a DO. In Denecke [2000], simple multidimensional feature structures replace the nodes in typed feature structures with vectors of elements. The definition of a dialog object used throughout this text can be basically divided into two major constituents: *(i)* the core components together with the parent-children relationships or the value of an DO reflecting semantic content and *(ii)* the more technical aspects covered e.g. by score, age, or originator. The design of the latter is open for extensions to include additional properties relevant for DymaLog.

8.2. Hierarchical Structures of Dialog Objects: Object-Oriented Interpretation Trees

The parent-child relations define member relations: from the parent to the children we zoom to a finer granular view on an object. E.g. the parent 'time' may contain the children 'hour', 'minute', and 'second', thus disassemble their common parent 'time' on a more detailed level.

Through the parent-child relations, we can derive tree structures with dialog objects forming the nodes of the tree. These trees are denoted *object-oriented interpretation trees (*o^2I*-Trees)*. The name is motivated as follows:

- The nodes of an o^2I-Tree are given by the DOs. A DO itself is a complex structure. It carries entities that can also have a (hierarchical) structure themselves. Therefore, DOs go clearly beyond single feature-value pairs.

- The o^2I-Trees are designated for interpretations of input into the dialog system. Correspondingly, each hypothesis derived from input by the user is represented by a set of o^2I-Trees, see section 8.6

- The operations performed on a hypothesis are mainly determined by operations on the nodes – the DOs – inside an o^2I-Tree. Properties and characteristics of these complex objects underly the decisions to be made, see e.g. section 10. Furthermore, the operations on a hypothesis inside the application-blind Aide are generic, i.e. that the processing steps do not depend on the applications.

The terms for inner and leaf dialog objects are derived from such tree structures. Note that the single dialog object itself forms a non-empty, but trivial, o^2I-Tree.
Consider figure 8-1. On the left side, we take a closer look on one of the dialog objects contained

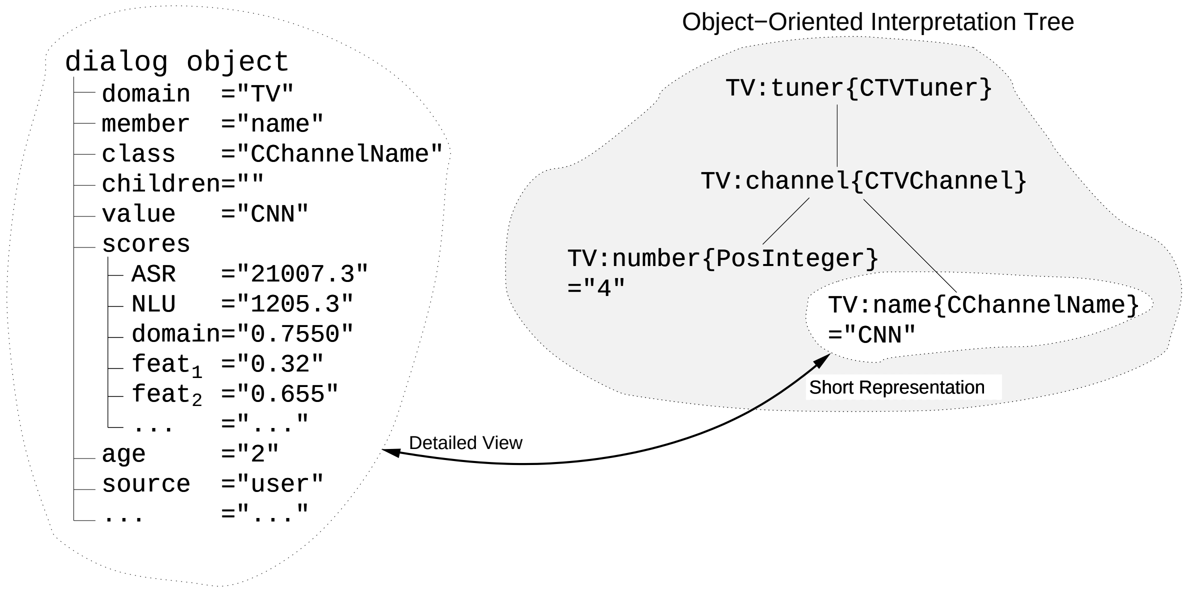

Figure 8-1: An o^2I-Tree representing the channel `CNN` as part of a TV tuner. The left side shows a closer look on the dialog object referring to the channel name.

in the o^2I-Tree at the right side. It shows that a DO itself has an inherent (tree) structure. However, we will mostly meet dialog objects in the context of object-oriented interpretation trees, thus using the short representation.
The applications define the object-oriented interpretation trees through ontologies (section 19). The o^2I-Tree in figure 8-1 refers to a concrete channel as it might be programmed into a TV-set, it could be a subtree of a more general o^2I-Tree representing the device 'TV-set'. Thus the `CTVTuner` tree can be regarded as an (partly) instantiated tree derived from an abstract o^2I-Tree which is provided by the ontologies of the TV-set application, defining the constituents and relations of its operation parameters (Russell [2000]). Such an abstract tree does not carry values. However, usually the dialog objects have counterparts in an application, e.g. the `CTVTuner` represents the tuner component of a TV-set, the channel number is included via the `TV:`***`number`*** {`PosInteger`} DO.

The explicit usage of hierarchically structured descriptions of the application operation parameters forming the basis of dialog management is given in Veldhuijzen van Zanten [1998].

In the implementation of DYMALOG, we use XML to represent of the DOs and o^2I-Trees. XML is also used to represent structures that combine several o^2I-Trees, see below.

8.3. Sources of Dialog Objects

The task of creating new dialog objects is mainly located at three different places:

1. the `analysis` together with the `transformation` process of semantic entities into dialog objects generates DOs directly from the user input,
2. the `domain model` connects single dialog objects received from the `transformation` process or `dialog knowledge processing` using object-oriented interpretation trees, thus generally requiring additional dialog objects to build up these trees, and
3. an application may insert dialog objects into an existing o^2I-Tree, e.g. when defaults are applied or the results of an operation are represented by additional object-oriented interpretation trees.

The user can therefore be the source for the creation of a DO via her input, or processing inside an application creates new DOs.

Integrating the current user input with the previous discourse leads to object-oriented interpretation trees consisting of nodes that were introduced in different turns. This is reflected by the age component of a DO. The age is computed relative to the current user input: the age of the current input is 1, the previous input has the age 2, and so on. Applications can make use of the age, e.g. to handle younger DOs with precedence over the older DOs.

A special domain that will be used for generic functionalities is the `system` domain. It is not associated with one of the custom applications but can be considered to feature a framework inherent application SYSTEM. The `system` domain is e.g. utilized in the navigation inside the discourse or the access to list entries.

8.3.1 Reuse of Object-Oriented Interpretation Trees. Once an o^2I-Tree is defined, it can be utilized during the definition of other object-oriented interpretation trees. This in turn allows us to model sections of the world once and reuse it in various object-oriented interpretation trees specialized for certain applications. Prominent examples are trees representing a time or a date, a person, or a place.
The usage of a structure representing the time in two different applications is outlined in the figures 8-2 and 8-3[10]. Figure 8-2 defines structures for time, date and – as a superstructure combining these two – point in time. An excerpt showing an o^2I-Tree belonging to an alarm clock application makes directly use of the `CTime`. A video recorder (VCR) application integrates the time indirectly via the `CPointInTime`. Figure 8-3 shows the fully expanded object-oriented interpretation trees as these would be used in instantiated form by AIDE.

The figures show that object-oriented interpretation trees integrated into other trees do not necessarily share the same domain, i.e. the trees might be defined by different applications. In DYMALOG, o^2I-Trees being integrated into other trees therefore inherit the domain of the higher-level structure, e.g. the `CTime` is used in the `AlarmClock` and `VCR` domains.

A second aspect can be observed in the figures; that is the usage of non-instantiated DOs, notation $o = $ `d:*{c}`. The instance '`*`' is used as root of abstract object-oriented interpretation

[10] Another example for the reuse of object-oriented interpretation trees might be the channel object from the o^2I-Tree representing a timer slot and the channel object from figure 8-1.

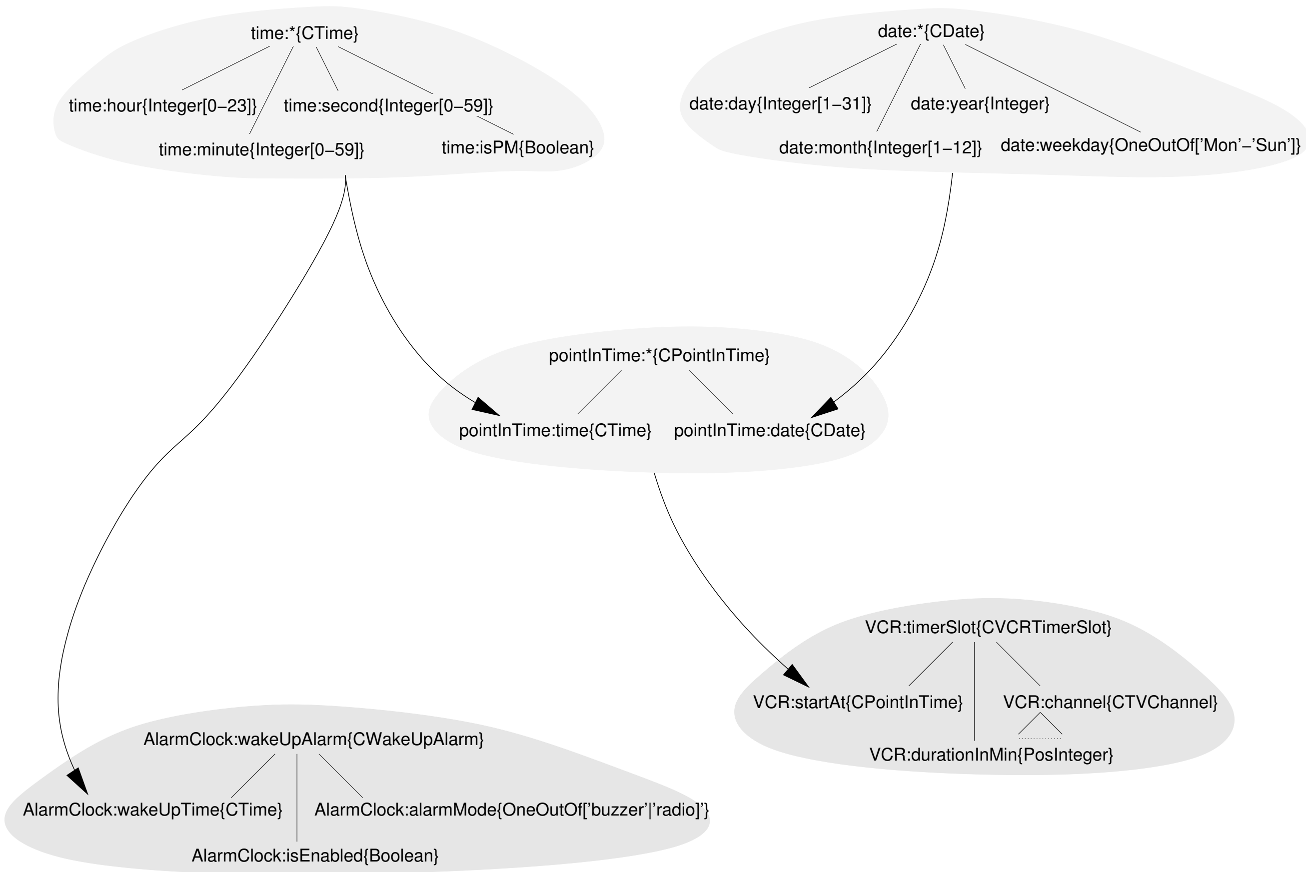

Figure 8-2: Reuse of object-oriented interpretation trees representing time, date, and point in time by object-oriented interpretation trees from the applications 'alarm clock' and 'video recorder' (VCR).

trees defining operation parameters, i.e. non-instantiated object-oriented interpretation trees. Instantiation connects the abstract trees to real world entities.
Let us consider the structure `time:*{CTime}`, which represents a time composed of hour, minute, second, and a flag for 'am' or 'pm' respectively in the above example. The structure as such is not related to a real world entity, thus the root object is not instantiated. However, its components represent concrete instances, e.g. the hour of a time can be represented by an integer value between 0 and 23.
The abstract structure representing a time is now instantiated through a relation with an entity. By declaring that the wake-up time of an alarm clock is of class `CTime` defined in the domain `time`, a concrete time with a reference to a real-world entity is instantiated. I.e. this time now has a non-abstract counterpart. The same holds for the starting point in time of a timer-slot utilized in a `VCR` that instantiates a `CPointInTime`.

8.4. Incremental Building of Instantiated Object-Oriented Interpretation Trees

Mostly, the content under discussion can be represented using a *partial* instantiated object-oriented interpretation tree. For instance, the time '10 minutes past 8' can be represented using the `time:*{CTime}` structure given in figure 8-2. The objects ***second*** and ***isPM*** are not required

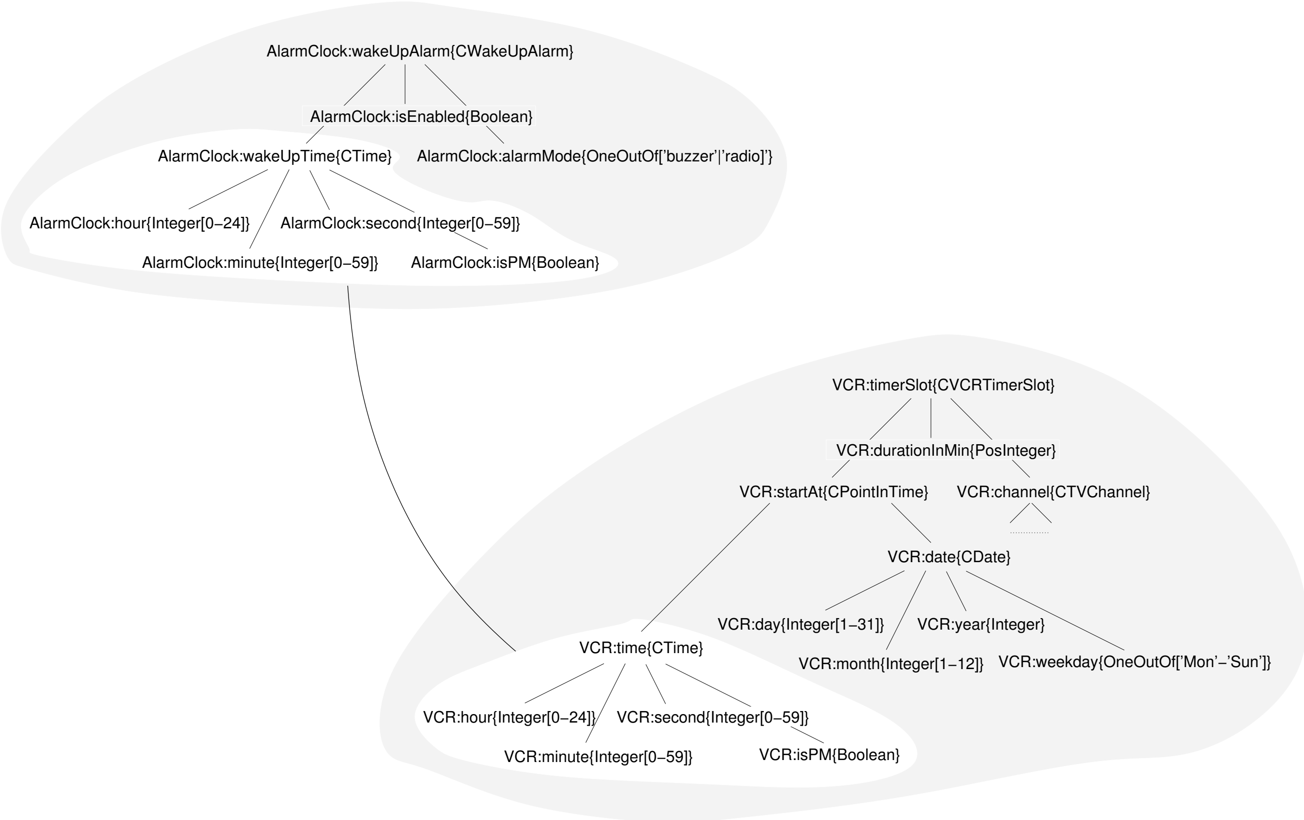

Figure 8-3: Expanded object-oriented interpretation trees of the applications 'alarm clock' and 'video recorder' (VCR), which integrate reused object-oriented interpretation trees from figure 8-2.

to represent this time – for now, we assume that these objects are not set to a default value. The instances of the object-oriented interpretation trees do only contain the necessary DOs, i.e. objects that have been under discussion previously. This includes objects derived from the user input and objects that have been brought into discussion by an application. In the previous example, a user may have uttered *"10 minutes past 8"*. If an application needs to distinguish between a.m. and p.m., it attaches an ***isPM*** object to the `time:*{CTime}` tree together with an indication that this information is needed.
During the discourse, an object-oriented interpretation tree can grow due to DOs introduced through user input or by applications. Single objects, or a small number of objects, from the user input may be merged with an already existing partial object-oriented interpretation tree instance. An application may attach DOs to such a tree during processing. The knowledge to create relationships between DOs is derived from knowledge sources provided by the applications, and consumed and made available to the dialog engine by the domain model (see also section 19 on ontologies).

Therefore, the object-oriented interpretation trees are minimal in a sense that the partial instantiated object-oriented interpretation trees do not carry unneeded DOs. That ensures the more efficient processing of such structures. The parsing and processing of these structures is limited to the relevant content. The memory consumption is lesser compared to complete trees that include superfluous objects, which is especially relevant when the object-oriented interpretation trees are used to exchange several competing hypotheses for each utterance between components.

8.5. Task Objects

In the discussion of dialog objects so far, we focused on the formulation of the operation parameters of applications. However, in order to perform an operation through an application it is generally not sufficient to just specify operation parameters. Therefore, this section introduces the concept of tasks.
Let us illustrate this issue on the basis of a VCR application, note figure 8-3. Assume the dialog system extracted the dialog object `VCR:`***name***`{CChannelName}="Fox"` from the user input. The user might e.g. want to record a broadcast on this channel, switch the VCR tuner to this channel, or obtain a list of recordings targeting on this channel. So how can we determine the operation to perform with the acquired dialog objects?

We introduce task objects that enhance the object-oriented interpretation trees:

8.2 Definition (Task object)**.** A *task object* is a tuple (T, ***o***, *a*). The constituents of a task object are:

- T: the *task* according to table 8-1,
- ***o***: the *dialog object* the task object is associated to, and
- *a*: the *age* of a task object in conformance with definition 8.1.

The notation XYZ is used for task 'xyz', a task object with task XYZ will be represented as TASK= "XYZ". ◇

Since the task is the constitutive part of a task object, the term 'task' will often also be used as a synonym for a task object.
Because of the association of a task object with a dialog object, the task objects appear as special dialog object children in the figures, as shown in figure 8-4. A particularity in the association of

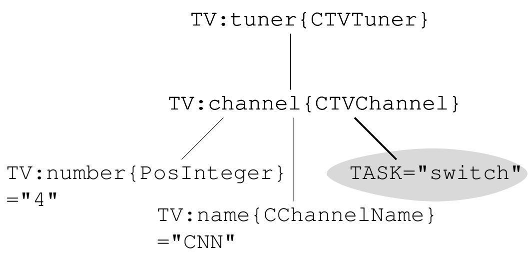

Figure 8-4: A potential representation of the user input *"show channel CNN"*. The TV application inserted the channel-number '4' for the channel 'CNN'. A SWITCH task is associated with the ***channel*** DO.

DOs and task objects is the capability of attaching a task object to a leaf DO. Remember that the status of a leaf DO excludes the existence of child DOs, which again illustrates the difference of DOs and task objects since the task can appear as 'child' of a leaf DO. The extraction of tasks from the user input is discussed in section 21. The attachment process of task objects to DOs is regarded in the sections 21 and 19.5.

For the task-oriented dialogs enabled by DyMaLog, we identified a collection of generic tasks listed in table 8-1 from sample interactions of users with existing dialog systems, e.g. Kellner and Portele [2002], Portele et al. [2003]. The turns from the dialogs are clustered and associated

task	description
INFO	give information about an entity
SWITCH	change a discrete value
ADJUST	change a numerical value
SELECT	select one or more entities
CREATE	create an entity
DELETE	delete an entity
HELP	provide help for an entity or in general
GREETING	greeting and farewell

Table 8-1: Generic tasks.

with tasks noted in the table, mostly one task per user input. For the applications realized in the *Marvin* dialog system (chapter III), it turned out that this collection of tasks is sufficient to operate these applications.
Even though it would be possible to extend the set of tasks when additional applications are prepared for the operation by the *Marvin* dialog system, we prefer a stable set of tasks. And we assume based on the observations during the development of the currently available applications that this set will converge and remain stable (and is already stable). This is a prerequisite for the application of methods inside the dialog engine AIDE based on tasks in an application-independent formulation. The main candidates for applying such methods are the communication management and the output creation to decide on the strategy and form for the reaction directed at the user.

Assume that an o^2I-Tree contains one or more dialog objects associated with tasks. The merger of such an o^2I-Tree and the related task objects will be referred to as o²I-*Tree including task objects*. However, the appendix 'including task objects' will be omitted most of the times, thus the term *object-oriented interpretation tree (*o²I-*Tree)* usually includes associated task objects.

8.6. Hypothesis Structures and Result Structures

The DOs – together with task objects – form the units to build up the o^2I-Trees. The o^2I-Trees themselves are the basic building blocks of two higher-level structures.

8.6.1 Hypothesis Structures. A set of object-oriented interpretation trees (o^2I-Trees) is denoted *hypothesis structure*. Each single hypothesis structure represents an interpretation of input into the dialog system, originating mostly from a user or from an application due to a certain event. While a hypothesis structure passes through the various processing steps, it is enriched.
The components of DYMALOG exchange such hypotheses. Once the user input is analyzed by the NLU, the resulting hypotheses are based on dialog objects, as indicated in figure 7-1 on page 54. Often, such a hypothesis structure carries only one o^2I-Tree. The relation of dialog object, o^2I-Tree, and hypothesis structure is shown in figure 8-5.

8.6.2 Result Structures. A special incarnation of a hypothesis structure is the *result structure*. The object-oriented interpretation trees inside the result structure do generally not

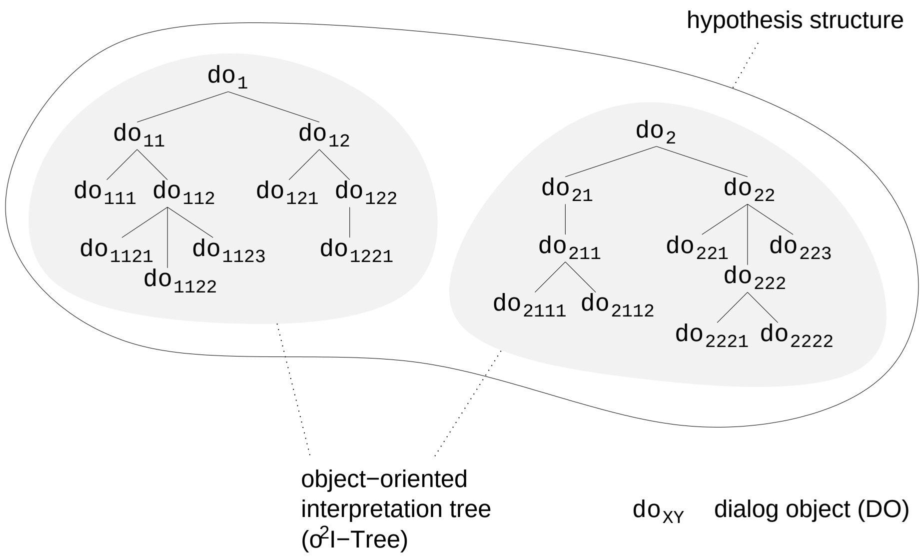

Figure 8-5: Relation of dialog object, o^2I-Tree, and hypothesis structure.

contain full-blown DOs as nodes – the objects restrict themselves to the representation of the information relevant to the application operation parameters. The result structure is used to represent the outcome of the processing of a hypothesis structure by the applications. As such, the number of object-oriented interpretation trees in the result structure usually does not necessarily conform with the number of object-oriented interpretation trees of the input hypothesis structure to the application.

E.g. when the user instructs the TV application to switch to channel `ABC`, the hypothesis structure and result structure can be very similar, both mainly representing the target channel and the related task. If the user requests a list of train connections between two cities in a train timetable application, the hypothesis structure might carry one o^2I-Tree specifying the constraints for the connections. The result structure usually would contain a structure that includes a list of connections fulfilling the restrictions posed through the hypothesis structure. Alternatively, it could contain a set of object-oriented interpretation trees representing the connections.

Figure 8-6 shows a potential result structure derived from an EPG application for the user input *"what is on TV tonight"*. The result structure contains only one o^2I-Tree that lists a set of broadcasts.

After the introduction of the fundamental data representation with dialog objects, task objects, and object-oriented interpretation trees, we will now turn towards the handling of this data by the dialog engine.

9. The Information State Representing the Comprehensive Discourse State

The analog to the Trindi information state is spread over several components of DyMaLog. The components `discourse memory` and `dialog knowledge processing` play a central role; as shown in the outline of the framework in figure 6-1, page 48. While the `discourse memory`

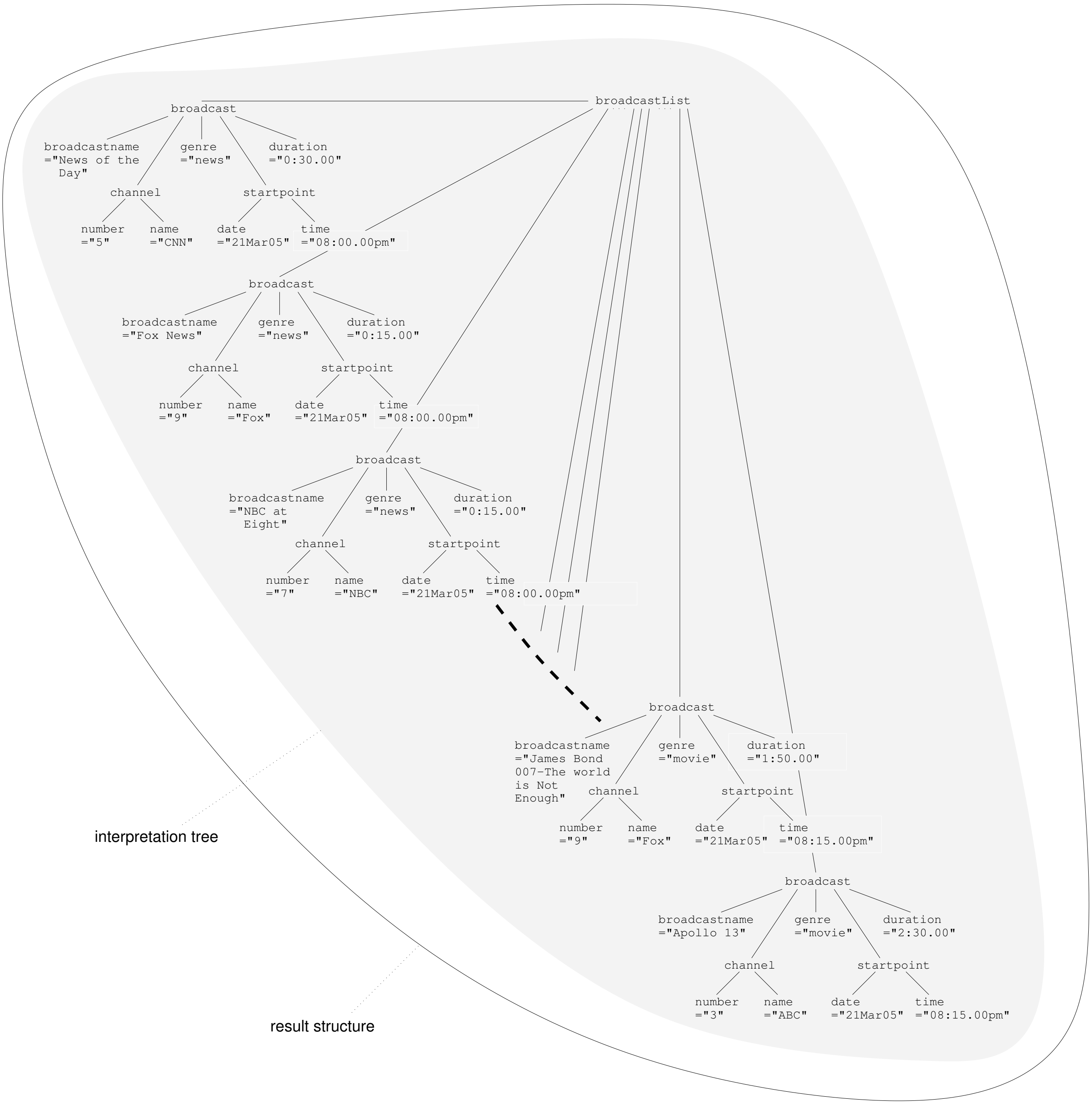

Figure 8-6: Example of a result structure containing a single o^2I-Tree for an EPG. The o^2I-Tree represents the broadcasts listed in table 3-1, page 32. The structures for date, time, and duration are collapsed for readability.

implements major parts of the Trindi information state, the `dialog knowledge processing` embeds the counterparts to significant parts of the Trindi dialog move engine.

Storing discourse related knowledge is concentrated in the `discourse memory` *centrally* for the dialog system (Flycht-Eriksson and Jönsson [2000]). All other components receive discourse related knowledge from the `discourse memory` if required.
Note that the components can still adapt to a certain situation, e.g. the recognizer might change its vocabulary and language model according to the user input expected next, or due to changes in the application setup. Thus a number of components are stateful with respect to the state of the discourse. But none of the components – except for the `discourse memory` – remembers data from the interaction of user and system directly, and thus keeps a history. This does not hold for the applications themselves. Knowledge acquired during the discourse generally changes the status of an application; these applications remember parts of this knowledge.

The `discourse memory` is the common ground of the discourse as perceived by the dialog system, Poesio and Traum [1997]. Presently, the storing does not distinguish between different levels of grounding but uses a binary decision. A future extension of the DOs may include a level of confidence to express (un-)certainty of the content of a DO or its substructures. Skantze [2005] presents a grounding model based on confidence scores utilized in error handling strategies that seems to be an appealing approach adaptable for use within DyMaLog.

Even though the `discourse memory` concentrates on o^2I-Trees, and we focus on this task in the section at hand, the `discourse memory` stores additional content. For instance, it stores the latest result structure. The result structure is used to identify lists, and then resolve references to entities in these lists, see also section 16. However, the result structures are not transferred into a special representation by the `discourse memory`.

9.1. Organization of the Memory

The object-oriented interpretation trees which store the processing result of an input event in the dialog system, including the result of the `recognition`, the `analysis` of user input, and the processing by an application, are not stored directly by the `discourse memory`.
The date storage is organized as follows.

9.1 Definition (Temporal Dialog Object Evolution)**.** The *temporal DO evolution* is a sequence of k modified dialog objects

$$\left(o_{(1)}, \ldots, o_{(k)}\right) \tag{9.1}$$

with $o_{(i)}$ being *derived* from $o_{(i+1)}$, $1 \leq i \leq k-1$. I.e. $o_{(i)}$ is a dialog object introduced into DyMaLog before $o_{(i+1)}$ in terms of turns. Furthermore, the DO $o_{(i+1)}$ is a modified version of the DO $o_{(i)}$ with $o_{(i)}$'s instance and class remaining constant.
The *modified dialog objects* are dialog objects with each child replaced by a reference on its temporal DO evolution. ◇

Among other things, definition 9.1 allows changes in the domain and children of the DO $o_{(i)}$. Therefore, the temporal DO evolution can handle the usage of the same dialog object over different applications.
Notice that the definition does not require a consecutive sequence of objects, one DO in the sequence for each turn. Referencing the temporal DO evolutions belonging to the children of a dialog object allows us to use the same temporal DO evolution in different contexts and different turns.

Analogous to the object-oriented interpretation trees, temporal DO evolutions can be used to build up trees. These trees carry as nodes temporal DO evolutions instead of dialog objects. We refer to these trees as *object-oriented interpretation trees with temporal evolution*.

Figure 9-1 shows an exemplary except of the `discourse memory` focusing on one temporal DO evolution. Two parental DOs use this evolution, and the children constellation for this

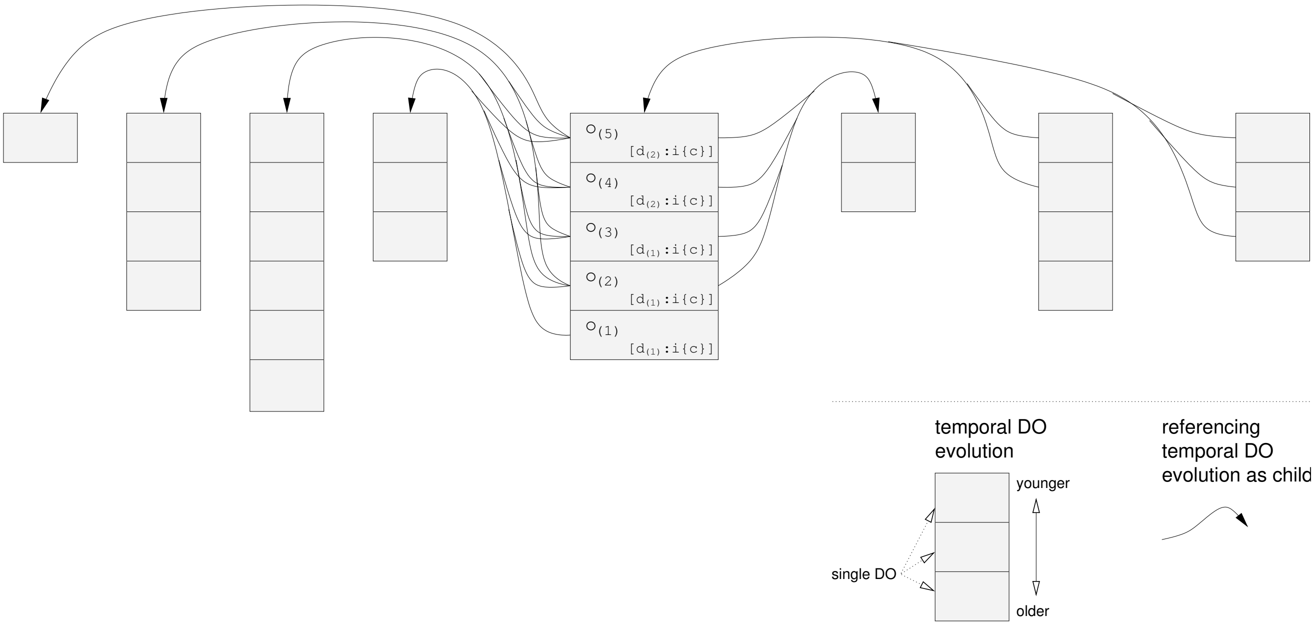

Figure 9-1: Excerpt of the `discourse memory` focusing on a single temporal evolution containing five changes of a dialog object during the dialog of user and system, including changing domains $\mathtt{d}_{(k)}$. The instance *i* and class `c` for the particular evolution remain constant.

particular evolution changed over time. Furthermore, the domain for the considered DO evolution changes from domain $\mathtt{d}_{(1)}$ to $\mathtt{d}_{(2)}$.
A more concrete example is given in figure 9-2. It sketches a section of the `discourse memory` and is centered around a representation of *time* given by the DO $t_{(i)}$. The storing after the first input simply reflects the hierarchical o^2I-Tree from the hypothesis structure, the DOs from the o^2I-Tree are trivial temporal DO evolutions wrapping only a single DO. Thus, the relations shown between the temporal DO evolutions build up a hierarchical structure being an exact mapping of the DO structure given by the o^2I-Tree. Note that the excerpt shown in the figure focuses on the *time*. The addressed structures below the *time* object are shown, ***hour***, ***min***, and ***pm***, but the DOs at higher-levels are reduced to the line of direct ancestors, e.g. the sibling ***date*** below the ***startpoint*** is omitted.
The second input *"show the movies starting at eight thirty"* also carries relevant information with respect to the section of the memory shown in the figure. The 'hour' part of the *time* remains constant, but the 'minute' entity is updated. We assume that *pm* is given by a default from the application. Figure 9-2 shows that the structure itself does not change. The ***hour*** is not modified, thus we can 'take over' from the previous input. The change in the ***min*** is reflected by adding a DO with *min*="30" derived from $m_{(1)}$ into the affected temporal DO evolution.
While dealing with the third input *"record the movie on Fox"* (assume that it complies to the previously defined time), the already existent `EPG:`*startpoint* with its substructure – and thus the inferior `EPG:`*time* – is ported into the `DVDRec` domain. Since the affected DOs are in principle

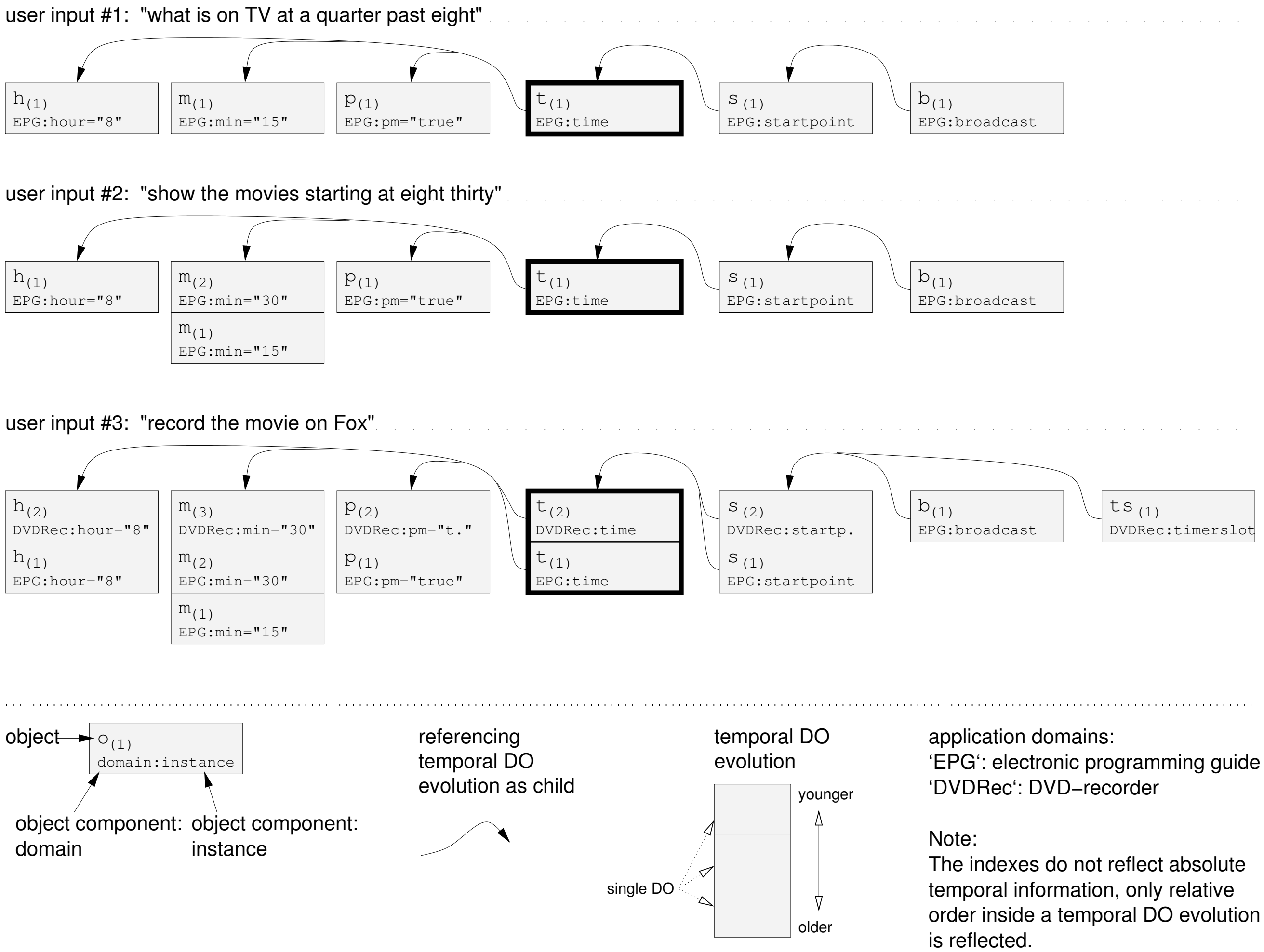

Figure 9-2: Excerpt from the `discourse memory` showing the development over time on the basis of three user inputs. The excerpt concentrates on the DO $t_{(i)}$ representing a time. Further objects are omitted, e.g. the date that would canonically be a part of the *startpoint* point in time, or the genre that is given in input #2. While the addressed children being part of the time are shown, the higher-levels are reduced to the direct line of ancestors of the time DO.

the same except for the updated domain, we continue the existing temporal DO evolutions and update these by appending an updated DO with the new domain to reflect the change in the domain. E.g. $\boldsymbol{h}_{(2)}$ is derived from $\boldsymbol{h}_{(1)}$ by changing the domain to `DVDRec`. Note that after user input #3 the timer-slot DO $\boldsymbol{ts}_{(1)}$ from domain `DVDRec` points to the temporal DO evolution containing $\boldsymbol{s}_{(i)}$, in the domain `EPG` the broadcast object $\boldsymbol{b}_{(1)}$ points to this evolution.

Even presently not realized in the framework, we want to remark that this structure would allow the user to refer to information previously introduced in another application: if a user enters *"I want to schedule a recording at the previous time"*, the EPG may restore the time `8:15.00pm`, which was originally introduced in the `EPG`, and is now used in domain `DVDRec`.

Most of the time we will be interested in the *most recent DO* contained in the temporal DO evolution. According to definition 9.1, the most recent DO is the object with the highest index, for sequence (9.1) object $\boldsymbol{o}_{(k)}$. Similarly the *i-th recent DO* in a temporal DO evolution is the i last dialog object in the related DO sequence, in case of definition 9.1 the $k-i+1$-th object $\boldsymbol{o}_{(k-i+1)}$.
Another selection of a dialog object from a temporal DO evolution is based on the age or creation time respectively of a DO. The *dialog object representing the state of a temporal DO evolution at some point in time* p or *a certain turn* t in the situation of definition 9.1 is given by $\boldsymbol{o}_{(i_t)}$ with

$$i_t = \max\Big\{i \in \{1,\dots,k\};\, \boldsymbol{o}_{(i)}\texttt{[}\boldsymbol{age}\texttt{]} \text{ is before or equal to } t\Big\}$$

whereas – if p is given and t is undefined – t is defined as the last turn before the point in time p.

9.2. Updates of the Discourse State

Once the `dialog knowledge processing` receives an o^2I-Tree identified by the `hypothesis selection` for the integration into the `discourse memory`, this tree is handed over to actually start the storage process.

9.2.1 Integration Procedure. The integration process is performed bottom-up, starting from the leaf DOs and then moving towards the root dialog object. The basic steps to be performed for the integration into the `discourse memory` are:

> Let $\boldsymbol{t}$ be an o^2I-Tree to be integrated into the `discourse memory`. The integration algorithm 9.1 disassembles the o^2I-Tree and performs the integrations one by one for each node of the tree representing a dialog object[11]. We have to distinguish three cases.
> If for a DO $\boldsymbol{o}$ in $\boldsymbol{t}$ and temporal DO evolution `e` in the `discourse memory` exists such that $\boldsymbol{o}$ is the most recent object in `e` (case 1), or $\boldsymbol{o}$ can be derived from the most recent object (case 2), then $\boldsymbol{o}$ is merged into `e` if required. Otherwise a new temporal DO evolution containing $\boldsymbol{o}$ is added to the `discourse memory` (case 3).
>
> Updates of the memory due to the integration of DOs do not mutually affect the integration of other objects of the same tree. The only exception is the usage of modified and newly created temporal DO evolutions as replacement of children of a DO by the related temporal evolutions.

[11]Algorithms are presented in pseudo-code oriented at the programming languages `C` or `Java` and script languages like `perl`. The symbol '`//`' is used to embed comments.
The algorithms presented here are not literally realized in the *Marvin* dialog system. E.g. for simplicity, optimizations to speed up the computations are omitted since the readability of the procedures generally suffers from that.

Algorithm 9.1: integration of an o^2I-Tree t into `discourse memory`.

// *Remark: Subfunctions are defined on the next pages.*

Data: o^2I-Tree t
Result: nodes of t integrated into `discourse memory`

```
foreach dialog-object o ∈ t (bottom-up) do
    // since the tree t is processed bottom-up, the nodes below o are already processed
    modified DO o_modified := replace_children_with_temp_DO_refs(o);
    temporal DO evolution e := return_origin_temp_DO_ref(o);
    if e ≠ ∅ then
        // o_modified can be integrated into an existing temporal DO evolution
        DO o_most recent := most recent modified DO in e;
        if o_modified=o_most recent then
            // Case 1.
            // i.e. o is already contained in the discourse memory
            // (o_modified=o_most recent is the most recent DO in e) → thus no action is required
        else
            // Case 2.
            // i.e. o_modified is a modification of o_most recent
            append o_modified as most recent DO to e;
        end
    else
        // Case 3.
        // o cannot be associated with an existing element from the discourse memory
        e = new temporal DO evolution containing the DO sequence (o_modified) of length 1;
        insert e into discourse memory;
    end
end
```

Function 9.2 `replace_children_with_temp_DO_refs`(DO o)

Data: DO o: the children of this DO will be replaced by the corresponding temporal DO evolutions
Result: modified DO o_{modified}: modified DO with temporal DO evolutions as children

```
modified DO o_modified := copy of DO o without child objects;
foreach child c ∈ o[children] do
    locate temporal DO evolution e in discourse memory containing c as most recent DO;
    // insert counterpart of c in o_modified
    append reference to e at o_modified;
end
return o_modified
```

Function 9.3 `return_origin_temp_DO_ref(DO o)`

Data: DO *o*: search for the corresponding temporal DO evolutions for this DO
Result: temporal DO evolution `e`: temp. DO evolutions corresponding to *o*, ∅ if no match was found

if exists temporal DO evolution `e` such that *o* is derived from the most recent DO of `e` **then**
  return `e`
else
  return ∅
end

To illustrate the integration algorithm with the example of figure 9-2, case 1 corresponds to the handling of the hour $h_{(1)}$ between user input #1 and input #2. The first input specifies *hour*="8", which is also addressed in input #2. Thus the hypothesis structure derived from the user input #2 would also contain *hour*="8". During the integration of input #2 into the `discourse memory`, there is no need to introduce an additional *hour* DO since the existing DO already represents the newly obtained knowledge.
In case 2, the DO representing the minute part of the *time*, $m_{(i)}$, needs to be updated from $m_{(1)}$="15" to $m_{(2)}$="30" from input #1 to input #2. However, if only the value of a DO is changed, we may use the already existent temporal DO evolution containing $m_{(i)}$ as done in figure 9-2. A DO is appended to the evolution as most recent DO reflecting the change in the value.
Case 3 requires the creation of a new temporal DO evolution that is independent of the already existent evolutions in the `discourse memory`. When new content is entered into an empty `discourse memory`, exclusively case 3 is applicable. In our example, also from user input #2 to input #3 a new temporal DO evolution is introduced carrying the DO $ts_{(1)}$.

During the integration of an o^2I-Tree into the `discourse memory`, it is not searched for temporal DO evolutions inside the memory from which the dialog objects can be derived. Instead, during the construction of the object-oriented interpretation trees the `dialog knowledge processing` tags dialog objects from these trees with an identifier, which connects these objects to temporal DO evolutions if possible. This is especially possible if objects from the dialog memory are integrated with the analysis result for a new user input. Also, the derivation of a new dialog object from a temporal DO evolution takes place before the integration of the DO into the memory, e.g. while merging an object from the user input and a DO from the memory (like updating the value), which leads to a derived object.

9.2.2 Exceptions from the Integration. During the integration process, o^2I-Trees that reside inside the `system` domain experience special treatment compared to the integration of an o^2I-Tree of a custom application. This is especially applicable to the navigation inside the discourse (section 15). Navigation instructions from the user demand changes of the `discourse memory` restricted on existing knowledge inside the component. The memory does not integrate additional information from the navigation instruction itself.

Compared to the Trindi framework, the update of the `discourse memory` is an unequivocally determined process in DyMaLog while the update of the Trindi information state (with the selection of the appropriate rules) rather relates to the integration of the previous discourse and the hypotheses in the `dialog knowledge processing` together with the following selection in the `hypothesis selection`.

9.3. Reconstruction of Object-Oriented Interpretation Trees from the Discourse Memory

Of course we have to provide means for the access to the `discourse memory` once elements are inserted. The integration of the previous discourse and the interpreted current user input heavily relies on these means.
To retrieve the o^2I-Tree representing the status of this tree at a certain point in time below a certain temporal DO evolution let `e` denote the evolution of interest. W.l.o.g. we can restrict ourselves to the status at a certain turn t instead of an arbitrary point in time. Algorithm 9.4 describes the process to rebuild the o^2I-Tree below `e` at a certain turn t.

Algorithm 9.4: reconstruction of o^2I-Tree from temporal DO evolutions

```
Data: temporal DO evolution e
Result: o²I-Tree t at a certain turn t

// recursive call builds DOs according to definition 8.1,
// thus an o²I-Tree can directly be derived
DO o_root := get_DO_for_temp_DO_evo(e,t);
o²I-Tree t := o²I-Tree with root DO o_root;
```

The algorithm can be described as follows: For each temporal DO evolution being 'relevant' for the tree spanned below `e`, determine the modified DO representing its state at turn t. For these

Function 9.5 `get_DO_for_temp_DO_evo`(temporal DO evolution `e`, turn t)

```
Data: temporal DO evolution e: extract state from e at turn t, i.e. the complying DO
Data: turn t: turn of interest
Result: DO o: DO representing state of e at turn t

modified DO o_modified := modified DO representing the state of e at turn t;
DO o := copy of o_modified without references to temporal DO evolutions;
foreach temporal DO evolution e_referenced referenced by o_modified do
    // start recursion
    DO o_from evo := get_DO_for_temp_DO_evo (e_referenced,t);
    append o_from evo as child to o;
end
return o
```

modified DOs compute the children DOs, which can be derived from the references to temporal DO evolutions. The modified DO together with the computed child DOs forms a classic DO. The set of classic DOs forms the nodes of the o^2I-Tree.

9.4. Utilizing the Discourse Memory

As we discussed before, the `discourse memory` serves as the central memory instance inside Aide. The organization of the content inside the `discourse memory` reflects that the interaction of a user with the system leads to a sequence of hypothesis structures. Thus, when the memory is inspected on the level of hypothesis structures, it contains a linear sequency. In principle, this

also holds on the level of o^2I-Trees that build up the hypothesis structures. This does not hold for the sub-units in an o^2I-Tree, the DOs, as they are stored in the temporal DO evolutions.
The content of an hypothesis structure is preserved completely when it is integrated into the discourse memory. This offers the opportunity to provide alternative views on the discourse history. However, that might require to develop algorithms that derive these alternative views. For instance, a view that partitions the discourse into sub-dialogs and that relates these sub-dialogs may operate mainly with the help of the *age* of the DOs. The outcome of such a view can be a hierarchical structure of the discourse.

Up to now, the main customer of the `discourse memory` is the `dialog knowledge processing` in order to integrate the discourse history with the latest user input and application processing result. Presently, the `communication management` bases its decisions purely on the hypothesis structure, see section 17. It does not make use of the `discourse memory` directly. Yet, more advanced strategies may rely on additional information from the previous discourse, see also section 17.6.
Apparently, the `discourse memory` also forms the basis to navigate inside the discourse, as it will be discussed in section 15. For this task, the content of the discourse memory is perceived as a linear sequence of hypothesis structures. This results in navigation operations similar to navigations in a web-browser. Another interesting option for the future would be the navigation on the level of sub-dialogs, given the `discourse memory` is capable of providing an appropriate view as explained above.

10. Evaluation of Interpretations in the Context of the Discourse

A major issue in dialog processing is the integration of the hypotheses representing the current user input with the knowledge acquired during the discourse (e.g. Alexandersson and Becker [2001], Cristea et al. [1998], Bunt [1999], see also Grosz [1996]). The present systems explicitly formulate for each accessible application how the integration of the operation parameters has to be performed. A broad field of research in the linguistic area is the resolution of ellipses and anaphora.

In this section, we present the approach to integrate application operation parameters by the application-blind DyMaLog. It heavily depends on the representation of knowledge using object-oriented interpretation trees as introduced in section 8, which in part takes over the resolution of ellipsis.

Integrating the hypothesis structure representing an hypotheses of the current user input isolated from the context and the previous discourse is a complex process. We will discuss the integration decomposed in a number of steps. The integration is inspired by the 'overlay' operation to merge hierarchical structures (Alexandersson and Becker [2003]), e.g. as applied in the SmartKom discourse model.
A flexible model for 'context resolution' – including the integration of current user input and previous discourse – is presented in Filisko and Seneff [2003]. However, the dialog systems utilizing the resolution component are single-domain applications, and 'all domain-dependent knowledge and constraints are developer-specified in external tables'. Thus, the dialog systems closely integrate knowledge from the applications into the dialog system, but the components are prepared for utilization in different domains.

Remember that we do not restrict the processing to one single hypothesis but allow competing hypotheses (section 7). The integration process is a major source for alternative interpretations of the input, as we will see later. Allowing alternative interpretations for the user input up to the `hypothesis selection` has two implications:

1. all interpretations must find the same conditions when combined with the previous discourse, and
2. the integration into the `discourse memory` can actually only be performed in a separate step after the selection process.

DYMALOG addresses the first point by updating or creating interpretations as the previous discourse is combined with the interpretation of the user input, without modifying the state of the `discourse memory`. This section focuses on this merger. Section 9.2 dealt with updating the `discourse memory` with the knowledge from the best interpretation.

The merger of an interpretation of the current user input with the previous discourse has its counterpart in the TRINDI update rules. However, DYMALOG postpones the selection of the best 'set of update steps' to allow competing views on the user input and a modularized design of the dialog engine.

10.1. Integration of Hypothesis Structures with the Previous Discourse

The procedure for the integration of a hypothesis structure h with the previous discourse as represented in the `discourse memory` can be divided into 3 steps:

1. Identify the temporal DO evolutions contained in the `discourse memory` relevant for the integration with h. For these evolutions, derive object-oriented interpretation trees that will be actually deployed during the integration.
2. Consider now the set of object-oriented interpretation trees T obtained during the first step and included in h. Applying knowledge on the relationships of DOs specified through the ontologies inside the `domain model`, the root-DOs from the trees in T are related by newly created hypothesis structures.
   Note that these relations are not necessarily unique, i.e. for a set of given root-DOs a set of different ways to combine these DOs may be obtained from the `domain model`.
3. The hypothesis structures from the previous step are made out of root-DOs from the first step. The object-oriented interpretation trees belonging to these root-DOs are now integrated below these root-DOs. Thus, the object-oriented interpretation trees in the hypothesis structures from step 2 are combined with the trees considered in step 1 and delivered by h.
   In general, the integration of object-oriented interpretation trees below the root-DOs is ambiguous.

The integration procedure realizes requirement 4.1(1) formulated in the criteria catalog.

The figures 10-2, 10-3, and 10-4 illustrate the three steps described above on the basis of a basic example. In the setting of this example, a user already interacted with the dialog system and just entered *"I want to leave from Paris Gare du Nord"*. This input is processed by the dialog system up to `dialog knowledge processing`, figure 10-1, and will be integrated with

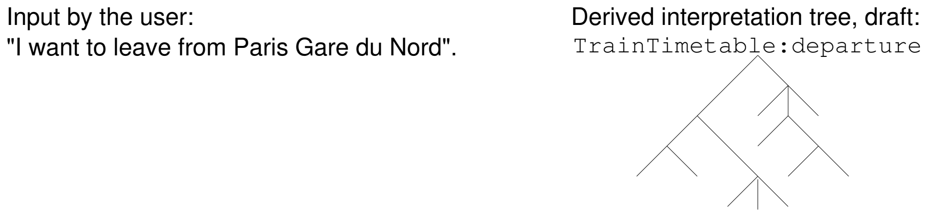

Figure 10-1: Draft of the o^2I-Tree derived from *"I want to leave from Paris Gare du Nord"* for a train timetable application, associated domain `TrainTimetable`.

the previous discourse next.
Step 1, figure 10-2, shows a snapshot of the `discourse memory` as perceived before the integration. The o^2I-Tree related to the input *"I want to leave from Paris Gare du Nord"* together

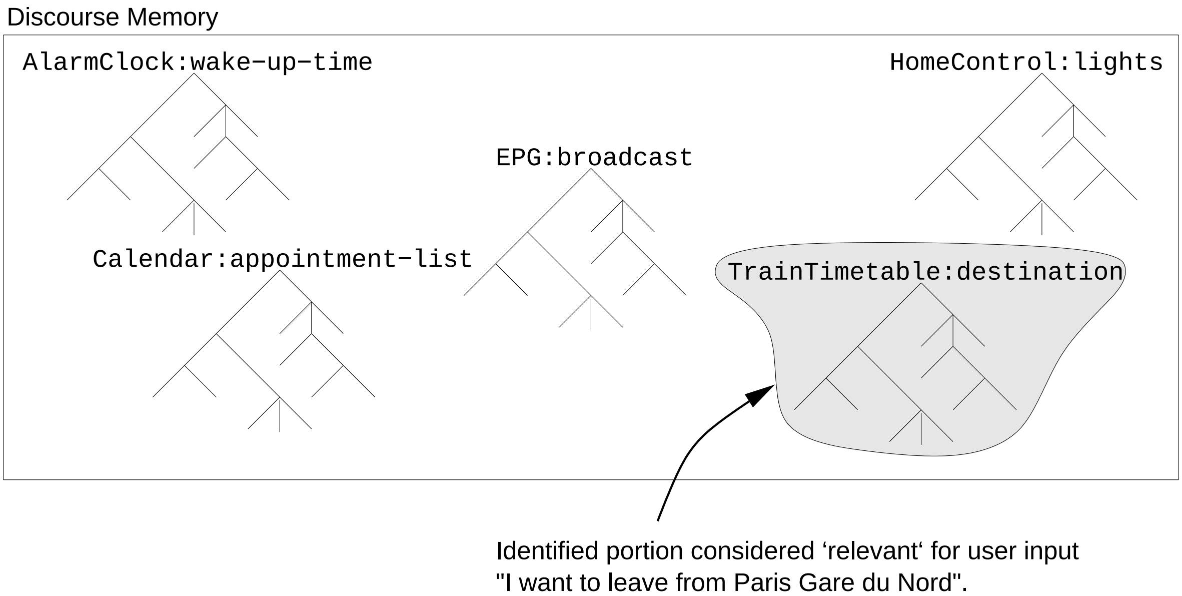

Figure 10-2: Integration of hypothesis structures with the previous discourse: *step 1*, identification of relevant content in the `discourse memory` for the user input. Snapshot of the `discourse memory` (fictive). Instead of a detailed view on the temporal DO evolutions, reconstructed o^2I-Trees are drafted to reproduce the content of the `discourse memory`. As the root-DOs are the only DOs required to illustrate the basic idea of the integration algorithm, the structures below the root-DOs are indicated only.

with the age of the elements in the memory currently define the conditions for the entities being relevant for the o^2I-Tree derived from the user input. Taking as basis the o^2I-Tree in figure 10-1, the elements in the tree with root `TrainTimetable:`*`destination`* are selected.
During the second step, figure 10-3, the `domain model` creates a relation of the root-DOs from the hypothesis structure that represents the user input (`TrainTimetable:`*`departure`*) with the root-DOs of structures that were identified during the first step, here: `TrainTimetable:`*`destination`*. The `domain model` utilizes ontologies delivered by the applications to establish a relation between the root-DOs. A trivial result of the relation process is obtained by treating the objects as independent units, i.e. no relation is created at all. In the given setup, we assume

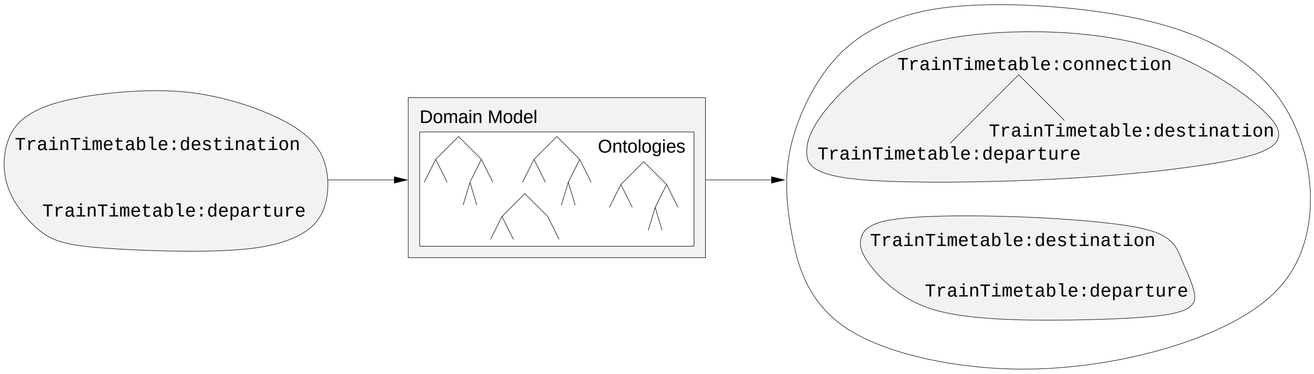

Figure 10-3: Integration of hypothesis structures with parts of the discourse history: *step 2*, relation of root-DOs according to ontologies in the `domain model`. The root-DO of the o^2I-Tree belonging to the current user input, `TrainTimetable:`***`departure`***, needs be related with the root-DOs of the related structures from the `discourse memory`, `TrainTimetable:`***`destination`***. In the given example, two alternative relations are found: one combines the given root-DOs, the second treats the objects as independent units.

that the `TrainTimetable:`***`departure`*** and `TrainTimetable:`***`destination`*** can be related by a common parent `TrainTimetable:`***`connection`***.
Figure 10-4 shows the result of step 3. For the example, embedding the trees from the user input

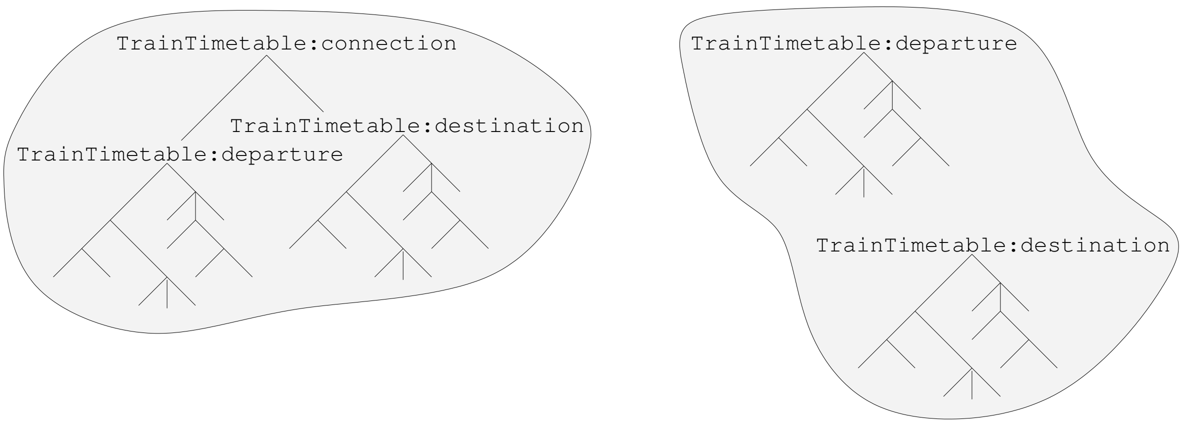

Figure 10-4: Integration of hypothesis structures with the previous discourse: *step 3*, embedding the trees from the hypothesis structure and `discourse memory` into the relations derived in step 2. The root-DOs from the o^2I-Trees in the hypothesis structure representing the user input and from the `discourse memory` serve as anchor points to integrated the trees below these root-DOs into the relation structures obtained in step 2. The two results shown here both illustrate the simplest case: the tree structures can just be appended below the root-DOs, no overlap during the integration with existing structures or other trees to be integrated needs to be considered.

and step 1 into the structures resulting from step 2 is straightforward. The root-DOs are different leaves in the relating structure, the trees belonging to the root-DOs can simply be appended to these leaves.

After sketching the basic idea, the core of the integration algorithm is split into a series of single units and presented in more detail in the remainder until section 10.8. Section 10.9 extends the core method to allow sharing of discourse information over the boundary of single applications. The subsequent subsections round up the discussion around the integration of structures.

For the description of the core algorithm, we first illustrate the idea of each single step using an example. Then the formal description is given. For the illustration, we stick to a single example that is developed further from step to step. Thus, for a more detailed overview than the 3-step glimpse given in this section one might rely on the introductive examples for each single step.

10.2. Anchor of the Integration Algorithm

Algorithm 10.1 is the entry point to the more formal presentation of the previous 3-step description. It structures the algorithm according to the three steps on the most general level. The functions 10.2 'get_o2I-trees_from_discourse', 20.1 'relate_DOs_using_o2I-tree_lists', and 10.4 'integrate_trees_related_to_root-DOs' hide complex operations on the algorithm entry level. Function 10.2 'get_o2I-trees_from_discourse' and 10.4 'integrate_trees_related_to_root-DOs' are discussed in more detail next. Function 20.1 'relate_DOs_using_o2I-tree_lists' is covered in more detail in section 20 dealing with the `domain model` (page 173).

The hypothesis structure h processed by algorithm 10.1 is one of the hypotheses being delivered to the `dialog knowledge processing`. The result of the algorithm is a set of hypothesis structures H. Each hypothesis structure contained in H represents an interpretation of the user input, i.e. a hypothesis for this input.

Processing the user input up to the point where the result is put into the context of the discourse yields a set of hypotheses. On the one hand, these 'non-discourse related' hypotheses are directly passed through to the `hypothesis selection`. On the other hand, they serve as a basis to actually integrate with the previous discourse. I.e. if at least one hypothesis is derived from the user input, the `hypothesis selection` collects hypotheses that integrate parts of the discourse, but also hypotheses that do not consider the discourse. Since $h \in H$, the integration result is never empty, thus $H \neq \emptyset$ always holds.

I.e. the outcome of algorithm 10.1 includes one hypothesis that does *not* incorporate temporal DO evolutions from the `discourse memory`, namely h being an interpretation of the user input without any context from the discourse – remember that the hypotheses obtained from the processing before the integration with the previous discourse takes place are already hypotheses representing the user input, even without integrating knowledge from the previous discourse.

10.3. Alternative Spans of Relevance from the Discourse History

A priori, the number of turns contained in the discourse history, which should be integrated with the present user input, is not predetermined. This number generally depends on the content of the user input. The dialog system that realizes DYMALOG allows limiting the number of turns from the history to be integrated with the hypotheses obtained from the present user input.

To avoid a binary decision – either no history elements are merged with the hypotheses for the present user input, or the maximum number of turns from the history allowed by the corresponding parameter – subsets of the maximum number of turns are also considered. We start from the o^2I-Trees that belong to the present user input without the integration of knowledge from the history. Successively, preceding turns from the discourse history are taken into account until a

Algorithm 10.1: integration of hypothesis structure and previous discourse

Data: hypothesis structure h: user input interpretation without context from previous discourse
Data: parameters $\mathtt{p}_{\mathtt{discourse\ memory}}$: parameters for identification of relevant content from `discourse memory`
Result: set of hypothesis structures H: competing results of integrating h with the previous discourse

set of hypothesis structures $H := \{h\}$;
set of domains `D` := domains from DOs in h;

// Step 1: get 'relevant' object-oriented interpretation trees from `discourse memory`
set of object-oriented interpretation trees T
:= `get_o2I-trees_from_discourse(D,`$\mathtt{p}_{\mathtt{discourse\ memory}}$`);`

// add the o^2I-Trees from h to the structures derived from the memory
$T = T \cup \{o^2I\text{-Tree } t; t \text{ is contained in } h\}$;

// subsets of T are considered: all o^2I-Trees in T whereof the root-DO's age
// is between 1 and n are used in the generation of hypotheses (n is varying)
integer `counter` := 0;
set of object-oriented interpretation trees $T_0 := \emptyset$;

while $T_{\mathtt{counter}} \neq T$ **do**
  `counter` = `counter`+1;
  // get all o^2I-Trees in T with the age of the root-DO between 1 and
  // `counter`; actually, the set obtained in the previous iteration is combined with
  // the set of trees with root-DOs of age `counter`
  set of object-oriented interpretation trees $T_{\mathtt{counter}}$
  $:= T_{\mathtt{counter}-1} \cup \{o^2I\text{-Tree } t;$ for root-DO o_{root} of $t \in T$, $o_{\text{root}}[\mathit{age}] = \mathtt{counter}$ holds$\}$;
  // Step 2: consult the `domain model` to establish a relationship between root-DOs
  set of DOs $O_{\text{root-DOs}} := \{$DO o_{root}; o_{root} is root-DO of $t \in T_{\mathtt{counter}}\}$;
  set of hypothesis structures $H_{\text{related root-DOs}}$
  := `relate_DOs_using_o2I-tree_lists(`$O_{\text{root-DOs}}$`);`
  // Step 3: integrate the trees from T_i belonging to the root-DOs in $H_{\text{related root-DOs}}$
  foreach hypothesis structure $h_{\text{related root-DOs}} \in H_{\text{related root-DOs}}$ **do**
    set of hypothesis structures $H_{\text{integrated hypothesis}}$
    := `integrate_trees_related_to_root-DOs(`$h_{\text{related root-DOs}}$, $T_{\mathtt{counter}}$`);`
    $H = H \cup H_{\text{integrated hypothesis}}$;
  end
end

given limit is reached. Whenever an additional turn is added, basically the three steps outlined in section 10.1 are executed. In algorithm 10.1, this mechanism is reflected by the 'while'-loop, which encloses step 2 and 3.
Of course, this procedure is a source of rivaling hypotheses.
To illustrate this procedure, assume that at most 3 turns stored in the history may be integrated with the present user input. The objects directly derived form the user input carry the age 1, thus elements of age 2 to 4 from the history are considered. In a first iteration, only the elements directly related to the present user input are relevant. I.e. the hypotheses generated from the user input are directly taken as hypotheses for the user input. Next, elements of age 2 from the history are integrated with the hypotheses from the user input. Therefore, the resulting hypotheses for the user input comprise knowledge from the immediately preceding turn. During the next iteration, the hypotheses for the user input are integrated with elements of age 2 and 3 from the history. The hypotheses generated during this iteration include the last and second last turns from the discourse history. Finally, the elements of age 2 to 4 from the history are integrated analogously to build hypotheses with a longer historical knowledge for the user input.

It should be noted that the age of a root-DO of an o^2I-Tree is updated if this root-DO is also the root-DO of an o^2I-Tree contained in the selected hypothesis. E.g., if the user accesses an EPG, an o^2I-Tree could represent the restrictions being used to filter entries from the EPG. Assume that a *broadcastList*-object is the root-DO of the o^2I-Tree. If the user updates the restrictions and the *broadcastList* remains a root-DO of the selected hypothesis structure, then the age of *broadcastList* is set to 1[12].
Among other features, the age of the root-DO is used to judge if the elements from the history are relevant for the integration with the hypotheses derived from the present user input.

10.4. The Accompanying Example: Browsing the Electronic Programming Guide

To illustrate the single steps of the integration algorithm, we will follow an example in the EPG domain. For the sake of the example assume that the user already has interacted with the system for some time and that the most recent user input is *"show details on the movie at ten with Pierce Brosnan"*. The input is processed up to the point where the integration with the previous discourse actually begins.

Figure 10-5 shows a section of the ontology for an EPG application that is used during the example. It allows to represent a list of broadcasts. When a user defines operation parameters, these may define the constraints to query a database containing the broadcasts. The broadcast list element from the ontology can then be used to define the format that carries the queried broadcasts.
A hypothesis structure representing the input *"show details on the movie at ten with Pierce Brosnan"* is shown in figure 10-6. Of course, the processing could also generate alternative hypotheses, but we limit our inspections on the given hypothesis structure made up of one o^2I-Tree according to the EPG ontology.

Since the example accompanying the description of the single steps in the integration algorithm is intended to make the fundamental ideas clearer, it does not cover every tiny bit handled by the algorithm nor will it perform the complete processing for every hypothesis, which is derived during the processing. The example will concentrate on the most interesting cases to understand

[12]An age of 1 relates to the presently processed turn. A DO, which is introduced or updated due to the processing of the present user input, obtains this age.

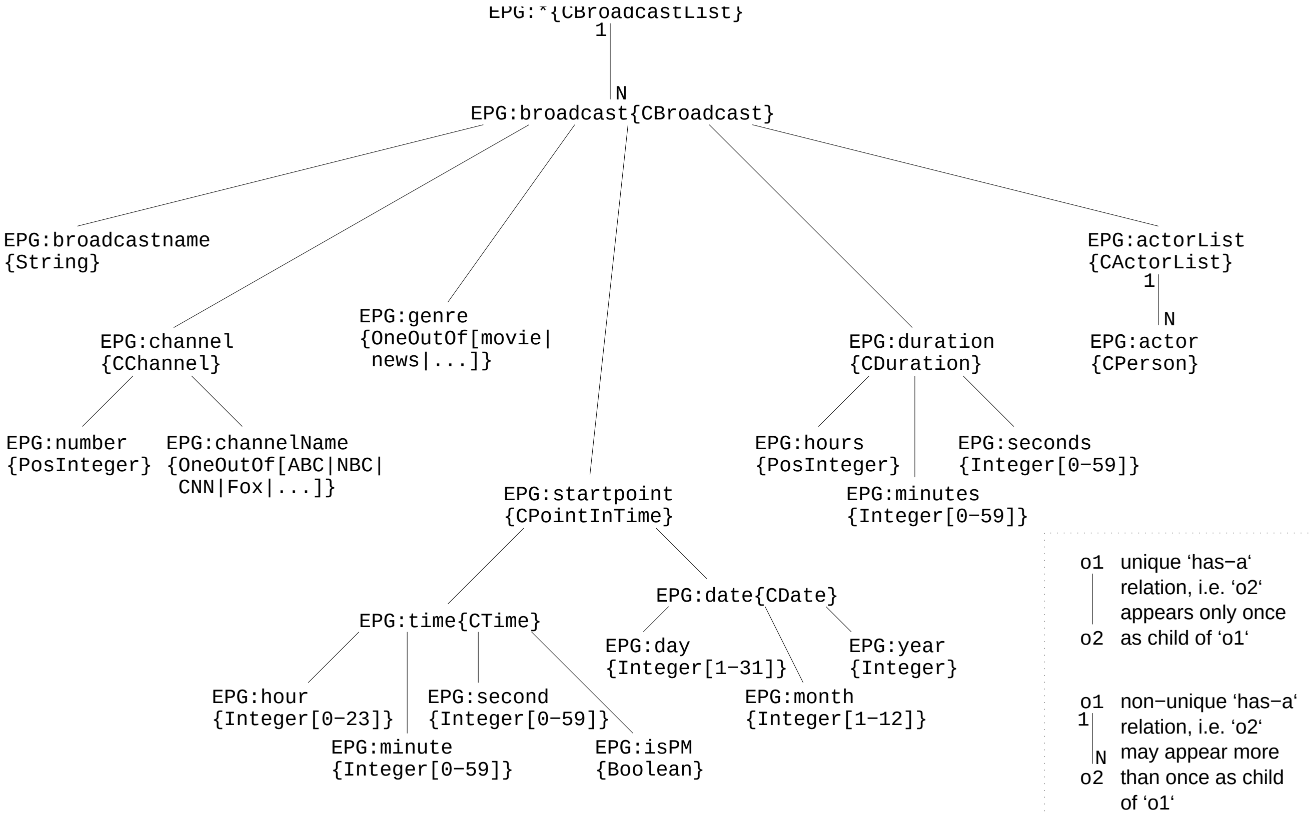

Figure 10-5: Exemplary ontological model for the representation of a list of broadcasts in an EPG application. 'has-a' and 'is-a' relations build up ontological structures. These two relations can be collapsed into pure 'has-a' structures being sufficient for deployment inside the `domain model` (section 19.3). The usage of 'has-a' only relations improves readability and is shown in this figure. Note that the o^2I-Trees also connect DOs via 'has-a' relations. Non-relevant parts with respect to the example are omitted, e.g. an entry for a short description or a picture from the broadcast might be of interest.

the procedure, but the illustration is not mandatory for the understanding of the algorithm.

10.5. Extraction of Discourse Knowledge Related to the Hypothesis

Now we turn towards the first step of the integration algorithm, the extraction of 'relevant' object-oriented interpretation trees from the discourse in the `discourse memory`.

Example 'EPG'. During the interaction of a user and the dialog system, the `discourse memory` keeps track of the discourse. In a first step, function 10.2 'get_o2I-trees_from_discourse' loops over the complete history and evaluates for each temporal DO evolution if its most recent DO contained is a root-DO of an o^2I-Tree with its age equal to the age currently under investigation. If such a DO is found, the o^2I-Tree below this root-DO is reconstructed. The reconstruction yields to a set of o^2I-Trees as shown in figure 10-7. Next, the decision has to be taken which of the trees are 'relevant' for the user input, done by function 10.3 'o2I-tree_matches_restrictions'. Presently, in the basic variant of the integration algorithm, this decision

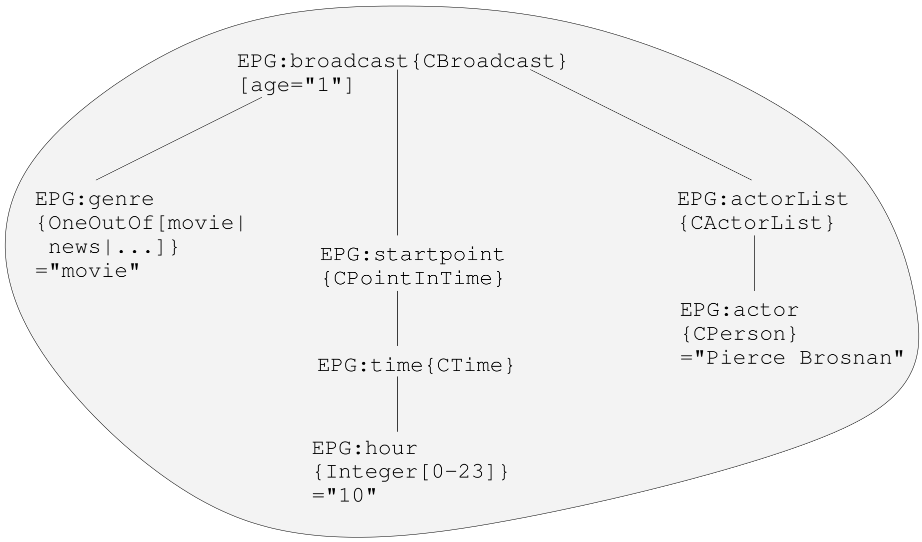

Figure 10-6: Possible hypothesis structure with one o^2I-Tree representing the user input *"show details on the movie at ten with Pierce Brosnan"* according to the ontology fragment for an EPG given in figure 10-5.

depends on the domains in the currently considered hypothesis structure from the user input and the root-DOs of the trees derived from the `discourse memory` before. Thresholds for ages of the root-DOs are given. E.g. only trees with root-DOs younger than the age of 6 are considered. If the domain of the root-DO can also be found in the hypothesis structure, then it should be younger than an age of 5. If it is not contained, it should not be older than an age of 2 (which makes the general age of 6 redundant). In addition, the maximum number of different domains can be restricted. I.e. the number of different domains in a hypothesis is determined, and a tree from the previous discourse is integrated with this hypothesis only if the resulting number of domains is below a given threshold. The o^2I-Trees, which comply with these constraints given the user input figure 10-6, are marked in figure 10-7.

The actual o^2I-Trees are mono-domain structures. Adding a tree of a domain that is not contained in a hypothesis structure into this hypothesis structure therefore just adds the o^2I-Tree without actually modifying the trees inside the hypothesis structure.

The extraction of object-oriented interpretation trees from the `discourse memory` restricts itself to relevant elements with respect to the interpretation of the user input, which should be integrated with the previous discourse. The relevance of an element from the stored discourse information is determined by means of a set of extraction parameters built up by the following elements:

- Maximal age of a DO (`maxAge`, integer).
  Upper bound of ages for DO's from the discourse history to consider for the integration, i.e. the current turn of age 1 up to the turn of age `maxAge` are potentially relevant.

- Maximal number of intra-domain turns (`memoryLengthInsideDomain`, integer).
  If the domain of the considered object from the `discourse memory` is contained in a DO of the hypothesis structure: numbers of turns before the current turn considered to be relevant for integration of these intra-domain objects.

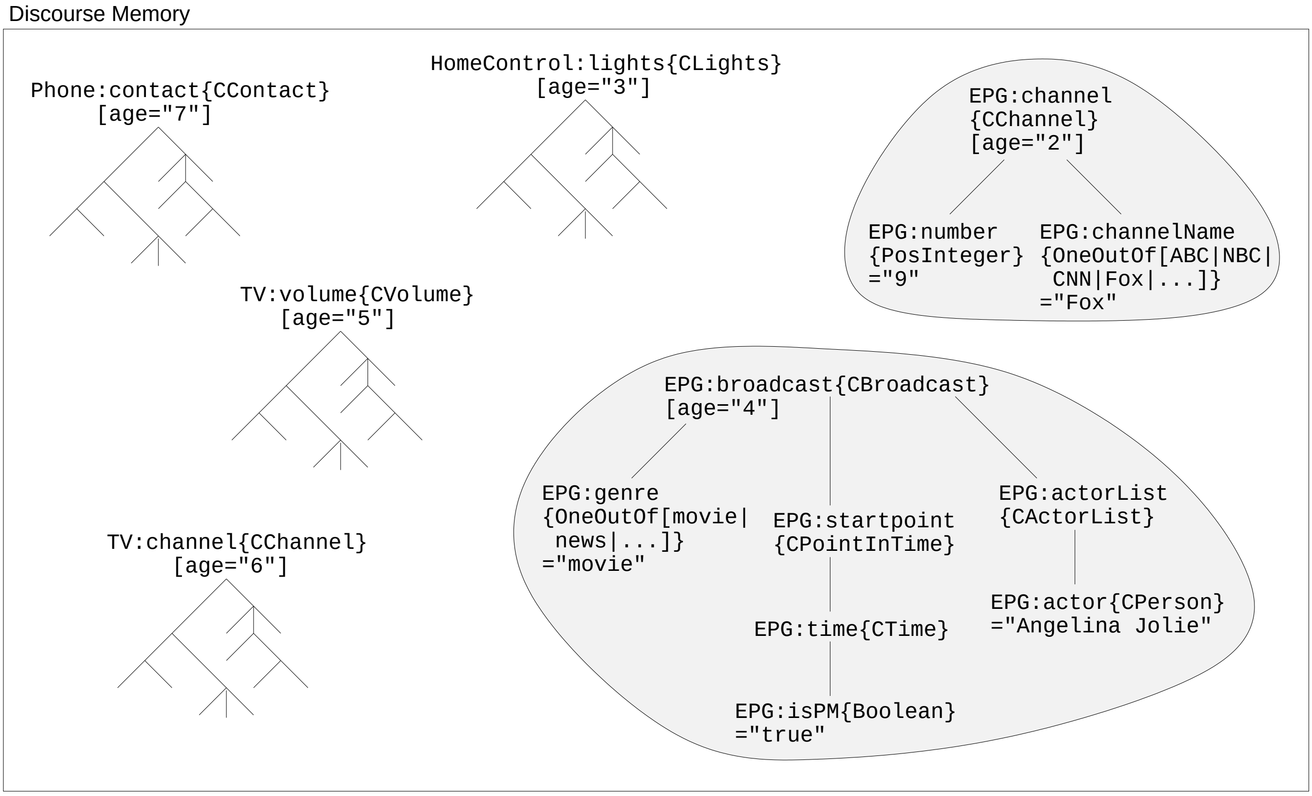

Figure 10-7: `Discourse memory` before the integration of the new user input. Reconstructed o^2I-Trees contained in the memory are shown. The user addressed different applications during the discourse. As the details on the non-EPG portions are irrelevant for our example, the exact structures are omitted. The current user input is of age 1 (and thus not contained in the memory so far), the previous input of age 2, etc. The marked structures are identified as being 'relevant'.

- Maximal number of extra-domain turns (`memoryLengthOutsideDomain`, integer).
  Analogous to the intra-domain turns, restricting on the non domain-matching case.
- Maximal number of different domains (`maxNumDifferentDomains`, integer).
  A domain that was not considered so far is discarded if the total number of different domains exceeds `maxNumDifferentDomains`.

These parameters restrict the search length inside the history in terms of turns, or limit the number of different domains in a single hypothesis.

The functions 10.2 'get_o2I-trees_from_discourse' and 10.3 'o2I-tree_matches_restrictions' formulate the identification method. For each turn, the algorithm considers all temporal DO evolutions where the age of the most recent DO contained in this evolution complies with the above restrictions. If such a DO serves as root of an o^2I-Tree in that turn, the o^2I-Tree with this root is reconstructed – thus the search for relevant objects heavily relies on the reconstruction of object-oriented interpretation trees from the temporal DO evolutions as presented in section 9.3, applied for the turn under consideration.
The reconstructed o^2I-Tree has to fulfill the constraints formulated in function 10.3 'o2I-tree_matches_restrictions' in order to be selected as a relevant o^2I-Tree. The constraints currently applied are related to the age of the root-DO of the considered o^2I-Tree and the domain with

Function 10.2 `get_o2I-trees_from_discourse`(set of domains `D`, extraction parameters `p`)

Data: set of domains `D`: standard domains for intra-domain considerations, usually the set of domains contained in the hypothesis derived from the user input, without integration into the previous discourse
Data: parameters `p`: parameterization for identification of relevant content from `discourse memory`
Result: set of object-oriented interpretation trees T: set of o^2I-Trees reconstructed from the `discourse memory`, candidates for the integration

```
// This function operates mainly on the discourse memory to extract 'relevant' entities.
set of object-oriented interpretation trees T := ∅;
integer counter := 1;
// walk through the stored discourse, from younger to older
while counter ≤ number of turns so far do
    // look up evolutions where the most recent DO is a root object
    // with an age of counter
    set of temporal DO evolutions E
    := set of temporal DO evolutions e in the discourse memory whereby
       for the most recent modified DO o of e
       • o[age] = counter, and
       • o is root-DO of an o²I-Tree integrated at age counter
       holds;
    // reconstruct o²I-Trees below the root-DO and check if the tree is 'relevant'
    foreach temporal DO evolution e ∈ E do
        o²I-Tree t
        := reconstruct most recent o²I-Tree for e according to algorithm 9.4;
        set of domains D_T := domains from DOs in u ∈ T
        // check if tree is 'relevant', if yes: add to set of relevant trees
        if o2I-tree_matches_restrictions(t, p, D, D ∪ D_T) then
            T = T ∪ {t};
        end
    end
    counter = counter + 1;
end
return T
```

Function 10.3 `o2I-tree_matches_restrictions`(o^2I-Tree t, extraction parameters `p`, set of domains `D`, set of domains D_{max})

Data: o^2I-Tree t: check validity of t with parameterization `p`
Data: parameters `p`: parameterization for identification of relevant content from `discourse memory` (see page 84 for an overview on the parameters)
Data: set of domains `D`: standard domains for intra-domain considerations
Data: set of domains D_{max}: set of relevant domains with respect to the maximum number of allowed domains
Result: boolean `b`: t is valid?

// Check if the tree t is 'relevant'.

DO $o_{\text{tree root}}$:= root-DO of t;

// criterion: is root-DO younger than a certain age specified in the parameters?
if $o_{\text{tree root}}$[*age*] > `p.maxAge` **then**
  return false
end

// criterion: is the domain contained in the hypothesis structure and younger than
// a certain age?
if domain of $o_{\text{tree root}} \in$ `D` and $o_{\text{tree root}}$[*age*] $\leq$ `p.memoryLengthInsideDomain` + 1 **then**
  return true
end

// criterion: is the domain *not* contained in the hypothesis structure and younger
// than a certain age
// and
// is the number of different domains in $T \cup \{t\}$ smaller than a given limit?
if domain of $o_{\text{tree root}} \notin$ `D`
  and $o_{\text{tree root}}$[*age*] $\leq$ `p.memoryLengthOutsideDomain` + 1
  and $|D_{max} \cup \{d;\ \text{DO } d{:}\, i\,\{c\} \text{ is contained in } t\}| \leq$ `p.maxNumDifferentDomains` **then**
  return true
end

// no criterion matches
return false

respect to the user input interpretation. Besides a global restriction of the age for the potentially relevant DOs, the evaluation allows to distinguish between root-DOs descending from a domain also contained in the current hypothesis and root-DOs for which this does not hold.

We have to note that the `domain model` honors the domain of DOs during its process of creating a relation between DOs. One o^2I-Tree can only 'live' in one domain, i.e. an o^2I-Tree can only belong to one application-domain. Therefore, combining a hypothesis structure of the user input and an o^2I-Tree from the previous discourse with non-matching domains leads to at least one additional o^2I-Tree in the hypothesis structure representing this interpretation.
Later on, we discover a modified version of the identification procedure, which will enable the combination of trees between differing domains. The modified version supports sharing of discourse information between applications as discussed in section 3.3.3.

10.6. Establishing a Relationship Between Root-Dialog Objects

As step 2 of algorithm 10.1 is discussed in the context of `domain model` and ontologies, we proceed directly with step 3 and refer to section 20.1. Yet, we illustrate the proceeding in step 2 on the basis of the EPG example.

> **Example 'EPG'.** During the previous step, a set of o^2I-Trees from the `discourse memory` was identified as being 'related' to the hypothesis structure representing the user input. The hypothesis structure also contains a set of o^2I-Trees, however in our example the hypothesis structure carries only one element. These outstanding o^2I-Trees need to be combined in order to embed the hypothesis structure into the previous discourse.
>
> To prepare the merger, we consider the root-DOs of the trees: from the previous discourse `EPG:`*`channel`*{`CChannel`} (age 2, see figure 10-7) and `EPG:`*`broadcast`* {`CBroadcast`} (age 4), from the hypothesis structure also `EPG:`*`broadcast`*{`CBroadcast`} (age 1, see figure 10-6). The `domain model` establishes a relationship between these DOs on the basis of the ontologies from the applications[13].
>
> The `domain model` starts with the integration by reducing the number of root-DOs if possible. 'Equal' DOs are merged while remembering that two or more root-DOs contributed to this single DO. The remaining DOs to relate are
>
> - `EPG:`*`broadcast`*{`CBroadcast`}, originating from two DOs, one in the previous discourse (age 4) and one from the hypothesis structure (age 1), and
> - `EPG:`*`channel`*{`CChannel`}, originating from the previous discourse (age 2).
>
> Creating a relationship is an iterative process. The reduced set of DOs serves as starting point for the process. The basic idea is to derive all possible permutations of DOs from the reduced set and combine the elements in such a permutation sequentially. For the two DOs in this example, obviously two sequences are derived:
>
> (`EPG:`*`broadcast`*{`CBroadcast`},`EPG:`*`channel`*{`CChannel`}) and
> (`EPG:`*`channel`*{`CChannel`},`EPG:`*`broadcast`*{`CBroadcast`}).
>
> Generally we can derive $n!$ different sequences of DOs covering all elements in a set of n different elements. For the time being, we arbitrarily choose one of the permutations

[13]Details on the integration are formulated in function 20.1 'relate_DOs_using_o2I-tree_lists'.

to further illustrate the relation process.
In order to integrate the DOs in the sequence (EPG:***broadcast***{CBroadcast}, EPG:***channel***{CChannel}) we start with the first object EPG:***broadcast***{CBroadcast}. During the initial step, the single DO is 'related' with itself. I.e. the ontology is searched for matches of the DO with corresponding objects in the ontology. In case of the ontology shown in figure 10-5, only one object canonically fits to EPG:***broadcast***{CBroadcast}. If there were more than one corresponding object, the following steps need to be applied separately for each of the corresponding objects. Applying the steps on the different corresponding objects could lead to alternative relations of the given DOs.
During the next step, the trivial relation of one object is connected to the next element in the sequence, EPG:***channel***{CChannel}. The 'shortest' relation[14] of these objects is given in figure 10-8. Another relation including the 'shortest' relation is shown in

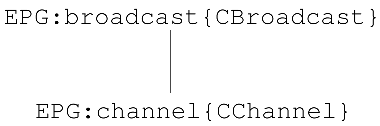

Figure 10-8: 'Shortest' relation of a ***broadcast*** and a ***channel*** DO according to the EPG ontology.

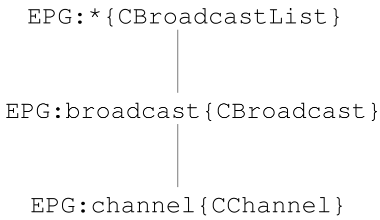

Figure 10-9: The relation of the ***broadcast*** and ***channel*** DOs from figure 10-8 remains a valid relation if it is extended by a parent DO of the ***broadcast*** according to the EPG ontology.

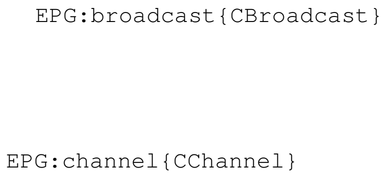

Figure 10-10: Treating a ***broadcast*** and a ***channel*** DO as atomic units: two disjoint entities are returned by the domain model.

figure 10-9. A trivial relation is obtained by treating the two the DOs as independent, figure 10-10. Also the complete ontological structure given in figure 10-5, or each structure that includes figure 10-8 non-trivially relates the two DOs. For efficiency reasons, the domain model already performs a selection on the resulting relations based on an internal rating. This rating prefers more compact trees, therefore we assume in this example that the three relations shown in the figures are the outcome of the relation process.
Note that in our example the search for the second DO EPG:***channel***{CChannel} in the ontological structure has a single canonical counterpart. Otherwise, each potential counterpart would require an independent relation to the first object, again potentially leading to a variety of alterative relations.

If the reduced set of DOs carries a third element, this third element would be connected to the relation of the first two DOs such that the existing relation is included in the relation of the three DOs[15]. The iteration would be continued until all DOs are covered.

[14] 'Shortest' according to an appropriate rating based on the accumulated path lengths between the covered objects.
[15] To be precise: each possible relation of the first two DOs needs to be combined with the third DO separately.

Note that the outcome of the relation process is a set of hypothesis structures; each hypothesis structure represents one possibility to relate the given set of DOs according to the ontologies known by the `domain model`.

10.7. Embedding the Trees from User Input Interpretation and Discourse into the Relation Trees

After identifying the relevant entities from the previous discourse for the user input interpretation and obtaining a skeleton to merge these structures, step 3 will actually perform the integration. An iterative process embeds the structures from hypothesis structure and discourse, successively embedding each tree. Embedding a single tree is potentially recursive. Again, additional alternatives to integrate the trees appear *during* the integration. The procedure has to reflect this.

> **Example 'EPG'.** The embedding is performed for each relation delivered by the `domain model` separately, function 10.4 'integrate_trees_related_to_root-DOs'. We pick out the most compact relation shown in figure 10-8 to demonstrate the integration. Note that this relation is made out of one tree only. If the relation consists of more than one tree, each of these trees is considered separately.
>
> To integrate the o^2I-Trees inside the hypothesis structure representing the user input (figure 10-6) and 'relevant' trees from the previous discourse (figure 10-7), the set of these trees $\mathcal{T}_{\text{for integration}}$ is sorted according to the age of their root-DOs. The integration of these trees is triggered one by one, starting with the tree belonging to the youngest root-DO, see function 10.5 'integrate_trees_related_to_root-DOs_into_o2I-tree'. Accordingly, first the integration of `EPG:`***broadcast***{`CBroadcast`} (age 1) from the hypothesis structure is performed, then `EPG:`***channel***{`CChannel`} (age 2) and finally `EPG:`***broadcast***{`CBroadcast`} (age 4) from the `discourse memory` is embedded.
>
> To embed the o^2I-Tree with root-DO `EPG:`***broadcast***{`CBroadcast`} (age 1, from hypothesis structure) into the skeleton from the `domain model`, see function 10.6 'integrate_single_DO_into_tree', we first identify the counterpart of the root-DO inside the skeleton. In the present skeleton given in figure 10-8, obviously the parental object originates from the root-DO currently investigated.
> Now the o^2I-Tree from the hypothesis structure with root-DO `EPG:`***broadcast***{`C-Broadcast`} (age 1) is actually being integrated into the skeleton tree from the `domain model`. The anchor for the integration is the counterpart of the root-DO inside the skeleton. The integration is performed in two stages: first, the object itself without children is merged, then the children are investigated.
> Merging the root-DO itself, without children from the anchor in the skeleton, basically means that the anchor-DO inherits the properties of the root-DO. In our illustrations, the only visible property that is inherited is the age. The strategy is to include properties of younger objects into older objects.
> After merging the object itself, the children of the o^2I-Tree from the hypothesis structure are considered for integration, done by function 10.7 'update_children_below_DO'. Each child is processed separately. However, the anchor-DO from the integration skeleton and the root-DO from the hypothesis structure only carry 'disjoint' children in this example. Therefore, we can simply append all children to the anchor-DO in this case. The outcome is shown as the result of integration #1 in

figure 10-11.
The resulting structure of integration #1 then forms the skeleton to integrate the next structure, the tree with root-DO `EPG:`*`channel`*{`CChannel`} (age 2) from the `discourse memory`. The embedding is performed analogously to the first integration and shown as integration #2 in figure 10-11. We can observe that the first two integration steps are non-ambiguous.
The most interesting step is the integration of the tree from the previous discourse with root-DO `EPG:`*`broadcast`*{`CBroadcast`} (age 4) that shares the anchor with the tree from the hypothesis structure. Since the anchor in the skeleton – `EPG:`*`broadcast`* {`CBroadcast`} of age 1 – is younger than the root-DO of the tree from the previous discourse, this object remains. Next, the children of both *`broadcast`*s are integrated into the `EPG:`*`broadcast`*{`CBroadcast`} of age 1. We loop over all children of the integration candidate from the previous discourse:

- `EPG:`*`genre`*{`OneOutOf[movie|news|...]`}
  The skeleton also owns an `EPG:`*`genre`*{`OneOutOf[...]`} object as child of the anchor-DO. So these objects could either be merged or be treated as siblings. However, the ontology figure 10-5 forbids more than one `EPG:`*`genre`*{`OneOutOf[...]`}. Therefore, these objects are merged according to the strategy outlined above.

- `EPG:`*`startpoint`*{`CPointInTime`}
  This object is a unique child of `EPG:`*`broadcast`*{`CBroadcast`}. In addition, down to the leaves, all children are unique. Therefore the recursive integration is a straight forward moving down to the leaves and can be compared to the previous procedure. At each object, the required processing is analogous to the processing we present here in detail for `EPG:`*`broadcast`*{`CBroadcast`}. Figure 10-12 shows the portion of the integration relevant to the DO `EPG:`*`startpoint`*{`CPointInTime`}.

- `EPG:`*`actorList`*{`CActorList`}
  The actor list is again a unique child of `EPG:`*`broadcast`*{`CBroadcast`}, thus the `EPG:`*`actorList`*{`CActorList`} objects have to be merged together. Hence, we look at the integration of these objects. Again, the objects `EPG:`*`actorList`* {`CActorList`} are merged without considering their children, the main criteria being their age. Next, the children are appended to the resulting object. According to the ontology, the `EPG:`*`actor`*{`CPerson`} is not unique. Therefore, the `EPG:`*`actor`*{`CPerson`} objects may be merged, or form two separate siblings, figure 10-13.

During the integration process we implicitly assumed that the age relations of the root-DOs can be transferred onto their subordinate DOs, e.g. `EPG:`*`actor`*{`CActor`}= `"Pierce Brosnan"` is younger than `EPG:`*`actor`*{`CActor`}=`"Angelina Jolie"` since the same holds for their root-DOs.

To round up the example, figure 10-14 lists two outcomes for the integration example based on the relation figure 10-8.

The example only marginally covers the multi-hypotheses handling. Whenever alternative possibilities of integrating the structures occur, the next integration step has to consider this directly. The formal description includes the multi-hypotheses handling and dealing with newly appearing

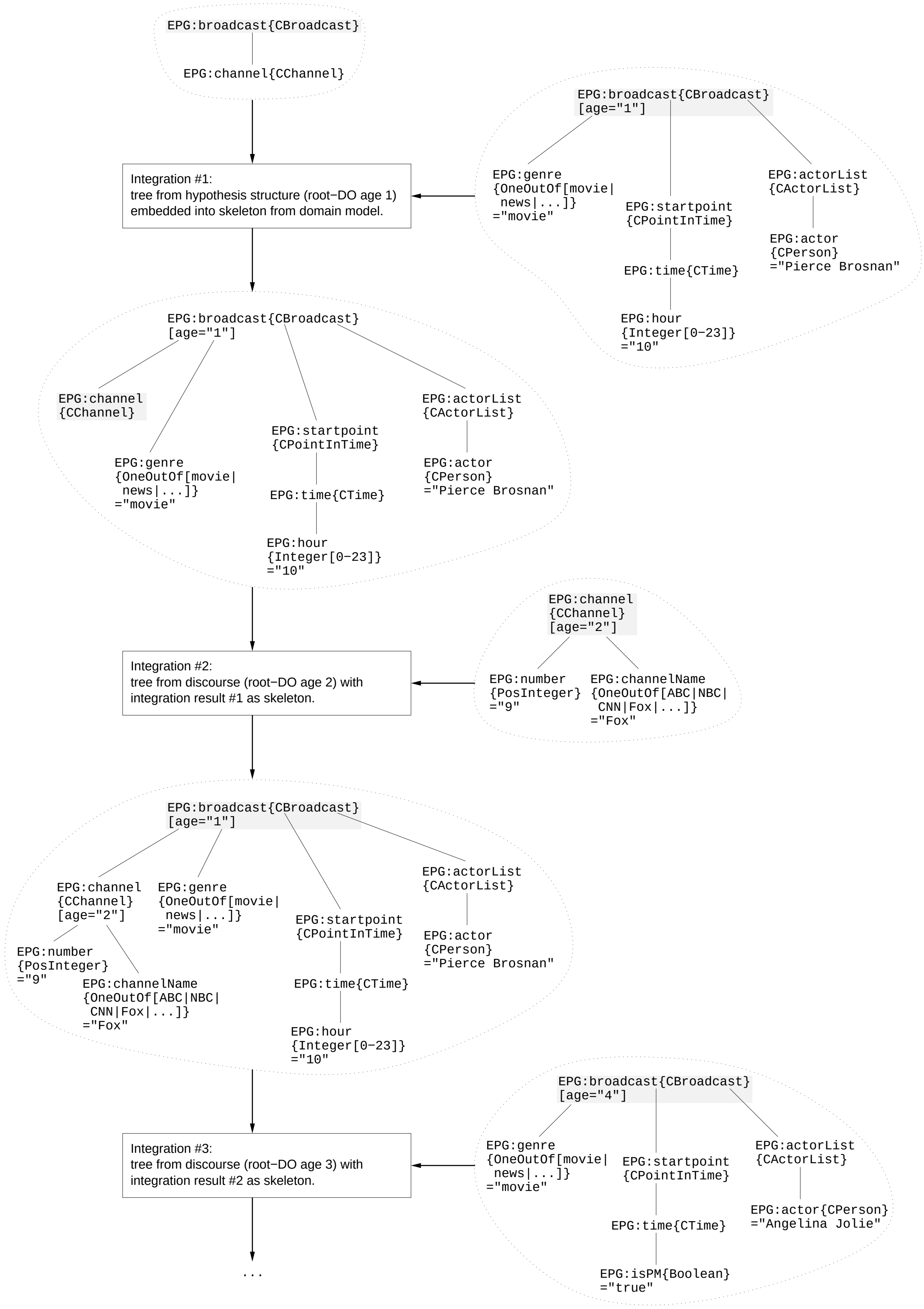

Figure 10-11: Regime of the integration of two o^2I-Trees into a skeleton from the `domain model`. These two integrations are non-ambiguous. The third integration is hinted at.

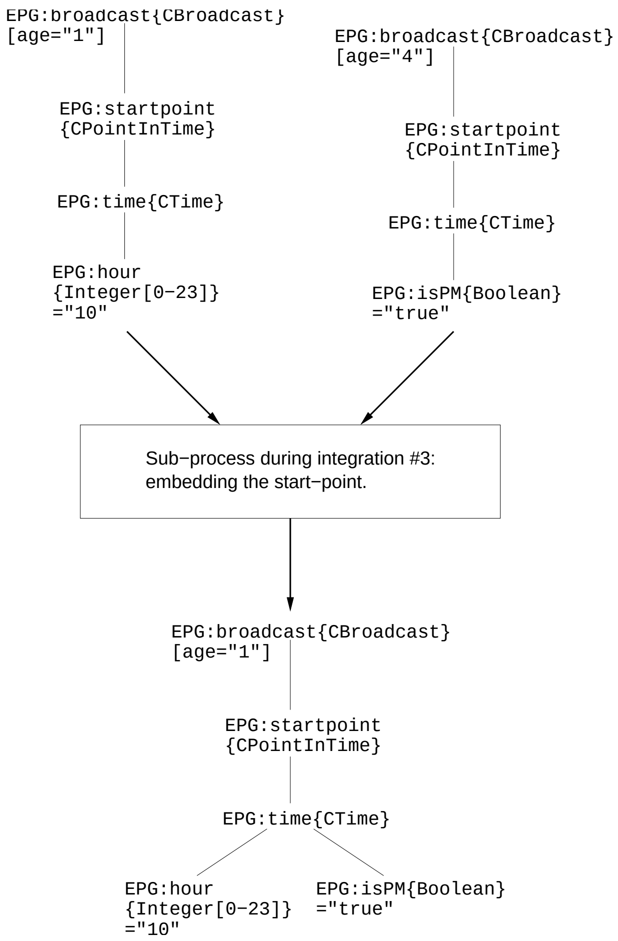

Figure 10-12: Illustration of a problem during the integration: embedding `EPG:`*startpoint*`{CPointInTime}`.

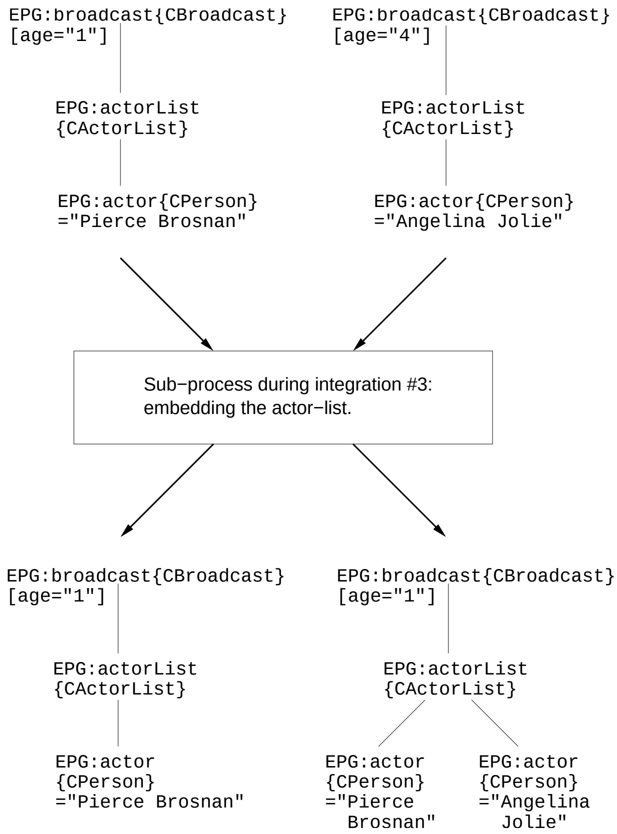

Figure 10-13: Non-unique children as source of ambiguous integrations: actors in an actor list.

hypotheses.
Now we come back to the formal presentation of the third step of the integration algorithm.

The prerequisite for step 3 can be summarized as follows:
A hypothesis structure $h_{\text{user input}}$ representing the user input and a set of object-oriented interpretation trees $T_{\text{discourse}}$ from the previous discourse are to be integrated. The `domain model` constructed a variety of hypothesis structures $h^{(1)}_{\text{related root-DOs}}, \ldots, h^{(n)}_{\text{related root-DOs}}$, each of these stands for one way of relating the root-DOs of the previously mentioned object-oriented interpretation trees in $h_{\text{user input}}$ and $T_{\text{discourse}}$. For step 3 we pick out one of the relations of root-DOs $h^{(i)}_{\text{related root-DOs}}$ and merge the object-oriented interpretation trees from $h_{\text{user input}}$ and $T_{\text{discourse}}$ into $h^{(i)}_{\text{related root-DOs}}$.

Function 10.4 'integrate_trees_related_to_root-DOs' is basically split up into two subunits. At first, each o^2I-Tree in the hypothesis structure connecting the root-DOs is considered separately for the integration of object-oriented interpretation trees below the contained root-DOs. Each integration concentrated on one o^2I-Tree results in a set of object-oriented interpretation trees, each carrying one valid integration result. The second part deals with the combination of the results from the single integrations, thus building up hypothesis structures, every single one forming a hypothesis structure for the user input.

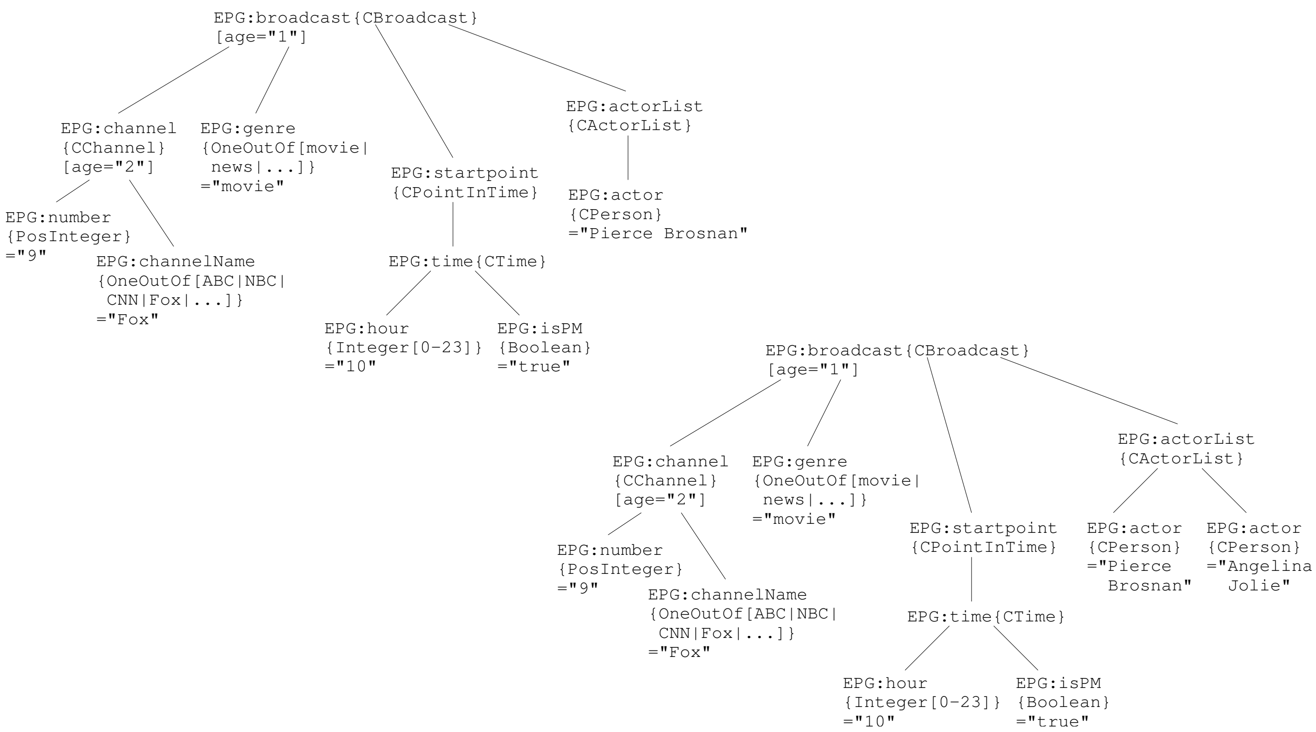

Figure 10-14: Result of the integration process starting from the relation of root-DOs shown in figure 10-8.

Function 10.4 `integrate_trees_related_to_root-DOs`(hypothesis structure $h_{\text{related root-DOs}}$, set of object-oriented interpretation trees $T_{\text{for integration}}$)

Data: hypothesis structure $h_{\text{related root-DOs}}$ comprising the root-DOs serving as anchor for integrating object-oriented interpretation trees

Data: set of object-oriented interpretation trees $T_{\text{for integration}}$, usually $T_{\text{for integration}} = h_{\text{user input}} \cup T_{\text{discourse}}$, to be integrated into $h_{\text{related root-DOs}}$

Result: set of hypothesis structures $H_{\text{result}} = \{h^{(1)}_{\text{integrated}}, \ldots, h^{(m)}_{\text{integrated}}\}$, each representing one hypothesis for the user input derived from $h_{\text{related root-DOs}}$

// embed trees from hypothesis structure and `discourse memory` into skeleton
// from `domain model` connecting their root-DOs; the structures inside a
// relation are integrated one after another
foreach o^2I-Tree $t \in h_{\text{related root-DOs}}$ **do**
    // integrate into the single tree t from the relation skeleton
    set object-oriented interpretation trees T_t
    := `integrate_trees_related_to_root-DOs_into_o2I-tree`(t, $T_{\text{for integration}}$);
end

// combine the results for each single structure from a relation into
// complete hypothesis structures
H_{result} := `build_o2I-tree_lists_through_combination`($(T_t)_{t \in h_{\text{related root-DOs}}}$);

10.7.1 Embedding Trees into a Skeleton Structure. Function 10.5 'integrate_trees_related_to_root-DOs_into_o2I-tree' considers a single tree t_{origin} that serves as integration skeleton together with a set of object-oriented interpretation trees forming candidates for the integration. The flow inside the function is depicted in figure 10-15. For each of the integration candidates,

Function 10.5 `integrate_trees_related_to_root-DOs_into_o2I-tree(`o^2I-Tree t_{origin}, set of object-oriented interpretation trees $T_{\text{for integration}}$`)`

Data: o^2I-Tree t_{origin}: originator for the integration of object-oriented interpretation trees belonging to the root-DOs that have been related by t_{origin}
Data: set of object-oriented interpretation trees $T_{\text{for integration}}$: candidates for the integration into t_{origin}
Result: set of object-oriented interpretation trees $T_{\text{result structure}}$: contains the different possibilities to integrate $T_{\text{for integration}}$ into t_{origin}

set of object-oriented interpretation trees $T_{\text{result structure}} := \{t_{\text{origin}}\}$;

// loop over all trees that need to be integrated
foreach o^2I-Tree $t_{\text{integrate me}} \in T_{\text{for integration}}$, tree with youngest root-DO first (according to age) **do**
  DO $o_{\text{root/integrate me}}$:= root-DO of $t_{\text{integrate me}}$;
  // loop over all currently available alternative integration results obtained so far
  // and integrate the tree $t_{\text{integrate me}}$ next
  foreach o^2I-Tree $t \in T_{\text{result structure}}$ **do**
    $T_{\text{result structure},t}$:= `integrate_single_DO_into_tree(`t, $o_{\text{root/integrate me}}$, $t_{\text{integrate me}}$`)`;
  end
  // collect all integration results
  $T_{\text{result structure}} = \bigcup_{t \in T_{\text{result structure}}} T_{\text{result structure},t}$;
end
return $T_{\text{result structure}}$

the integration into t_{origin} is performed in a single iteration. Each of these integration processes generally leads to a set of alternative integration results. Thus, before each step in the integration sequence is performed, the set of trees serving as skeleton to incorporate the next tree must be updated. The redefinition of $\texttt{T}_{\text{result structure}}$ in the function reflects the need for the update.

10.7.2 Attachment of Trees at Anchor Positions. Again function 10.5 'integrate_trees_related_to_root-DOs_into_o2I-tree' postpones the actual integration to a more specialized method: function 10.6 'integrate_single_DO_into_tree'. The integration problem is further brought down for the skeleton tree being currently under investigation and a single root-DO representing a candidate tree for the integration: focus 'only' on this root-DO to perform the integration. This is the entry point into the potentially recursive process once the anchor for the root-DO of the integration candidate inside the skeleton has been identified.

The integration of a single DO described in function 10.6 'integrate_single_DO_into_tree' – shown in figure 10-16 – is divided into two stages:

1. merge the DO node contained in the skeleton with the corresponding DO in the integration candidate, and

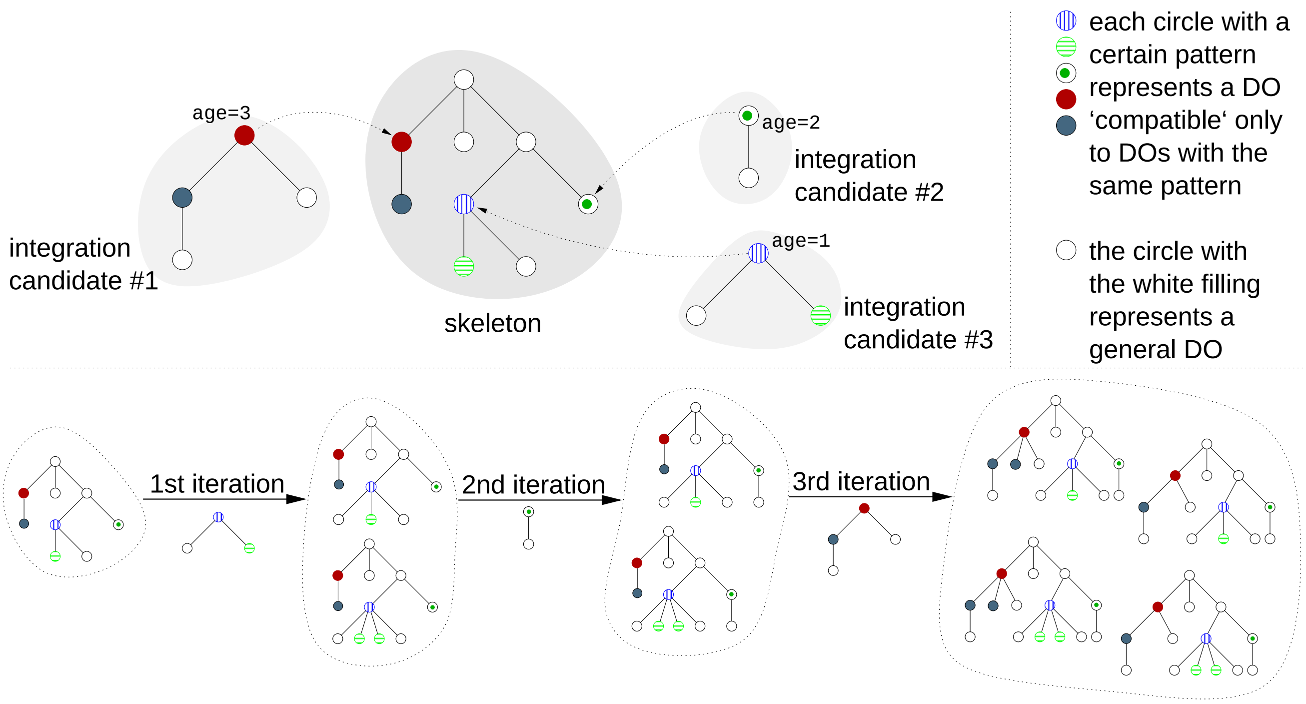

Figure 10-15: Integration of three trees into one skeleton tree (illustration of function 10.5 'integrate_trees_related_to_root-DOs_into_o2I-tree'). The upper half shows one skeleton tree that is obtained from the `domain model`. Three candidates are to be integrated into this skeleton. The age of the root-DO is shown near the root-DO of the integration candidates. The pattern and filling of the circles indicate compatible DOs, the positions where the trees are going to be integrated are hinted by arrows. The lower half of the figure illustrates the sequential integration of the candidates, starting with the 'youngest'. Notice that from a single skeleton multiple integration results are obtained after the 1st and the 3rd iteration.

2. determine the children for the DO node in the skeleton by aligning it with the children of the corresponding DO in the integration candidate.

It is the entry point for the recursive integration process walking down the children below the DOs considered here.
During the first stage (upper part of figure 10-16), the DO supposed to be merged with a certain node into the skeleton is considered, yet this step ignores its children. Mind that the prerequisite, under which this function is called, ensures that the instance and class of nodes to be integrated coincide. The age of the candidate for the integration and the existing DO, which serves as anchor, is compared. The procedure of the node merge is as follows: if the candidate is not younger than the target node with respect to the age of these DOs, the state of the DO in the skeleton tree is maintained. Otherwise, the target DO inside the skeleton adopts the components from the candidate DO except for the child-DOs – the original child-DOs remain.
The child-DOs of the potentially updated anchor DO in the skeleton tree are enriched with children from the candidate DO if they exist. The update of the children is treated by function 10.7 'update_children_below_DO'. During the derivation of the children in general ambiguities occur, thus forming another source for alternative hypotheses for the user input.

Let us now move on with the determination of the children for the target node through the

Function 10.6 `integrate_single_DO_into_tree(`o^2I-Tree $t_{\text{integrate here}}$, DO $o_{\text{integrate pos}}$, o^2I-Tree $t_{\text{integrate me}}$`)`

Data: o^2I-Tree $t_{\text{integrate here}}$: the originating tree used to integrate $t_{\text{integrate me}}$
Data: DO $o_{\text{integrate pos}}$: $t_{\text{integrate here}}$ contains $o_{\text{integrate pos}}$ directly or in derived form, it is used as anchor for the integration of $t_{\text{integrate me}}$
Data: o^2I-Tree $t_{\text{integrate me}}$: tree that is considered for integration into $t_{\text{integrate here}}$
Result: set of object-oriented interpretation trees $T_{\text{result structure}}$: contains the different possibilities to integrate $t_{\text{integrate me}}$ into $t_{\text{integrate here}}$

// the original DO at the integration position $o_{\text{integrate pos}}$ might have been
// modified during the integration of a 'previous' (younger) DO
DO $o_{\text{integration node}}$:= counterpart of $o_{\text{integrate pos}}$ in $t_{\text{integrate here}}$;

// now actually integrate $t_{\text{integrate me}}$ at the position of $o_{\text{integration node}}$ into $t_{\text{integrate here}}$ in
// a two stage process:
DO $o_{\text{root for integration}}$:= root-DO of $t_{\text{integrate me}}$;

// 1. merge the node $o_{\text{root for integration}}$ with $o_{\text{integration node}}$; might update $o_{\text{integration node}}$
if $o_{\text{integration node}}$`[`*age*`]` $\geq$ $o_{\text{root for integration}}$`[`*age*`]` **then**
    $o_{\text{integration node}}$ = overwrite $o_{\text{integration node}}$ with differing content from $o_{\text{root for integration}}$, do not touch $o_{\text{integration node}}$`[`*children*`]`;
end

// 2. update the children for $o_{\text{integration node}}$ by considering the children of
// $o_{\text{integration node}}$ *and* $o_{\text{root for integration}}$
set of object-oriented interpretation trees $T_{\text{result structure}}$
:= `update_children_below_DO(`$t_{\text{integrate here}}$, $o_{\text{integration node}}$, $o_{\text{root for integration}}$`[`*children*`])`;

return $T_{\text{result structure}}$

enrichment by children of the candidate DO, function 10.7 'update_children_below_DO'. Each child $o_{\text{integrate me}}$ of the integration candidate DO $o_{\text{root for integration}}$ (from function 10.6 'integrate_single_DO_into_tree') is considered separately. Alternative solutions can be determined while merging the children of the two DOs. Thus, again the algorithm in function 10.7 'update_children_below_DO' has to honor this ambiguity: each alternative solution is considered separately to merge with the current child. After each child from $\mathtt{T}_{\text{result structure}}$ was processed, $\mathtt{T}_{\text{result structure}}$ is updated with the processing result of the last round. The process then starts again with the next child for integration.

How is a single child actually merged into the set of children below the node of the skeleton being approached in current turn? Basically there are two cases which must be covered: (i) the DO $o_{\text{integrate pos},t}$ from the skeleton already possesses one or more children of the same instance and class content as $o_{\text{integrate me}}$ (i.e. $\mathtt{O}_{\text{similar children}} \neq \emptyset$) or (ii) the children of $o_{\text{integrate pos},t}$ and the potential child $o_{\text{integrate me}}$ are 'disjoint' (i.e. $\mathtt{O}_{\text{similar children}} = \emptyset$). The second case is the easy one.

Case (i). In this case, alternative solutions of the integration are derived. For each child of $o_{\text{integrate pos},t}$ with the same instance and class content as $o_{\text{integrate me}}$ we jump back to the integration of a single DO onto an existing DO in the skeleton tree, function 10.6 'integrate_single_DO_into_tree', with the potential child and its similar counterpart as arguments. The skeleton tree that serves as the basis of the integration is a clone of the skeleton tree t currently considered for the integration. Thus for each similar child a new alternative integration solution is determined. In addition, case (ii) is also performed in case (i) if none of the children of $o_{\text{integrate pos},t}$ or the DO

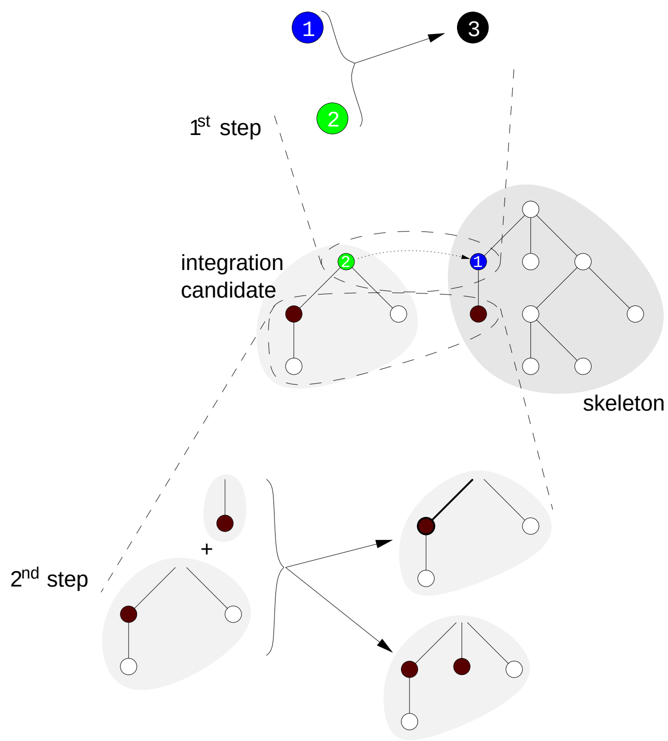

Figure 10-16: Integration of a single DO into the skeleton tree (illustration of function 10.6 'integrate_single_DO_into_tree'). Each circle represents a DO. The numbered circles with a colored filling are 'compatible', the same holds for the circles with the dark red (nearly black) filling. Circles with a white filling are ordinary DOs. The integration candidate is going to be embedded into the skeleton. During the 1st step, the root-DO of the integration candidate and the compatible DO inside the skeleton are merged without paying attention to their children. For the resulting DO, its children are derived during the 2nd step. This figure shows two alternative integration results leading to two and three children respectively.

to be integrated as child $o_{\text{integrate me}}$ is explicitly marked as being unique by the corresponding DO property.
Case (ii). In this case – either none of the existing children is similar to the potential new child or none of these DOs is marked unique – the $o_{\text{integrate me}}$ is appended as a new child of $o_{\text{integrate pos},t}$ into t.

10.8. Combination of Integration Results for Single Skeletons

Given the accompanying EPG example (see also pages 83, 88, and 90), the combination of different integration results is trivial.

> **Example 'EPG'.** When we start with a hypothesis structure carrying a single o^2I-Tree and consider only skeletons to relate the root-DOs made out of a single structure, e.g. figures 10-8 and 10-9, the derived hypothesis structure is also restricted to a single o^2I-Tree.
> However, when the hypothesis structure contains more than one o^2I-Tree or the `domain model` delivered a non-single relation structure (like figure 10-10), then we

Function 10.7 `update_children_below_DO`(o^2I-Tree $t_{\text{integrate here}}$, DO $o_{\text{integrate pos}}$, set of object-oriented interpretation trees $T_{\text{children for integration}}$)

Data: o^2I-Tree $t_{\text{integrate here}}$: the trees from $T_{\text{children for integration}}$ are considered for integration as children of $o_{\text{integrate pos}}$ into $t_{\text{integrate here}}$
Data: DO $o_{\text{integrate pos}}$: object whose set of children might be updated with children represented by $T_{\text{children for integration}}$
Data: set of object-oriented interpretation trees $T_{\text{children for integration}}$: candidates for children of $o_{\text{integrate pos}}$ that might update its existing children or be added as new children
Result: set of object-oriented interpretation trees $T_{\text{result structure}}$: contains the different possibilities to integrate the children $T_{\text{children for integration}}$ below $o_{\text{integrate pos}}$ into $t_{\text{integrate here}}$

set of object-oriented interpretation trees $T_{\text{result structure}} := \{t_{\text{integrate here}}\}$;

// check: is there something to integrate?
if $T_{\text{children for integration}} = \emptyset$ **then**
| **return** $T_{\text{result structure}}$
end

// [... continued on page 100 ...]

can perform the integration separately for each o^2I-Tree in the hypothesis structure and each relation structure. For the structure in figure 10-10, we would get a set of possible integration results for each of the trivial structures. Afterwards, each possible integration result for `EPG:`*`broadcast`*`{CBroadcast}` is combined with each of the results for `EPG:`*`channel`*`{CChannel}` to derive the hypothesis structure that integrate user input and discourse (in our case there would be two results for the first trivial 'structure' and only one result for the latter).

To complete the integration algorithm, the combination of independent integration results is regarded.

Once all trees are integrated using the described (potentially recursive) procedure, the alternative integration results for each single skeleton tree must be combined to obtain a collection of hypothesis structures, which represent alternative interpretations of the user input.

The combination of the integration of trees into skeletons outlined in the last section is covered by function 10.9 'build_o2I-tree_lists_through_combination'. Since we apply a downstream `hypothesis selection` (section 11.1), the combination of the alternative integration results can 'simply' create *all possible* combinations and postpone the comparison of these hypotheses to the selection process. Note that the number of generated results can be computed as $\prod_{i=1}^{r} s(i)$ using the notations introduced in function 10.9 'build_o2I-tree_lists_through_combination'. However, in most cases $r = 1$ or $r = 2$ holds.

The combination of the alternative integration results concludes the algorithm given by function 10.4 'integrate_trees_related_to_root-DOs' by determining a set of hypothesis structures the `hypothesis selection` can choose from.

Function `update_children_below_DO`; *continued*

// [...continuation of 10.7 'update_children_below_DO' (page 99) ...]

// loop over all candidates for integration
foreach o^2I-Tree $t_{\text{integrate me}} \in T_{\text{children for integration}}$ **do**
  // loop over all different integration results from the last integration steps and
  // merge the current candidate into all of these
  foreach o^2I-Tree $t \in T_{\text{result structure}}$ **do**
    set of object-oriented interpretation trees $T_{\text{result structure},t} := \emptyset$;
    DO $o_{\text{integrate pos},t}$:= counterpart of $o_{\text{integrate pos}}$ in t;
    DO $o_{\text{integrate me}}$:= root-DO of $t_{\text{integrate me}}$;

    // children of the integration position DO $o_{\text{integrate pos},t}$ in t being 'similar'
    // to the DO to be integrated
    set of DOs $O_{\text{similar children}}$
    := children of $o_{\text{integrate pos},t}$ with instance and class equal to instance and class of $o_{\text{integrate me}}$;

    // create for each child being similar to $o_{\text{integrate me}}$ a new hypothesis by
    // merging these DOs
    foreach $o_{\text{similar child}} \in O_{\text{similar children}}$ **do**
      o^2I-Tree $\hat{t}$:= copy of t;

      // recursively walk down the tree
      set of object-oriented interpretation trees $\hat{T}$
      := `integrate_single_DO_into_tree(`$\hat{t}$, $o_{\text{similar child}}$, $t_{\text{integrate me}}$`)`;
      $T_{\text{result structure},t} = T_{\text{result structure},t} \cup \hat{T}$;
    end

    // if there are no children similar to $o_{\text{integrate me}}$ or the children are not unique
    // for their parent, append the tree for integration as new child
    if $O_{\text{similar children}} = \emptyset$ or ($o_{\text{integrate me}}$`[`*isUnique*`]` $\neq$ `true` and for all $o \in O_{\text{similar children}}$: o `[`*isUnique*`]` $\neq$ `true`) **then**
      o^2I-Tree $\hat{t}$
      := t with $o_{\text{integrate me}}$ appended as child to $o_{\text{integrate pos},t}$ (thus inserting the complete tree $t_{\text{integrate me}}$);

      $T_{\text{result structure},t} = T_{\text{result structure},t} \cup \{\hat{t}\}$;
    end
  end
  $T_{\text{result structure}} = \bigcup_{t \in T_{\text{result structure}}} T_{\text{result structure},t}$;
end

return $T_{\text{result structure}}$

Function 10.9 `build_o2I-tree_lists_through_combination`(vector of sets of object-oriented interpretation trees $(\mathsf{T}_t)_t$)

// [w.l.o.g. assume that $(\mathsf{T}_t)_t = (\mathsf{T}_i)_{i=1,\dots,r}$ is numbered by integers and $\mathsf{T}_i = (\mathsf{T}_{i,j})_{j=1,\dots,s(i)}$]

Data: set of object-oriented interpretation trees $(\mathsf{T}_i)_i$: alternative results of the integration of trees into the i-th skeleton

Result: set of hypothesis structures H_{result}: set with all possible combinations of integration results for each single skeleton

set of hypothesis structures $H_{\text{result}} := \emptyset$;

foreach combination $\mathtt{c} = \{c_1, \dots, c_r\}$; $c_i \in \{1, \dots, s(i)\}, i = 1, \dots, r$ **do**
  hypothesis structure $h := (\mathsf{T}_{i,c_i})_{i=1,\dots,r}$;
  $H_{\text{result}} = H_{\text{result}} \cup \{h\}$
end

return H_{result}

10.9. Sharing Discourse Information Between Domains

An essential capability of our application-blind DymaLog is the sharing of discourse information between domains as introduced in section 3.3.3 and listed as 4.1(4) in the criteria catalog for enabling the support of dynamic application setups. I.e. knowledge that was acquired from the user by the system while interacting with some application A is also used later when the user interacts with another application B. Sharing discourse information requires the DO o – or the o^2I-Tree with the root-DO o – that was used in the domain of application A to be 'compatible' with the domain of application B. The 'overlay' operation in Alexandersson and Becker [2003] relies on the SmartKom multi-domain ontology (Gurevych et al. [2003]) to enable sharing of discourse information between domains using the structure of the underlying ontology explicitly. E.g., a broadcast on TV and a presentation of a movie in a cinema may share a common superstructure 'entertainment', and thus subknowledge like start-time might be shared.

> **Example 'EPG/DVD-Recorder'.** To illustrate sharing of discourse information between applications, we adopt the example of the EPG application. The relevant section of the ontology for the DVD-Recorder is given by figure 10-17.
>
> Instead of the user input represented by figure 10-5, we presume the user uttered *"schedule a recording at eight fifteen"*, see figure 10-18.

10.9.1 Compatibility between Dialog Objects. The compatibility of DOs is derived from the definition of the object-oriented interpretation trees through ontologies (section 19) inside the applications. The `domain model` is responsible for providing a mapping

$$\mathfrak{c} : O \times O \longrightarrow \{\text{true}, \text{false}\}$$

$$(10.1) \qquad (o, \tilde{o}) = (\mathtt{d}{:}\,i\,\{\mathtt{c}\}, \tilde{\mathtt{d}}{:}\,\tilde{i}\,\{\tilde{\mathtt{c}}\}) \longmapsto \begin{cases} \text{true} & \text{if } o \text{ and } \tilde{o} \text{ are `compatible'} \\ \text{false} & \text{else} \end{cases},$$

$$\text{with } O := \{o\,;\text{DO } o \text{ available in one of the connected applications}\}.$$

For compatible DOs the equality of their class, $\mathtt{c} = \tilde{\mathtt{c}}$, holds. The compatibility of DOs is a consequence of the reuse of structures when defining the application operation parameters. Thus

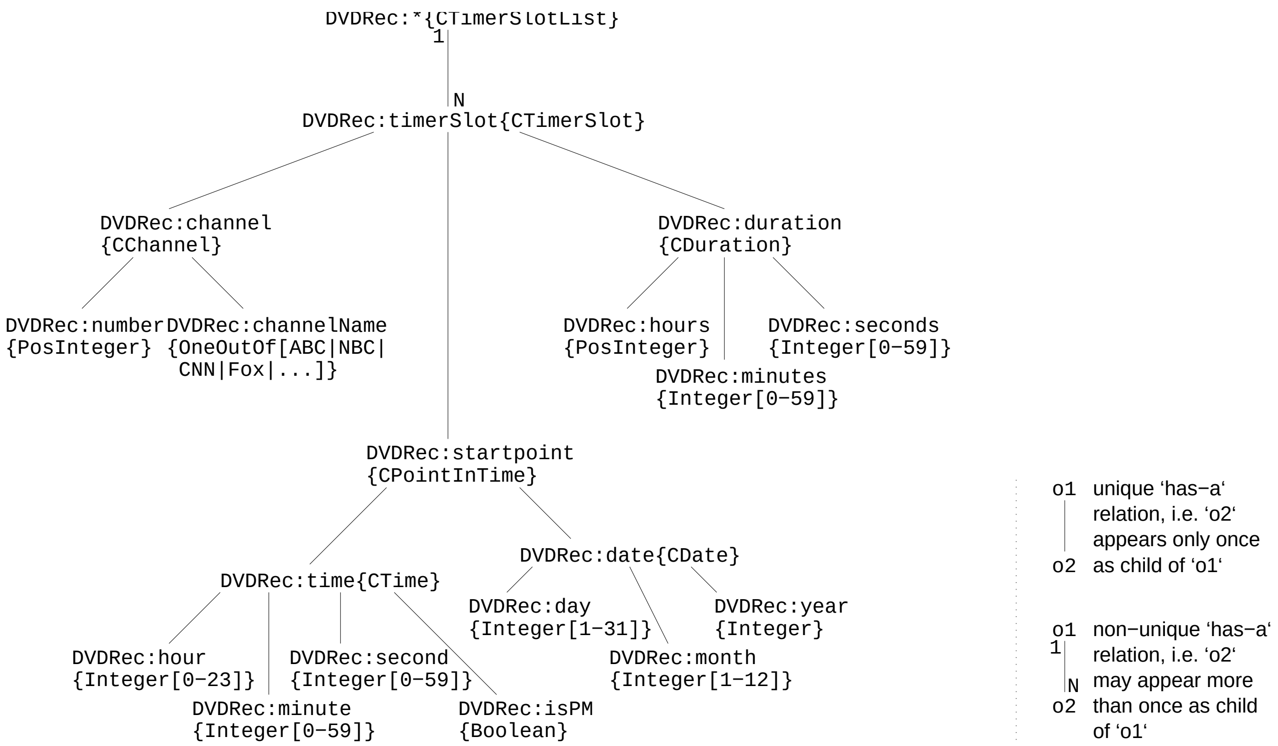

Figure 10-17: Exemplary ontological model for the representation of a list of timer-slots in a DVD-Recorder application. ‘has-a’ and ‘is-a’ relations build up the ontological structures. These two relations can be collapsed into pure ‘has-a’ structures. The outcomes of the `domain model` (section 19.3) are relating structures in which only ‘has-a’ relations connect the objects. The usage of ‘has-a’ only relations to represent the ontology improves readability and is shown in this figure. Note that the o^2I-Trees also connect DOs via ‘has-a’ relations.

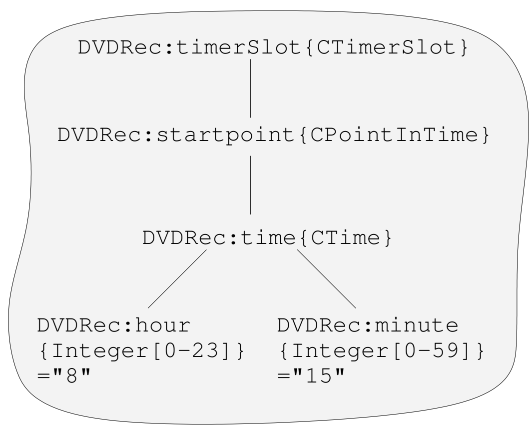

Figure 10-18: Possible hypothesis structure with one o^2I-Tree representing the user input *"schedule a recording at eight fifteen"* according to the ontology fragment for a DVD-recorder given in figure 10-17.

an o^2I-Tree defined in one domain, which is also used in the operation parameters of another domain leads to a set of compatible DOs with respect to these two domains. More details can be found in section 19.4.

Example 'EPG/DVD-Recorder'. For the example, the relations

$$\mathfrak{c}(\texttt{EPG:}\textit{channel}\,\{\texttt{CChannel}\},\texttt{DVDRec:}\textit{channel}\,\{\texttt{CChannel}\}) = \text{true},$$
$$\mathfrak{c}(\texttt{EPG:}\textit{startpoint}\,\{\texttt{CPointInTime}\},\texttt{DVDRec:}\textit{startpoint}\,\{\texttt{CPointInTime}\}) = \text{true, and}$$
$$\mathfrak{c}(\texttt{EPG:}\textit{duration}\,\{\texttt{CDuration}\},\texttt{DVDRec:}\textit{duration}\,\{\texttt{CDuration}\}) = \text{true}$$

should hold. I.e. we assume that the ***channel***, ***startpoint***, and ***duration*** objects are inherited from generic domains in the DVDRec application, the same should hold for the EPG application.

As we see later, together with the compatibility of a root-DO of a structure also its children are compatible, e.g. the ***hour***{`Integer[0-23]`} is therefore also compatible with respect to `EPG` and `DVDRec`.

10.9.2 Modifications of the Integration Algorithm 10.1 to Enable Discourse Information Sharing. Algorithm 10.1 presented before honored the domain to determine the relevant entities from the `discourse memory` for integration. However, the domains remained constant and were not modified – remember further that o^2I-Trees have mono-domains. To allow the integration between two different domains, we have to face the compatibility and updates of the domain.

The compatibility of DOs influences the original integration process of the hypothesis structure representing the user input before the integration of the discourse knowledge and object-oriented interpretation trees from the `discourse memory`, algorithm 10.1.
Assume that an o^2I-Tree t scheduled for integration originates from a domain not contained in the hypothesis structure that serves as container in which t is to be integrated. Since the `domain model` does not try to directly mediate between domains, t will simply be appended to the hypothesis structure as a separate element. The o^2I-Tree t can be merged with one of the trees from the hypothesis structure if the domain of t can already be found in the hypothesis structure.

Now the compatibility of DOs comes into play. Starting from an o^2I-Tree t, which should be integrated with some hypothesis structure h, the compatibility is especially of interest for the first of the above cases: the domain of t is not contained in h. If we want to merge trees of different domains, this is possible if the relevant DOs are compatible in a predefined sense – in our case with respect to mapping (10.1).

Example 'EPG/DVD-Recorder'. Starting from the hypothesis structure shown in figure 10-18, the `domain model` needs to be questioned to find candidates for the integration with the hypothesis structure from the user input. The memory shown in figure 10-7 does not contain elements from `DVDRec` (we assume further that the domain `DVDRec` is not compatible with a non-`EPG` domain). However, reusing information from the `EPG` seems to be reasonable.
Therefore, the algorithm to identify the relevant portions of the discourse is modified to deal with compatible objects.

When the tree with root-DO `EPG:`*`channel`*{`CChannel`} is considered, sharing the discourse information from the `EPG` is straightforward. We can simply transfer the *`channel`* into the `DVDRec` domain.
For the second tree of interest – root-DO `EPG:`*`broadcast`*{`CBroadcast`} – the situation is more complicated. The root object is not compatible with `DVDRec`, nevertheless the time information could be useful. Therefore, trees not being compatible with the target hypothesis structure are searched for (largest possible) compatible substructures. I.e. in the present setting, the structure below `EPG:`*`startpoint`* {`CPointInTime`} would be identified for the integration.

After the identification of relevant trees from the previous discourse, the trees are transformed into the compatible target domain, i.e. in this case from `EPG` into `DVDRec`. The transformed structures are added to the candidate o^2I-Trees for the integration with the hypothesis structure from the user input.

Then, the integration is performed as discussed before.

Let us now elaborate on the modifications in the identification of relevant entities from the `discourse memory`.

To allow the `dialog knowledge processing` to share discourse information between domains, an updated function 10.2 'get_o2I-trees_from_discourse' is used in the integration algorithm 10.1 incorporating mapping (10.1). Two major changes in function 10.2 'get_o2I-trees_from_discourse' are applied, a modified version of the selection function for object-oriented interpretation trees from the discourse is given by function 10.10 'get_o2I-trees_from_discourse'':

1. Instead of restricting itself towards the DOs in the `discourse memory` that served as root of an o^2I-Tree, subtrees are now considered for the integration. Consider a single temporal DO evolution `e` in the `discourse memory`. Let o_{root} be the root-DO of the recent o^2I-Tree reconstructed from `e` (remember section 9.3).
If the domain of o_{root} is also contained in h or compatible with a domain in h according to mapping (10.1)[16], this o^2I-Tree with the root-DO o_{root} will be considered to be relevant for the integration with h.
Otherwise, the subtrees of the o^2I-Tree with root o_{root} are studied. Walking recursively down from o_{root}, each path to a leaf is examined. As soon as a DO o with a domain compatible to the domains in h is found on such a path, the o^2I-Tree with root-DO o is added to the set of object-oriented interpretation trees being relevant for the integration with h. Further the search for compatible DOs below o is canceled. I.e. one or more disjoint substructures below o_{root} can be identified to merge with h. Figure 10-19 illustrates the integration using subtrees.

2. Assume that one or more object-oriented interpretation trees are identified for the integration into h. Instead of simply merging h with these object-oriented interpretation trees, each of the object-oriented interpretation trees is first transferred into a compatible domain contained in h if such a domain exists.

These major differences to the original algorithm allows knowledge gathered during the interaction with some other application than the current – even if it is hidden deep inside an o^2I-Tree constructed during such an interaction – to be incorporated in the current interpretation of the

[16] I.e. $\mathfrak{c}(\texttt{d:}i\{\texttt{c}\}, \tilde{\texttt{d}}\texttt{:}i\{\texttt{c}\})$ = true for some domain $\tilde{\texttt{d}}$ that is contained in h, $o_{\text{root}} := \texttt{d:}i\{\texttt{c}\}$. Note that especially in the case $\texttt{d}=\tilde{\texttt{d}}$ the compatibility holds.

Function 10.10 `get_o2I-trees_from_discourse'`(extraction parameters `p`, set of domains `D`)

// [Replacement for function 10.2 'get_o2I-trees_from_discourse'].

Data: set of domains `D`: list of domains the integration process targets on, usually all domains from user input interpretation
Data: parameters `p`: variables used to parameterize the selection of object-oriented interpretation trees from discourse
Result: set of object-oriented interpretation trees T: set of o^2I-Trees constructed from the `discourse memory`, candidates for the integration

set of object-oriented interpretation trees $T := \emptyset$;
integer `counter` := 1;

while `counter` ≤ number of turns so far **do**
  set of temporal DO evolutions `E`
  := set of temporal DO evolutions `e` in the `discourse memory` whereby
    for the most recent modified DO o of `e`
    - o [*age*] = `counter`, and
    - o is root-DO of an o^2I-Tree integrated at age `counter`

    holds;

  foreach temporal DO evolution `e` ∈ `E` **do**
    o^2I-Tree t_e
    := reconstruct most recent o^2I-Tree for `e` according to algorithm 9.4;

    // modification #1: the derivation of 'related' o^2I-Trees needs to
    // identify DOs from compatible domains (details: see updated function p. 106)
    set of domains $\mathtt{D}_T$:= domains from DOs in $u \in T$

    set of object-oriented interpretation trees $T_{\text{integrate}}$
    := `get_o2I-trees_for_integration'`(t_e, `p`, `D`, $\mathtt{D}_T$);

    // modification #2: transform trees from compatible domains into target domain
    foreach $t_{\text{integrate}} \in T_{\text{integrate}}$ **do**
      DO `d`:m{`c`} := root-DO of $t_{\text{integrate}}$;
      if `d` ∈ D **then**
        $T = T \cup \{t_{\text{integrate}}\}$;
      else if exists $\tilde{\mathtt{d}} \in D$: $\mathfrak{c}$(`d`:m{`c`}, $\tilde{\mathtt{d}}$:m{`c`}) = true **then**
        o^2I-Tree $t_{\text{transformed}} := t_{\text{integrate}}$ transformed into domain $\tilde{\mathtt{d}}$;
        $T = T \cup \{t_{\text{transformed}}\}$;
      else
        $T = T \cup \{t_{\text{integrate}}\}$;
      end
    end
  end

  `counter` = `counter` + 1;
end
return T

Function 10.11 `get_o2I-trees_for_integration'`(o^2I-Tree t, extraction parameters `p`, set of domains `D`, set of domains $\mathtt{D}_{\max}$)

Data: o^2I-Tree t: repository for the extraction of integration candidates
Data: parameters `p`: variables used to parameterize the selection of object-oriented interpretation trees from discourse
Data: set of domains `D`: list of domains the integration process targets on, usually all domains from user input interpretation
Data: set of domains $\mathtt{D}_{\max}$: set of relevant domains with respect to the maximum number of allowed domains
Result: set of object-oriented interpretation trees T: set of object-oriented interpretation trees constructed from the `discourse memory`, candidates for the integration

DO $o_{\text{tree root}} = \mathtt{d}{:}m\{\mathtt{c}\}$:= root-DO of t;

// first, the replica of the 'classic' check for relevance (restricted to matching domains)
if $d \in \mathtt{D}$ and `o2I-tree_matches_restrictions(`t, `p`, `D`, $\mathtt{D} \cup \mathtt{D}_{\max}$`)` **then**
| **return** $\{t\}$
end

set of object-oriented interpretation trees $T := \emptyset$;

// the classic check failed

// walk down the structure from the root to the leaves and search for the first DO
// on these paths that is compatible to one of the target domains

// loop over all paths from the root to the leaves
foreach directed path `p` from $o_{\text{tree root}}$ to a leaf in t **do**
| // check if there is a DO compatible to one of the target domains on the path, the
| // one closest to the root is the one we want
| DO $o_{\text{match on path p}}$
| := first DO o ($o = \mathtt{d}{:}m\{\mathtt{c}\}$) on path `p` with $\mathfrak{c}(o, \mathtt{d'}{:}m\{\mathtt{c}\})$ = true for a $\mathtt{d'} \in \mathtt{D}$;
| // if a compatible object is found, add the related tree to the candidates for integration
| **if** $o_{\text{match on path p}} \neq$ `null` **then**
| | $T = T \cup \{t_{\text{match}}; t_{\text{match}}$ is o^2I-Tree below $o_{\text{match on path p}}$ in $t\}$;
| **end**
end

// finally – if nothing was found so far – the 'classic' check for relevance in case
// of non-matching domains, i.e. we prefer intra-domain matches before
// compatibility matches before non-matching domains
if $T = \emptyset$ and $\mathtt{d} \notin \mathtt{D}$ and `o2I-tree_matches_restrictions(`t, `p`, `D`, $\mathtt{D} \cup \mathtt{D}_{\max}$`)` **then**
| **return** $\{t\}$
end

return T

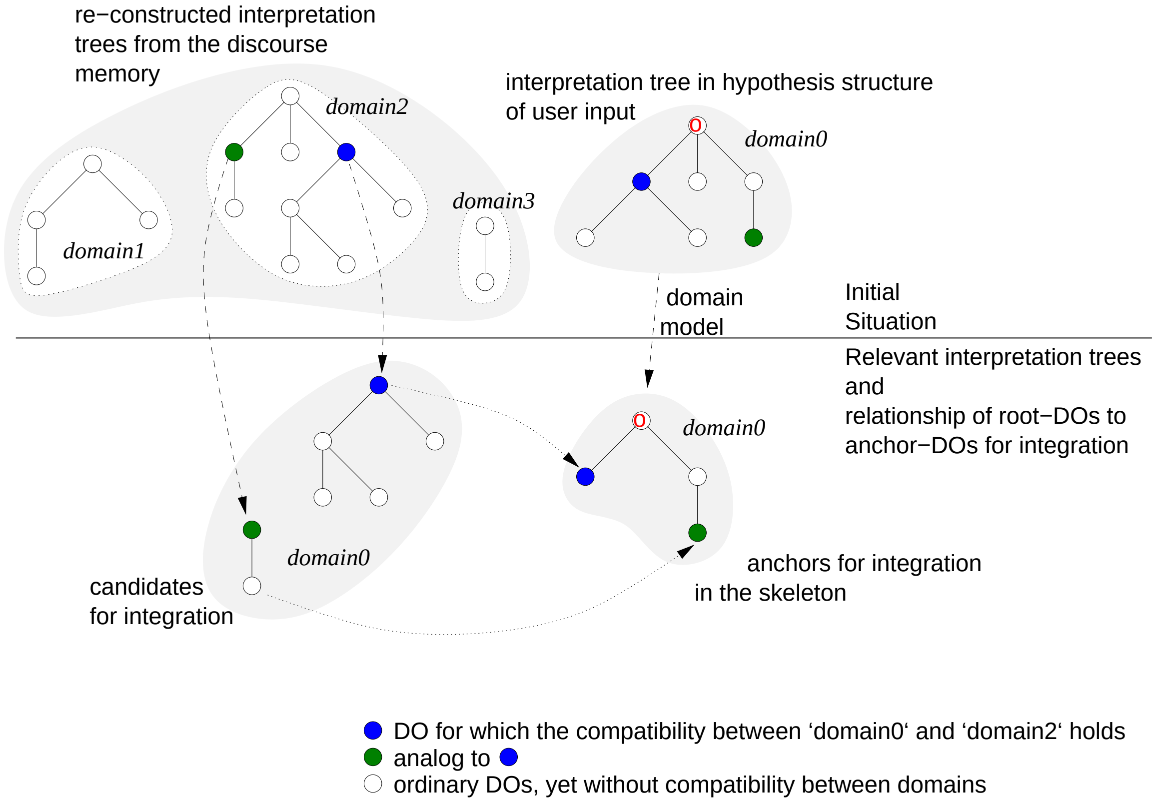

Figure 10-19: Usage of substructures from the `discourse memory` for the integration enabled by the introduction of compatible DOs (related to function 10.10 ‘get_o2I-trees_from_discourse‘). At first, the trees from the `discourse memory` cannot be merged with the o^2I-Tree in the hypothesis structure from the user input due to different domains. By searching the trees from the `discourse memory`, compatible substructures are identified, i.e. their root-DOs being compatible with DOs in the target domain. The lower half shows the identified substructures and a potential skeleton from the `domain model` together with hints where the substructures are going to be integrated.

user input. Therefore, applications are enabled to share acquired discourse information. Remember the example of a train timetable information system where the arrival information can be used as origin for a flight connection in section 3.3.3. The underlying hierarchically organized representations of operation parameters combined with the reuse of structures during the definition of these representations allow the `dialog knowledge processing` to provide generic functionality for sharing discourse information between domains. Thus, no explicit formulation to describe this interchange is required, which in turn makes the independent formulation of applications for use inside the dialog system possible.

We can observe that in the case of a compatible root-DO of the integration candidate or a candidate built up from non-compatible DOs the integration is performed analogously to the original selection process described in function 10.2 ‘get_o2I-trees_from_discourse‘. Yet, this might require the transformation of the domain.

The major difference is the search for compatible DOs below the root-DO.

10.10. From a Flat Structure of Semantic Entities Towards Hierarchical Structures

The analysis of the user input provides a set of semantic entities, each semantic entity carries ontological information (the domain and class of an object), optionally together with a value for this entity. In addition, or as an alternative, a task might be associated to the semantic entity. For the case of the NLU, this is described in section 21.2. The analysis provides a flat structure of semantic entities, i.e. a flat list of atomic entities. The semantic entities are used to derive DOs and/or task objects. If a DO and a task are contained together in a semantic entity, the task is associated with this DO.
Together with the domain model, section 20, a set of competing hypothesis structures are derived. Each hypothesis structure represents a possibility to relate the semantic entities obtained during the input processing.

10.11. Relaxation by Pruning Object-Oriented Interpretation Trees

Generally, the more operation parameters that are specified for an application, the more precise an entity is described by the o^2I-Tree containing these operation parameters. Suppose that the `hypothesis selection` is configured such that it targets at integrating the user input with the previous discourse, the user gets increasingly specific in consecutive inputs. When e.g. an application performs a database query on the basis of the current o^2I-Tree, the set of results from the query is increasingly restricted.

The reverse process of restricting the operation parameters for an application is *relaxation*[17]. Basic relaxation is supported by the framework by providing the means to stop the integration of o^2I-Trees during the recursive process described in the function 10.7 'update_children_below_DO'. During the design of the knowledge sources for the `analysis`, the designer may attach a 'relax'-tag to a semantic entity. When the analysis generates such a semantic entity, the tag is then adopted by the DO constructed from this semantic entity.
If we reach a child DO tagged with 'relax' in function 10.7 'update_children_below_DO', before the algorithm walks recursively down to function 10.6 'integrate_single_DO_into_tree' with this child as argument, further integration below this child is stopped. The o^2I-Tree is cut off below this DO if applicable and the node is marked, so no integration will proceed below this node.

E.g. assume that in an EPG the user specified the ***channel*** with ***channelname*** `Fox`. The EPG table shows broadcasts from channel `Fox`. To see all channels again, the user may utter *"show all channels"*. From the analysis result, a DO representing a channel is obtained, tagged with a 'relax'. If the ***channel*** DO from the new user input is merged with the already existing ***channel*** with child ***channelname***, the DO ***channelname*** will no longer be part of the resulting o^2I-Tree.

10.12. Relating Dialog Objects with Temporal Dialog Object Evolutions in the Discourse Memory

The temporal DO evolutions maintain the history of the evolution of a DO. During the integration of the o^2I-Trees directly derived from the analysis of the user input and the structures from the `discourse memory`, DOs are again taken out of the context of the `discourse memory` (section 9.3). The newly generated DOs obviously do not even have an association towards a

[17] Remember that we leave automatic relaxation to the applications.

certain temporal DO evolution – there may no evolution exist that matches the new DO. Yet, we cannot foresee if a temporal DO evolution, which *could* be associated with a new DO from the user input, will later on be used during the integration of the new user input with knowledge from the previous discourse.

The DOs derived from the `discourse memory` carry an identifier that allows us to merge this DO back into its originating temporal DO evolution during the integration of the selected hypothesis into the `discourse memory` (section 9.2). The merger is feasible in two cases:

- the DO from the `discourse memory` was not updated during the integration process, or
- the DO was merged with a new DO from the input or with older DOs from the previous discourse.

A DO directly derived from the current user input, which is not merged with a DO from the previous discourse, presently creates a new temporal DO evolution in the `discourse memory`.

11. Selection of the Best Hypothesis

After discussing where the hypotheses are created and how the selection is triggered, we elaborate on the selection process. Since the creation is distributed over different components, the details on the creation can be found in the next sections of this chapter.

The selection process can easily be outlined:

11.1 Definition (Selection of Hypothesis)**.** Let $\mathcal{U}$ be the space of all possible inputs by the user. Given a set $\{h_1, \ldots, h_n\}$, $n \geq 1$, of n hypotheses and a measure $\mathfrak{m}_u$ on the space of all possible hypotheses[18] $\mathcal{H}$ for each $u \in \mathcal{U}$, the *hypothesis selection process* is a mapping

$$s_{\mathfrak{m}} : \mathfrak{P}(\mathcal{H}) \times \mathcal{U} \longrightarrow \mathcal{H}$$
$$(\{h_1, \ldots, h_n\}, u) \longmapsto h_{i_0}, i_0 \in \{1, \ldots, n\} \tag{11.1}$$

with $\mathfrak{m}_u(h_{i_0}) \geq \mathfrak{m}_u(h_i)$ for all $i \in \{1, \ldots, n\}$ and $\mathfrak{m} := (\mathfrak{m}_u)_{u \in \mathcal{U}}$. ◇

Let us consider properties of the selection process of definition 11.1.

1. For a given $\mathfrak{m} = (\mathfrak{m}_u)_{u \in \mathcal{U}}$, (11.1) does not imply the uniqueness of the selection process $s_{\mathfrak{m}}$: If a certain set $\tilde{H} := \left\{\tilde{h}_1, \ldots, \tilde{h}_k\right\}$ and $\tilde{u} \in \mathcal{U}$ with the property

$$\mathfrak{m}_{\tilde{u}}\left(\tilde{h}_{j_0}\right) = \mathfrak{m}_{\tilde{u}}\left(\tilde{h}_{j_1}\right) \geq \mathfrak{m}_{\tilde{u}}\left(\tilde{h}_j\right)$$

for $j_0, j_1 \in \{1, \ldots, n\}$, $j_0 \neq j_1$, and all $j \in \{1, \ldots, n\}$ exists and $\tilde{s}_{\mathfrak{m}}$ according to (11.1) fulfills

$$\tilde{s}_{\mathfrak{m}}\left(\left\{\tilde{h}_1, \ldots, \tilde{h}_k\right\}, \tilde{u}\right) = \tilde{h}_{j_0},$$

then we can derive a selection process $s'_{\mathfrak{m}}$ as

$$s'_{\mathfrak{m}}(H', u') := \begin{cases} \tilde{h}_{j_1} & \text{if } H' = \tilde{H} \\ \tilde{s}_{\mathfrak{m}}(H', u') & \text{else} \end{cases},$$

[18] The space of all hypotheses $\mathcal{H}$ contains the set of all hypotheses restricted to the set of applications currently connected to the system. Generally, the set of hypotheses related to connected applications will differ from $\mathcal{H}$.

$H' \in \mathfrak{P}(\mathcal{H})$ and $u' \in \mathcal{U}$. The selection process $s'_{\mathfrak{m}}$ then also complies with (11.1) and $s'_{\mathfrak{m}} \neq \tilde{s}_{\mathfrak{m}}$ holds.

If the stronger condition

$$\mathfrak{m}_u(h_{i_0}) > \mathfrak{m}_u(h_i) \text{ for all } i \in \{1, \ldots, n\}, \tag{11.2}$$

$u \in \mathcal{U}$, in (11.1) is requested, the selection process for a given $\mathfrak{m}$ is unique:
Let $s'_{\mathfrak{m}}$ and $\tilde{s}_{\mathfrak{m}}$ be selection processes according to definition 11.1 respecting the stronger condition (11.2). Assume that $s'_{\mathfrak{m}}(H,u) \neq \tilde{s}_{\mathfrak{m}}(H,u)$ holds for an $H \in \mathfrak{P}(\mathcal{H})$ and $u \in \mathcal{U}$. Then $s'_{\mathfrak{m}}(H,u) = h'$ and $\tilde{s}_{\mathfrak{m}}(H,u) = \tilde{h}$ with $h' \neq \tilde{h}$, $h', \tilde{h} \in H$. From (11.2)

$$\mathfrak{m}_u(h') > \mathfrak{m}_u(h) \text{ for all } h \in H, h \neq h',$$

can be derived. In particular $\mathfrak{m}_u(h') > \mathfrak{m}_u\left(\tilde{h}\right)$ holds, which contradicts $\tilde{s}_{\mathfrak{m}}(H) = \tilde{h}$.

2. The set of hypotheses $\mathcal{H}_{\mathcal{A}}$ relevant for the application set $\mathcal{A}$ currently connected to the dialog system is generally changing in the case of changes in the application setup $\mathcal{A}$. Since the set of applications prepared for the operation by the dialog system can be enlarged after the dialog system was put into operation, previously unseen hypotheses might be added to $\mathcal{H}_{\mathcal{A}}$ when such a new application is added to $\mathcal{A}$.

   Note that due to the dynamic application setup the number of possible hypotheses contained in the set $\mathcal{H}$ cannot be bounded to a certain set beforehand:
   Assuming the finiteness of $\mathcal{H}$, practically only a limited number of applications can be operated by the system. Let $\mathcal{H}'$ contain the elements of $\mathcal{H}_{\mathcal{A}}$ for all application setups $\mathcal{A}$. $\mathcal{H}'$ could be constructed such that it contains $|\mathcal{H}'| < \infty$ elements due to the finiteness of $\mathcal{H}$. Let further each hypothesis be unambiguously addressable by the user.
   During k inputs, the user can at most non-ambiguously enter $(|\mathcal{H}|+1)^k$ different sequences. This is the maximum number of combinations that can be generated from $|\mathcal{H}|$ different hypotheses in k turns, adding a 'noting understood' rival to the hypotheses in $\mathcal{H}$. If, after each input, a different state is entered, no more than $(|\mathcal{H}|+1)^k$ different states can be reached, including unlikely input sequences like 1st: *"switch to CNN"*, 2nd: *"switch to CNN"*, and 3rd: *"switch to CNN"* where after each user input a different state is entered. If we inductively connect applications adding new functionality to the system, the system should also be able to address this new functionality. In the borderline case, this means that $(|\mathcal{H}|+1)^k$ must be increased by either longer sequences used to address the functionality (increase k) or increasing the number of hypotheses, which can be addressed (enlarge $\mathcal{H}'$).

   Certainly, we do not want to place a burden on the user to remember too long sequences to address certain functionality. The strategy of DyMaLog is not to restrict the set of possible hypotheses in principle. Representations, which differ from the representations incorporated by the already existing functionalities, hypothesize additional applications or additional functionalities respectively. I.e. the selection process needs to be capable of dealing with additional hypotheses introduced by new applications and additional functionalities, thus cannot restrict the selection on a predefined fixed set of hypotheses.

Actually, to deploy the selection process it is sufficient to concentrate on $\mathfrak{m} = (\mathfrak{m}_u)_{u \in \mathcal{U}}$. The hypotheses generated from a single user input form the set $H \in \mathfrak{P}(\mathcal{H})$, and we want to extract the best hypothesis $h \in H$ for the user input u according to $\mathfrak{m}$. Let us now take a closer look on the measure underlying the `hypothesis selection`.

11.1. *Quality Measure for the Selection*

The selection process is based on a probability measure $\boldsymbol{P}$. Similar to Young [1999], the desired hypothesis $\hat{h}$ for a certain user input u is defined by

$$\hat{h} = \underset{h \in H}{\operatorname{argmax}} \boldsymbol{P}(\mathtt{H} = h \,|\, \mathtt{U} = u)\,, \tag{11.3}$$

the set H representing the set of hypotheses collected by the `hypothesis selection`. The random variable `H` denotes the potential outcome of the `hypothesis selection` process, the random variable `U` represents the user input. Let `IP` denote the random variable describing the result of the input processing. Using Bayes rule, from equation (11.3)

$$\hat{h} = \underset{h \in H}{\operatorname{argmax}} \sum_{ip} \boldsymbol{P}(\mathtt{H} = h \,|\, \mathtt{IP} = ip, \mathtt{U} = u)\, \boldsymbol{P}(\mathtt{IP} = ip \,|\, \mathtt{U} = u) \tag{11.4}$$

can be derived. If D terms the result of the processing by the `domain model`, (11.4) leads to

$$\begin{aligned} \hat{h} = \underset{h \in H}{\operatorname{argmax}} \sum_{ip} & \boldsymbol{P}(\mathtt{IP} = ip \,|\, \mathtt{U} = u) \\ & \sum_{d} \boldsymbol{P}(\mathtt{H} = h \,|\, \mathtt{D} = d, \mathtt{IP} = ip, \mathtt{U} = u)\, \boldsymbol{P}(\mathtt{D} = d \,|\, \mathtt{IP} = ip, \mathtt{U} = u)\,. \end{aligned} \tag{11.5}$$

From the hypotheses, a set of features summarized in the random variable $\mathtt{F} = (\mathtt{F}_1, \ldots, \mathtt{F}_k)$ is extracted. The features do not directly embed measures already contained in (11.5). However, `F` denotes features derived from processing results or directly from the structure of a hypothesis. As `F` summarizes the features relevant for the selection, we replace `H` by `F`:

$$\begin{aligned} \hat{h} = \underset{h \in H}{\operatorname{argmax}} \sum_{ip} & \boldsymbol{P}(\mathtt{IP} = ip \,|\, \mathtt{U} = u) \\ & \sum_{d} \boldsymbol{P}(\mathtt{F} = \mathrm{f}(h) \,|\, \mathtt{D} = d, \mathtt{IP} = ip, \mathtt{U} = u)\, \boldsymbol{P}(\mathtt{D} = d \,|\, \mathtt{IP} = ip, \mathtt{U} = u)\,, \end{aligned} \tag{11.6}$$

the function f computes the relevant features from a hypothesis. The measure $\boldsymbol{P}^{\mathtt{F}\,|\mathtt{D},\mathtt{IP},\mathtt{U}}$ can be further decomposed, taking into account the vector structure of `F`. Therefore, the resulting measure for the selection process takes the form:

$$\boldsymbol{P}^{\mathtt{H}\,|\mathtt{U}} = \prod_{\text{model } M} \boldsymbol{P}_M\,, \tag{11.7}$$

$\boldsymbol{P}_M$ being the conditional distribution belonging to model M. Actually, $\boldsymbol{P}_M$ will be the approximation of the conditional distribution belonging to model M. Even though the elements in `F` are not stochastically independent, we approximate the measure with (11.7). A model providing a measure may originate from a single component, like the ASR or the NLU, or being derived from the hypothesis in the `hypothesis selection`.

While a probability distribution forms a measure in the mathematical sense, the measures actually computed by the components or through the annotated hierarchical structures have – at least partly – not this property, we will therefore speak of *ratings*.

Thus the `hypothesis selection` deploys an approximation of equation (11.6) and (11.7) respectively. The equation also points out the structure of the processing of hypotheses inside the dialog system organized as a chain.

11.2. The Applied Measure

The `hypothesis selection` task formulated in equation (11.3) can be transferred into the equivalent problem of maximizing scores, e.g. by applying the logarithm:

$$\hat{h} = \operatorname*{argmax}_{h \in H} S\left(\mathtt{H} = h \,|\, \mathtt{U} = u\right) = \operatorname*{argmax}_{h \in H} \sum_{\text{model } M} S_M\left(h\right), \tag{11.8}$$

remember that S_M generally originates from a conditional distribution. While some of the measures have a strong stochastic background, e.g. the measure provided by the PHICOS speech recognition (Steinbiss et al. [1995]), especially the feature based ratings are not directly related to a probability measure.

Taking this into consideration and denoting the rating of model M by R_M, the measure actually applied in the `hypothesis selection` rates hypotheses according to a weighted sum of scaled single ratings:

$$\hat{h} = \operatorname*{argmax}_{h \in H} R_\omega\left(\mathtt{H} = h \,|\, \mathtt{U} = u\right) = \operatorname*{argmax}_{h \in H} \sum_{\text{model } M} \omega_M f_M\left(R_M\left(h\right)\right), \tag{11.9}$$

ω_M being the weight of model M, $\omega := \left(\omega_M\right)_{\text{model } M} \in \mathbb{R}^{|\{\text{models} M\}|}$, and $f_M(\cdot)$ scales the ratings $R_M(\cdot)$ of model M. If necessary, $f_M(\cdot)$ can be used to renormalize the rating of model M or to transform a rating, e.g. an underlying probability distribution, into the score space. The introduction of weight parameters becomes necessary because of approximations made in the models and since the ratings of the models might not be a probability distribution at all. Generally, the value of a rating is proportional to the quality of the hypothesis.

While the model introduced in this section tries to identify the most likely hypothesis based on the user input and the preceding processing stages, the probabilistic model in Fleming and Cohen [2001] formulates benefits and costs to decide on actions to take in the interaction with the user. Another example of probabilistic dialogue management can be found in Roy et al. [2000], where the decisions are based on (partially observable) Markov decision processes. However, in these papers probabilistic dialog management is applied, while we perform probabilistic hypothesis selection.

Next, we investigate the single ratings R_M for the selection of the best hypothesis in equation (11.9).

The separation of the dialog framework from the application requires the measure underlying the hypothesis selection process to be application-independent. Thus, this also needs to hold for the single ratings that constitute the overall measure. The ratings cannot utilize application related knowledge and have to rely on 'generic' features that can be obtained from the hypotheses.

Dialog state vectors in Denecke [2000] are an approach in the same direction, where each single component has a limited value range. However, the dialog state vectors are not used as a measure to determine the best hypothesis out of a set of hypotheses, but help the dialog system to determine the next step.

We will now discuss the single ratings in equation (11.9) in more detail. Table 13-2 on page 129 summarizes the single ratings.

11.3. Rating from the Input Processing

The measure related to the input processing in equation (11.6) is given by $\boldsymbol{P}^{\mathtt{IP}|\mathtt{U}}$. It evaluates the input processing result given a certain user input.

The measure for the input processing in DYMALOG is a combination of ratings obtained from the ASR and NLU (the weight used in section 13 related to this rating will be denoted ω_1).

11.4. Rating from the Domain Model

The contribution from the `domain model` in equation (11.6) is the evaluation of the `domain model` result given the user input and result of the input processing $\boldsymbol{P}^{\mathtt{D}\,|\mathtt{IP},\mathtt{U}}$.

Actually, the hypothesis selection applies an approximation of $\boldsymbol{P}^{\mathtt{D}\,|\mathtt{IP},\mathtt{U}}$. The combination of semantic entities and/or existing structures by the `domain model` does not utilize the raw user input, like the acoustic waveform, but the result of the input processing. That implies

$$\boldsymbol{P}^{\mathtt{D}\,|\mathtt{IP},\mathtt{U}} \approx \boldsymbol{P}^{\mathtt{D}\,|\mathtt{IP}} \tag{11.10}$$

(related weight for section 13: ω_2).
Details on this rating are given later in section 20.3.

11.5. Ratings Derived Directly from the Hypothesis Structure

In equation (11.6), $\boldsymbol{P}^{\mathtt{F}\,|\mathtt{D},\mathtt{IP},\mathtt{U}}$ represents the evaluation of the measures calculated inside the `hypothesis selection` based on the annotated hypothesis given the user input, the result of the input processing, and the result from the `domain model`.

The measure computed from a hypothesis suppresses the direct dependency of $\boldsymbol{P}^{\mathtt{F}\,|\mathtt{D},\mathtt{IP},\mathtt{U}}$ from the user input `U`. Once the semantic entities are extracted from the input, the further processing steps do not fall back on the raw input. Thus

$$\boldsymbol{P}^{\mathtt{F}\,|\mathtt{D},\mathtt{IP},\mathtt{U}} \approx \boldsymbol{P}^{\mathtt{F}\,|\mathtt{D},\mathtt{IP}} \tag{11.11}$$

will be used. $\boldsymbol{P}^{\mathtt{F}\,|\mathtt{D},\mathtt{IP}}$ denotes the probability for a set of characteristics calculated inside the `hypothesis selection` based on the annotated hypothesis structure given the result of the input processing and `domain model`.

A complete list of ratings directly obtained from the hypothesis structure is presented next. For this purpose, let h denote a hypothesis structure. The o^2I-Trees of h are $(t_i)_{i=1,\ldots,n}$, w.l.o.g. $n \geq 1$.
The ratings are presented in a form that suits to equation (11.9). In order to apply (11.9), the approximation $\boldsymbol{P}^{\mathtt{F}\,|\mathtt{D},\mathtt{IP}} = \boldsymbol{P}^{(\mathtt{F}_1,\ldots,\mathtt{F}_k)\,|\mathtt{D},\mathtt{IP}} \approx \prod_{i=1}^{k} \boldsymbol{P}^{\mathtt{F}_i\,|\mathtt{D},\mathtt{IP}}$ was used. As we will notice, the $\mathtt{F}_i$ are not stochastically independent, and we refer again to ω, since it was introduced as a correction.

11.5.1 Present Turn and Relevance of History. Information introduced into the discourse – either by the user, an application, or the dialog system – has different periods of validity. It could, for example, be relevant only for the present turn, or also influence the subsequent turns. Also, information that is more recent may superimpose older, already existent knowledge.
A set of measures related to the age of DOs inside a hypothesis is contained in the overall measure.

Mean Age of DOs in Structures. Three ratings are based on the mean age of certain sets of DOs in the hypothesis structure. Due to the integration of the hypothesis structure representation of the user input with the previous discourse, the resulting hypothesis structure has a certain age distribution in its DOs.
To compute these ratings let

$$\begin{aligned} O_{\text{all}}(h) &:= \{o\,;\,DO\ o \text{ is non-task node in } t_i \text{ for an } i \in \{1,\ldots,n\}\}\,, \\ O_{\text{inner}}(h) &:= \left\{o \in O_{\text{all}(h)};\, o \text{ has non-task children}\right\}\,, \text{ and} \\ O_{\text{leaf}}(h) &:= \left\{o \in O_{\text{all}(h)};\, o \text{ has no children except for task DOs}\right\}\,, \end{aligned}$$

thus $O_{\text{all}}(h) = O_{\text{inner}}(h) \cup_{\text{disjoint}} O_{\text{leaf}}(h)$.
The first rating incorporates *all DOs* inside the hypothesis structure. It is computed as the reciprocal of the mean age of the DOs, i.e. the accumulated age for all DOs divided by the number of these DOs:

$$R_{\text{mean age}}(h) = \frac{1}{\text{mean age of DOs in } h} = \frac{1}{\frac{1}{|O_{\text{all}}(h)|}\sum_{o \in O_{\text{all}}(h)} o\,[age]}. \tag{11.12}$$

$R_{\text{mean age}}$ favors hypothesis structures with a 'younger' age distribution (the weight used in section 13 related to this rating will be denoted ω_3).

Similarly, also ratings based on the mean age of the inner DOs and leaf DOs respectively are available:
The rating for the mean age of the *inner nodes* is given by

$$R_{\text{mean age inner DOs}}(h) = \frac{1}{\text{mean age of inner DOs in } h} = \frac{1}{\frac{1}{|O_{\text{inner}}(h)|}\sum_{o \in O_{\text{inner}}(h)} o\,[age]} \tag{11.13}$$

(related weight for section 13: ω_4). Analog to the inner dialog objects, for the *leaf nodes* we use

$$R_{\text{mean age leaf DOs}}(h) = \frac{1}{\text{mean age of leaf DOs in } h} = \frac{1}{\frac{1}{|O_{\text{leaf}}(h)|}\sum_{o \in O_{\text{leaf}}(h)} o\,[age]} \tag{11.14}$$

(related weight for section 13: ω_5).
The score from these ratings is set to 1 if the set used in the denominator is empty[19].

In Glass et al. [2000], the 'query density' measures the number of 'new concepts' per user query. It is meant to quantify the effectiveness of the communication between human and machine: 'The higher the density, the more effectively a user is able to communicate concepts to the system'. The next two ratings are also based on the comparison of new content to existing content, i.e. they target in the same direction.

Ratio of New Leaves Compared to New DOs. The hypotheses do not allow mixed-mode nodes, i.e. a DO may either have other DOs as children or contain a value. Thus, only leaves can carry values. To operate an application, often a user specifies certain values of an application operation parameter. This can be observed from applications presently implemented for a dialog system that realizes DYMALOG. The share of newly introduced leaf DOs, which may carry a value, in the set of newly introduced DOs takes this observation into account.

Restricting ourselves to the perspective on the DOs of age 1 in h, the fraction of leaves in h of these ages compared to the overall number of DOs of age 1 provides the next rating:

$$R_{\text{new leaves to new DOs}}(h) = \frac{|\{o \in O_{\text{leaf}}(h); o\,[age] = 1\}|}{|\{o \in O_{\text{all}}(h); o\,[age] = 1\}|} \tag{11.15}$$

(related weight for section 13: ω_6). The rating prefers new DOs being leaves of the hypothesis structure. I.e. a higher rating is obtained when the structure remains constant but the leaves change.

New Leaves of Age 1 vs. Existing Leaf DOs. Introducing a leaf DO by the user input could result in a new leaf inside a hypothesis or the modification of an already existing leaf. The following measure compares the number of existing leaves, no matter if modified or not in the present turn, with the number of newly introduced leaves. While the previous rating concentrates

[19]The following ratings are evaluated as 0 in degenerated cases.

purely on newly introduced DOs, this rating focuses on leaves, but drops the restriction that only newly added or modified DOs are considered.

Focusing on leaves, we compare the number of existing leaf DOs, including DOs modified in the current turn, with leaves introduced in the current turn

$$\begin{aligned}
&R_{\text{new vs. existing leaves}}(h) \\
&\quad= \frac{1}{|O_{\text{leaf}}(h)|} \operatorname{abs}\big(|\{o \in O_{\text{leaf}}(h); o\,[age] > 1\}| \\
&\qquad\qquad + |\{o \in O_{\text{leaf}}(h); o \text{ was updated during current turn}\}| \\
&\qquad\qquad - |\{o \in O_{\text{leaf}}(h); o \text{ was introduced in current turn}\}|\big) \\
&\quad= \frac{1}{|O_{\text{leaf}}(h)|} \operatorname{abs}\big(|\{o \in O_{\text{leaf}}(h); o \text{ originally introduced before current turn}\}| \\
&\qquad\qquad - |\{o \in O_{\text{leaf}}(h); o \text{ newly introduced in current turn}\}|\big)
\end{aligned} \tag{11.16}$$

(related weight for section 13: ω_7). $R_{\text{new vs. existing leaves}}$ prefers either a large number of existing leaves together with few new leaves or a high number of new leaves while reusing only a small number of old leaves. An equal number of existing and new leaves is scored badly by this rating.

11.5.2 Tasks. In general, tasks are associated with operations by applications depending on their context, as outlined in section 8.5. A hypothesis without a task object does usually not lead to the execution of an operation by an application. On the other hand, applications could realize strategies to deal with more than one task. E.g., an application could restrict the processing to the youngest task objects according to the age, or process task objects depending on the distance to the root-DO of a tree, the 'closer' the task is to the root, the earlier it is processed.
The `domain model` takes care that the association of a task object to a DO is valid. It requires knowledge on allowed structures, which is contained in the ontologies utilized by the `domain model`. Thus, the task rating presented in this section does not need to evaluate if the associations of tasks to DOs are valid.

To present the task related measure, the set of all task objects in h is given by

$$O_{\text{task}}(h) := \{\mathrm{T}; \mathrm{T} \text{ is a task object in } t_i \text{ for an } i \in \{1, \dots, n\}\}.$$

The task rating is defined by

$$R_{\text{tasks}}(h) = \begin{cases} \frac{1}{1+|O_{\text{task}}(h)|-\exp\left(|O_{\text{task}}(h)|-\sum_{\mathrm{T}\in O_{\text{task}}(h)}(\mathrm{T}[age])\right)} & \text{if } |O_{\text{task}}(h)| > 0 \\ 0 & \text{if } |O_{\text{task}}(h)| = 0 \end{cases} \tag{11.17}$$

(related weight for section 13: ω_8). The rating favors smaller numbers of tasks $|O_{\text{task}}(h)|$ on the one side and prefers younger tasks on the other, the 'exp' argument being in charge of the second property.

11.5.3 Occurrences of Domains. The `domain model` provides a rating based on the relation of objects. The relation of objects is performed on the basis of the ontologies from the applications. In contrast to this rating, the next ratings are based on the number of occurrences of different domains in a hypothesis. Cristea et al. [1998] judges the (global) coherence of the discourse by a smoothness score. Remembering that the measures need to be independent of the underlying applications, the domain-based measures formulate coherence measures for DymaLog on the domain level.

Number of Different Root-DO Domains. The first domain related measure concentrates on single o^2I-Trees in a hypothesis structure. Looking on the root-DOs in h, the reciprocal of the number of different domains for these DOs delivers a rating biased towards lesser number of different domains

$$R_{\text{different root domains}}(h) = \frac{1}{|\{\mathtt{d};\, o = \mathtt{d}{:}m\{\mathtt{c}\} \text{ is root-DO of } t_i, i = 1, \ldots, n\}|} \tag{11.18}$$

(related weight for section 13: ω_9).

Ratio of Root-DO Domains vs. Domains in h. Usually, the o^2I-Trees inside the hypothesis structure belong to the same domain. However, since we allow sharing discourse information between applications, an o^2I-Tree might jointly use knowledge from different domains. Comparing the number of domains used in the root-DOs with the total number of domains in h provides a measure on the smoothness of such a resulting o^2I-Tree:

$$R_{\text{ratio root-/all-DO domains}}(h) = \frac{|\{\mathtt{d};\, o = \mathtt{d}{:}m\{\mathtt{c}\} \text{ is root-DO of } t_i, i = 1, \ldots, n\}|}{|\{\mathtt{d};\, o = \mathtt{d}{:}m\{\mathtt{c}\} \text{ is DO in } t_i, i = 1, \ldots, n\}|} \tag{11.19}$$

(related weight for section 13: ω_{10}). It favors o^2I-Trees residing completely in one domain. Usually, the domains are adopted when information is shared. Thus, in general

$$R_{\text{ratio root-/all-DO domains}}(h) = 1$$

should hold.

11.5.4 Properties of Dialog Objects. The ratings given in section 11.5.1 already make use of the properties of a DO. The bases for these ratings are sets of DOs of certain ages, usually combined with the DO's function (inner node or leaf) in an o^2I-Tree. The ratings presented here can be considered to be related to measures inside the dialog state vectors presented in Denecke [2000] ('quality of current input', 'overall quality').
In this section, ratings based on certain properties of DOs are discussed, but the age property is kept outside.

Ratio of Leaf DOs with Value vs. Leaf DOs. Leaf DOs may or may not carry a value, e.g. the channel-number a user wants to switch her TV to. Leaf-DOs without a value indicate that a certain parameter was addressed, but this parameter specifies no value. As an example, the utterance *"I want listen to an album"* could generate a `album` DO, which is a leaf according to the underlying ontology, without specifying the concrete album. Several applications are realized for the system that implements DymaLog. The leaves in the associated ontologies are generally capable of carrying a value.
To evaluate if the leaves carry a value, the rating

$$R_{\text{ratio leaf with/without value}}(h) = \frac{|\{o \in O_{\text{leaf}}(h);\, o\,\mathtt{[value]} \text{ is set for } o\}|}{|O_{\text{leaf}}(h)|} \tag{11.20}$$

compares the number of leaves carrying a value with the total number of leaves (related weight for section 13: ω_{11}). Ideally, with respect to this measure all leaves contain a value, thus leading to a rating of 1.

Fraction of Properly Defined DOs. During the construction of the hypothesis structure, some of the DOs could lack a complete definition, e.g. if no domain is associated to such a DO. The rating

$$R_{\text{properly defined}}(h) = 1 - \frac{|\{o \in O_{\text{all}};\, o \text{ is not properly defined}\}|}{|O_{\text{all}}|} \tag{11.21}$$

gives a score for the completeness of a hypothesis structure h (related weight for section 13: ω_{12}). Usually, $R_{\text{properly defined}}(h)$ should be equal to 1. Otherwise, it hints on problems during the input processing, e.g. due to inconsistencies in the definition of some of the knowledge sources or faults during the processing.

Existence of List Reference. A more technically motivated rating is related to the existence of a list reference. A hypothesis that includes a reference to a list item could be heavily modified due to this reference. A sub-o^2I-Tree may be inserted at the DO that contains the reference. This sub tree however consists of nodes with an age equal to or larger than 2, i.e. DOs from the `discourse memory`.
Thus, we introduce a trigger for the existence of a list reference, given by

$$R_{\text{ex. of list ref.}}(h) = \begin{cases} 1 & \text{if a DO with reference to a list item exists in } h \\ \frac{1}{2} & \text{if no list reference is contained in } h \\ 0 & \text{if a DO with an irresolvable list reference exists in } h \end{cases} \tag{11.22}$$

(related weight for section 13: ω_{13}). A hypothesis structure, which does not contain any reference, scores $\frac{1}{2}$. One or more successfully resolved references result in the ideal rating of 1, yet only if no irresolvable reference is contained in h. In the latter case, the worst scoring of 0 is obtained.

11.5.5 Segmentation and Understanding of User Input. Bernsen et al. [1996] lists principles of cooperative human-computer dialogs. In the field of informativeness, a generic principle states that a contribution should not be more informative than required. One principle in the area of manner demands brief statements, i.e. avoidance of prolixity. These principles are in line with the I-principle stated in van Rooy [2001], which 'advises the speaker to say no more than he must to fulfill his[these] goals'. We can associate these requirements to a measure based on the semantic content extracted from an utterance.

The user input, which formed the basis of the creation of the hypothesis structure h, is first recognized by the ASR and then segmented during the understanding process. Each recognition result is associated to a (possibly empty) set of semantic entities. In addition, the fraction of words from the recognition result used in this association to semantic entities differs for the various recognition results and different analysis results.
The rating to analyze the recognition and analysis result is based on the segmentation of the recognition result by the analysis, and the association of the obtained segments to semantic content. Let $(s_1, \ldots, s_p)$ be the segmentation obtained in the `NLU`. The rating

$$\begin{aligned} &R_{\text{seg. and underst. of input}}(h) \\ &\quad = 1 - \frac{|\{s_i;\, \text{semantic content for } h \text{ could } \textit{not} \text{ be extracted from } s_i, i = 1, \ldots, p\}|}{p} \end{aligned} \tag{11.23}$$

punishes the existence of segments that did not contribute to the semantics contained in h (related weight for section 13: ω_{14}). I.e. an utterance should not make the contribution more informative than required, and prefers brief, non-prolix, utterances (Bernsen et al. [1996]). In this rating, the

s_i span the maximal possible number of words. If e.g. the segment *"I want to"* is not associated to a semantic entity, these three words may form a first segment s_1. Another valid segmentation could be $s_1 =$*"I"* and $s_2 =$*"want to"*, both not being related to semantic content. However, such segmentation would contradict the requirement of a span of maximum length.

The ratings presented before are partly obvious features derived from the structure of the knowledge representation. Furthermore, the analysis of the hypothesis structures as received by the `hypothesis selection` showed ambiguities in the selection process with a limited number of underlying measures. Note that for many hypothesis structures, the combined ASR and NLU score remains the same since they are derived from the same recognition/understanding result. In order to differentiate between such equally rated hypothesis structures, the additional ratings were developed.

12. Rank-Based Parameter Estimation for the Selection Measure

Equation (11.9) represents the criterion underlying the selection of the best hypothesis[20]. A critical issue for selecting the appropriate hypothesis is the choice of the underlying weights $(\omega_M)_{\text{model } M}$. The transformation functions $(f_M)_{\text{model } M}$ are used to transform the underlying measures into a common score space, mainly applying a logarithm on a probability distribution. Manually choosing and optimizing the weights $(\omega_M)_{\text{model } M}$ can be time-consuming and imperfect. Thus, we concentrate on the data driven computation of the combination weights.

Instead of using a parameter study to obtain $(\omega_M)_{\text{model } M}$, a rank-based method for the computation is used and presented in this section motivated by rank-based statistical methods for parameter estimation. We will refer to the estimation algorithm as *global-rank optimization algorithm (*GRAIL*)*.

Consider a selection process in the sense of definition 11.1. For one turn of the interaction of a user with a dialog system, the `hypothesis selection` collects a set of hypotheses $\{h_1, \ldots, h_n\}$ for the user input $u \in \mathcal{U}$. Applying the underlying rating $R_\omega(\mathtt{H} = \cdot \,|\, \mathtt{U} = u)$ allows us to sort the hypotheses:

$$\{h_1, \ldots, h_n\} \longmapsto \{h_1^*, \ldots, h_n^*\} \tag{12.1}$$

with $R_\omega(\mathtt{H} = h_i^* \,|\, \mathtt{U} = u) \geq R_\omega\left(\mathtt{H} = h_j^* \,\middle|\, \mathtt{U} = u\right)$ for $i > j$, $i, j \in \{1, \ldots, n\}$. Ideally, weights $(\omega_M)_{\text{model } M}$ can be determined such that the hypothesis with the best rating, h_1^*, is the best possible interpretation of the user input by the system. The judgment on the 'best possible interpretation' representing the user input u depends on the view of the observer being in charge of (manually) evaluating (or annotating respectively) all interpretations. Though the optimization has to be performed in such a way that it can be applied for arbitrary input u from the user.

12.1. Estimation Setup

Given a sequence of user inputs $u_1, \ldots, u_t$ from t turns and a set of hypotheses

$$H_i = \left\{h_{i,1}, \ldots, h_{i,n(i)}\right\}$$

[20]We assume the situation of section 11, especially the notations, to hold also in this section.

for input u_i collected by the `hypothesis selection`, $1 \leq i \leq t$ (w.l.o.g. the hypotheses are sorted as in mapping (12.1)). The *rank* r_ω of an interpretation $\tilde{h}_i \in H_i$ is defined as

$$r_\omega\left(\tilde{h}_i, H_i\right) := \left|\left\{h \in H_i; R_\omega\left(\mathtt{H} = h \,|\, \mathtt{U} = u_i\right) > R_\omega\left(\mathtt{H} = \tilde{h}_i \,\middle|\, \mathtt{U} = u_i\right)\right\}\right| + 1, \tag{12.2}$$

i.e. the position of $\tilde{h}_i$ in the ordered n-best list of hypotheses H_i. If h_i^* is the best possible interpretation for turn i, the short notation $r^*_{\omega,i} := r_\omega\left(h_i^*, H_i\right)$ is used for the rank of the best interpretation. The *mean rank* for a sequence $\left(\tilde{h}_i\right)_{i=1,\ldots,t}$ is defined as

$$r_\omega^{\text{mean}}\left(\left(\tilde{h}_i, H_i\right)_{i=1,\ldots,t}\right) := \frac{1}{t}\sum_{i=1}^{t} r_\omega\left(\tilde{h}_i, H_i\right). \tag{12.3}$$

The short notation $r_\omega^{\text{mean}^*} := r_\omega^{\text{mean}}\left(\left(h_i^*, H_i\right)_{i=1,\ldots,t}\right) = \frac{1}{t}\sum_{i=1}^{t} r^*_{\omega,i}$ is used for the mean rank of the best possible hypotheses $\left(h_i^*\right)_{i=1,\ldots,t}$.
Table 12-1 illustrates the setup. The hypothesis with the best possible interpretation is marked

	turn 1	·	·	·	**turn i**	·	·	·	**turn t**
	$h_{1,1}$	·	·	·	$h_{i,1}$	·	·	·	$h_{t,1}$
	·	·	·	·	·	·	·	·	·
	·	·	·	·	$h_{i,r^*_{\omega,i}-1}$	·	·	·	·
	$h_{1,r^*_{\omega,1}-1}$	·	·	·	$*h_{i,r^*_{\omega,i}}$	·	·	·	·
	$*h_{1,r^*_{\omega,1}}$	·	·	·	$h_{i,r^*_{\omega,i}+1}$	·	·	·	$h_{t,r^*_{\omega,t}-1}$
	$h_{1,r^*_{\omega,1}+1}$	·	·	·	·	·	·	·	$*h_{t,r^*_{\omega,t}}$
BETTER	·	·	·	·	·	·	·	·	$h_{t,r^*_{\omega,t}+1}$
↑	·	·	·	·	·	·	·	·	·
$R_\omega\left(\mathtt{H} = \cdot \,\|\, \mathtt{U} = u_i\right)$	$h_{1,n(1)}$	·	·	·	·	·	·	·	·
↓			·	·	·		·		$h_{t,n(t)}$
WORSE					$h_{i,n(i)}$		·		

Table 12-1: Illustration of the n-best lists sequence for the input sequence $u_1, \ldots, u_t$. The best possible interpretation is tagged with '$*$'. (W.l.o.g. the hypotheses for each turn are sorted as in mapping (12.1).)

with an asterisks '$*$'. In the ideal case, the best possible interpretation appears on top of the n-best list for each turn, i.e. $r^*_{\omega,i} = 1$ for all i. In general, for a given ω some of the h_{i,r_i^*} do not have the best score. Thus for these $r^*_{\omega,i}$, $r^*_{\omega,i} > 1$ holds.

12.2. n-Best List Optimization Criteria

In order to estimate the combination weights ω given a setup as described in section 12.1, a criterion to rate the performance for a certain ω is needed. The criterion used in this text will be the minimization of the mean rank of the best hypotheses $r_\omega^{\text{mean}^*}$. It serves as measure for the simultaneous optimization of the rank of the best possible hypotheses. For table 12-1 this criterion leads to a positioning of the best possible hypothesis as high as possible in each column simultaneously.

One could think of a variety of other measures that might be applied alternatively. Among these are the

- minimization of a variation of the weighted mean rank

$$r'_\omega\left(\left(\tilde{h}_i, H_i\right)_{i=1,\dots,t}\right) := \frac{1}{t}\sum_{i=1}^{t} 2^{r_\omega\left(\tilde{h}_i, H_i\right)-1},$$

- the concentration on the *top rank only*

$$r_\omega^{\text{top rank}^*}\left(\left(\tilde{h}_i, H_i\right)_{i=1,\dots,t}\right) := \frac{1}{t}\sum_{i=1}^{t}\left(\mathbb{1}_{\{1\}}\left(r_\omega\left(\tilde{h}_i, H_i\right)\right) + \alpha\mathbb{1}_{\{2,\dots,n(i)\}}\left(r_\omega\left(\tilde{h}_i, H_i\right)\right)\right)$$

  with $\mathbb{1}$ being the indicator function[21] and the penalty α significantly larger than 1, or

- a combination of both[22].

Displacing the weights between the possible ranks of a hypothesis can e.g. be used to stronger punish 'bad' ranks (illustrated by using 2^{r-1} instead of r in equation (12.3)). Focusing on the top rank using $r_\omega^{\text{top rank}}$, all best possible hypothesis with $r^*_{\omega,\cdot} > 1$ are punished equally – no matter how close to the top rank this hypothesis is located.

12.3. Relationship of Estimation Rank and Set of Inequalities

Using the definition from equation (12.2), table 12-1 can be translated into the following set of inequalities

$$(12.5)\quad \begin{aligned} 0 &< R_\omega\left(\mathtt{H} = h_{i,1}\,|\,\mathtt{U} = u_i\right) - R_\omega\left(\mathtt{H} = h_i^*\,|\,\mathtt{U} = u_i\right)\\ &\vdots\\ 0 &< R_\omega\left(\mathtt{H} = h_{i,r^*_{\omega,i}-1}\,\middle|\,\mathtt{U} = u_i\right) - R_\omega\left(\mathtt{H} = h_i^*\,|\,\mathtt{U} = u_i\right)\\ 0 &\geq R_\omega\left(\mathtt{H} = h_{i,r^*_{\omega,i}+1}\,\middle|\,\mathtt{U} = u_i\right) - R_\omega\left(\mathtt{H} = h_i^*\,|\,\mathtt{U} = u_i\right)\\ &\vdots\\ 0 &\geq R_\omega\left(\mathtt{H} = h_{i,n(i)}\,\middle|\,\mathtt{U} = u_i\right) - R_\omega\left(\mathtt{H} = h_i^*\,|\,\mathtt{U} = u_i\right) \end{aligned}$$

for $i = 1, \dots, t$.
Up to now, we assumed an arbitrary given fixed ω, which allowed us to compute the $r^*_{\omega,i}$.

For the inverse approach targeting on the computation of ω let us select $(p_1^*, \dots, p_t^*) \in \mathbb{N}^t_{>0}$. The p_i^* are supposed to predetermine the wanted rank of the best possible hypothesis, thus corresponding to $r^*_{\omega,i}$ in the situation of section 12.1. In the case of an optimization criterion for the best hypothesis according to equation (12.4), the quality for such a given rank-vector can be

---

[21] The *indicator function* is defined as $\mathbb{1}_X(x) := \begin{cases} 1 & \text{if } x \in X \\ 0 & \text{if } x \notin X \end{cases}$.

[22] The measures shown here can be derived from the more general form

$$(12.4)\quad r_\omega^{\text{general}}\left(\left(\tilde{h}_i, H_i\right)_{i=1,\dots,t}\right) := \frac{1}{t}\sum_{i=1}^{t}\sum_{k=1}^{n(i)} f_k\left(r_\omega\left(\tilde{h}_i, H_i\right)\right),$$

$f_k : \mathbb{N} \to \mathbb{R}, x \mapsto f_k(x)$ for all $k = 1, \dots, \max\{n(i); 1 \leq i \leq t\}$.

directly computed from the p_i^*. In conformance with (12.5), the predefinition of targeted ranks can be reformulated to

$$\begin{aligned} 0 &< R_\omega(\mathtt{H}=h_i \,|\, \mathtt{U}=u_i) - R_\omega(\mathtt{H}=h_i^* \,|\, \mathtt{U}=u_i) && \forall h_i \in H_i^{<} \\ 0 &\geq R_\omega(\mathtt{H}=h_i \,|\, \mathtt{U}=u_i) - R_\omega(\mathtt{H}=h_i^* \,|\, \mathtt{U}=u_i) && \forall h_i \in H_i^{\geq} \end{aligned} \tag{12.6}$$

whereby $H_i = H_i^{<} \cup_{\text{disjoint}} H_i^{\geq}$, $|H_i^{<}| = p_i^* - 1$, and $h_i^* \in H_i^{\geq}$ for each $i = 1, \ldots, t$. These systems of inequalities partition the space $\mathbb{R}^{|\{modelsM\}|}$ of the ω, as illustrated in figure 12-1 for two models M_1 and M_2. Solving the system of inequalities for each division of the H_i must lead to at least

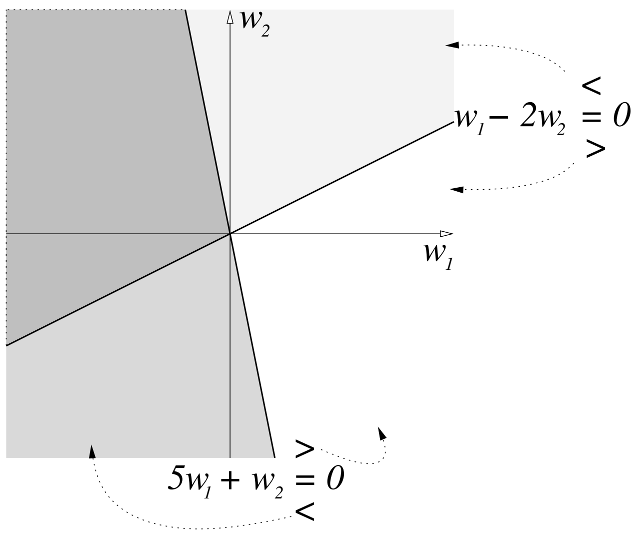

Figure 12-1: Partitioning of the $\mathbb{R}^2$ through combinations of the two systems $\omega_1 - 2\omega_2 < 0/ = 0/ > 0$ and $5\omega_1 + \omega_2 < 0/ = 0/ > 0$ and (parameterization (ω_1, ω_2) for two models M_1 and M_2).

one non-empty set of ω that form the solution space for one of these systems.

12.4. The Basic Idea Underlying Grail

Starting from the best rank-setup according to the underlying quality measure, all possible related decompositions of the H_i leading to this setup are considered. If a non-empty solution space for the set of inequalities (12.6) in ω for one of the decompositions exists, optimal combination parameters are found. Otherwise, the next worse rank-setups are considered comparably. This procedure is iteratively continued until a non-empty solution space is found.

The ideas are the basis of the computation of ω as we described (te Vrugt and Portele [2006]). The hypothesis selection algorithm of the intention recognition applied in SmartKom served as starting point for the `hypothesis selection` in DyMaLog.
The algorithm sketched before was successfully applied to data collected with the SmartKom system. However, on corpora collected with the *Marvin* dialog system the search space of the estimation algorithm was too large. Even though SmartKom provides a highly complex, flexible, and powerful dialog framework, the components in the SmartKom system generate significantly less hypotheses. SmartKom is aware of the benefits of maintaining alternative hypotheses, and thus postpones the final decision to a dedicated hypothesis selection component. However, by sensibly applying knowledge early in its various components, SmartKom aims at finding the

balance between providing a useful number of alternative hypotheses as long as required and discarding superfluous hypotheses as soon as possible. Within the bandwidth from systems that do not allow any alternative hypotheses to systems without restrictions on the number of hypotheses for some user input, SmartKom is located around the 'center' representing systems that allow for a moderate number of hypotheses. DyMALog is oriented towards the more radical approach of an unrestricted number of hypotheses in a system. However, to enable an online system, DyMALog is able to deploy constraints on the number of hypotheses that can be generated by certain components.
The SmartKom recognition and analysis components strongly limit the number of hypotheses. In addition, the domain model and context related modules, including the discourse memory, do not focus on the generation of alternative hypotheses. Therefore, only very few additional hypotheses are generated. In contrast, the dialog system based on DyMALog postpones decisions on the best enhancement of hypotheses to the `hypothesis selection`, and therefore a significantly larger number of alternative hypotheses are generated. Together with the notably larger number of ratings utilized in the quality measure underlying the selection process, this leads to an increase in computation time of the estimation algorithm such that no solutions are found within a reasonable time-frame.

Estimation of the Complexity. The i-th turn has $n(i)$ hypotheses, i.e. the best hypothesis needs to be compared with $n(i)-1$ rivals. This is done for t turns. Thus, each system of inequalities which has to be solved consists of $\sum_{i=1}^{t}(n(i)-1)$ inequalities.
If all possible partitions of the H_i are to be checked, there are

$$|\text{partitions}\,(H_1,\dots,H_t)| = \underbrace{\prod_{i=1}^{t} \underbrace{\sum_{k=1}^{n(i)} \underbrace{\binom{n(i)-1}{k-1}}_{\substack{\text{selection of } H_i^< \\ \left|H_i^<\right|=k-1}}}_{\text{place } h_i^* \text{ at rank } k}}_{t \text{ turns}} \tag{12.7}$$

systems of inequalities that have to be solved. A lower bound for the number of partitions is given as

$$|\text{partitions}\,(H_1,\dots,H_t)| \geq \prod_{i=1}^{t} n(i) \geq \left(\min\{n(i); i=1,\dots,t\}\right)^t. \tag{12.8}$$

E.g. assume that 50 turns are annotated, each carrying 10 hypotheses. If the computation of the solution space for one system of inequalities takes 1 second, then $|\text{partitions}\,(H_1,\dots,H_{50})| \geq 10^{50}$ holds and a complete computation of all solution spaces would need more than $3.17 * 10^{42}$ years. As noted before, for data collected with the SmartKom system a solution using an optimized, more efficient version of the exact algorithm was found. To compute an optimized parameter setup in DyMALog, we rely on an approximation.

12.5. *Application of the Simplex Algorithm for Reduced Computation Times*

In order to obtain applicable parameter setups within reasonable time, we developed an algorithm utilizing the downhill simplex algorithm from Nelder and Mead described in the numerical recipes (Press et al. [2002, chapter 10, section 10.4 'Downhill Simplex Method in Multidimensions']) on

top of the rank-based formulation.
The downhill simplex method in n dimensions starts with a non-degenerated simplex constructed from $n+1$ points. The function under consideration is evaluated for this simplex. Depending on the outcome, the simplex is modified in at least one, but not more than n, points. Modifications are e.g. the reflection of a single point, expansion, or contradiction in a single direction or in multiple directions. The update of the simplex is chosen according to given rules in order to walk into the direction of a (local) minimum. Since the minimum might be a local one, the actually implemented parameter estimation algorithm Grail starts with different shapes of the simplex, and compares the outcome with the already available results to obtain the 'best local minimum'. For a given starting simplex, the stopping condition is based either on a distance with respect to the movement of the presently used points, or the decrease in the function value evaluated for the given simplex.
A more detailed discussion of the method, together with sample code, can be found in the numerical recipes (Press et al. [2002]).

12.6. Remarks on the Selection Process

The ratings derived directly from the hypotheses are subject of section 11.5 after the hierarchical structures are formally introduced.
Allowing changes in the setup further motivated us to design the selection process to be application-independent – otherwise introducing a new application into the system would also require an update of the selection process. Therefore, the ratings are purely based on features of a hypothesis without directly relying on the application itself.

In case of $|\{h \in H_i; r_\omega(h, H_i) = 1\}| > 1$ the first of these is selected and forwarded to the `communication management` since the `communication management` is currently designed to deal with a single best hypothesis. A future advanced version of the communication management might include means to handle equally rated top hypotheses.

13. The Parameterized Hypothesis Selection Process

The `hypothesis selection` (section 11) applies a measure according to the form given in equation (11.9). The success of the selection process depends on the sensible selection of the weights $(\omega_M)_{\text{model } M}$.

The underlying model evolved together with the dialog system. The development was performed in an iterative process, see also Glass [1999] and Degerstedt and Jönsson [2001]. During this process, the capabilities of the system increased. The initial version provided very limited capabilities, and the application accessibility was also preliminary. Logical steps being performed in the iterative process are:

1. Interactions of a user with the dialog system.
2. Evaluation and analysis of the interactions.
3. Update of the dialog system and/or applications, including the modification of knowledge sources and adoption of the parameterization. In particular, the update includes
   - the extension of the system's capabilities, and/or

- the re-estimation of the weights $(\omega_M)_{\text{model } M}$.

4. Restart at step 1 for the next iteration.

During the development phase, the set of models M, which contribute to the overall measure, was extended.

The behavior of the measure underlying the selection process is investigated on the basis of a parameter study. Using Grail for the estimation of the combination weights (section 12), an optimal weight set can be computed.

13.1. Dependency of the Selection Measure on its Parameterization

The combination of n measures according to equation (11.9) grows exponentially in the number of possibilities for a measure. If, for example, k different weights ω_M are checked for each model M, then k^n different combinations are possible.

13.1.1 Data Collection. In this section, we investigate data collected during interactions with one of the latest versions[23] of the *Marvin* dialog system. This system also formed the basis of the interactions presented in chapter III. Table 13-1 summarizes features of the data. The corpus is denoted $\mathtt{C}_1$. It includes interactions with the applications activation & identification,

number of turns	82
total number of hypotheses	1 574
mean number of hypotheses per turn	19.2
number of turns with only 1 hypothesis	4
number of turns with 50 or more hypotheses	8
number of evaluated weight sets $(\omega_M)_{\text{model } M}$ with	
... 1 non-zero weight	70
... 2 non-zero weights	2 275
... 3 non-zero weights	45 500
... 4 non-zero weights	625 625

Table 13-1: Characteristics of the data corpus $\mathtt{C}_1$ for analysis of the influence of the various measures on the hypothesis selection performance.

TV, music player, and image browser (and also the system application). Only 15 of the turns are related to the EPG or HD recorder applications.
In order to evaluate the potential of the `hypothesis selection`, the error rate for a given set of weights $(\omega_M)_{\text{model } M}$ is computed. The number of weights, which differ from 0, was restricted to at most 4 weights. Figure 13-1 shows the distribution of hypotheses per turn, ranging from 1 to 50. During the generation of the corpus, the number of hypotheses for each turn was restricted to at most 50. If more hypotheses are generated for a single turn, the best 50 hypotheses according to the underlying measure and given parameterization are extracted. Actually, if the next m hypotheses ($m \geq 1$, i.e. hypotheses number 51 to $50 + m$) have the same score than hypothesis

[23]The source code and related knowledge source files of the *Marvin* dialog system are managed within a version control system. To collect the corpora $\mathtt{C}_1$ and $\mathtt{C}_2$ we used the *Marvin* dialog system as of May 13th, 2004, and June 7th, 2004, respectively. More recent versions of the *Marvin* dialog system add minor bug-fixes that were discovered during online interactions with the system.

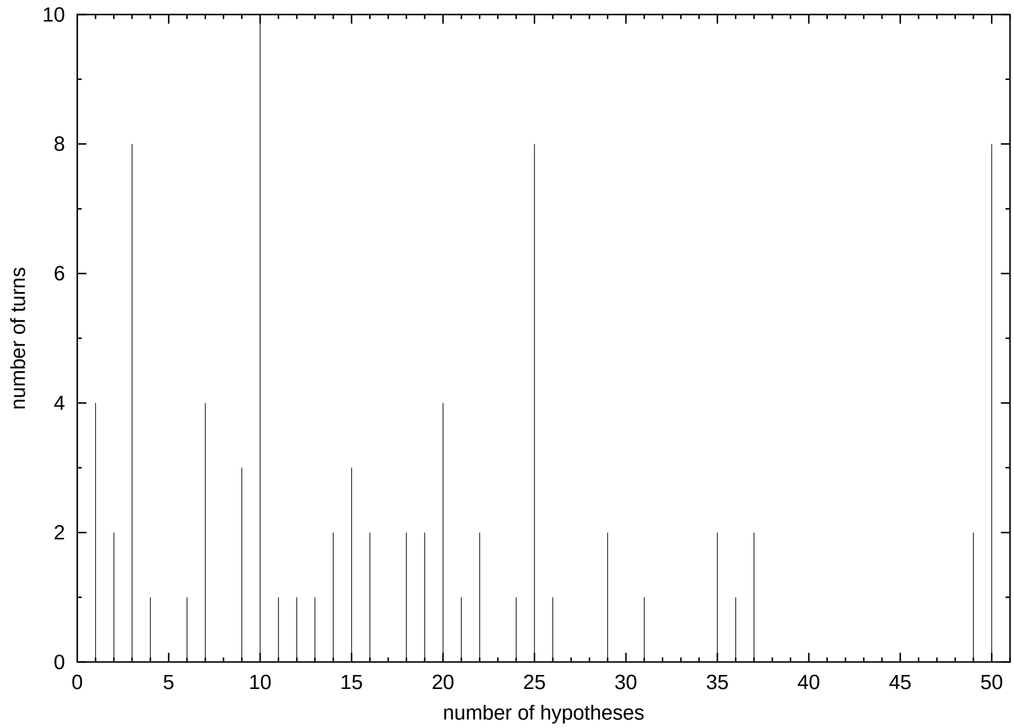

Figure 13-1: Distribution of the number of hypotheses for a single turn of the corpus $\mathtt{C}_1$ summarized by table 13-1.

number 50, then these hypotheses are also listed in the corpus. By the given parameterization, we mean the parameterization of the system that was actually used for the interaction with a user in order to gather the data for the corpus generation.

13.1.2 Annotation. The data were manually annotated[24], i.e. for each turn exactly one hypothesis was selected. This hypothesis is considered to be the best representation of the user input given the discourse history. For the given corpus $\mathtt{C}_1$ (and also for corpus $\mathtt{C}_2$ introduced later), the annotation criteria applied on the corpus are:

1. *Restriction of the hypothesis structures to single domains.* At present, a single utterance mostly addresses a single application. Correspondingly, a hypothesis should also be limited to a single domain.

2. *Stay in domain.* In an ongoing interaction, the continuation of the interaction with the same application is preferred over the change between applications.

3. *Integrate with history from same or compatible domain.* To enable applications beyond command-and-control, the user input is brought into the context of the discourse. Canonically, the hypothesis extracted directly from the user input is enriched with discourse content from the same domain or a compatible domain. Integrating content from a different, non-compatible domain would lead to more than one domain in the discourse. This contradicts to item 1.

[24]The author annotated the data collected with the *Marvin* dialog system using the tool shown in figure 13-2.

4. *Reuse and extension of existing structures, if reasonable.* If (parts of) an existing hypothesis structure can be used to represent content from the user input, the reuse is preferred over the substitution by a comparable structure. Often, (parts of) an existing hypothesis structure can be used as skeleton to represent the user input integrated with the previous discourse. This skeleton is then extended to include additional content from the user input.

5. *Maximization of the entropy on the user input.* A semantic representation is derived from the user input. After performing an appropriate segmentation of the input, each segment is associated with a semantic entity. Alternatively, a segment may not contribute to the set of semantic entities. During the manual selection of the best hypothesis, hypotheses that include a more complete coverage of the input with segments related to semantic entities are preferred. For spoken language input, i.e. sentences which can be completely parsed by the NLU without leaving parts unconsidered are favored.

A screenshot of the annotation tool is given in figure 13-2. During the annotation, the hypothesis that has been actually selected during the interaction must be considered as discourse history when dealing with the next turn. I.e. the discourse history being relevant for the annotation of the next turn may differ from hypotheses actually rated best by the annotator for previous turns.

13.1.3 Ranks and Equal Scores. To compare the performance of different sets of weights $(\omega_M)_{\text{model } M}$, a two-stage rating is applied. The major component of the rating is defined by the mean rank of the best hypothesis[25] on the corpus. An optimal set of weights yields to a mean rank of 1; for each turn, the measure given the weights top-rates the hypothesis selected by the annotator. The second element of the two-stage rating is the mean number of hypotheses rated equally than the hypothesis selected by the annotator. Ideally, the selected hypothesis has a unique score, so it can be distinguished from its rivals.

13.1.4 Parameter Study on Performance of Single Ratings. The influence of single ratings on the selection process is investigated using a parameter study. Between 1 and 4 weights were given a non-zero weight. The remaining weights keep a zero weight and do not contribute in the selection process.
Since we focus on the analysis of the relation between the selection performance and the single measures, the complete corpus $\mathtt{C}_1$ is taken as evaluation corpus.
For comparison, an optimal set of weights $(\omega_M)_{\text{model } M}$ is estimated on corpus $\mathtt{C}_1$ using Grail. The performance given this set of weights indicates the optimal achievable selection result. However, the performance of the selection algorithm is also evaluated later by estimating the weights on a training corpus and the evaluation of the weights on a disjoint evaluation corpus, section 13.2.

The selection measure of the latest versions of the *Marvin* dialog system is a combination of 14 single ratings. The parameter study, which was carried out on the corpus $\mathtt{C}_1$, evaluated six different values for each weight: -10, -1, 0, 1, 10, 100. Combinations of up to 4 weights with a non-zero value are considered.
Figure 13-3 contains four plots to illustrate the distribution of the two-stage rating for different sets of weights. Figure 13-4 integrates these four plots, and also zooms closer to the interesting area near to the origin. The two-stage rating translates as follows into the plots: the abscissa gives the mean rank of the hypothesis structures selected by the annotator. The ordinate contains the mean number of rivals with the same score according to the underlying measure given a certain

[25] I.e. the hypothesis selected as best hypothesis by the annotator.

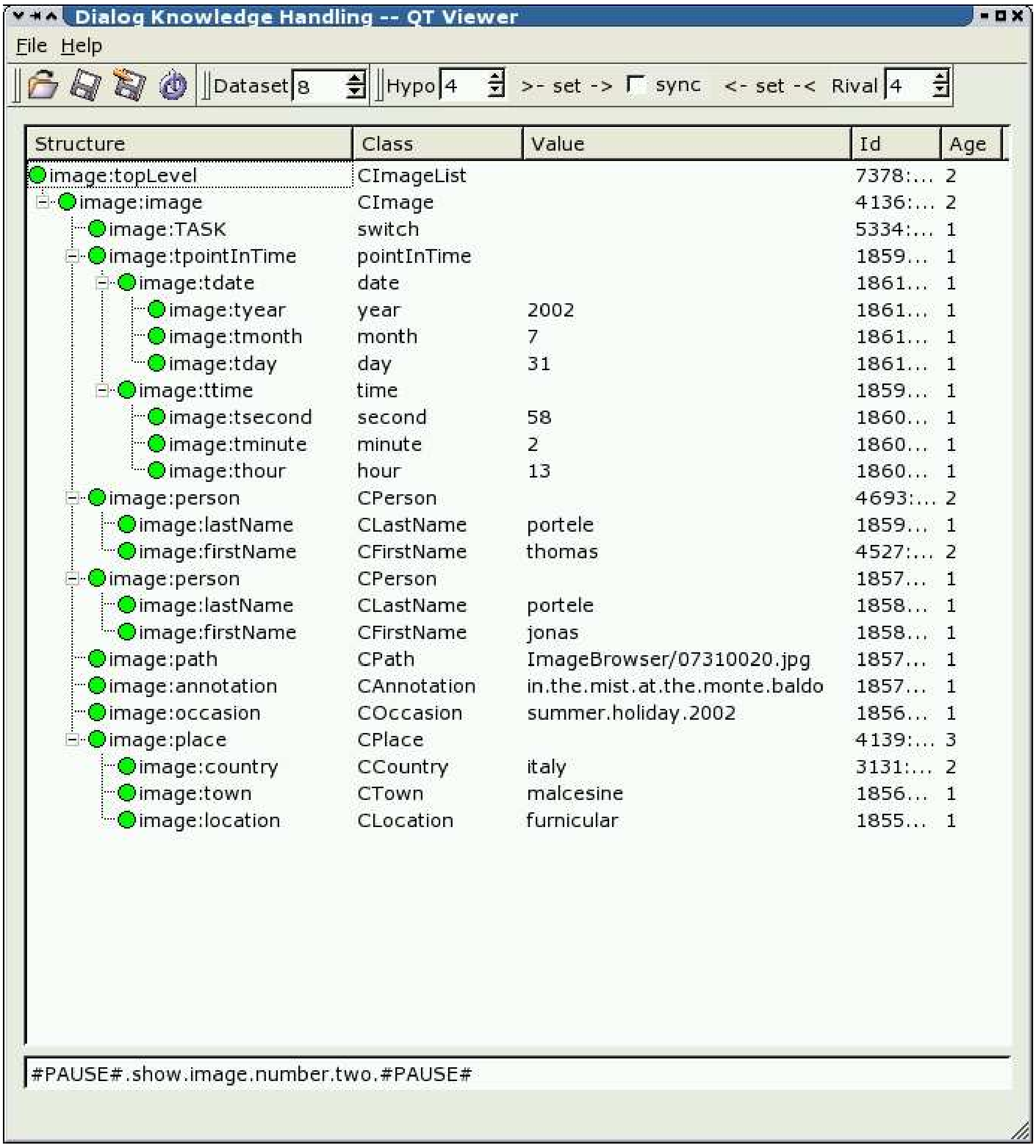

Figure 13-2: Screenshot of the hypothesis structure viewer and annotation tool. The user selects the different sets of hypotheses belonging to a certain user input via 'Dataset' (here: #8). A hypothesis is selected with the 'Hypo' property (#4), the user can compare the selected hypothesis with another hypothesis using the 'Rival' property (here: equals selected 'Hypo'). The bottom line states the recognition result.

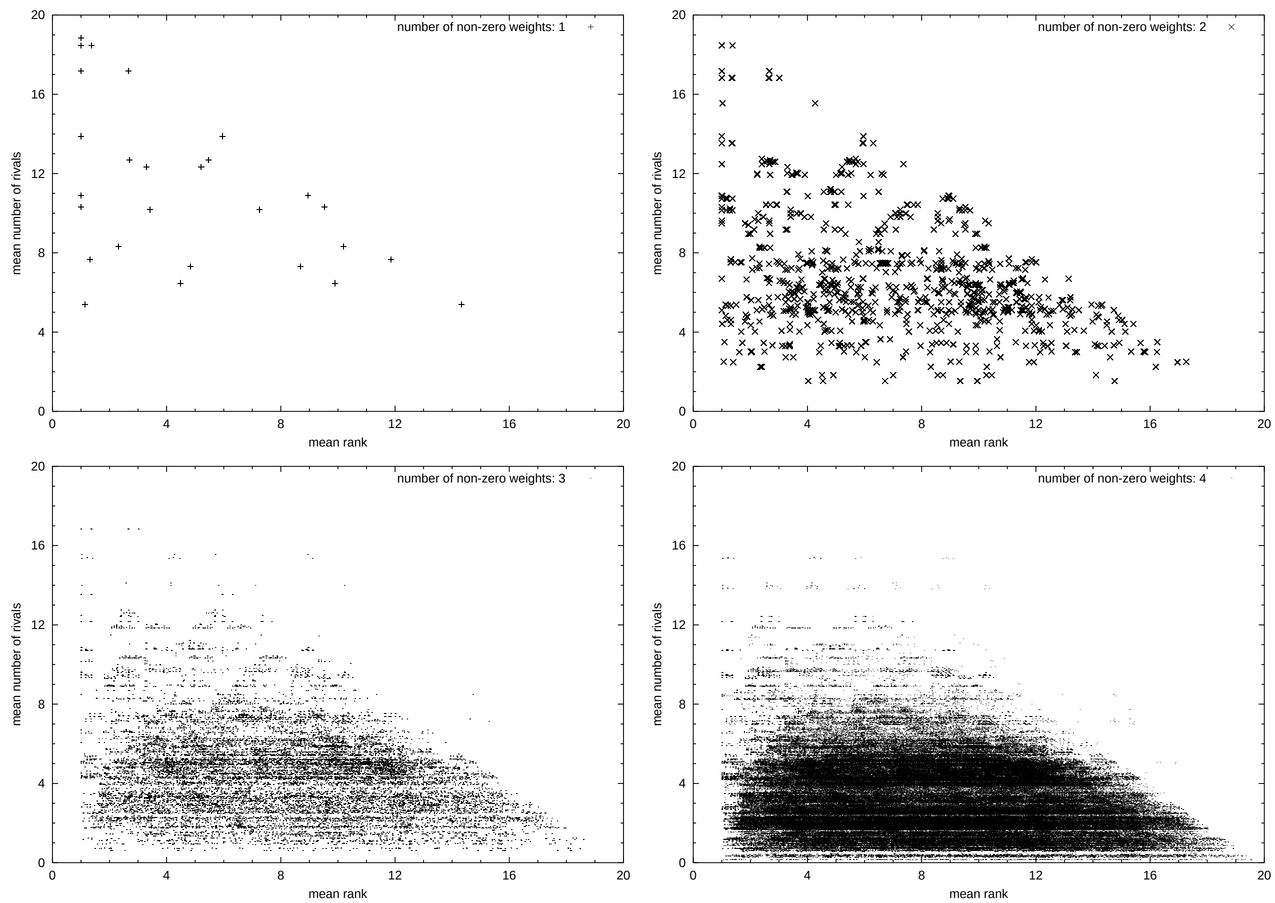

Figure 13-3: Mean rank and mean number of rivals with the same score than the hypothesis selected by the annotator for a variety of weight setups $(\omega_M)_{\text{model } M}$ on the corpus $\mathtt{C}_1$. Each marker represents a single setup. The weights ω_M take one of the values $\{-10; -1; 0; 1; 10; 100\}$. The figures contain setups with up to four non-zero weights. The plot in the upper left contains the results of weight setups with *(a)* a single non-zero weight. In the upper right, *(b)* two weights carry a non-zero value. Setups with *(c)* three non-zero weights are contained in the lower left plot. The remaining plot in the lower right shows setups with *(d)* four non-zero weights.

set of weights[26].

In the optimal case, the hypothesis selected by the annotator is rated top for each turn by the measure given a certain set of weights. Thus, the closer the mean rank of the selected hypotheses is to 1, the better. Also, a unique hypothesis is preferred, i.e. the mean number of rivals should be as close as possible to 0. To summarize, we search for weight configurations that lead to points close to the origin of the graphs.

The figures contain around 1.3 million evaluations of weight sets. Table 13-3 picks out weight setups as close as possible to the origin. A summary of the single ratings, which contribute to the overall rating, and their related weights can be found in table 13-2. The need to balance the mean rank and mean number of rivals is taken into account by including a weight configuration that leads to a better mean rank, but the mean number of rivals may be much

[26]The selected hypothesis structure itself is not counted, i.e. if there are no other hypotheses with the same score at all, the number of rivals would be 0.

rating	*weight*	*source*	*description*
$R_{\text{ASR\&NLU}}$	ω_1	ASR and NLU	combination of internal ratings from ASR and NLU
R_{domain}	ω_2	`domain model`	relation of a set of ontological objects by the `domain model`
$R_{\text{mean age}}$	ω_3	`hypo.selection`	mean age of DOs in a hypothesis structure (HS)
$R_{\text{mean age inner DOs}}$	ω_4	`hypo.selection`	mean age of *inner* DOs in a HS
$R_{\text{mean age leaf DOs}}$	ω_5	`hypo.selection`	mean age of *leaf* DOs in a HS
$R_{\text{new leaves to new DOs}}$	ω_6	`hypo.selection`	fraction of the new *leaf* DOs compared to all new DOs in a HS
$R_{\text{new vs. existing leaves}}$	ω_7	`hypo.selection`	compare new leaf DOs with existing leaf DOs in a HS
R_{tasks}	ω_8	`hypo.selection`	number and age of task objects in a HS
$R_{\text{different root domains}}$	ω_9	`hypo.selection`	domains of the root-DOs of the o^2I-Trees in a HS
$R_{\text{ratio root-/all-DO domains}}$	ω_{10}	`hypo.selection`	fraction of root-DO domains compared to all DO domains in a HS
$R_{\text{ratio leaf with/without value}}$	ω_{11}	`hypo.selection`	fraction of leaf DOs that contain a value compared to all leaf DOs in a HS
$R_{\text{properly defined}}$	ω_{12}	`hypo.selection`	fraction of properly instantiated DOs compared to all DOs in a HS
$R_{\text{ex. of list ref.}}$	ω_{13}	`hypo.selection`	existence of a list reference in a HS
$R_{\text{seg. and underst. of input}}$	ω_{14}	`hypo.selection`	fraction of segments with an associated semantic entity compared to total number of segments in a segmentation of the recognition result by the NLU

Table 13-2: Overview of the single ratings that contribute to the overall measure for the hypothesis selection in the *Marvin* dialog system, see equation (11.9). The table also shows a variable, which will be used to refer to certain ratings, the source where a rating is computed, and a rough description. A more detailed discussion of these ratings was presented in section 11.

worse.
The setups, which yield to the optimal mean rank of 1.0, share a relatively high mean number of rivals compared to other reasonable setups (around twice as much or more).
The more non-zero weights are permitted, the closer the mean rank of a balanced setup approaches 1, while, at the same time, the mean number of rivals approaches 0. The mean number of rivals with the same score generally never reaches exactly 0: during the processing, the 'same' hypotheses with respect to the underlying measure as the hypothesis selected by the annotator may be generated. In addition, different paths in processing could generate equal hypotheses. Even though some components remove equal hypotheses, different paths in creation may prevent the removal of all equal hypotheses.

The weight ω_1 in table 13-3 determines the contribution of the mutual score of ASR and NLU[27]. In setups with a single non-zero weight, a non-zero ω_1 yields the best results.
The rating related to the weight ω_{10} is based on the ratio of the number of different domains in

[27]The ratings related to ω_i are introduced in section 11, see also table 13-2.

ω_1	ω_2	ω_3	ω_4	ω_5	ω_6	ω_7	ω_8	ω_9	ω_{10}	ω_{11}	ω_{12}	ω_{13}	ω_{14}	**mean rank**	**mean num. of rivals**
+														1.13333	5.38667
													+	1.0	10.3067
+									+					1.06667	2.50667
+							+							1.4	2.48
									+				+	1.0	5.10667
+							+		+					1.06667	1.17333
+		−					+							1.49333	0.72
							+	+	+					1.0	4.28
+		−						+	+					1.02667	0.72
+		−					+		+					1.05333	0.146667
							+	+	+				+	1.0	2.68
+	+	−	−	+	−	+	+	+	+	+	+	+	+	1.01333	0.106667

Table 13-3: Selection from the set of weight setups $(\omega_M)_{\text{model } M}$ shown in figure 13-3. The qualitative representation of the weights uses '+' to indicate a positive weight and '−' to indicate a negative weight. The other weights are zero. Setups with the same number of non-zero weights are grouped together, from 1 non-zero weight up to 4 non-zero weights. For comparison, the results of an optimal weight setup estimated on the corpus $\mathtt{C}_1$ by GRAIL is given in the last row. The exact weights are given in table A2-1 (page 250).

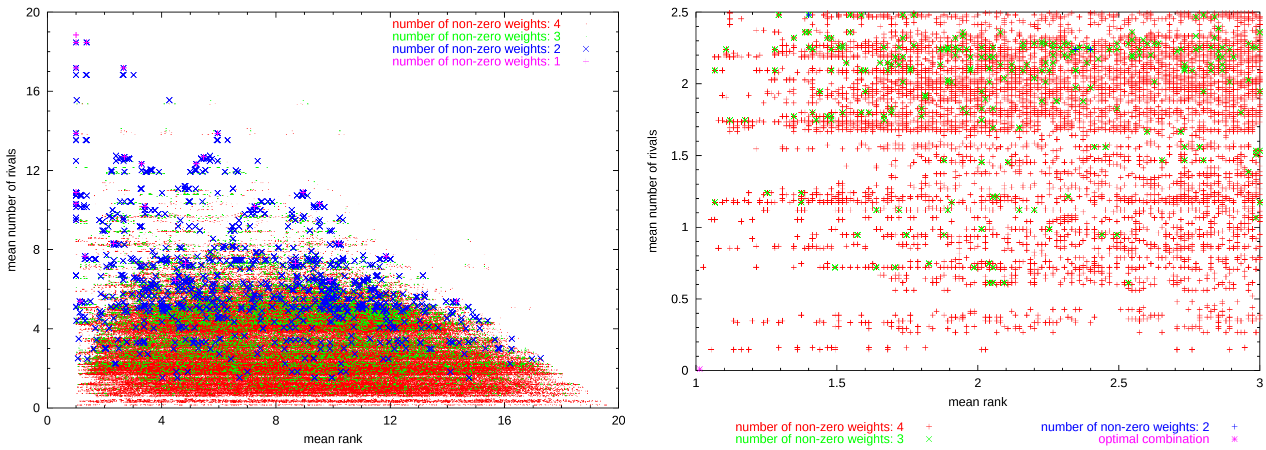

Figure 13-4: Integration of the plots in figure 13-3 into a single plot. On the left, *(a)* the same viewport is shown while the plot on the right, *(b)* zooms into the plot at the origin.

the root-DO vs. the number of different domains in all DOs of the given hypothesis structure ($R_{\text{ratio root-/all-DO domains}}$). This weight rounds up ω_1 when searching for an optimal solution for two non-zero weights.
If three non-zero weights are considered, a task related score comes into play: ω_8 (related rating: R_{tasks}).
In case of four non-zero weights, the weight ω_9 related to the number of domains of the root-DOs in a hypothesis structure ($R_{\text{different root domains}}$) and the mean age ω_3 are relevant ($R_{\text{mean age}}$). The latter gets a negative non-zero value.
Also, weight ω_{14} related to the segmentation and understanding of the ASR result in the NLU appears in the table for setups with the optimal mean rank 1 ($R_{\text{seg. and underst. of input}}$).

We can observe that filling up the setups with non-zero weights – up to four non-zero weights – takes the previous setup of non-zero weights as basis. This consistency also seems to hold for the trained optimal setup, even though the exact values of the weights differ.

The last line of the table states the optimal result given by Grail. The algorithm is free to use non-zero weights for all ratings. It however strongly focuses on the mean rank, the mean number of rivals is clearly subordinate. Trained on this corpus, the mean rank comes close to 1. In addition, the mean number of rivals in this optimal solution is very close to 0.

To summarize, the ratings actually applied in the *Marvin* dialog system to compute the overall rating of a hypothesis structure enable the system to position the hypothesis selected by the annotator mostly on top of the set of all hypotheses for a turn. Furthermore, given an appropriate set of weights, the measure can distinguish between most hypotheses, i.e. the ratings of different hypotheses differ in general.

13.2. Performance of Weight Sets on an Evaluation Corpus

The optimal weight setup $(\omega_M)_{\text{model } M}$ contained in table 13-3 is used in further interactions of users with the dialog system. From these interactions, another corpus named $\mathtt{C}_2$ was generated. The corpus $\mathtt{C}_2$ is annotated analog to corpus $\mathtt{C}_1$. The characteristics of corpus $\mathtt{C}_2$ are summarized in table 13-4. It includes more hypotheses than $\mathtt{C}_1$. Also, the number of hypotheses per turn

number of turns	116
total number of hypotheses	3 065
mean number of hypotheses per turn	26.4
number of turns with only 1 hypothesis	4
number of turns with 50 or more hypotheses	27

Table 13-4: Characteristics of the data corpus $\mathtt{C}_2$ for evaluation of the performance of weight sets derived using corpus $\mathtt{C}_1$.

is higher than for corpus $\mathtt{C}_1$. The distribution of turns with a certain number of hypotheses is shown in figure 13-5. It should be noted that the number of turns with more the 50 hypotheses is clearly higher than for $\mathtt{C}_1$. Together with the higher number of hypotheses per turn, we can derive that there are in average more rivals per single turn. This may lead to a higher mean rank and/or mean number of rivals.

Table 13-5 applies the weight sets underlying table 13-3 on corpus $\mathtt{C}_2$. I.e. corpus $\mathtt{C}_1$ served as training corpus while the disjoint corpus $\mathtt{C}_2$ serves as evaluation corpus. On the evaluation corpus $\mathtt{C}_2$, the same trends can be observed than for the training corpus $\mathtt{C}_1$. For both, the mean rank and the mean number of rivals, the evaluation corpus shows higher results. For the

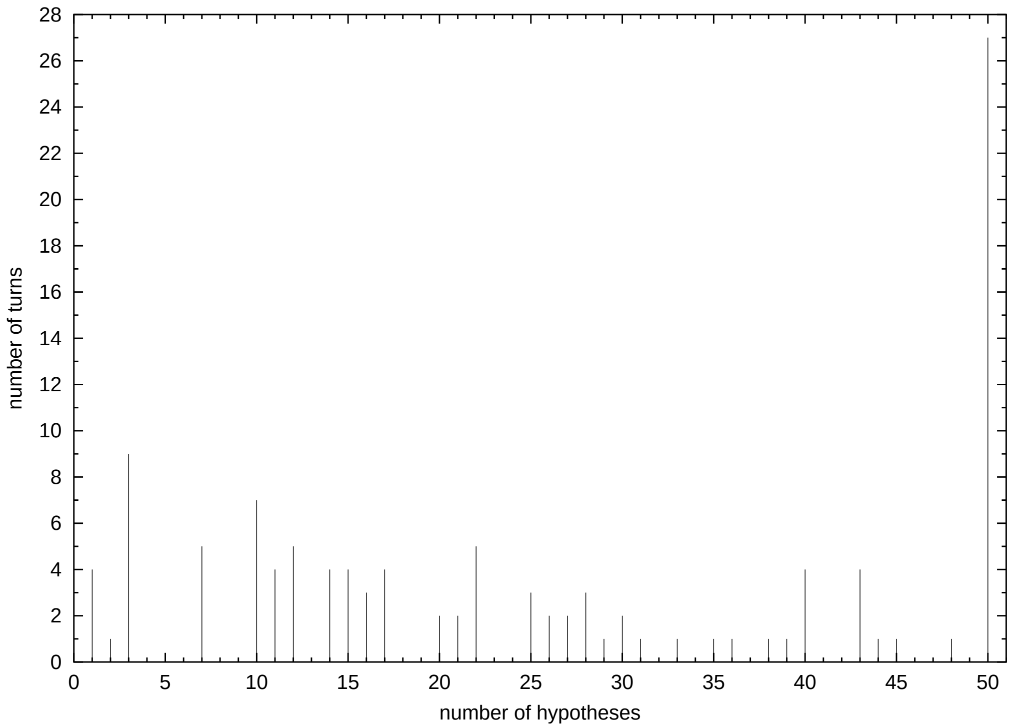

Figure 13-5: Distribution of the number of hypotheses per single turn of the corpus C_2 summarized by table 13-4.

weight setups with a restricted number of non-zero weights, the degeneration is stronger than for the automatically estimated setup. E.g. the first setup with 4 non-zero weights the mean rank worsens from 1.03 towards 1.45. In addition, the mean number of rivals degrades from 0.72 to 1.35. For the automatically computed weight configuration, the mean rank on the training corpus C_1 is 1.01 and on the evaluation corpus C_2 1.1. The corresponding mean numbers of rivals are 0.11 and 0.38 respectively.

To conclude, we can observe a consistent behavior of the different weight setups for these corpora. The automatic estimation of weights with GRAIL yields to the best performance on the training corpus. On the evaluation corpus, the differences between the automatic method and the weight sets from the parameter study with a limited number of non-zero weights becomes even more evident.

For completeness, a sub-corpus $C_{2/EPG+HDRec}$ of corpus C_2 is considered. It includes mainly interactions with the EPG and HD RECORDER application; in total 28 turns are contained with a mean number of 26.6 hypotheses per turn. Some of the results for the considered weight sets are summarized in table 13-6. The results are in general worse than the results for corpus C_2. Correspondingly, the results on the complementary corpus of $C_{2/EPG+HDRec}$ in C_2 are better than for the overall corpus C_2, e.g. the mean rank is 1.08 and the mean number of rivals with the same rank is 0.35 on the 88 turns given the automatically computed (optimal) weight set. However, training corpus C_1 is not equally distributed with respect to the applications, the EPG and HD RECORDER application are underrepresented. This could lead to the slightly worse performance on $C_{2/EPG+HDRec}$, together with a large proportion of turns with many hypotheses (half of the turns have 25 hypotheses or more).

ω_1	ω_2	ω_3	ω_4	ω_5	ω_6	ω_7	ω_8	ω_9	ω_{10}	ω_{11}	ω_{12}	ω_{13}	ω_{14}	**mean rank**	**mean num. of rivals**
+														2.80909	9.72727
													+	1.0	16.6182
+									+					1.76364	4.06364
+							+							1.81818	2.74545
									+				+	1.0	7.80909
+							+		+					1.53636	1.5
+		−					+							1.8	1.05455
							+	+	+					1.0	4.3
+		−						+	+					1.44545	1.35455
+		−					+		+					1.52727	0.463636
							+	+	+				+	1.0	2.70909
+	+	−	−	+	−	+	+	+	+	+	+	+	+	1.1	0.381818

Table 13-5: The weight sets underlying the result shown in table 13-3 are applied on C_2. However, again the qualitative representation is used for clarity. The exact weights are given in table A2-1 (page 250).

ω_1	ω_2	ω_3	ω_4	ω_5	ω_6	ω_7	ω_8	ω_9	ω_{10}	ω_{11}	ω_{12}	ω_{13}	ω_{14}	**mean rank**	**mean num. of rivals**
+														3.26923	10.0385
+									+					2.65385	4.61538
+							+							3.03846	3.0
+							+		+					2.5	1.5
+		−					+							3.07692	1.19231
+		−						+	+					1.65385	1.65385
+		−					+		+					2.19231	0.538462
+	+	−	−	+	−	+	+	+	+	+	+	+	+	1.15385	0.5

Table 13-6: Performance of selected weight sets on the sub-corpus $C_{2/EPG+HDRec}$ of corpus C_2. Results for corpus C_2 were given in table 13-5.

14. Distribution of Hypotheses During the Processing of the User Input

The *Marvin* dialog system has three units that have major influence on the generation of hypotheses for a certain user input. These units allow to influence the generation through parameterization.

The `ASR` and `NLU` together define the first unit. The end result provided by these components is a variety of interpretations of the input represented in a lattice. Yet, we will not change the parameterization of these components directly, but restrict the number of hypotheses being generated from this lattice.

The second unit, which mainly contributes to the generation of new hypotheses, is the `domain model`. The number of possible relations of a set of DOs can be restricted. I.e. a limited number of relations is returned for a request. The actually returned relations are selected according to an internal measure of the `domain model`, only the n best are included.

The `dialog knowledge processing` also includes a parameterization that influences the hypothesis generation process. The parameterization relates to the age of objects in the `discourse` `memory` to determine the relevance for the integration with the present user input.

14.1. Corpora and Methodology

During the iterative development of the *Marvin* dialog system, the initial parameterization evolved. It defines a compromise between speed, i.e. time required to process the user input, and the number of available hypotheses for a given input. It served as the basis of the evaluation of the distribution of hypotheses during the processing of the user input. The parameters for a single unit are altered, and the other parameters are kept fixed on the basis of the initial parameter configuration.

To obtain comparable results, the distribution of hypotheses at different stages are computed on the two corpora C_1 and C_2. The initial parameterization is given in table 14-1 by setup 1 and 2 respectively. Instead of speaking directly into the microphone, the user input is emulated by feeding the result of the ASR one by one into the system. The word-lattices used for this purpose are taken from the interactions with the dialog system to generate the corpora C_1 and C_2.

The change in the (number of) generated hypotheses could also lead to a different hypothesis being actually selected compared to the original interactions to generate C_1 and C_2. Therefore, the corpora may differ in some turns or in the overall number of turns. If the processing result differed, but the interaction could be continued in a meaningful way, then the interaction is continued. If a meaningful continuation is not possible, steps that build upon such a failed input are omitted.

To count the number of hypotheses transferred between certain components, we make use of capabilities of the platform underlying the system: MultiPlatform can log the messages being sent between the modules. The number of hypotheses transferred between two components can be computed from these logs. However, the logs only contain the hypotheses being actually transferred. The request-/response-communication towards the `domain model` includes a caching mechanism, thus reduces the number of hypotheses interchanged between the `domain model` and another module.

In addition, some modules check their output before delivery towards the next component, and suppress hypotheses that have been sent already before. To summarize, the number of hypotheses actually generated during the processing of the user input is generally higher than indicated in table 14-2 due to the filtering of equal hypotheses.

14.2. Evolution of the Number of Hypotheses

The quantitative examination of the number of hypotheses is performed for a variety of parameter setups. We alter parameters of different components of the *Marvin* dialog system. Four components are primarily affected by the variation: `ASR` together with the `NLU`, `transformation`, `dialog knowledge processing`, and `domain model`.

The setups used for the examination are given in table 14-1. The corpora C_1 and C_2 have been collected from interactions of a user with the dialog system. The underlying parameter setups that have been applied during the data collection are setup 1 and setup 2 respectively. These setups serve as the basis to investigate the effects of different parameter settings on the number of generated hypotheses.

setup	corpus	ASR/ NLU **n-best list length**	dialog knowledge processing max-Age	maxNum-Different-Domains	memoryLength-InsideDomain	memoryLength-OutsideDomain	domain model **number of results limit**
1	C_1	5	2	2	1	1	3
1-1	C_1	**2**	2	2	1	1	3
1-2	C_1	**10**	2	2	1	1	3
1-3	C_1	5	2	2	1	1	**1**
1-4	C_1	5	2	2	1	1	**10**
1-5	C_1	5	**1**	**1**	**1**	**0**	3
1-6	C_1	5	**3**	**3**	**2**	**2**	3
1-7	C_1	**2**	**1**	**1**	**1**	**0**	**1**
1-8	C_1	**10**	**3**	**3**	**2**	**2**	**10**
2	C_2	5	4	2	3	2	3
2-1	C_2	**2**	4	2	3	2	3
2-2	C_2	**10**	4	2	3	2	3
2-3	C_2	5	4	2	3	2	**1**
2-4	C_2	5	4	2	3	2	**10**
2-5	C_2	5	**2**	**1**	**2**	**0**	3
2-6	C_2	5	**6**	**3**	**4**	**3**	3
2-7	C_2	**2**	**2**	**1**	**2**	**0**	**1**
2-8	C_2	**10**	**6**	**3**	**4**	**3**	**10**

Table 14-1: Parameter setups used during the quantitative evaluation of the hypotheses generation. The setups 1 and 2 are the starting point for the variation of parameters on the corpora C_1 and C_2 respectively. For each component, smaller and larger parameters are considered. For the `dialog knowledge processing`, the complete related parameter set is varied as a whole (setups 1-5, 1-6, 2-5, and 2-6). For comparison, also the setups based on the extreme parameters are evaluated (setups 1-7 and 2-7 contain the minimal values, 1-8 and 2-8 the maximal values). The parameter 'n-best list length' limits the number of hypotheses extracted from the analysis-lattice resulting from the `ASR` and `NLU`. The parameters of the `dialog knowledge processing` have been discussed in more detail in section 10.5. For the `domain model`, the maximum number of relations for a single set of DOs is determined by the 'number of results limit'. (*Remark:* 'memoryLengthInsideDomain' and 'memoryLengthOutsideDomain' count the number of previous turns in the history, i.e. these values are 1 smaller than the ages.)

14.2.1 Parameters. The final outcome of the processing by the `ASR` and `NLU` is an analysis-lattice that contains semantic entities computed on the basis of the speech recognition results. The first n best paths through the lattice are further processed. The parameter 'n-best list length' defines an upper bound for the number of hypotheses being extracted from the lattice.

Similarly, the maximum number of responses generated by the `domain model` for each request can be limited. Changing the parameter 'number of results limit' has effects on the number of hypotheses generated inside the `transformation` and `dialog knowledge processing`, since these components make use of the `domain model` to establish a relationship between DOs.
The parameterization of the `domain model` affects the number of hypotheses generated by the `transformation`, yet the `transformation` has no additional internal parameters to control the number of generated hypotheses.

The parameters of the `dialog knowledge processing` are discussed in more detail in section 10.5. The parameters influence the number of previous turns used for the integration with the present user input (`maxAge`, `memoryLengthInsideDomain`, and `memoryLengthOutsideDomain`). In addition, the total number of different domains in a single hypothesis can be limited (`maxNumDifferentDomains`). However, this limit does not affect the number of domains extracted directly from the user input but only deal with additional domains added through the history. We consider three different combinations of the parameters related to the `dialog knowledge processing`.

14.2.2 Execution of the Experiments. The evaluation is based on the corpora C_1 and C_2. From the data, which was logged during the interaction of a user with the dialog system, the lattices resulting from the recognition and analysis are extracted. After adjusting the parameters, the lattices are fed again into the system.

Changes in the parameter setups heavily influence the number of generated hypotheses, and thus the set of available hypotheses for user input as such. Therefore, also the hypothesis selection is affected. A change in the selected hypotheses generally has impact on the discourse. The lattices are manually fed into the system, the outcome is judged manually if the next lattice representing the next user input is adequate.
In case of a negative assessment, the next lattices in the sequence are considered sequentially. Another option is to stop the present discourse.
During the execution of the experiments, some lattices were skipped to continue the discourse in a meaningful way. In only one case, the discourse was stopped before the regular last lattice. Though in a significant number of cases, the reaction on some user input differed from the reaction obtained using setup 1 and 2 respectively[28].

14.2.3 Number of Hypotheses. The evolution of the number of hypotheses is investigated by means of the results given in table 14-2. The table lists the mean number of output hypotheses generated for each single input hypothesis at certain stages in the processing chain ('multiplier'). The processing stages are represented by components.
In addition, some of the components form a processing sequence. For this sequence, the combination of the multipliers is given as the mean accumulated number of hypotheses ('accu.'): starting from a single user input, several components multiply the number of hypotheses. Obviously, the product of the multipliers should result in the accumulated number of hypotheses. However, the accumulated number of hypotheses slightly differs from the products. This is due to two reasons: *(i)* the caching mechanisms and *(ii)* the filtering of duplicate hypotheses applied in some

[28]Differences to the 'original' reactions are not annotated, and thus not quantified.

setup	ASR/ NLU multiplier	accu.	transformation multiplier	accu.	dialog knowledge processing multiplier	accu.	domain model multiplier
1	3.69	3.69	1.53	5.64	3.36	18.95	1.60
1-1	1.91	1.91	1.50	2.87	3.16	9.06	1.58
1-2	5.99	5.99	1.63	9.73	3.56	34.61	1.67
1-3	3.69	3.69	0.99	3.66	2.88	10.52	1.00
1-4	3.69	3.69	1.57	5.81	3.54	20.56	1.79
1-5	3.71	3.71	1.53	5.65	1.01	5.69	1.58
1-6	3.61	3.61	1.51	5.45	5.01	27.30	1.67
1-7	1.91	1.91	0.98	1.87	1.02	1.91	1.00
1-8	5.96	5.96	1.72	10.25	6.34	65.00	2.13
2	3.90	3.90	1.60	6.23	7.86	48.95	1.73
2-1	1.94	1.94	1.56	3.02	7.28	22.00	1.74
2-2	6.27	6.27	1.68	10.55	7.61	80.22	1.81
2-3	3.90	3.90	1.00	3.90	6.68	26.06	1.00
2-4	3.90	3.90	1.88	7.32	8.67	63.51	2.43
2-5	3.90	3.90	1.60	6.22	3.87	24.11	1.77
2-6	3.91	3.91	1.60	6.26	10.44	65.36	1.76
2-7	1.94	1.94	0.99	1.92	3.32	6.39	1.00
2-8	6.28	6.28	2.07	13.03	11.96	155.82	2.70
$1 \cup 2$	3.82	3.82	1.56	5.97	6.49	38.80	1.71

Table 14-2: Quantitative evaluation of the hypotheses generation through the components `ASR` and `NLU`, `transformation`, `dialog knowledge processing`, and `domain model`. The *multiplier* states the mean number of hypotheses generated by a component for each single input hypothesis. The accumulated number of hypotheses (*accu.*) shows the mean number of hypotheses generated for a single user input in the processing sequence from ASR to the `dialog knowledge processing`. In this chain, the `domain model` supports the `transformation` and `dialog knowledge processing` in the hypotheses generation, and is therefore indirectly contained in the related multipliers. Due to this special role, the effect on the accumulated number of hypotheses is complex and is not given in this table. The different parameterizations underlying the hypotheses generation process are given in more detail in table 14-1.

components.
The outstanding role of the `domain model` to support the `transformation` and `dialog knowledge processing` is reflected in the table – as already pointed out before, the `domain model` contributes to the increase of the number of hypotheses, but it is not a direct part of the previously mentioned linear processing sequence. Therefore, no accumulated number of hypotheses is given for this component.

The graphs given in the figures 14-1 and 14-2 illustrate the numbers listed in table 14-2, separately for the corpora C_1 and C_2. The upper graph concentrates on the single multiplier for each single component, the lower graph shows the accumulated number of hypotheses. The connected data points refer to the linear processing sequence.

Canonically, changing the 'n-best list length' directly influences the number of hypotheses extracted from the lattice generated by the `ASR` and `NLU`. The number of hypotheses extracted from the lattice stays below the given limit, because some lattices representing the user input comprise a lesser number of hypotheses than the limit specifies. This becomes more obvious for larger limits, e.g. a multiplier of 1.91 in setup 1-1 for a given limit of 2 compared to a multiplier of 5.99 in setup 1-2 at a limit of 10. One can observe that a change of the limit also affects the other components. A lower limit principally lowers the multipliers of the other components, while a higher limit increases the multiplier.
The accumulated number of hypotheses behind the `knowledge processing` clearly differs for a low and high number of hypotheses extracted from the lattices, e.g. 9.06 vs. 34.61 on corpus C_1.

For the `domain model`, the limit of responses for a single user input varied between 1 and 10. When concentrating on this parameter – setups 1-3, 2-3, 1-4, and 2-4 – the `domain model` completely utilizes the allowed range for the lower bound. Exactly one response answers each request. The dependent components show a clearly reduced number of generated hypotheses. The multiplier of the `transformation` even drops below 1. This is due to the elimination of double hypotheses, i.e. hypotheses that have already been sent. On the other hand, the increase in the number of allowed responses is not that obvious for corpus C_1, but more clear for C_2. In conjunction, the components relying on the `domain model` show no significant change for the high number of allowed responses for C_1. For C_2, the influence is much stronger, especially for the `dialog knowledge processing`. During the creation of corpus C_2, more complex tasks have been carried out by the user than during C_1. I.e. less pure command-and-control style input is used, and the history had to be taken into account more often. Also, the initial parameter setup (setup 2) was designed to allow a higher number of hypotheses per user input, see also table 14-2. This leads to more complex hypotheses with more DOs on average. For corpus C_1, the mean number of DOs per hypotheses that have been sent to the `hypothesis selection` is 13.1 (setup 1). For C_2 (setup 2), this number is 15.9 (+21.8%). This means that more complex hypotheses are created. In addition, more requests for the `domain model` are created (C_1: 5.7 requests per turn, setup 1; C_2: 9.7 requests per turn, setup 2). The additional requests therefore seem to trigger a higher number of possible relations of the DOs given in the request.

The changes in the parameter setup of the `dialog knowledge processing` show a clear effect. On C_1, the configuration of the parameter setup to ignore the history (setup 1-5) leads to a multiplier close to 1. The value is slightly above 1 because of the possible use of DOs in different domains. Similar to the setups 1, 1-1, and 1-2, setup 2-5 considers one turn from the history in addition to the present user input. The resulting multiplier (3.87) approaches the multiplier of the C_1 setups. The parameter settings that lead to high multipliers, setup 1-6 and 2-6, show a strong effect on the number of generated hypotheses. The `dialog knowledge processing` is the last component of the linear arranged components, and therefore the overall effect of the change in the `ASR` and `NLU` parameter is stronger for the given variations.

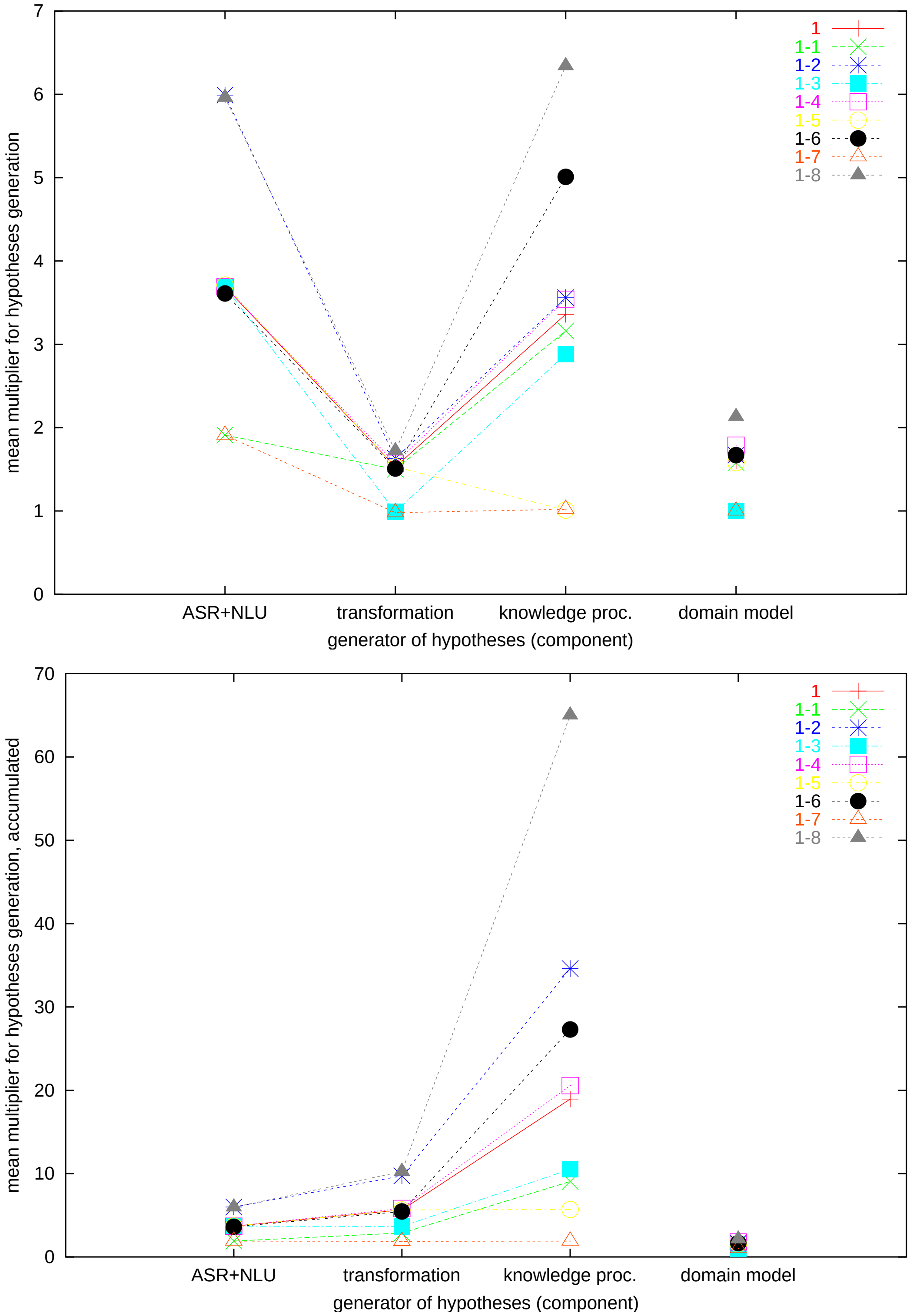

Figure 14-1: Illustration of the data from table 14-2 for corpus $\mathtt{C}_1$ (setups 1 and 1-X). The components `ASR` and `NLU`, `transformation`, and `dialog knowledge processing` form a processing sequence, which is indicated by the connections. The `domain model` stands out since it assists the latter two components during the hypotheses generation. The graphs show the multiplier (upper graph) and accumulated number of hypotheses (lower graph, multiplier for the `domain model`).

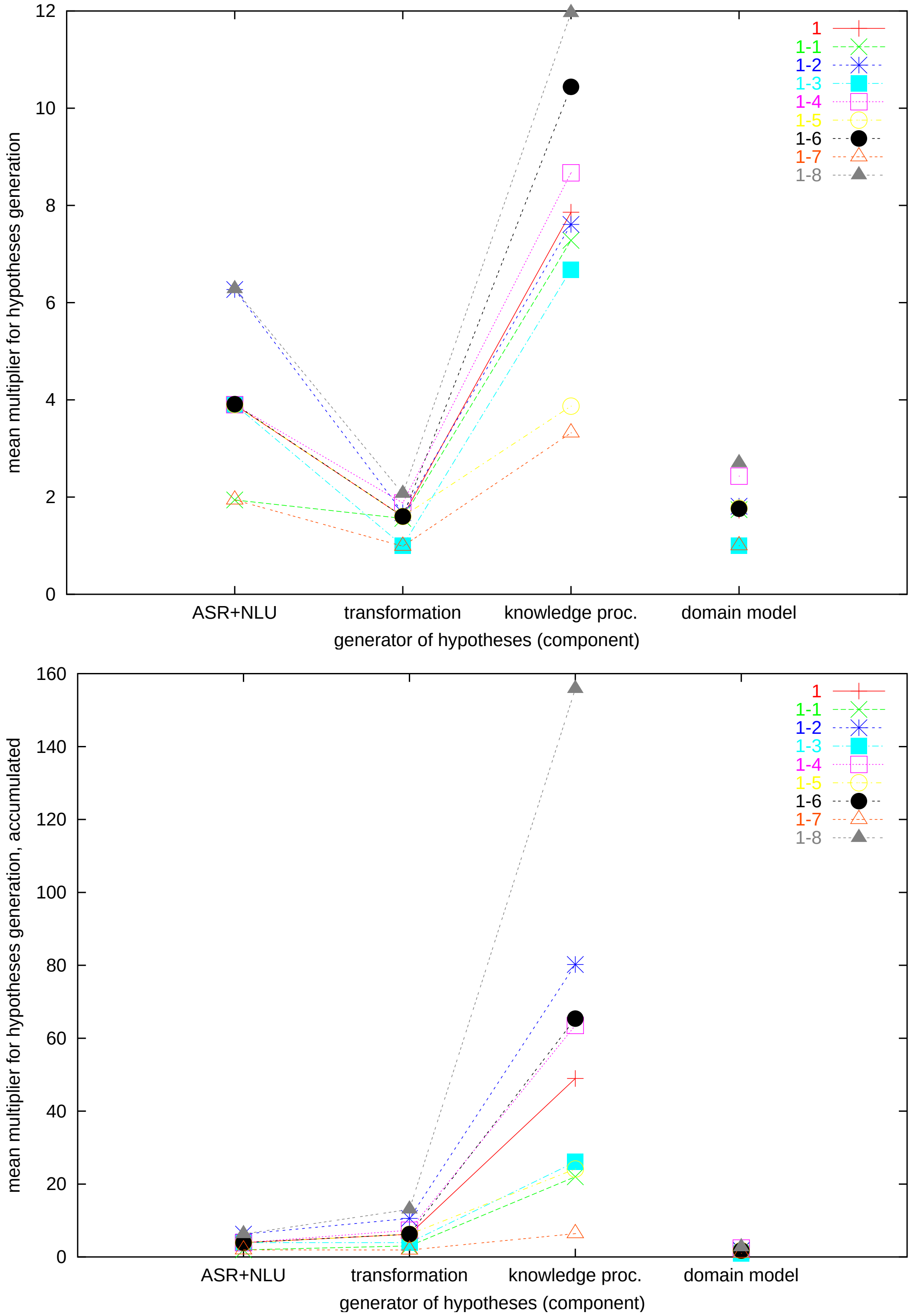

Figure 14-2: Illustration of the data from table 14-2 for corpus $\mathtt{C}_2$ (setups 2 and 2-X). The components `ASR` and `NLU`, `transformation`, and `dialog knowledge processing` form a processing sequence, which is indicated by the connections. The `domain model` stands out since it assists the latter two components during the hypotheses generation. The graphs show the multiplier (upper graph) and accumulated number of hypotheses (lower graph, multiplier for the `domain model`).

Finally, the parameter setups 1-7, 1-8, 2-7, and 2-8 combine the single parameter variations such that settings, which lead to lower multipliers, are combined (setup 1-7 and 2-7), analogously for the higher multipliers (1-8 and 2-8). The setups 1-7 (2-7) and 1-8 (2-8) form an envelope of the original setup and the setups that vary the parameters of a single unit for C_1 (C_2). The mean numbers of hypotheses generated for a single user input are between 1.91 and 65.00 for corpus C_1, and 6.39 and 155.82 for C_2. The differences between the corpora are clearly originating from the different initial parameter setups. The setups related to C_2 principally take more turns from the discourse history into account.

The overall picture shows that the range in multipliers for the evaluated settings is most obvious for the `dialog knowledge processing`, followed by the `ASR/NLU`. The latter transmits the broad range on the accumulated multiplier, due to the position at the start of the considered processing chain. The influence of the `domain model` is visible, however not that strong. The `transformation` attenuates the effects of the `domain model`.
The parameters for `ASR/NLU` and `domain model` directly give and upper bound for the multiplier of these units. The multiplier, which is the mean number of generated hypotheses per input hypothesis, is significantly below the given upper bound (e.g. setup 2-2: $6.27 < 10$, `ASR/NLU`, and setup 2-4: $2.43 < 10$, `domain model`). Especially for the `domain model`, the effect of increasing the parameter mirrors the tendency in a limited way.

15. Direct Access to the Discourse Memory by the User

The navigation according to criterion 4.2.2(1) realized inside DymaLog allows the user to explicitly move between the states of a linear ordered sequence. The states of this sequence correspond to the states of the `discourse memory` after an event was considered by all components which should react on the event. Such an event might be input by the user or some event that lets an application provide the dialog engine with some information. A state thus includes all updates of the `discourse memory` that might be performed until the reaction to an event is presented to the user (or the decision is taken not to present anything to the user).

Navigation capabilities offer the user a broader flexibility in the interaction with our dialog system. Failures in the `analysis` process, e.g. errors in ASR due to noisy environments, or changes in mind can easily be compensated by restoring previous states in the discourse even though the explicit navigation in the discourse might appear seldom in human-human communication.
Some present systems try to compensate for failures during the input recognition or analysis by detecting these problems and initiating subdialogs for correction. Yet, Denecke [2002] implements an 'undo interaction pattern' (triggered e.g. through *"no, not ... "*) that can be used to revoke the very last input.

15.1. Navigating in the Discourse

Navigation in a linear sequence of states is a well-known concept for users of today's 'normal' personal computers. Restoring previous states, or even annulling a preceding restore, is often applied in daily use. There are two prominent realizations for these concepts.
State of the art web browser like Microsoft Internet Explorer, browser based on the Mozilla Web

Browser[29], Konqueror, and Safari implement means to jump back to previous web pages and after moving backward in the page history, under certain conditions, also steps forward again in the previously discarded pages are possible.
Many present software programs provide 'undo' and 'redo' capabilities, often restricted to a number of steps. We can refer to a variety of genres in which most programs support these capabilities, e.g. the text processing and spreadsheet programs in office suites (Microsoft Office, OpenOffice Suite, ...) and graphics processing software (Photoshop, The Gimp, ...).

DymaLog realizes these concepts in the dialog engine Aide. In addition to the 'backward' and 'forward' movements inside the discourse, the 'cancel' direction is introduced. Let us look at the effects of the operations in more detail as these are applied in the dialog domain.

15.2. Effects of Navigation on the Discourse Memory

The *backward* and *cancel* operations both modify the central `discourse memory`[30]. The current state inside the `discourse memory` is discarded in favor of the state preceding that state. These operations can be applied multiple times. A snapshot representing the updated state of the discourse is transferred to the subsequent components. The snapshot is tagged such that these components can identify the operation and react on the backward or cancel operation. Since Aide decouples from the applications, the functionality of the applications cannot be accessed directly. At this point, we have to rely on the applications to implement guidelines on the behavior for backward and cancel. In case of a 'backward', applications are asked to reexecute the action, which was executed in the restored state. The application's reaction on a 'cancel' should be to cancel the last currently ongoing action – if some action is performed – and to restore the previous state inside the dialog system without executing the associated action. Note that the two navigation operations addressing past information share their effect on Aide but not on the applications.
The *forward* operation – if available – integrates a state that was previously removed from the `discourse memory` by a 'backward' or 'cancel' back into the memory. A snapshot of the updated `discourse memory` is again transferred to the subsequent components. Applications should reexecute the actions of the reinserted state. Thus, the forward operation is the direct counterpart of the backward. However, at present it can also be applied after cancel operations.

Figure 15-1 picks up the example from section 9.1 and shows the effects of two successive backward operations. After a backward operation, a forward operation might be used to restore the state before the backward operation. E.g., a forward following the user input #4 restores the state shown as result of user input #3. After two backward directed navigation commands, user input #4 and input #5, two forward operations can be applied. The first 'forward' after two 'back' would restore the state after input #4, another 'forward' retrieves the state after input #3. More than one forward operation can only be applied after a sequence of backward, and possibly also forward, operations as this section will show.
Figure 15-2 shows an abstract illustration of the effects of a backward instruction from the user on the `discourse memory` which can be inverted by a 'forward'.

[29] E.g. Mozilla Suite Browser or Mozilla Firefox.
[30] Remember that Aide features a central component (`discourse memory`) which holds discourse related knowledge, the other components are stateless with respect to discourse knowledge.

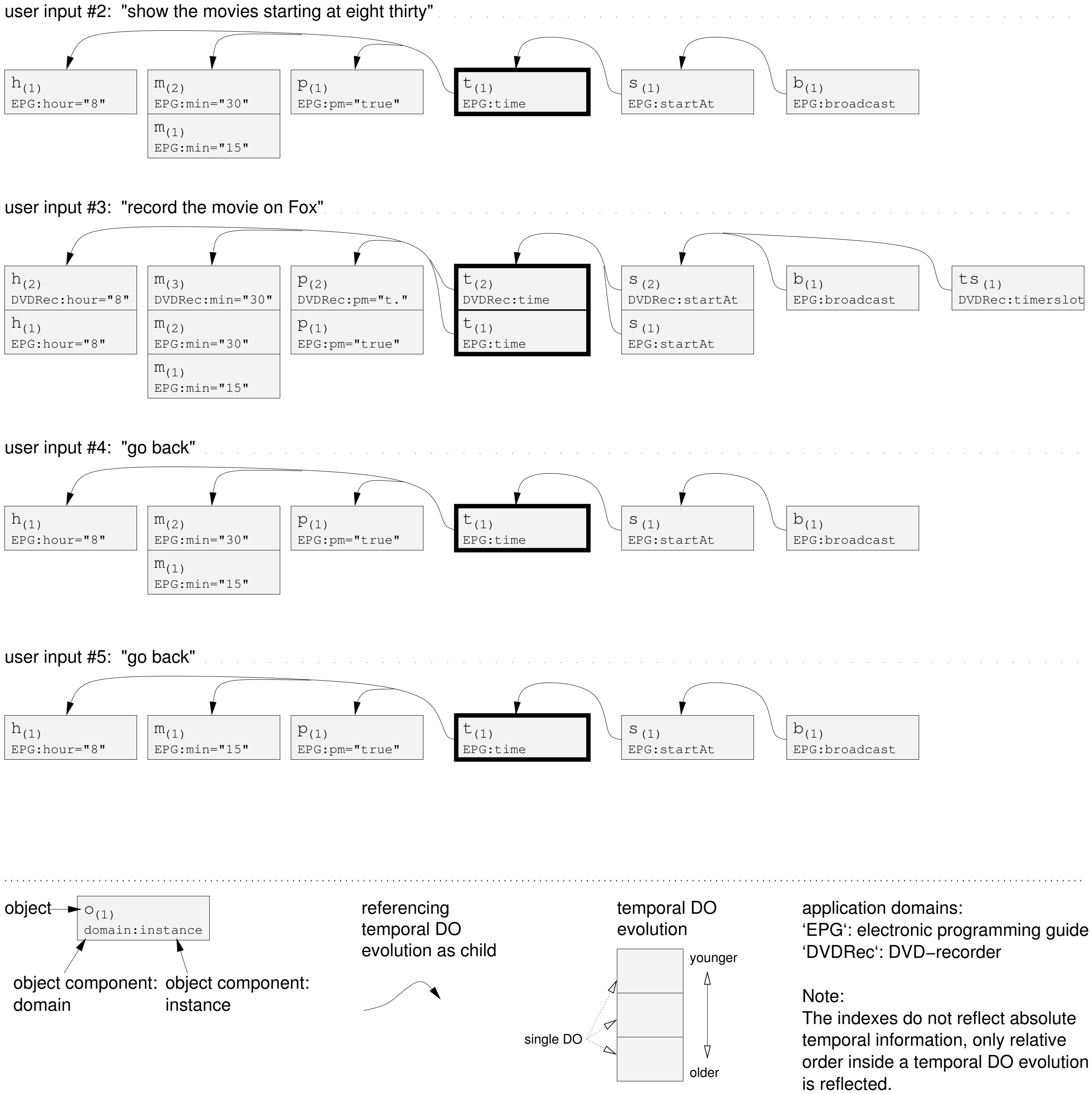

Figure 15-1: Continuation of the example given in section 9.1, especially figure 9-2. User input #2 and #3 are repeated for clarity. The figure shows the effect of input #4 *"go back"*, where the user moves back from the `DVDRec` to `EPG` again. Another *"go back"* restores the state after user input #1.

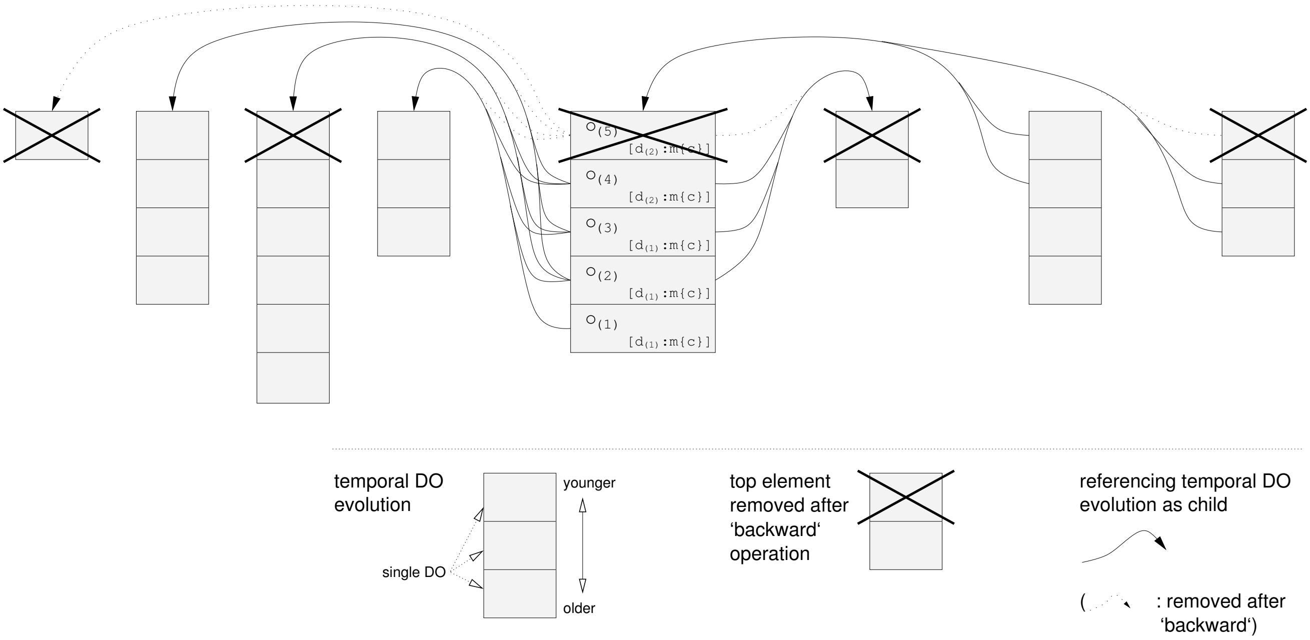

Figure 15-2: Excerpt of the `discourse memory` as introduced in figure 9-1. The user removed the most recent elements from selected temporal DO evolutions through a 'backward' operation. An immediately following 'forward' might restore the removed DOs again.

15.3. Navigation Schemes Identified for DYMALOG

The following interaction sequences might occur during the interaction of a user and the dialog system:

- Single 'backward' or 'cancel' operation between normal user input[31].
  A single backward operation is illustrated in figure 15-4, e.g. input sequence (3)→(4)→(5). After one or more user inputs, the user requests a backward operation, which restores the state before the current state. Starting from this 'old' state, the interaction is carried on as if the user pursues with normal input.

- Single 'forward' operation between normal user input.
  Figure 15-3, input sequence (13)→(14)→(15), shows a single 'forward' (14) between common user input. Since the forward is not preceded by a backward or cancel, the forward stack (section 15.4) is empty and the system remains in the state before the forward instruction (`discourse memory` state 'F').

Both realizations of navigation, 'forward/backward' and 'undo/redo', share a restriction. After a series of one or more 'backward' or 'undo' operations, at most the same number of 'forward' or 'redo' operations can only be performed if the sequence of forward directed operations is not interrupted by a deviating operation. To illustrate this, consider a text processing application. After a user entered `"hello world"` and performed an undo operation, assume that `"hello"` remains on the screen. The removed item `"world"` was put on a stack of removed elements. If the user now selects the redo operation, `"hello world"` is restored. However, if the user types

[31]The term 'normal user input' will be used to refer to user input which does not trigger a 'backward', 'cancel', or 'forward' operation.

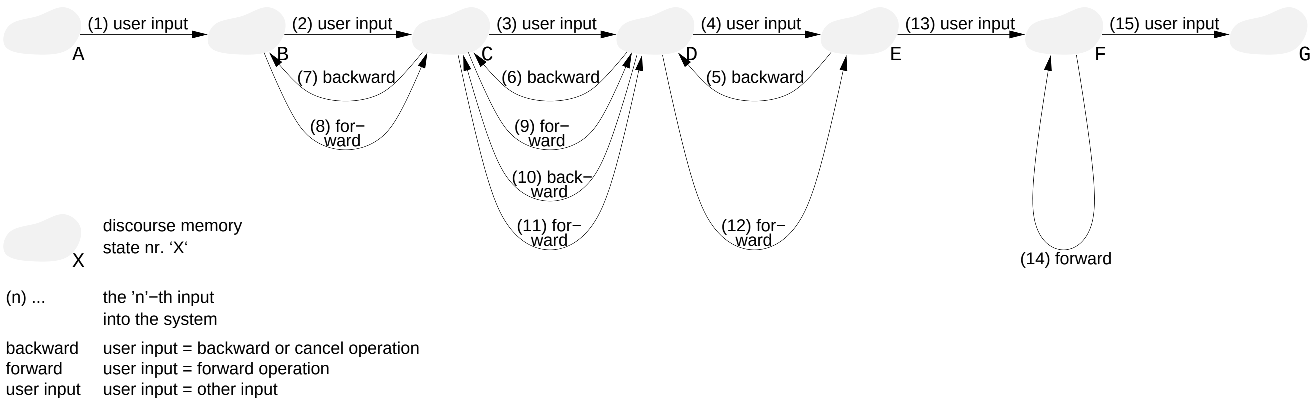

Figure 15-3: Navigation on a linear sequence of states using 'forward' and 'backward' operations mixed with normal user input. The numbers in brackets preceding the input, '(n)', numbers the input. E.g. '(7) backward' is the 7th input by the user into the system indicating that the user enters input to trigger a backward operation. Letters number the suggested states of the `discourse memory` to enable references.

`"computer"` before the redo (which leads to `"hello computer"`), the redo operation is no longer available.

- Sequence of 'backward'-'forward' operations embedded into normal user input.
  Figure 15-3 also includes examples how the usage of backward and forward operations lets the `discourse memory` jump between existing discourse states as stored in the `discourse memory`, input (4)-(13). Backward operations remove the current state from the `discourse memory` and push the state on a separate stack. A forward operation pops the top state of the stack and inserts it again into the `discourse memory`. The case of an empty forward stack was shown before (see user input (14)) and occurs if more forward than backward operations are requested in turn.
  Obviously, the cases of a single 'backward' or 'forward' operation are special cases of this case.

Figure 15-4 combines the above occurrences of navigation commands and shows how branches may appear during an interaction, leaving also dead ends. An overhang of backward-directed navigation operations leads to these branches.

15.4. Implications for the Dialog Knowledge Processing

Enabling DyMALog to support navigation operations explicitly requested by the user mainly affects the `dialog knowledge processing`, which interacts with the `discourse memory`. It hosts the update rules for these operations. Applications must honor the tags attached to hypothesis structures to inform that this hypothesis structure was derived from a navigation operation.
The `discourse memory` was extended to support navigation. In addition to the memory structure holding the system's view on the discourse, a separate stack, the *forward stack*, is introduced. It is required to enable the forward navigation and stores the states removed from the memory by backward or cancel operations.
For the present framework only one step navigation operations are supported, i.e. it is only possible to jump from one state to the direct predecessor or access the top most element from the

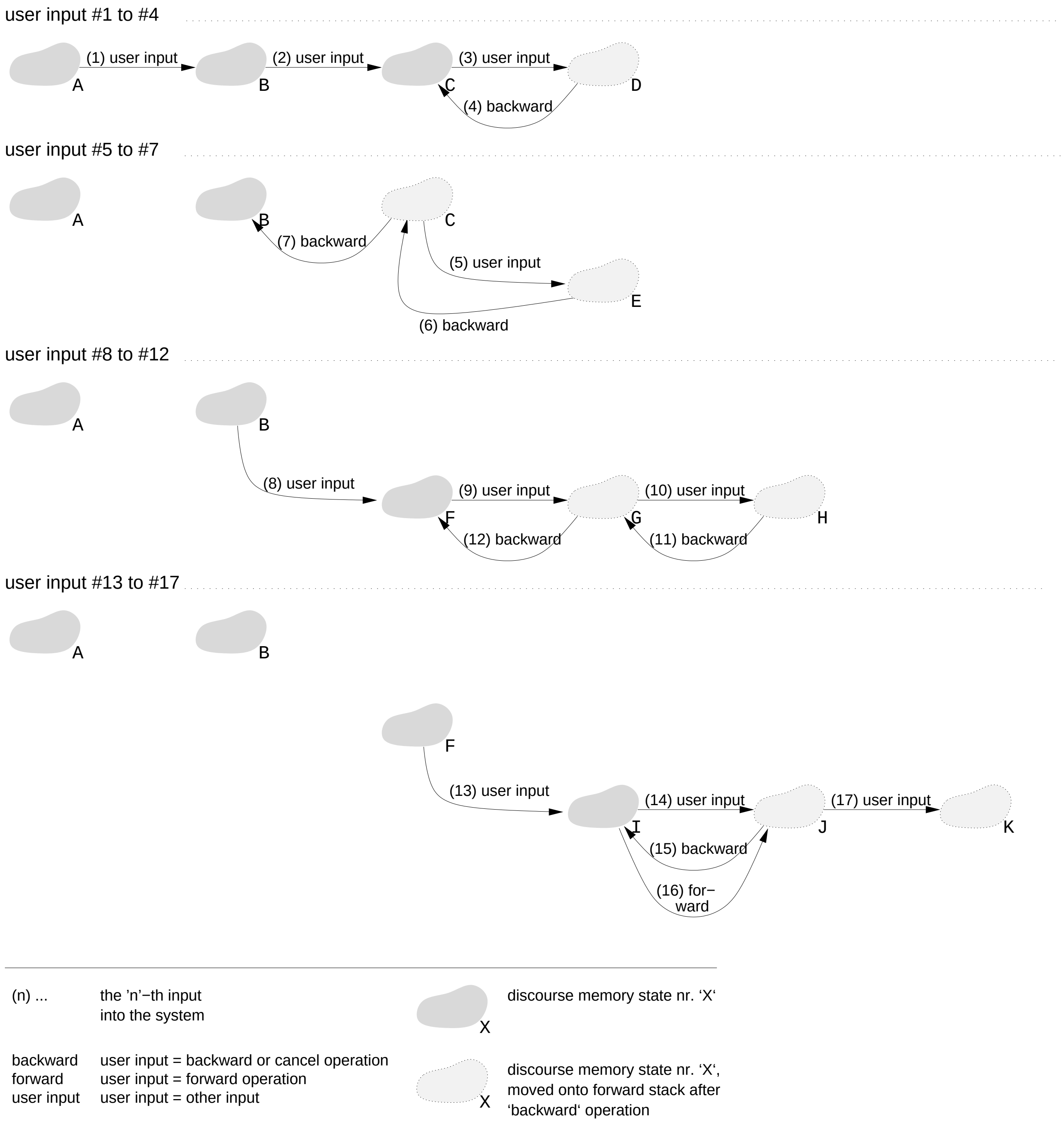

Figure 15-4: Development of a sequence of `discourse memory` states when a non-matching number of 'forward'-'backward' operations in turn are applied. E.g. the first backward operation (4) originating from state 'D' is not followed by a forward but a normal user input (5) which leads to a branch, state 'E'. The development is broken into temporal sequences to show states vanishing when a 'backward' is not followed by 'forward' but normal user input. Remember that '(n)' preceding the input numbers the user input in chronological order. The letters numbering the `knowledge states` are not directly related to a temporal sequence.

forward stack.

The processing of a user interpretation that represents a navigation operation by the `dialog knowledge processing` deviates from the processing of normal user input as presented in section 10 and section 9.2. Instead of integrating the content from the interpretation selected by the `hypothesis selection`, a modification of the state of the `discourse memory` is obtained without integrating the current user interpretation. Let us now address the question what happens inside the `dialog knowledge processing` when it receives a navigation instruction.

15.4.1 Backward and Cancel Operations. The difference between backward and cancel is concentrated in the applications, the dialog engine AIDE does not distinguish between these two. Thus, we limit ourselves to the backward operation.
As soon as the `dialog knowledge processing` receives a backward instruction, the set of DOs with age one in the discourse is determined:

$$O_{\text{current}} := \bigcup_{\substack{\text{temporal DO evolution } \mathtt{e} \\ \text{contained in the } \mathtt{discourse\ memory}}} \{o\,; \text{DO } o = \text{most recent DO in } \mathtt{e},\ o\,[age] = 1\}\,.$$

O_{current} is pushed onto the forward stack. Together with O_{current}, for each DO in O_{current} the related temporal DO evolution and its relations to children is pushed onto the forward stack. Simultaneously, for each temporal DO evolution `e` two actions are performed:

- if the most recent DO of `e` is contained in O_{current}, this DO is removed from `e`, and
- the age of each DO in `e` is reduced by 1, i.e. a DO with age 2 becomes the current DO with age 1 etc.

After the former current DOs are removed, for each temporal DO evolution that contains a DO of age 1 (thus previously of age 2) that further served as root-DO, the related o^2I-Tree is reconstructed from the `discourse memory`. The derived object-oriented interpretation trees build up a hypothesis structure that forms the snapshot of the discourse. It is forwarded to the subsequent components as the result from the current user input enriched with the information that the snapshot is the result of a backward or cancel operation.

15.4.2 Forward Operations. The reverse process of the backward operation, the forward operation, is trivial if the forward stack is empty. No modifications of the `discourse memory` are carried out and an empty hypothesis structure is forwarded to the successors as the result.
Starting from a non-empty forward stack, the top most item is reintegrated into the `discourse memory` when a forward instruction is received. The DOs from O_{current} of the top entry on the stack are used to restore a previous state of the memory. The knowledge on the relationships between these DOs and temporal DO evolutions, also being part of the forward stack, supports this restoration. The closure of this process is the removal of the top element O_{current} from the forward stack.

15.4.3 Normal User Input. Even though normal user input is not processed as the instructions above but follows the procedures from section 10, it also has influence on the navigation. When the `hypothesis selection` chooses normal user input as best hypothesis, the forward stack is cleared. For figure 15-4, this means because of the creation of a new branch (e.g. input (5), (8), and (13)) that the forward stack is cleared. In such a case – if the stack was not empty – the previous states corresponding to the elements of the stack cannot be reached by forward operations again since these states are removed from the memory (figure 15-4, `discourse memory` state 'D', 'C', 'E', 'H', or 'G').

15.4.4 Changes in Restored States. Note that we assumed that a backward, cancel, or forward operation literally restores previous states of the `discourse memory`. However, the result of processing the snapshot of the discourse being the consequence of a navigation instruction by the applications might differ. Reasons for this can be influences of changes in the environment of the application or varying processing stages inside the application if the result of a navigation command is detected. Variations in the application result find their way into the `discourse memory` and therefore its state might require an update. I.e. in figure 15-3 and 15-4 the arrows showing the flow between the states must sometimes not be connected to the 'previous' states but to a variant of these (not shown in the figures).
E.g. when a previous state is restored due to a 'backward' operation, the o^2I-Trees inside the hypothesis structure, which have been rebuilt after the 'backward', are marked appropriately. These marks are added to the restored state in the `discourse memory`.

15.5. Representation of Navigation Instructions

As already indicated, navigation instructions to trigger the appropriate update operations of the `discourse memory` are embedded into object-oriented interpretation trees. These object-oriented interpretation trees are located in the `system` domain being the carrier for generic functionalities. The navigation instructions do not directly add information to the `discourse memory` but update its internal state.

16. Access to Entries in the Result Lists

During human-human or human-computer interaction, the need to reference an entity that is addressed in the adjacent interaction arises, e.g. to execute some action. It includes referring to a real world entity like 'the chair on the left-hand side of the table' (von Hahn et al. [1980]).
Inside DYMALOG, different means to refer to elements of a list can be provided to the user. Besides the content-based access to a list entry, we can use its relative position in the list. To illustrate this, consider a table representing an excerpt from the TV program supplied by an EPG. We might address the shows and movies in the table by their titles, e.g. *"license to kill"* to select the movie 'James Bond 007 – License to kill' if it is shown in this table. However, using the content from such a list requires specific knowledge on the entries, like lexical knowledge or variants of a textual representation to address them. Thus, application specific knowledge is needed. Even when we enable the use of partial matches to reference list items, as shown in the example above, still some knowledge specific to an entry is required. As an alternative generic way especially suitable for our application-blind approach, the position of an item inside the list can be used, e.g. the 'first', 'last', n-th entry, or the 'item on position n'. Note that inter-mixed relative addressing is currently not included in the framework, e.g. 'the next show below James Bond'. The resolution of ordinal references in current dialog systems with application-specific dialog management is e.g. addressed in Filisko and Seneff [2003].
Note further that it is not restricted to lists presented by a certain modality. Indeed mostly visual

output is used to present lists, but one could for example think of speech output for (short) lists. Referencing list elements by their position is of course strongly supported if such an itemization is numbered in the presentation to the user.

Three units enable generic references to list items:

1. Analysis of the application's processing result to identify list structures.

2. Identification and extraction of list references from the user input.

3. Embedding referenced structures into the hypothesis.

The `discourse memory` stores the latest result structure and makes it available for the dialog engine. The dialog engine identifies lists in the result structure and accesses the referenced entities, if applicable.

This section introduces means to allow (semi-)generic references to lists, thus providing a solution for demand 4.2.1(3) in the criteria catalog.

16.1. Identification of List Structures from the Application Result

After Aide has finished the processing of the user input, also involving the interaction with an application, the selected hypothesis together with an optional result representing the processing result from an application is ready to be utilized for the determination of the system's output. The outline of the framework in figure 6-1 shows these relations. At this point, parallel to the generation of the response, the `dialog knowledge processing` analyses the result structure in order to identify and extract a listing of entities if contained.
Instead of automatically identifying a list in the result structure, lists could already be 'manually' marked by their generators, i.e. the applications. However, then an application designer would have to implement means to explicitly tag or construct each list to enable a user to reference the items. Consequently, the dialog engine would rely on the consistency of the tagging of lists in the results. Therefore, Aide takes over the responsibility to identify lists in the result structures from the applications, and thus removes another dependency between the dialog engine and the applications. By this means, the automatic identification of lists eases the creation of applications for the application designer.
The identification process is performed in parallel to the update of the discourse state (section 9.2). Then the user can reference the lists during the next input.

The analysis of the result structures, which is discussed next, is illustrated in figures 16-1, 16-2, 16-3, and 16-4. The procedure to identify a list structure in the result structure omits the trivial case of an empty result structure:

> Let R be the target result structure for the identification process. Two foci are considered during the process.
>
> 1. *Lists on root level.* Consider the core components (domain, instance, and class) of the root-DOs in R. For each setup of core components, which matches more than one root-DO, the object-oriented interpretation trees with these root-DOs form one single list.
> Figure 16-1 shows a list that is identified on the root level for an EPG application. When a list on the root level is identified, no further investigation on the o^2I-Trees utilized in this list is performed.

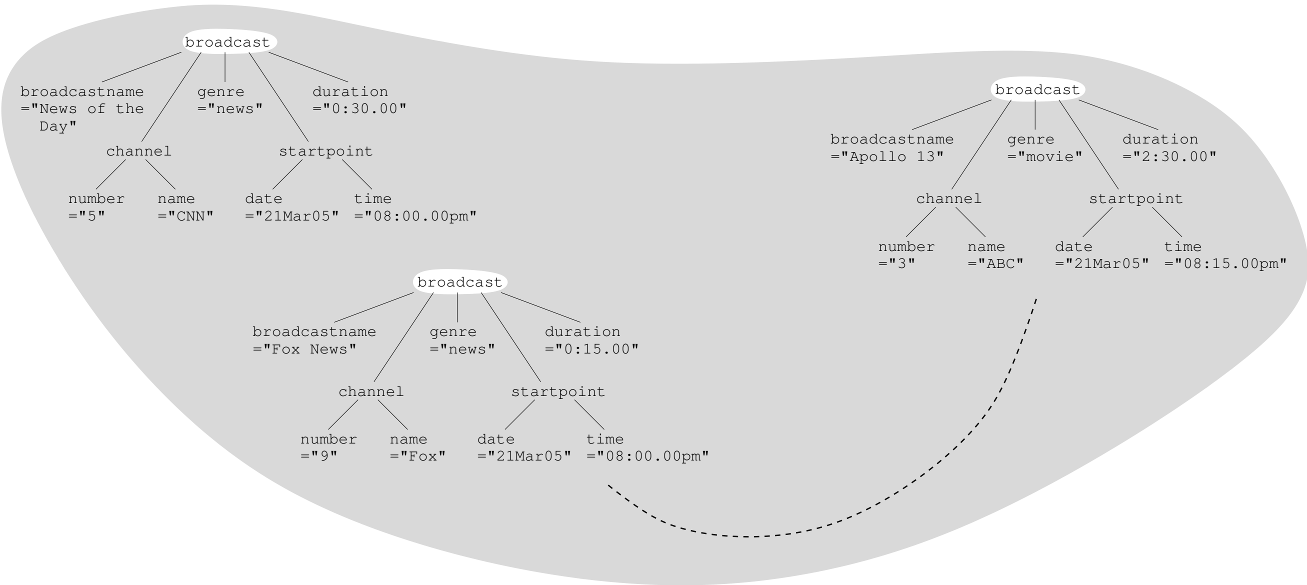

Figure 16-1: Identification of list elements on the root level. Picking up the result structure shown in figure 8-6, an EPG application may package its result as a set of separate o^2I-Trees in one result structure as shown here – the ***broadcastList*** root-DO that connects the ***broadcast*** s in figure 8-6 into a single o^2I-Tree is missing. The ***broadcast*** objects are the most high-level elements that form this list.

Each remaining object-oriented interpretation tree $t \in R$ not utilized in a list on the root level is processed by step 2 in order to search for a list inside the tree. However, if no list inside the tree is identified, t itself forms a trivial list with one entry (this case can be found in figure 16-4).

2. *Lists inside an* o²I-*Tree.* For a single o^2I-Tree t, the identification process searches for a list inside t. The current identification process searches for a unique list below the root-DO, i.e. a set of child DOs below one DO ***o*** with the same core components. The children do not need to be direct children of the root-DO; the path between the root-DO and the parent of the list items must not contain more than one DO on each level.
E.g. in figure 16-2, the ***broadcast*** structures do not form atomic o^2I-Trees in the result structure but are part of a ***broadcastList***. The list identified in this case is comparable to the list extracted in figure 16-1. Figure 16-3 shows a list of ***actor*** objects that is not directly located below the root-DO. Since there are no branches between the root-DO and the parent of the list, the procedure classifies the ***actor*** DOs as items of a list.

The setup in figure 16-4 is similar to figure 16-3. However, the ***actorList*** has siblings and thus the ***actor*** objects are not treated as list items. In this case, the tree itself defines a trivial list with one item on the root level according to the algorithm.

We anticipate that the *Marvin* dialog system is configured to favor the interaction with a single application per user input. The applications currently controlled by this system return a single list in form of a root-DO list (see figure 16-1) or embedded in an o^2I-Tree (see figure 16-2 and 16-3).

The `discourse memory` stores the lists identified from the result structures. Note that a

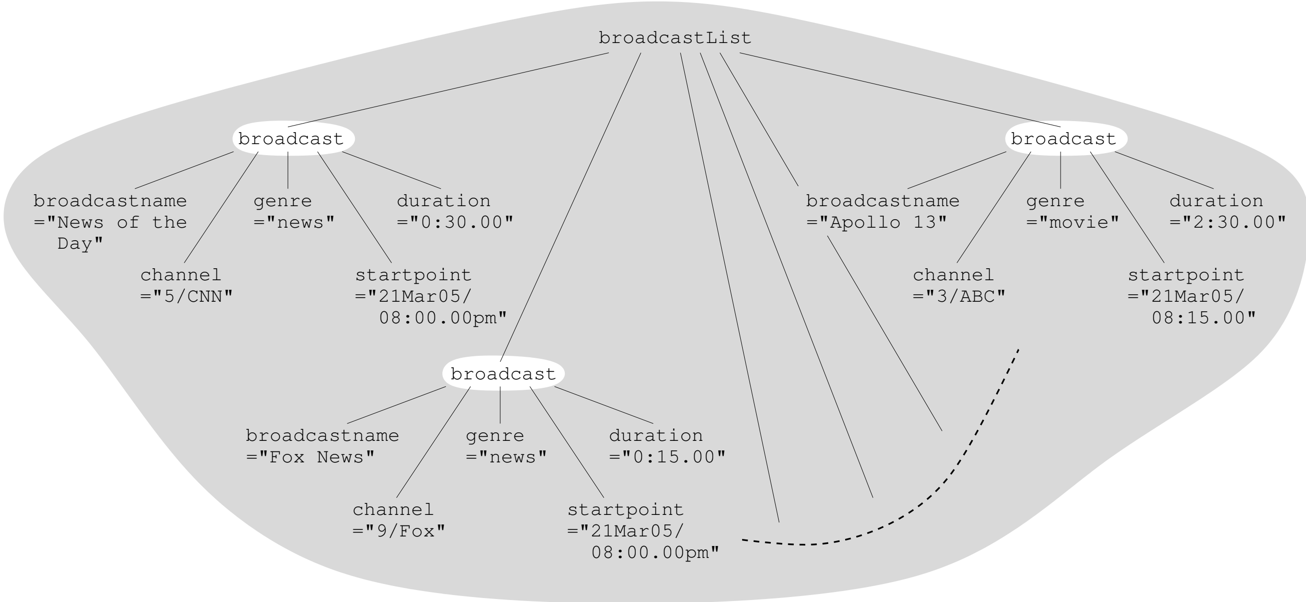

Figure 16-2: Identification of list elements inside an o^2I-Tree of a result structure. The result structure from figure 8-6 contains a structure that carries a list of broadcasts, the root-DO for this list is given as ***broadcastList***. When the list shown here is not an element of a non-trivial list on root level, the structure itself is searched for an inherent list structure. The substructures representing a broadcast being direct children of the root-DO define such a list.

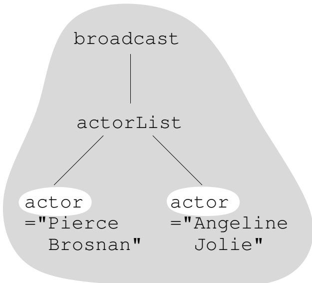

Figure 16-3: Identification of list elements inside an o^2I-Tree of a result structure, based on the example from figure 10-14. Since there are no branches between the root-DO and ***actorList***, the children ***actor*** of ***actorList*** are identified as a list.

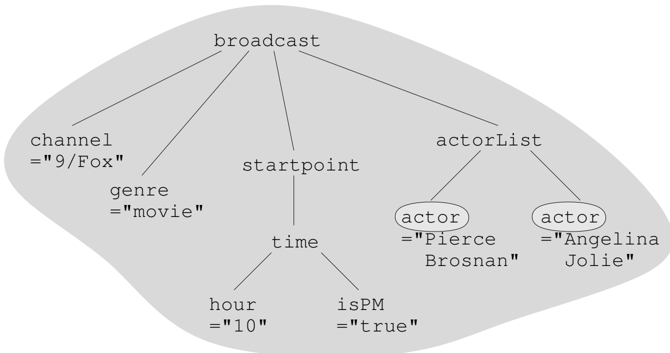

Figure 16-4: Identification of list elements inside an o^2I-Tree of a result structure. In contrast to figure 16-3, branching off from the path from the root-DO to ***actorList*** is possible since ***actorList*** is not the only child of ***broadcast***. The identification algorithm then does not treat the ***actor*** objects as a list, since otherwise a single tree might contain more than one list. Note that in this case the complete structure itself is identified as a trivial list with one item on the root level.

navigation operation generally triggers the interaction of the dialog engine with an application, which in turn delivers a related result structure that is analyzed by the `dialog knowledge processing`.

16.2. Referencing List Entries in the User Input

The current realization of referencing list items differentiates between (semi-)generic processes discussed in this section and application-specific approaches like addressing via the entry's content.

For generic spoken language access[32] to lists the result from the applications is analyzed as described in section 16.1. The NLU extracts semantic entities from the ASR outcome, section 21. Besides the domains of the applications, we introduced a special system domain using the identifier `system`. The generic functionality can be regarded to be located in the special SYSTEM application.
A variety of formulations is covered by the combination of ASR and NLU that lead to the construction of DOs in the `system` domain representing the reference to an entry in a list, including:

> *"the first item"*, *"the top element"*, *"the* 7*-th entry"*, "13*-th item"*,
> *"item at position one"*, *"entry at number three"*, and *"element position five"*.

Together with fragments like *"give me info on ..."*, *"delete the ..."*, or *"select ..."*, which can be associated with tasks, a complete generic instrumentation to access functionality is provided. Besides these generic approaches, embedding application-specific fragments into the generic formulations leads to semi-generic formulations, e.g.

> *"the first* show*"*, "song *number eleven"*, "the picture *at position thirteen"*, and *"the fifth* train*"*.

The NLU honors the additional information that a certain application operation parameter inside an application was addressed (the show, song, picture, or train in the above examples). Therefore, the DOs are enhanced to include the reference to a list item if such a reference can be extracted from the user input. In order to resolve list references, the `dialog knowledge processing` cannot restrict itself on the observation of the system domain but must also resolve references in application o^2I-Trees.

16.3. Integration of Object-Oriented Interpretation Trees and Referenced Item

Once a list reference is identified, this information can be used to combine the currently considered interpretation of the user input with the previously identified lists. Let us assume that a certain DO $o_{\text{ref to list}}$ with a reference to a list entry in a hypothesis structure exists.

At this stage, two prominent sources of problems for resolving the reference in $o_{\text{ref to list}}$ exist:

- The identification process did not extract a list except for a trivial list, either because of the absence of such a list in the result structure or the process failed to extract an existing list.

[32]Remember that we currently focus on speech as input modality.

- A position outside the identified list is addressed, e.g. the extracted list contains ten entries but the entry at position twenty is asked for.

In these cases, the framework provides methods to handle these identifiable problems in the `communication management`.

In order to formulate the integration process, consider the case of n lists $\mathsf{L}_i = (\mathsf{l}_{i,j})_{j=1,\ldots,m(i)}$, $1 \leq i \leq n$, with $m(i)$ entries each extracted from a result structure and $\mathsf{o}_{\text{ref to list}}$ references the k-th element in one of the lists. Basically, the referenced object is looked up in the lists and then it takes over the position from $\mathsf{o}_{\text{ref to list}}$ in the hypothesis structure. The identification of the referenced object is performed as follows:

As a first step, for each list L_i with $m(i) \geq k$, $1 \leq i \leq n$, the k-the entry is extracted. Let $\left\{\tilde{\mathsf{l}}_i; i = 1, \ldots, \tilde{n}\right\}$ denote the list entries being referenced by $\mathsf{o}_{\text{ref to list}}$ in one of the L_i, $1 \leq i \leq n$, thus $\tilde{n} \leq n$.

- If $\tilde{n}$ equals 0, then the reference cannot be resolved.
- If $\tilde{n}$ is larger than 0 and $\mathsf{o}_{\text{ref to list}}$ or the o^2I-Tree that includes $\mathsf{o}_{\text{ref to list}}$ contains a non-system application domain, the entries from $\left\{\tilde{\mathsf{l}}_i; i = 1, \ldots, \tilde{n}\right\}$ being related to this domain are extracted. If the extracted set contains at least one element, the first element of this set is considered to be the referenced item[33]. Note that the selection is arbitrary if more than one element matches the domain (however in the setup presented in section III we did not observe this case).
- If no element was identified in the previous case and $\tilde{n}$ equals 1, this single identified entry is regarded as the referenced element. This case includes fully generic references.
- Otherwise, if no referenced entry can be identified one of the problematic cases occurred.

Let $\tilde{\mathsf{l}}_0$ be the referenced entry identified according to the previously mentioned methods. Thus, we assume that an entry could be located. The o^2I-Tree $\tilde{\mathsf{l}}_0$ is then integrated at the position of $\mathsf{o}_{\text{ref to list}}$, i.e. the root-DO of $\tilde{\mathsf{l}}_0$ replaces $\mathsf{o}_{\text{ref to list}}$ in the hypothesis structure being the origin of $\mathsf{o}_{\text{ref to list}}$ and $\tilde{\mathsf{l}}_0$ is inserted.

Figure 16-5 shows the resolution for an exemplary setup. However, mostly in the actual setup the number of different lists, and thus p, is 1. The current model of references does not allow additional children DOs below a DO like $\mathsf{o}_{\text{ref to list}}$, which carries the reference. Therefore, we do not need to merge the referenced tree with objects already contained in the hypothesis structure. The present process is realized such that both, referencing relative to the first and last element of a list, is possible. I.e. the user may search for the n-th element from the beginning or the end of a list.

[33] Actually the method uses domain and instance of $\mathsf{o}_{\text{ref to list}}$ to find 'compatible' list entries. The search inside each entry of an identified list is not restricted to the root-DOs but also allows children in such trees to be used as referenced object; if more than one match is found the first one is taken.

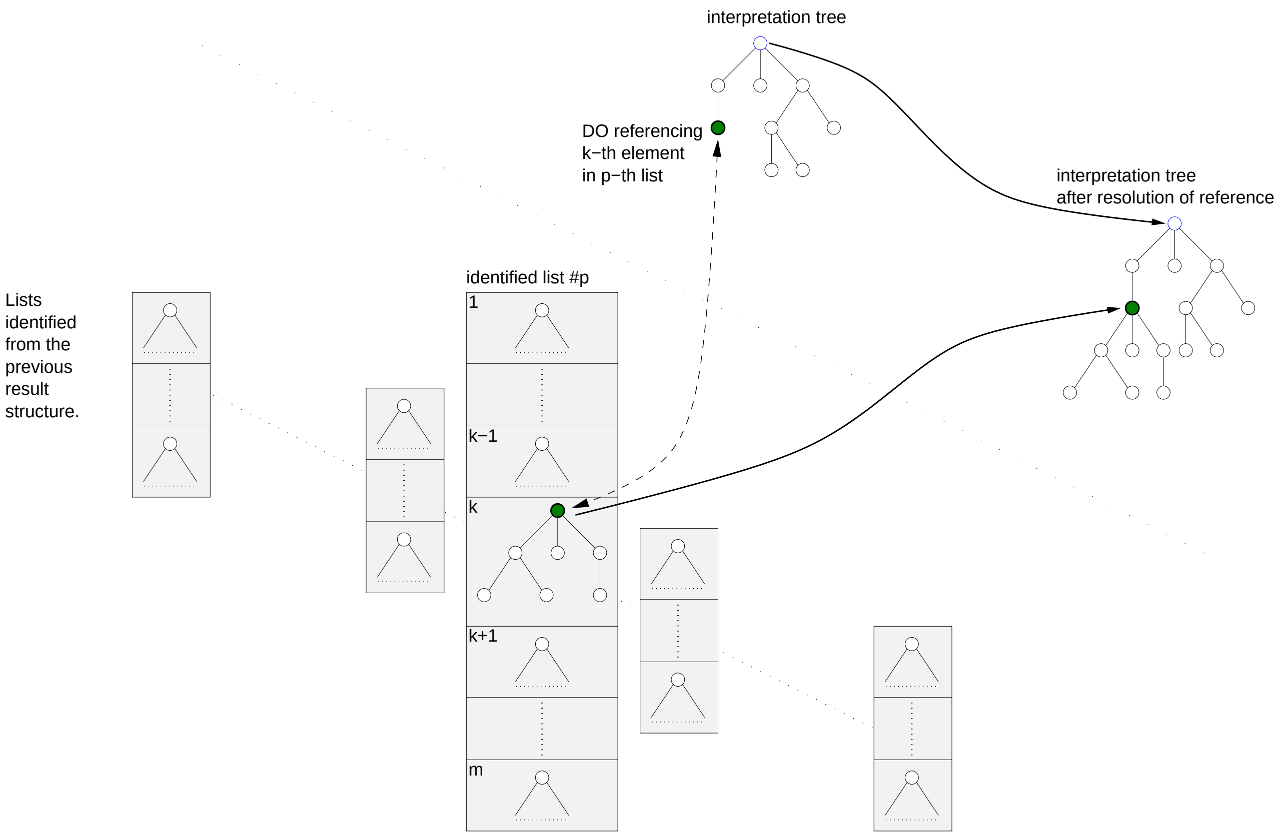

Figure 16-5: Resolution of a list reference to build up an o^2I-Tree integrating the referenced structure. The o^2I-Tree on the top references the k-th item in the p-th list, shown in the figure at the lower left. Also, an o^2I-Tree that integrates the referenced item can be found on the right.

17. Managing the Communication Between User, Dialog System, and Applications

To interact with a user, a dialog system must be capable to perform the analysis of the input from the user, evaluate this interpretation, and determine and present an appropriate reaction on the input. The communication management is in charge to translate between (i) the transformation result of the user input into a meaningful interpretation in the context of the previous discourse and (ii) the reaction directed at the user due to this interpretation. The translation is a non-trivial process. In DyMALog, it comprises the organization and execution of the interaction with the outsourced applications.

As the focus of this work is biased towards the analysis of the user input in an application-independent dialog framework, the communication management provides a skeleton to support this, open for enhancement in the future†. A more sophisticated `communication manager` for the dialog system may adhere to the basic principles in the communication of two partners, especially in the constellation 'human-machine', e.g. Allwood [1997], Doran et al. [2001], Grice [1975].

†Note the remarks on the communication management on page 8.

17.1. Organization of Communication Facilitating the Interaction

The current communication management is mainly involved in organizing the interaction of a user and the dialog system as well as the interaction of the dialog system and the application management layer (see also the overview on the framework in figure 6-1). It performs bookkeeping on

- the hypothesis structures obtained from the `hypothesis selection`, then routed to the application management, and
- the hypothesis structures together with the result structures returned from the applications to be delivered further to the output creation.

The task of the `communication management` is to provide a mapping from the set of all possible hypothesis structures into the joint space of all hypothesis structures and result structures

$$\begin{aligned} \texttt{communication management} : \mathcal{H} &\longrightarrow \mathcal{H} \times \mathcal{R} \\ h_0 &\longmapsto (h, r). \end{aligned} \tag{17.1}$$

In order to perform the mapping, the applications are contacted. I.e. if a proper hypothesis structure is received by the `communication management` it interacts in a request/response manner with the application manager, which itself drives the applications.
We do not explicitly exclude empty hypothesis structures h or empty result structures r. Especially in case of problems empty result structures might occur. Furthermore, the derived hypothesis structure h does not need to be equal to h_0 or to be a direct modification of the original hypothesis structure, but in general it enhances h_0.

17.1.1 Feedback on the Processing Status. Since the `communication management` is responsible for driving the output creation, it can use these capabilities to trigger the presentation of acknowledgments and information on the dialog engine state (Karsenty [2002]). The two cases covered by the communication management are:

- *Acknowledgment of Received Input.* As soon as hypothesis structures are received, the communication management intents to react on this information by initiating an acknowledgment, which is presented to the user. The reaction depends on the content that is contained in the obtained hypothesis structures. In case of a problem, see below, the dialog engine tries to provide an appropriate reaction on the problem – the applications are not bothered. Otherwise, an acknowledgement output informs the user that the framework engages in operating on the user's input. A common option for the acknowledgments are audio 'back-channel responses', e.g. choosing randomly between various short utterances or sounds like *"ok"*, *"aha"*, or *"emh"*[34] (Bell and Gustafson [2000], Okato et al. [1998]).
- *Feedback on Ongoing Operations.* For certain operations, an application controlled by the dialog engine through the application management may need a certain amount of time to complete such an operation, e.g. a database lookup over a network. Therefore the response towards the dialog engine, and thus a reaction directed at the user, is also delayed[35]. To indicate the need for a prolonged processing time by an application, the `application`

[34] The *Marvin* dialog system uses this variant.
[35] Besides synchronous interaction of Aide and applications, asynchronous interactions are to be integrated in the future. The event mechanisms provide the basics of asynchronous interactions to drive the dialog engine.

`management` automatically reacts on an outstanding answer from an application by posing an appropriate output. The output can be triggered once or repeated after certain intervals without response from the applications. Again, spoken language output can be used, e.g. randomly chosen phrases like *"please wait"*, *"just a second"*, or *"please stand by"*[36]. The interface between dialog engine and applications could be extended in the future to allow for more application related status information. Currently the `communication management` restricts itself to monitoring the existence of a reaction from an application.

17.1.2 Generic Instructions – The System Application. Since the generic functionalities are not part of a custom application, the communication management provides means to determine appropriate answers on these functionalities. However, it is not responsible for actually executing such generic operations, this is e.g. carried out by the `dialog knowledge processing`. As we have seen before, sample reactions are informing the user on an executed navigation command (section 15) or on the activation of a new application for the control through the dialog system (section 18.3).

Generic capabilities being mainly handled by the communication manager are the problem detection and handling outlined next.

17.1.3 Detection of Missing Application Responses. To prevent the dialog system from waiting infinitely long for an answer, the `communication management` regularly checks if it has already got all expected responses. After a reasonable time without a reaction, the communication management stops waiting for a reaction and initiates a response indicating that a problem in the interaction with an application occurred. The `communication management` has to ensure that the application is informed that the dialog system is waiting no longer for a reaction of this application – the application is responsible to appropriately deal with this information. If an application reaction (as direct answer to the request) is received anyhow after this decision, the dialog system discards the reaction for further processing by the dialog system. To deal with the 'abortion', the application may e.g. decide to stop processing the request and undo all changes being a consequence of this request. Alternatively, it could go on processing the request and initiate a communication with the dialog system later during the discourse to inform the user of the results.

The reason for a missing response might be an error or crash in one of the applications. Or the application was disconnected from the dialog system.

17.2. Problem Detection and Handling

Maintaining a proper flow in the interaction of a user and the dialog system requires the identification of problems occurring during the dialog. Detecting problematic situations from linguistic clues in the discourse is a broad field of research. In contrast to the automation of phone-based services, a dialog interface e.g. in the CE domain cannot fallback to a human-operator when problems are detected, Wright Hastie et al. [2002].
The communication management concentrates on problems being a consequence of the way of knowledge representation and handling introduced here, thus concentrates more on the dialog system inherent aspects.

[36] This feedback method is also implemented in the *Marvin* dialog system.

Missing Application Responses. The first type of problems is the *absence of an application response* already addressed before in section 17.1.3. If an application does not react on a request after a certain time, the user is informed and responses to the request received after this point in time are discarded.

Empty User Interpretation. An *empty hypothesis structure* indicates that the analysis of the user input failed. Reasons can e.g. be the incomplete coverage of the user input by the lexicon of the ASR or the inability to extract semantic entities in the NLU. In this case, an appropriate output is triggered to indicate the failure in the extraction of information the framework can handle, and to ask the user for her next input.
Another special reason leading to an empty hypothesis structure are navigation operations in degenerated situations. This includes performing a forward operation even though the forward stack is empty or a backward operation without earlier turns, which can be recovered. The additional knowledge is used to adapt the system reaction to help the user understand the problem.

Unresolved List Reference. The same holds in the case of an *irresolvable list reference*. Knowing that the selected hypothesis contains a reference, which has not been resolved, is communicated towards the user. A more detailed analysis may be given to the user, e.g. if a list contains only 10 items, thus a 20-th entry cannot be addressed.

Multi-Domain Hypothesis Structures. Note that the current lightweight communication manager is not capable of *handling a multi-domain hypothesis structure*. I.e. input like *"play the new messages on the answering machine and turn on the radio"* is not handled and the user is informed on this restriction (of course, we assume for this that the answering machine and radio reside in different domains). A starting point to enable such multi-domain requests is to sequentially interact with the `application management` for each o^2I-Tree of the hypothesis structure one by one. However it can only be the first approach, since e.g. an open question is how to proceed if one o^2I-Tree contains a 'proper' interpretation for one of the custom applications and one contains a navigation instruction or error. A different approach would let the application management take over the responsibility on handling multi-domain hypothesis structures. This would require only very limited adaptations in the communication management.

More Than One Hypothesis Structure. Further, the communication management issues a warning to the user if it receives *more than one hypothesis structure*. A reasonable treatment would – among other things – require an understanding on the differences to decide how to go on with this set of hypothesis structures. E.g. equal structures with only one differing value at a leaf DO might easily be disambiguated (like selecting a destination for a train, `Hamburg` vs. `Homburg`) or require more complex handling.

17.3. Dialog Initiative

In a dialog between two partners, the partner who takes the initiative is usually in control. DymaLog does presently not include a sophisticated communication management, and it does thus not define advanced strategies with respect to initiative taking. However, the *Marvin* dialog system implements both, user driven and system driven interaction strategies, and realizes therefore a mixed-initiative dialog system. For example, the system takes the initiative when manually programming a recording, and, when browsing the image collection, the initiative lies with the user[37].

The event based trigger process that invokes the communication management forms the basis of the flexible initiative handling. It is able to react on events generated from the input by the user and events fired by an application. The events contain annotations to support the communication management in its decision process for the next reaction directed at the user. It is the obligation of the communication management to decide if the system should retain, take, or give away the initiative in the dialog. As already pointed out before, the strategy presently implemented in the *Marvin* dialog system realizes means to handle user initiative as well as system initiative. In the latter case, the user may take over the initiative: e.g. when the system prompts for the parameters to schedule a recording, the user may provide the content in another order than requested, possibly overspecified, or change her mind and perform an unrelated step, which could include switching to another application or task.

Section 18.4 explains how the event mechanism can even be used by the dialog system to start off a new dialog with a user.

17.4. Interfacing with the Applications

Interfacing with the applications is performed through the `Application Management`. Both, application management and applications, are the focus of section 18. The mapping (17.1), page 155, is strongly influenced by the integration of the applications in the execution of the mapping through the interaction sequence shown in figure 17-1. Remember that the hypothesis structures h_i transport the knowledge obtained during the analysis of the input and integration into the previous discourse. The application is also capable of enriching h_2 as we will see later. The result structure r transports the outcome of processing h_3 inside the application with respect to an execution of one or more actions by the application. In the domain representing a TV-set, this might for example represent that it was switched to another channel. Another example is the result list in a flight information application carrying a set of connections between two cities, as the user might have requested these.

17.5. Interaction Dependent Adaptation of Component States

The communication management is responsible for adapting the state of other components in the dialog system. This does not mean that an excerpt of the discourse is transferred and stored in an affected component but the state is adopted towards the current dialog situation.

A fundamental task taken over by the `communication management` is the handling of the modality state. This especially holds for the audio input channel, thus controlling the activation status for speech recognition. Presently, the audio channel is closed for recognition between the end of the user input and presenting a response on this input back to the user, i.e. during processing by the dialog engine and the application side. Furthermore the communication manager is involved in controlling output modalities, e.g. the viewport (section 22.3.3).

---

[37] When manually programming a recording, the *Marvin* dialog system prompts the user to gather the required content, e.g. channel, time, and date. The user may specify more content than requested by the system.
In the image browser application, the system does usually not request specific information from the user. Instead, the user is free to operate on the image database, like defining certain constraints on the images that she wants to see.
See chapter III and the video recordings on the DVD bundled with this thesis for examples.

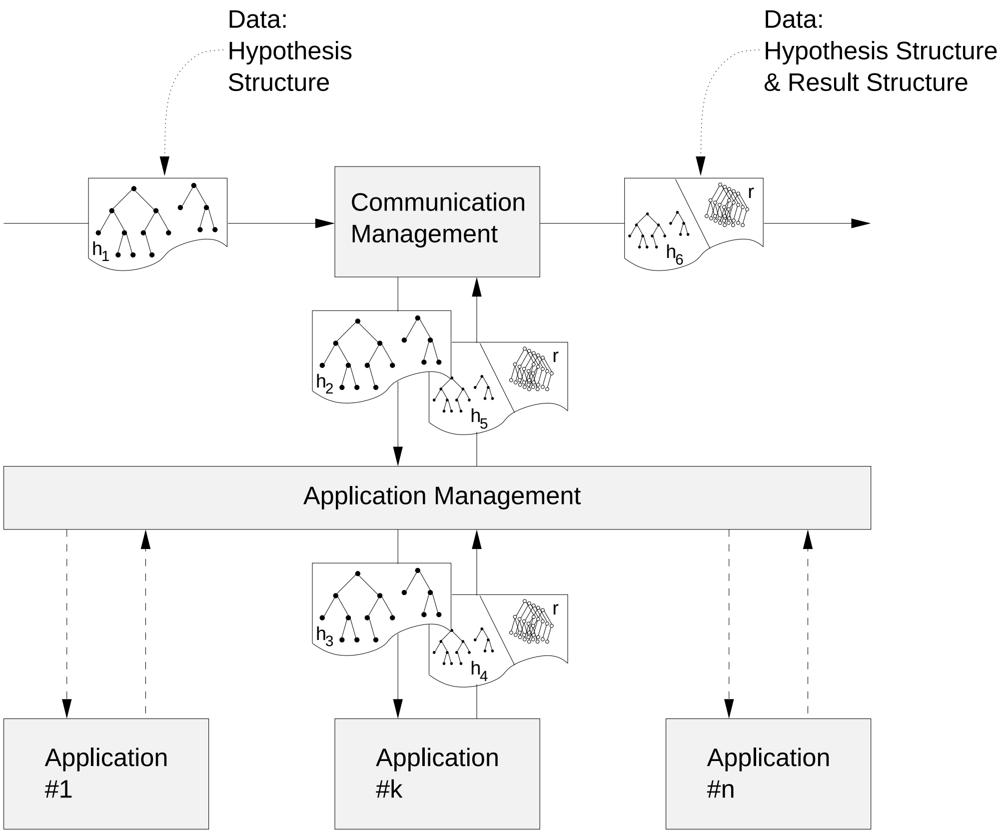

Figure 17-1: Flow of data between `communication management`, `application management`, and applications. Exchanged data types are hypothesis structures and result structures.

Even though the communication management is prepared to perform further state dependent configurations of components, these rely on an enhanced communication management. Such adaptations include adapting the lexicon of the ASR, e.g. raising the probability of yes-/no-answers if the communication management decides to pose such a question for clarification.

Note that the adaptation of knowledge sources due to changes in the application setup lies in the responsibility of the application management discussed in more detail in section 18.1.

17.6. Towards Advanced Communication Management

At present, the output creation partly adopts the determination of the output, namely the `generation` (section 22.1). With support from the applications, transformations to map certain hypothesis structures and the related result structures into a format suitable for rendering are used to explicitly formulate the reactions, see section 22.1.

Since the application logic needs not to be included in the dialog engine logic, the dialog logic only needs to model application-independent issues (Chu-Carroll [1999], Macherey and Ney [2003]). The dialog manager is application independent, Papineni et al. [1999]. Veldhuijzen van Zanten [1998] gives an example how dialog management can utilize a hierarchical structured slot organization representing the operation parameters for an application.
As already stated in this section, a more advanced and general approach may be based on the speech act theory (Searle [1969]). Analyzing the hypothesis structure to determine the speech acts derived from the current user input together with the history on the speech acts lets us apply

a model for estimating speech acts that should be used for the output creation. To estimate the next speech acts, we can use a probabilistic model based on Markov models (Nagata and Morimoto [1994], Garner et al. [1996]).
Once the speech acts for the user input are determined and ranged into a history of speech acts, the `communication manager` may prepare an appropriate response based on the sequence of acts up to the present user input. Obviously, the response can be based on speech acts, possibly combined with features inside the hypothesis structures (e.g. an annotation of a DO that indicates that further information for this object is required).
Enabling the `communication management` to compute a speech act based response will also require adaptations in the generation components.

Besides the speech acts, future advanced methods in the `communication management` may rely on the tree based representation of application operation parameters. Dialog management using hierarchical structures is shown in Hochberg et al. [2002], Macherey and Ney [2002].

As a starting point to derive dialog strategies for the `communication management`, we could fall back on the strategies derived in present systems and try to extend these for a framework decoupled from the applications. To derive such a strategy, corpus-based statistical methods might be applied (e.g. Bernsen et al. [1996], Bohnenberger [2000], Roy et al. [2000], Fleming and Cohen [2001], Levin et al. [2000], and Litman et al. [2001]). Also systems built upon the theory of (shared) plans are appealing as a starting point towards a more advanced communication management, e.g. Litman and Allen [1990], Lochbaum [1998]. Techniques to automatically derive plans from corpora or examples are used to learn strategies, Garland et al. [2001]. However, many of the systems described in these papers rely on domain knowledge when learning optimal strategies, often with limited state spaces, see also Paek and Chickering [2005]. The abstraction from the dependency on the controlled applications remains one of the future challenges.

Lemon et al. [2002a] presents another interesting topic for a multi-application dialog system for a single application system: dealing with multiple concurrent tasks in the interaction of human and machine.

17.7. Interrupting the Computer: Barge-In

The reasons for disturbing the dialog system during its processing, or even during the presentation of its response to the user, can be manifold: the user may change her mind, the user may discover that the system does not perform the requested operation, the user gets impatient, and so on[38].
The effect of barge-in on the dialog system state is a priori not evident. It takes some time to process the user input that contains the utterance, and when the intention of that utterance becomes clear to the dialog system, it has to evaluate its internal state and execute the appropriate actions.
Once the system identified barge-in from the user input, it has to decide how this affects the processing of the last regular input. The simplest reaction would be to ignore the barge-in, as it is presently done by the implementation of DYMALOG. Another straight forward approach is the interruption of the processing as soon as the barge-in was detected, and restoring the internal state either before the last utterance, at the point in time when the user input was received, or when the barge-in was finally detected. Instead of just restoring the state before the barge-in, the system may execute a compensation operation, e.g. to undo an action performed by an

[38]We will concentrate on barge-in input where the user intents to interrupt the processing of the dialog system, and the execution of a certain action, to sketch major effects of barge-in. However, additional sorts of barge-in exist, e.g. a user providing additional content that could clarify the next action to take when the system takes 'too much' time to think about the last input.

application.

At present, DymaLog does not cover barge-in. Due to the modularized approach, the main handling of barge-in can be dealt with by a separate component in the future; the underlying MultiPlatform allows to easily integrate additional modules. Yet, it could also be integrated into an existing component, canonically the `communication manager`.
To enable barge-in, some of the components will need adaptations. For instance, the speech synthesizer and renderer in their present implementation do not provide means to interrupt a currently executed speech output process.
Other already available concepts can support barge-in, like the mechanisms underlying the navigation in the discourse. These mechanisms are useful to restore previous states.

A major challenge in a system separating the dialog system and the applications is the handling of barge-in by the applications, in contrast to the dialog system itself. If the barge-in can be processed *before* an application was contacted, only the state of the dialog system needs to be updated to reflect the request. In the case where the application was already addressed before barge-in was detected, the state of the application has to be taken into account. The operations already performed by an application can be (partly) reversible, or irreversible: turning on the light can easily be annulled, while an already sent e-mail cannot be fetched back. Advanced concepts are needed to deal with interruptions by barge-in, especially when an application already performed some operation.

18. Interaction and Interfacing with Applications

Section 17 covered the interaction of the dialog engine Aide and the application management. We now concentrate on the intermediate layer between the dialog system and the applications, the `application management`, the applications itself, and their interaction.
SmartKom applies an advanced approach towards the separation of applications from the dialog system (Torge et al. [2002]). The functional model makes the functionality from the applications available to the dialog system through an appropriate representation. Changes in the application setup require an update of the functional model. Affected modules in the SmartKom system need to adapt to these changes. As an advantage, applications may be combined to execute complex tasks, compare also Bühler and Minker [2002]. However, the knowledge on the applications inside the dialog system is much more explicit, e.g. ‘the planning module is always aware of the current state of each device in the network’.

18.1. Application Management

Two main aspects can basically divide the tasks of the application management:

18.1.1 Identifying and Addressing the Proper Application. The main work to address the proper application for a hypothesis structure is delegated to the hypothesis selection process. The o^2I-Trees inside the hypothesis structures are associated with applications through the domain property of the DOs. Therefore, once a hypothesis structure is selected the possible target applications become clear. Routing the hypothesis structure forward towards the applications lets them perform an operation according to the content of the obtained hypothesis structure. Of course, the applications can easily decide if a hypothesis structure is relevant for themselves

through the domain. The applications expect that the domain of the root-DO represents the domain of all DOs contained in an o^2I-Tree.
In the present set of applications available for use in the realization of DyMaLog, each domain is handled by exactly one application. However, possibly multiple competing applications might be addressed by the same hypothesis structure. It would then lie in the responsibility of the `application manager` to decide on the preferred application. However, the decision process is an open research issue and does not affect our present prototype setup.

The `application management` serves as single central entry point for Aide to interact with applications. The other way round, the application management is also the single modifier of knowledge sources being dependent on the current application setup†.

In a possible extension of the present `application management`, it might allow to perform tasks by the complex interplay of different applications as described in Herfet et al. [2001] or Bühler and Minker [2002]. However, our preferred approach to such a complex interplay would provide the means for such an interplay of applications in a separate application, instead of integrating these means into the `application management`. For the moment, we limit the scope to the operation of a single application out of a dynamic set of applications together with the capability of automatically sharing discourse information between these applications.

18.1.2 Management of the Application Setup. The dialog system described here serves as hub $\mathbf{h}$ in a dynamic application setup $(\mathcal{A}_{\text{conn}}, \mathcal{A}_{\text{non-conn}}, \mathbf{h})$. Changes in the set of connected applications $\mathcal{A}_{\text{conn}}$ generally affect the available functionality that can be addressed through the dialog system (Pakucs [2003b]). There are two obvious processes:

1. Connecting an application to the hub, i.e. moving an application from $\mathcal{A}_{\text{non-conn}}$ to $\mathcal{A}_{\text{conn}}$. In the context of DyMaLog, this means that an application is added to be controlled by the dialog system.

2. The counterpart is the disconnection of an application from the hub: an application is moved from $\mathcal{A}_{\text{conn}}$ to $\mathcal{A}_{\text{non-conn}}$. It complies with removing an application from the influence of the dialog system.

Both processes have in principle the same effect on the framework (except that in one case the set of functionalities to be controlled by the dialog system increases, in the other case decreases). After the `application management` detects a change in the application setup, this change is announced to the `communication management`, which is free to trigger an appropriate reaction, usually informing the user on the change. An appropriate hypothesis structure generated by the `application management` transports the information between application management and communication management.
More effort must be spend on the second process that is stimulated by a change in the application setup as described next:

1. Application setup changes.

2. `Application management` adapts its internal state on applications and their related knowledge sources (and the `communication management` is informed on the change).

3. `Application management` prepares knowledge sources for use in the affected components.

    (a) Integrate knowledge sources into a single unit and provide the result.

†Note that parts of the application management are contributed by Dr. Portele, see also page 8.

(b) Provide components being able to handle knowledge sources from multi-domains with changes.

4. Affected components reconfigure with updated knowledge sources.

At present, knowledge sources from all three areas, which made up DymaLog, contain components that need to be updated. For the input processing, these components need to adapt to a new application setup:

- `Recognition (ASR).`
  - *Lexicon.* The vocabulary and the related pronunciation have to be adapted for the available functionality that can be addressed through the dialog system. The integration of lexica from different applications is straightforward. The lexicon creation process from a list of words starts with a large background lexicon and uses automatic transcription for non-covered words.
  - *Recognizer network.* The network ASR restricts the valid utterances by a network. For each application, including the system application, a single network is generated that is integrated into a single global recognizer network.
- `Analysis (NLU).` The speech understanding uses a grammar to represent the relations between word sequences and semantic entities. The semantic entities are atomic DOs that might have a task attached to it. Analog to the network applied during the recognition, the grammars are built up for each domain and then merged into a single grammar.

Inside our application-blind Aide, direct dependencies on knowledge sources from the applications are concentrated at one point:

- `Domain Model.` The `domain model` bases the process of establishing a relation between single DOs or a set of o^2I-Trees on ontological descriptions, see section 20. It obtains the collection of all ontological descriptions of connected applications from the `application management`.

Finally, also the output creation adapts to changes in the application situation:

- `Generation.` The `rendering` components are basically application-independent. Yet, a speech synthesizer might make use of the lexicon created for the ASR.
  The generation of the input for the rendering from the hypothesis structures and result structures utilizes 'stylesheets' as transformation instructions (see section 22.1). Since the applications assume that a single o^2I-Tree is related to one domain, it is sufficient to apply the transformations related to that domain on an o^2I-Tree. Therefore, the 'stylesheets' can be independently made available to (or removed from) the `generation`.

The 'DHomme' project also addresses integration of knowledge sources from different applications. Rayner et al. [2001] investigate the challenges of a consistent NLU grammar for dynamic application setups for an in-home network to control simple devices.

18.2. Applications

Criterion 4.1(3) postulates the clear separation between logic related to the dialog processing and related to the applications: the application-blindness of the system. While the major part of this chapter deals with procedures and methods applied inside the dialog framework, this section will allow for an insight how the application logic is added in the DyMALog approach.

We can identify two aspects that have to be addressed for applications enabled to be controlled through the dialog system. The first aspect concentrates on the requirements posed by DyMALog on an application to allow the operation through the dialog system. The second aspect engages in the realization of an application, i.e. how the actions provided by the application are realized.

***18.2.1 Requirements on Applications through* DyMALog.** In order to enable an application for use in a dialog system that allows for a dynamic application setup, a clear interface is required to allow applications to (dis-)connect and to interact with the dialog system. Our application-blind approach even facilitates the formulation of the interface.

Connecting to and Disconnecting from the Dialog System. Once connected to a connected environment the dialog system belongs to, the application announces its arrival through the network such that the dialog system can observe it. Correspondingly, a disconnection from the network is detected and the dialog system adjusts to the new situation. The criterion 4.1(5) formulated this as a necessity for voice interfaces in dynamic application setups.

Presently, the applications suitable for running in DyMALog explicitly send a message to the `application management` on a (dis-)connect. The `application management` informs the affected components on changes in knowledge sources and supplies the `communication management` with an overview on the changes[39].

18.2.2 Realization: Interfacing with the Dialog Engine and Adopting the Hypothesis Structure. Basically, we can distinguish between two starting points when creating an application that might be controlled through the framework: existing applications or newly designed applications. In addition, applications might or might not depend on application specific hardware. Thus, a second feature to discriminate applications is the realization purely in software or requirement of hardware.

The actual implementation of an application is irrelevant for the dialog system as long as it honors the interfaces (and performs the operations we expect it to perform). Therefore we encapsulate the applications[40], which together with the definition of a clear interface complies with requirement 4.1(6) of our criteria list.
As already indicated in figure 17-1, the applications get a hypothesis structure (h_3) as input and return a hypothesis structure (h_4) and result structure (r) as outcome for the dialog system. As long as the format of hypothesis structures is respected and the resulting hypothesis structure corresponds to the ontologies formulated for the application of interest, an application is free to modify the hypothesis structure by:

Adding DOs. Adding DOs refers to both, adding one or more atomic DOs into a hypothesis structure and adding a substructure made of more than one atomic DO. Applying defaults in an operation performed by an application is the major case leading to the insertion of

[39]Upcoming standards, e.g. 'plug-and-play' standards in the field of networked CEs applications and devices, will provide more robust means to handle connections and disconnections of applications and devices in connected environments.

[40]Especially due to the hierarchical organization of the hypothesis structures, object-oriented programming languages are a good approach to model the operation parameters in software-based applications or interfaces; each DO in such a model corresponds to one class. The software classes correspondingly recreate the reuse of structures. Inheritance can be used to adapt reused structures to certain applications.

DOs. E.g. when turning on a TV-set, the channel watched before turning off the device is restored. Or in a train timetable application, we may assume that the customer wants to depart from her residence today in about one hour from ‘now’ if no further information on the departure is given.

Removing DOs. During processing by an application, a hypothesis structure can also be thinned out by removing DOs. Removing an inner DO also has effect on the subordinated DOs, i.e. its children etc. These are removed together with the higher-level DO. Removing DOs becomes relevant in the case where contradicting content is contained in a single o^2I-Tree. E.g. let a user search for connections between two cities in a train timetable information system. Assume that the user requested connections for tomorrow 8 a.m. during the previous turns and assume further that it is now 2 p.m.. Changing the date from tomorrow to today would make the time invalid, the resulting point in time is history. Removing the time from the o^2I-Tree thus would make the structure again valid (another option would be to replace the time by some default).

Updating existing DOs. During the analysis of the starting point for operation h_3 or during the operation, DOs already contained in h_3 can be updated. Besides setting or adjusting the value of a leaf DO, other updates of a DO include:

- Tagging a DO with its creator: a DO that is introduced in a hypothesis structure generally originates either directly from the user input (including intermediate DOs used to connect DOs created e.g. by the `analysis` components) or processing by the application. DOs resulting from processing by the application can be clustered into two groups: computed results and defaults, which have been applied to perform an operation.
- Marking a DO for inquiry: the application might require additional information to properly perform an action. Such an inquiry ranges from an open request for information (which might e.g. trigger output like *"at what time do you want to record a show?"*) via requests for one out of a class of values (e.g. a set of cities for discrimination) to verifications (implicit and explicit, especially yes-/no-questions).

Especially the latter marks are used during the output creation to obtain accurate reactions. In a speech act based communication management, these hints would influence the determination of the next speech acts for the system reaction.
Note that all modifications on a hypothesis structure have to comply to the structures defined by the ontologies delivered with the application. Thus, the application designer has to ensure consistency between the constituents of an application.

The second outcome of an application as reaction on handling a hypothesis structure is the optional result structure. It usually carries the results from the operation performed on the basis of the received hypothesis structure. The entries in the result structure are instances of structures derived from the ontologies, e.g. the broadcast entries delivered by an EPG or train connections between certain cities.

The mechanisms used to update the knowledge sources utilized in the dialog system when an application is (dis-)connected (section 18.1.2) also allow applications to update the provided knowledge sources due to changes of the application state. Due to limitations in the ASR and NLU, at present ‘larger’ applications make use of this by distinguishing between

- global elements, which become available to the dialog system as soon as an application connects to the dialog system, and

- specialized elements being activated after the user enters an application, i.e. some globally available functionality of the application is requested.

The specialized elements might be hidden again, e.g. after a certain number of turns when the application was not contacted or when the application was explicitly left by the user.

18.2.3 Realization: Performing an Operation. Up to here, we discussed how the dialog engine AIDE drives an application through the `application management`. To actually perform an operation, e.g. switching a channel on a TV-set, performing a database query to extract certain train connections, or requesting information on a picture shown on the screen, the data has to be connected to applications.
AIDE does not care for the realization of an application as long as it respects the interface: an application can be a pure software solution, maybe based on an existing application enhanced by an interface layer, or a hybrid solution of a device together with an interface layer. Thus, the application is free on how to operate on the given data, with the exception of adhering to the interface. An application packages knowledge sources for the dialog framework, including the ontological descriptions of its operation parameters.

Knowing the descriptions of the operation parameters, an application usually has to perform three steps:

1. Analyze the input from the dialog engine.
2. Depending on the structure obtained in the first step, the application might perform an appropriate operation.
   The decision which operation to perform is part of the application logic. It should be based on the specified application operation parameters and the task(s) contained in the o^2I-Tree obtained from AIDE.
3. Update the input o^2I-Tree and generate the result structure for the performed operation.

The last step includes inserting new DOs into the o^2I-Tree if required or the annotation of DOs, e.g. to indicate that further information is needed before an operation can be performed.

18.3. Activating and Deactivating Connected Applications

In this section, we will not discuss the connection and disconnection of applications to the dialog system. Here we assume that an application is connected. As already stated earlier in this text, an application can differentiate between knowledge that is available as soon as an application is in range of the dialog system and knowledge that becomes available when an application enters a certain state.
For applications requiring 'larger' knowledge sources, two states were introduced: *activated* and *deactivated*. If an application distinguishes between these states, global sections of the knowledge sources are visible to the dialog system in both states, while specialized knowledge is only usable if the application is active (section 2.2). To summarize, an application should adhere to the following descriptions of these states:

Deactivated Application. In the deactivated state, an (already connected) application provides a minimal set of knowledge sources to announce functionality that is globally available for access by the user as soon as an application connects to the dialog system (global parts).

Activated Application. An activated application provides a set of enhanced knowledge sources to allow the user access to its larger set of functionality (global and specialized parts).

The deactivated state compares to a stand-by mode while an application is on full operation mode when it is activated. The minimal set of knowledge sources may deliver means to switch between these two states. In addition, a timer could be used to change from activated to deactivated, e.g. after a certain number of input turns when the user did not address that application.

DymaLog delivers means to perform the activation and deactivation of applications postulated in requirement 4.2.3. Generic ASR and NLU resources as well as ontologies prepare the infrastructure to explicitly change the activation status for an application – the application only has to provide the means how to address it. I.e. for speech input, that the name(s) of an application (together with a semantic 'tag') must be specified.

18.4. Application Initiated Interactions

Up to now, we concentrated on the case where the user input initiates the interaction of a user and the dialog system, thus an application is contacted and reacts on it. However, the framework foresees means to deal with hypothesis structures (together with empty result structures) delivered by an application without an initiative from the user. It can be used to establish an interaction with the user due to the occurrence of some event (Grisvard and Gaiffe [1999]). Commonly events occur if a certain point in time is reached (e.g. alarm clock) or a change in the surrounding is observed (e.g. user enters range of dialog system). The means required to allow application initiated interactions do not differ that much from the means required for user initiated interactions: at present, only appropriate handles in the output `generation` need to be created. For the deployment outside research settings, the decision of initiating a dialog with a user should include awareness about the presence of the user addressed, thus requiring means to enable the system to (automatically) determine the persons within reach.

The notification on changes in the application setup is an example of an application driven event without preceding input from the user[41], which generally leads to a system output directed at the user.

19. Ontologies: Modeling the Application Operation Parameters and their Realizations

Several times in the preceding sections we pointed out that the building blocks of a hypothesis structure, the o^2I-Trees, are derived from ontologies provided by the applications and exploited in the `domain model`. Ontologies are appropriate for describing parts of the real world (Forkl and Hellenschmidt [2002]). We use them to describe the operation parameters in an application[†]. The basic elements to build a description of the operation parameters for a certain application are ontological classes and ontological objects.

[41] Yet, the event fired by an application might be due to an interaction of the user directly with the application (e.g. pressing a button on the device itself), but not involving the dialog system in the interaction.

[†] Note that significant parts of the ontologies as used in this framework are prepared by Dr. Portele, see also page 8.

19.1 Definition (Ontological Classes and Objects)**.** *Ontological classes* formulate entities and their relations of the application operation parameters to represent elements in the real world. It refers to an abstract description of a section of the world without relating to a concrete instantiation.
An instantiated ontological class generates an *ontological object*. i.e. an ontological object points to a concrete entity. The 3-tuple $(\mathbf{d}, \mathbf{i}, \mathbf{c})$ of the constituents

- domain **d**: creates an association to an application,
- instance name **i**: denotes an ontological object related to an instantiated entity, and
- class **t**: defines the species or type of an object

characterizes an ontological object. The notation $\mathbf{o} =$ **d:i{t}** is used for an ontological object $(\mathbf{d}, \mathbf{i}, \mathbf{c})$, short: **i{c}**. For an ontological class – i.e. a *non-instantiated ontological object* – the notation **d:*{c}** (or ***{c}**) is used. ◇

The definition already indicates the close relation of DOs according to definition 8.1 and ontological objects.
During this section, the term *object* is used as short notation for an ontological object.

19.1. Relations Between Ontological Objects

Based on ontological objects, the two relations between the objects are introduced (Denecke and Waibel [1997]). These will also be used to create ontological classes. For the description of the first relation, the 'has-a' relation, we will waive the inner structure of an ontological object.

19.1.1 Member-Relation: 'has-a'. The *member relation* connects ontological objects among each other. It defines an one-to-many parent-children relation. More concrete, if $\mathbf{p} \xrightarrow{\text{has-a}} \mathbf{k}_i$ holds for ontological objects $\mathbf{p}$ and $\mathbf{k}_i$ $(1 \leq i \leq n)$, the $\mathbf{k}_i$ are children of the parent $\mathbf{p}$. We postulate that no loops are defined through the 'has-a' relation, i.e. $\mathbf{o}_i \xrightarrow{\text{has-a}} \mathbf{o}_{i+1}$ for $i = 1, \ldots, m-1$ and $\mathbf{o}_m \xrightarrow{\text{has-a}} \mathbf{o}_1$ is not valid.
E.g. consider an object **p** to represent a point in time. Let **t** denote a time object and **d** the date, then we would expect

$$\mathbf{p} \xrightarrow{\text{has-a}} \mathbf{t}$$
$$\mathbf{p} \xrightarrow{\text{has-a}} \mathbf{d}$$

to hold. The time and date objects themselves have children:

$$\mathbf{t} \xrightarrow{\text{has-a}} \mathbf{hr} \qquad \mathbf{t} \xrightarrow{\text{has-a}} \mathbf{min} \qquad \mathbf{t} \xrightarrow{\text{has-a}} \mathbf{sec} \qquad \mathbf{t} \xrightarrow{\text{has-a}} \mathbf{am/pm}$$

$$\mathbf{d} \xrightarrow{\text{has-a}} \mathbf{day} \qquad \mathbf{d} \xrightarrow{\text{has-a}} \mathbf{mon} \qquad \mathbf{d} \xrightarrow{\text{has-a}} \mathbf{yr}$$

with the obvious abbreviations. Figure 19-1 shows the overall picture, a natural model for a point in time (with a resolution down to seconds).

The 'root' of an ontological class as such is not instantiated. As soon as an object is used as a child of an object, it must be instantiated. In the previous example, the definition of the point

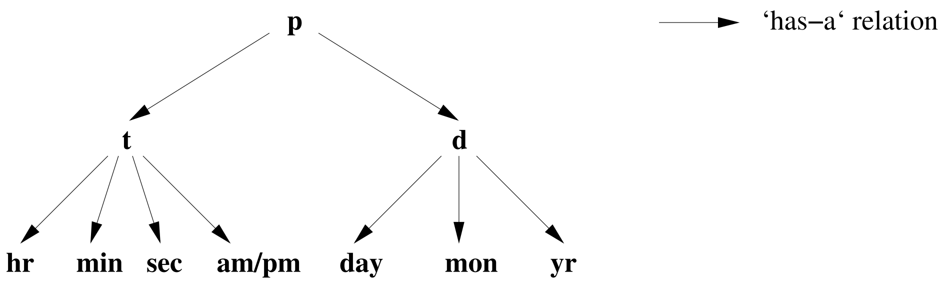

Figure 19-1: Example for the member relation ('has-a' relation): representation of a point in time.

in time is abstract (i.e. **p** = ***{CPointInTime}**; we will use the capital letter 'C' as a prefix to denote the class of an ontological object), yet the direct children are instances for a time and date (i.e. **t** = **time{CTime}** and **d** = **date{CDate}**).

The 'has-a' relation between parent and children has its counterpart in object-oriented programming: the parent corresponds to a class, the members of the class connect to the children. This relation determines the denomination of this relation. In programming languages like `C++` and `Java`, the class definition itself is also abstract while the members are instantiated entities.

19.1.2 Type-Relation: 'is-a'. The *type relation* determines the content an ontological object can carry. The valid types range from atomic to structured types.
Atomic types are string, integer, double, and one out of a list of values (especially boolean). E.g. in the above example of a point in time, it is natural to let **hr** be an integer between 1 and 12, **yr** is an integer, and **am/pm** is either 'am' or 'pm'.
The structured types are defined through ontological classes, i.e. an ontological class determines the type of an ontological object. To go on with the point in time example, **p** can be used to define the type of specialized derivations like the starting point of a period of time, or the wake-up time in an alarm clock application. All of these would be derived from the class **CPointInTime**. If in this situation an ontological class **CTime** abstractly representing the time was defined elsewhere, this class may be used to define the type of the time instance **t** in **p**.

Again, the connection to object-oriented programming can be drawn: the atomic types can be directly related to the according data types of widespread programming languages (e.g. `C++`, `C#`, `Java`). A DO of atomic data type carries content in conformity to this basic type definition, thus atomic values like `CNN`, `42`, or `true`. Using an ontological structure to define the class of an object is comparable to simple inheritance – in particular, the children of the originator are migrated as children of the inheritor.
To illustrate structured type-definitions through classes, assume that the time **t** of class **CTime** according to figure 19-1 is given. If an application now requires a time with a resolution in the range of milliseconds, it can start from **CTime** to define this more precise measure of time $\mathbf{t}_{\text{m-sec}}$. Since $\mathbf{t}_{\text{m-sec}}$ should represent a time, we specify $\mathbf{CTime}_{\text{m-sec}} \xrightarrow{\text{is-a}} \mathbf{CTime}$. The class of $\mathbf{t}_{\text{m-sec}}$ is defined as $\mathbf{CTime}_{\text{m-sec}}$. Further the sub-second resolution is embedded by $\mathbf{CTime}_{\text{m-sec}} \xrightarrow{\text{has-a}}$ **m-sec**, **m-sec** standing for the milliseconds. The resulting member-relations for $\mathbf{t}_{\text{m-sec}}$ are shown in figure 19-2.

As in M3L, we do not allow an atomic ontology class to have children at present (Herzog et al. [2004]). Apparently, that means that an object with an atomic class as type must be a leaf in such an ontological class tree.

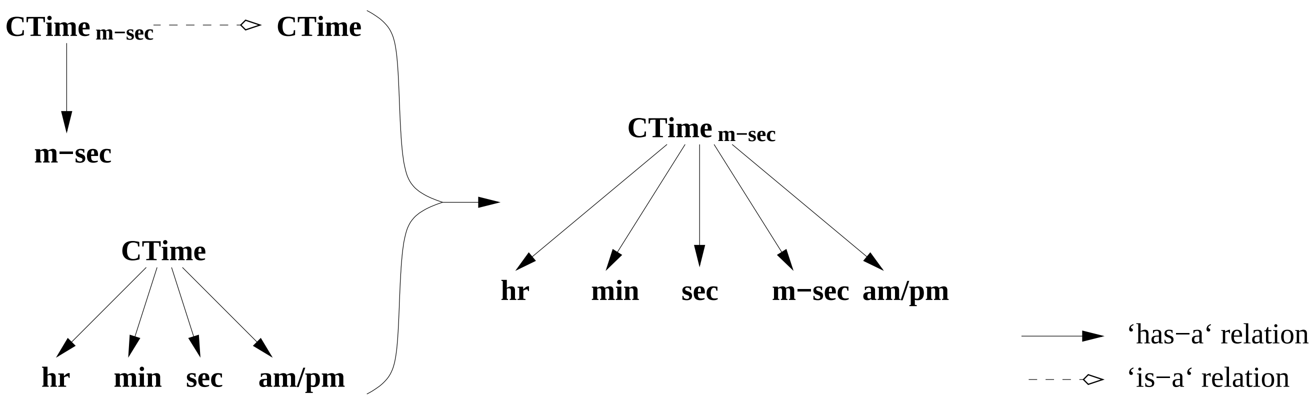

Figure 19-2: Derivation of a time with millisecond resolution from an ontological class representing a time with a resolution of seconds.

19.2. Reuse of Ontological Structures Between Domains

So far, we concentrated on the class of an ontological object and discussed the instantiation state of objects. Two relations can be used to connect objects and classes respectively. Type and member relation become even more powerful to describe the operation parameters of an application if we introduce means to access ontological structures between applications (or domains respectively). Domains are used as part of the world related to a certain application. The domains must obviously not necessarily be constructed disjoint. In addition, DyMALog allows to define domains detached from 'real' applications being controlled by the dialog system, e.g. a domain to describe 'time', 'date', and 'point in time'. Presently, the framework provides a set of such generic domains to ease the design of new applications and interfaces to existing applications respectively. The applications and interfaces can make use of such generic domains. E.g. an alarm clock may rely on the generic domain including 'time'. The generic domain may deliver means to describe the operation parameters of 'time', together with knowledge sources for recognition and analysis, etc. Such domains include the description of entities related to time and date, persons, places, shows, or channels. Of course, the set of these 'generic' domains can be extended in future to provide further building blocks the applications can rely on. An outlook given in Filisko and Seneff [2003] states that newly emerging generic capabilities may be integrated into a central place for reuse in different dialog systems without the need of major adaptations, e.g. means to deal with date, time, or geography.
Parts of the `system` domain, which hosts descriptions relevant to (semi-)generic functionalities, are also available to be utilized in the design of an application, e.g. to access items from a list or for (de-)activating single applications being connected to the system.

Applications can access elements of another domain by importing this domain. After inserting the other domain, ontological classes from the imported domain can be used to build up ontological structures in the original application. The domain of the imported structure, i.e. of all ontological objects related with such a structure, is adapted to the application's domain. E.g. the time and date domain is utilized by various applications, or the description of a person can be used by applications like EPG (actors in a movie), image browser (person names in annotation of a picture), or an application to support personalization (identified user).
Existing ontological classes can be used to define members of an ontological class or to derive additional classes based on an existing class. The combination of 'has-a' and 'is-a' thus supports a modularized approach to define the operation parameters of applications.

19.3. From Ontologies to Object-Oriented Interpretation Trees

The definitions of ontological objects (definition 19.1) and DOs (definition 8.1) already indicate their strong relation, the core components of the DO are a consequence of the derivation from an ontological object.
However, to obtain the o^2I-Trees made up of the DOs the 'is-a' inheritance relations are resolved. I.e. the inheritor assimilates the children of the originator defined through a 'has-a' relation. Resolving the inheritance relations is performed gradually. So, the o^2I-Trees purely rely on 'has-a' relations. During the assimilation process, the inherited children take over the domain of the inheritor.
Figure 19-3 illustrates the derivation of an o^2I-Tree from an ontology. A more concrete example

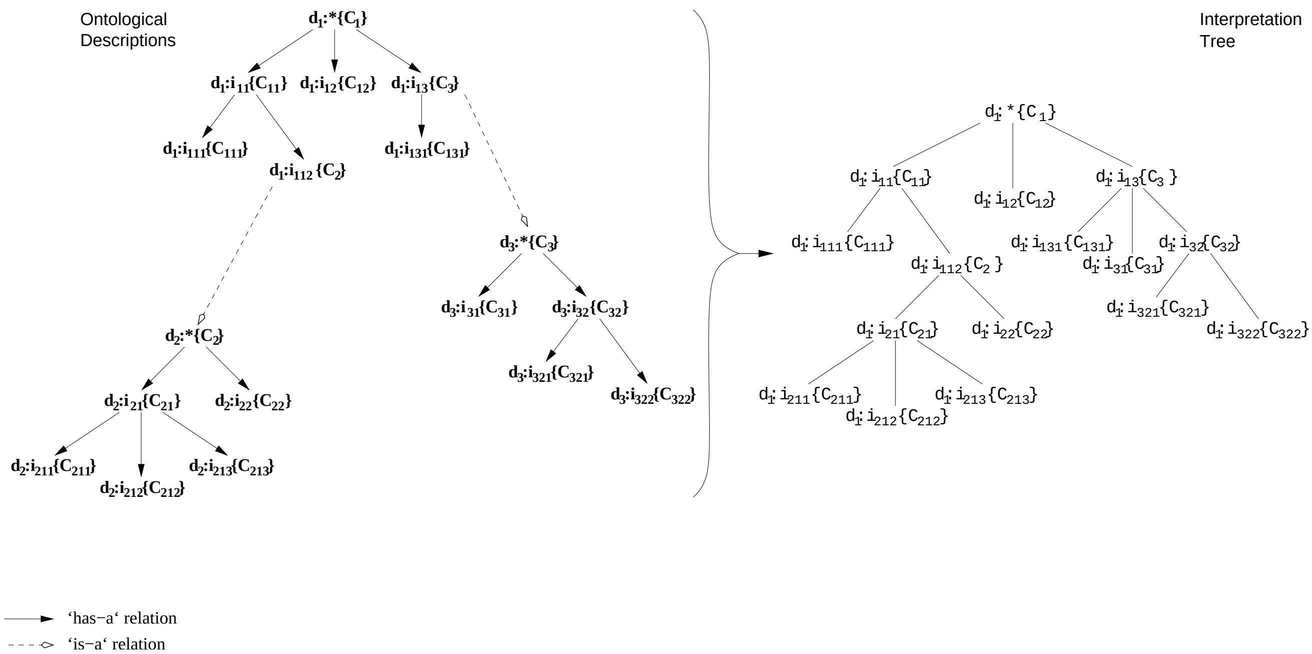

Figure 19-3: Resolving of ontological 'is-a' relations and transfer of domains in the derivation process from ontological classes (on the left) to an o^2I-Tree (on the right).

is given in figure 19-4. To model an alarm clock, we fall back on generic descriptions for the time to define the wake-up time and an audible signal played when the wake-up time is reached. For the o^2I-Tree, the tree of the ALARM CLOCK representing the operation parameters for this simple application embeds the generic structures.

19.4. Compatible Ontological Classes in Different Domains

Section 10.9 already showed how discourse information sharing is realized in the `dialog knowledge processing`. Discourse information sharing is based on mapping (10.1) (page 101), which serves as an indicator whether two objects are compatible. The `domain model` provides this mapping by deriving compatibility of two objects from the ontologies as delivered by the applications.

Two ontological objects $\mathbf{o} = \mathbf{d{:}m\{c\}}$ and $\tilde{\mathbf{o}} = \tilde{\mathbf{d}}\mathbf{:}\tilde{\mathbf{m}}\{\tilde{\mathbf{c}}\}$ are considered to be *compatible* if the following holds:

A domain **d'** exists such that **c** is equal to or derived via 'is-a' relations from an

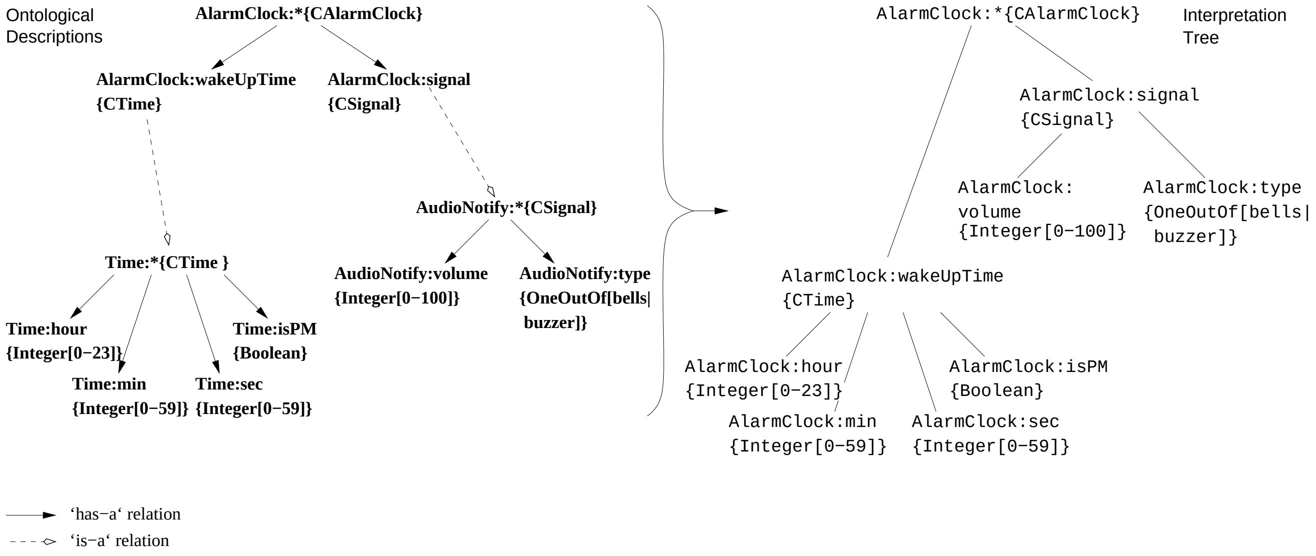

Figure 19-4: Derivation of an o^2I-Tree for a basic ALARM CLOCK application (right side) from ontological descriptions (left side). The ALARM CLOCK is modeled using generic domains to define the wake-up time and the type of signal to awake the user when the wake-up time is reached.

ontology class **c'** in the ontological description belonging to **d'**. Further $\tilde{\mathbf{c}}$ is also obtained from **d':*{c'}**.

I.e., **c** and $\tilde{\mathbf{c}}$ have a common origin. Of course, equal ontological objects are compatible. Two DOs are *compatible* if their corresponding ontological objects are compatible.

If e.g. in the situation of figure 19-3 an ontological object $\mathbf{o}_4$ in domain $\mathbf{d_4}$ is derived from $\mathbf{d_2{:}^*\{C_2\}}$, then this object $\mathbf{o}_4$ is compatible to $\mathbf{d_1{:}i_{112}\{C_2\}}$ and the same holds for the corresponding DOs.

19.5. Relation of Tasks with Ontological Objects

The ontologies provided by the applications also keep track which tasks are associated with certain ontological objects. Section 8.5 already showed how tasks are utilized in o^2I-Trees. However, the `domain model` actually performs the association of tasks and objects based on the ontologies.

During the creation of ontologies the designer can manually attach none, one, or more tasks as listed in table 8-1 (page 66) to each single ontological object. We can look at a task object as a special ontological object. It has a 'one out of a list' type, the list being obviously the tasks shown in table 8-1. The member relation can further connect this special object to each non-task ontological object, including objects of an atomic type. The relation algorithm thus does not have to differentiate between ordinary and task objects, except that tasks can be members of an object of atomic type.

20. Relating Objects via Hierarchical Structures

The basic duty of the `domain model` is to associate DOs or structures of DOs, the o^2I-Trees, according to ontological knowledge[†]. We already showed how exactly ontologies are incorporated by DymaLog.

Integrating single DOs obtained during the `analysis` and embedding the interpretation of the current input into the previous discourse requires single DOs, o^2I-Trees, or combinations of both to be connected in a meaningful way for one of the available applications. The integration algorithm 10.1 to merge hypothesis structure and previous discourse heavily relies on the `domain model` to provide such a meaningful relation.

20.1. Establishing Relations Between Ontological Entities in the Domain Model

Let **O** denote the set of all ontological objects known by the `domain model`. Furthermore the set $\mathcal{T}$ should contain all trees that can be derived due to the member- and type-relations between ontological objects according to the derivation of o^2I-Trees described in section 19.3. Thus, the relations between the objects building up the trees in $\mathcal{T}$ are 'has-a' relations. The `domain model` provides a mapping[42]

$$\texttt{domain model:} \qquad \mathfrak{P}(\mathbf{O}) \longrightarrow \mathfrak{P}(\mathcal{T}) \\ \{\mathbf{o}_1, \dots, \mathbf{o}_n\} \longmapsto \{t_1, \dots, t_k\}, \tag{20.1}$$

$k, n \in \mathbb{N}^{>0}$.

Once a set of ontological objects $\{\mathbf{o}_1, \dots, \mathbf{o}_n\}$ is given, the strategy at present followed in mapping (20.1) is presented in function 20.1 'relate_DOs_using_o2I-tree_lists'. It aims at creating one or more 'has-a' trees that cover all given objects $\{\mathbf{o}_1, \dots, \mathbf{o}_n\}$. Under the assumption that information of one domain given to the system by the user is 'closely' related and that the information of the same domain was gathered in consecutive turns, the coverage should be minimal with respect to some distance measure, see figure 20-1.

A more detailed example for an EPG application was given in section 10.6. Here we summarize shortly this example:

The hypothesis structure from the user input is made out of a single o^2I-Tree with root-DO `EPG:`***`broadcast`***`{CBroadcast}` with age 1, since it is the current user input. From the `discourse memory`, two trees are identified for the merger with the hypothesis structure: one o^2I-Tree with root-DO `EPG:`***`channel`***`{CChannel}` (age 2) and another o^2I-Tree `EPG:`***`broadcast`***`{CBroadcast}` (age 4).

During the first step of the algorithm, equivalent objects (with respect to the relation process) are merged; the resulting set for the relation contains two objects: {`EPG:`***`broadcast`***`{CBroadcast}`, `EPG:`***`channel`***`{CChannel}`}. However, we have to remember that the ***`broadcast`*** object refers to two different root-DOs.

Next, for the set of merged objects all possible permutations are considered. The permutations are then considered separately. The objects in each permutation are related iteratively (e.g. from 'left to right'). At the beginning, a relationship between the first two objects in the permutation

[†]Note that major parts of the domain model are contributed by Dr. Portele, see also page 8.

[42]We can assume the all ontological objects known by the `domain model` are used to build up descriptions of application operation parameters, therefore mapping (20.1) can be defined such that $k \geq 1$ must hold if $n \geq 1$.

Function 20.1 `relate_DOs_using_o2I-tree_lists`(set of ont. obj. **O**)

Data: set of ontological objects **O**: ontological objects to be connected by one or more trees describing 'has-a' relations derived from the ontologies

Result: set of tree sets `T`: each tree set in `T` describes a possibility to relate the objects in **O**

```
// merge functional equal candidates ('equal' objs, derived types, ...)
set of ont. obj. O_merged := ∅;
foreach ont. obj. o ∈ O do
    if some o_equiv ∈ O_merged exists such that o is 'functional equivalent' to o_equiv then
        //discard o
        mark o_equiv with the information that o is functional equivalent;
    else
        O_merged = O_merged ∪ {o};
    end
end

// get all sequential arrangements of the objects in O_merged
set of permutations Π := {π; π is a permutation of the objects in O_merged};

// create all possible relations for the n objects in each permutation separately
foreach permutation π = (o_1, ..., o_n) ∈ Π of objects in O_merged do
    // initialize the iteration: start with the set of (trivial) relations, i.e. the
    // trivial trees containing only o_1
    set of 'has-a' covering tree sets T(π, 1)
    := {t; t is a trivial tree containing only o_1 (according to the underlying
        ontologies)};

    // iteratively create a relation between the objects in π (if n > 1),
    // i.e. loop over all objects in π from 'left to right'
    for integer i from 2 to n do
        // merge the different relations of the first i − 1 objects with the i-th object
        foreach relation t ∈ T(π, i-1) do
            set of 'has-a' covering tree sets T(π, i)
            := {t'; t' is a relation that includes t and combines it with o_i
                (according to the underlying ontologies)};
        end
        increase i by 1;
    end
end

// collect all complete relations that cover all objects
set of 'has-a' covering tree sets T := ⋃_{π∈Π} T(π, n)

return T
```

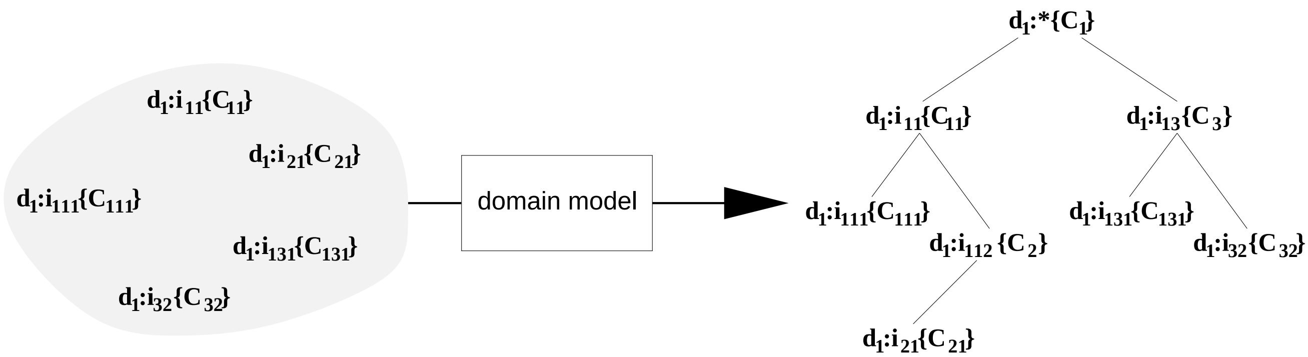

Figure 20-1: Connection of ontological objects according to the ontological classes as given in figure 19-3. The resulting tree corresponds to the smallest possible coverage.

is created. Then, we establish a relation between the result of this relation operation and the third object, and so on. For the EPG example, of course only two objects need to be related, thus only the initial step of the iteration needs to be performed.
Note that we did not discuss ambiguities during the relation process in this recapitulation here.

20.2. Algorithm to Establish Relationships Between Objects

Function 20.1 'relate_DOs_using_o2I-tree_lists' describes the algorithm to embed a set of given ontological objects $\{\mathbf{o}_1, \ldots, \mathbf{o}_n\}$ into trees derived from the ontologies of the applications. For the relation process, we generally consider non-instantiated ontological objects. E.g. the `analysis` of the user input can identify the class of an object, however it cannot decide with entity in the real world is addressed. To illustrate this, assume that a user uttered *"at eight o'clock"*. This is likely to generate an object of a class **CTime**. The user might refer to an alarm clock, program a DVD-recorder, specify a departure time, etc.

The process to establish a relation is not restricted to relations between root-DOs from a hypothesis structure with the roots of o^2I-Trees derived from the `discourse memory`. The result of the input processing (section 21) is a set of DOs. These DOs need to be related to derive a hypothesis structure. The relation algorithm is also utilized to obtain the valid relations.
In the example that will clarify the relation algorithm, we show how the algorithm establishes a relation between a set of single DOs from the user input – equal to the creation of relations between root-DOs.
The example deals with an ALARM CLOCK, the relevant section of the ontology can be found in figure 19-4. We assume that a user uttered *"wake me up tomorrow morning at a quarter past seven with the bell sound"* to specify

`AlarmClock:`*hour*`{Integer[0-23]}="7"`,
`AlarmClock:`*min*`{Integer[0-59]}="15"`,
`AlarmClock:`*isPM*`{Boolean}="false"`, and
`AlarmClock:`*type*`{OneOutOf[bells|buzzer]}="bells"`.

As the DOs can be clearly identified through their instance portion, we omit the domain and class in the following.

20.2.1 Grouping Similar Objects. The algorithm begins with grouping $\{\mathbf{o}_1, \ldots, \mathbf{o}_n\}$ together by merging objects being functionally equivalent during the computation of the coverage. Functional equivalent objects are objects with equal domains, instances[43], and classes. For the classes also similarity in the sense of classes related through 'is-a' relations is sufficient. For domain and instance, wildcards are permitted. These can be used if the domain or instance for an object cannot be definitely identified, e.g. when the domain cannot be determined (compare the above time example).

The DOs given for the ALARM CLOCK cannot be merged.

> **Example 'Alarm Clock'.** Since the DOs derived from the user input cannot be merged, a relation of the set of objects $\{\mathit{hour}, \mathit{min}, \mathit{isPM}, \mathit{type}\}$ according to the `AlarmClock` ontology needs to be derived. I.e. the 'merged' set of objects is equal to the original set.

In the example of the EPG application, section 10.6 and summarized above, the set of objects for the relation was reduced from 3 to 2.

20.2.2 Permutations of the DOs. After normalizing the set of DOs for relation to pairwise different objects, next the iterative relation step is prepared. Computing all different permutations for the set of relevant DOs does this.

> **Example 'Alarm Clock'.** The algorithm will establish a relation between four different DOs. The four objects can be ordered in $4! = 24$ sequences, e.g.
>
> $(\mathit{hour}, \mathit{min}, \mathit{isPM}, \mathit{type})$,
> $(\mathit{type}, \mathit{hour}, \mathit{min}, \mathit{isPM})$,
> $(\mathit{isPM}, \mathit{type}, \mathit{hour}, \mathit{min})$,
> $(\mathit{min}, \mathit{isPM}, \mathit{type}, \mathit{hour})$, and
> $(\mathit{hour}, \mathit{min}, \mathit{type}, \mathit{isPM})$.
>
> For the illustration of the iteration, we concentrate on the last permutation.

Since the set of DOs consists of pairwise different objects, the number of permutations is given by $n!$.

Remark: If for each permutation all possible relations were computed, the outcome of the iterative process would be equal for all permutations. However, presently the algorithm searches for the shortest or nearly shortest relation according to an underlying measure; a major contributor to this measure is the number of connections between the DOs. Then the resulting relations might differ for different permutations.

20.2.3 Iterative Derivation of Relating Trees. Finally, for each permutation an iterative process determines a structure that relates the DOs.

> **Example 'Alarm Clock'.** Consider exemplary the creation of the relation of
>
> $$(\mathit{hour}, \mathit{min}, \mathit{type}, \mathit{isPM}).$$
>
> Starting from ***hour*** (regarded as trivial tree, this is a 'relation' of only ***hour***), next ***min*** is merged. Figure 20-2 shows the result of the created relation, iteration #1, according to the underlying ontology. In iteration #2, ***type*** is added to the result of iteration #1. Then the result of the second iteration is related with ***isPM***.

[43] Objects are generally not related to instances when the `domain model` is requested.

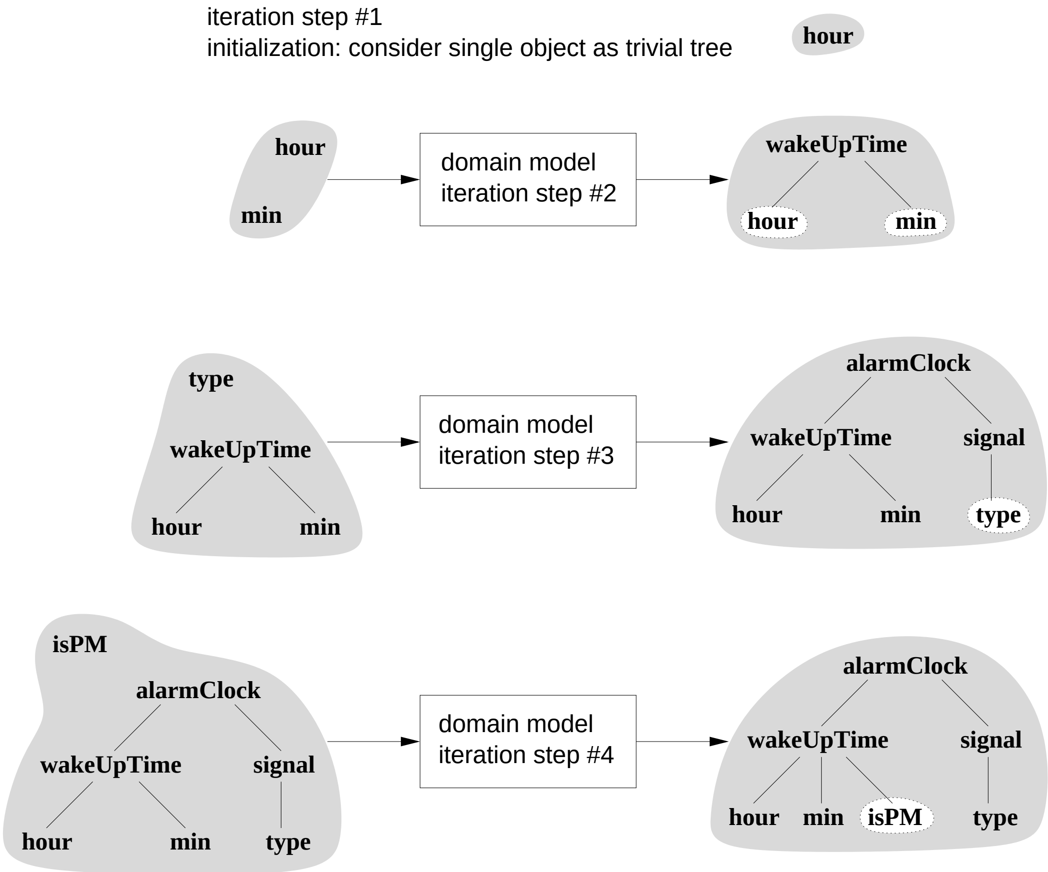

Figure 20-2: Iterative relation of the sequence (*hour*, *min*, *type*, *isPM*) according to the ontology given in figure 19-4 for the ALARM CLOCK example. The newly added objects are marked in the integration result. The ontology does not lead to ambiguous integration results during one of the iterations.

The iterative process starts with the first object in a sequence (iteration 1). A single object can be considered as a trivial relation with only one DO. The ontology may deliver more than one position where the object can be used. E.g. a time for a train timetable could represent an arrival or departure time.

Assume that the first $i-1$ objects are related, $i > 1$. The $i-1$ objects are connected by $n(i-1) > 0$ different relations. To each of these $n(i-1)$ relations t, a relation of the i-th object with t is created. When this operation is performed for all $n(i-1)$ relations from iteration $i-1$, a set of $n(i) \geq n(i-1)$ relations for the first i objects is obtained.

The demanded relations are the outcome of the n-th step of the iteration, i.e. when all objects are treated.

20.3. Evaluating Relations of Ontological Objects

The distance measure for a set of trees $\{t_1, \ldots, t_k\}$ connecting the set of ontological objects $\{\mathbf{o}_1, \ldots, \mathbf{o}_n\}$ according to mapping (20.1) is defined as

$$
\begin{aligned}
&R_{\text{domain}}\left(\{t_1, \ldots, t_k\}\right) \\
&\quad = 1 - \left(\sum_{\substack{\text{ont.obj. } o \\ \text{in } t_i, 1 \le i \le k}} \frac{1}{c_{\text{upper bound}} \cdot n} \right. \\
&\qquad\qquad \cdot \left(\max\left\{1, c_{\text{ref}} - c_{\text{penalty ref}} \cdot \left|\{\mathbf{o}_j; o \text{ references } \mathbf{o}_j, 1 \le j \le n\}\right|\right\} \right. \\
&\qquad\qquad\quad \left. + \mathbb{1}_{\{o'; o' \text{ is leaf in } t_j, 1 \le j \le k\}}(o)\right) \\
&\qquad \left. + c_{\text{penalty trees}} \cdot k \right).
\end{aligned}
\tag{20.2}
$$

Basically, we sum up over single scores for all objects contained in the trees $\{t_1, \ldots, t_k\}$. The single scores are based on the number of objects from $\{\mathbf{o}_1, \ldots, \mathbf{o}_n\}$ being associated with such an object (parameterized by c_{ref} and $c_{\text{penalty ref}}$) and the location of the object as inner node or leaf, leaves get an extra 'penalty'. Further, for each single tree needed to cover the objects, a constant penalty is added (parameterized by $c_{\text{penalty trees}}$). The resulting score is 'normalized' by the number of objects in the origin set of objects multiplied by an estimated upper bound of objects which might be created per object in a tree during the connection process (parameterized by $c_{\text{upper bound}}$). Finally, the rating is transformed by subtracting the resulting value from 1. We consider coverages with high ratings of R_{domain} to be better that results with lower scores[44].

Based on rating (20.2), the `domain model` can execute a preselection on the different alternative covering tree sets to disburden the consecutive steps in the dialog engine from a high payload of a large number of different relations.

20.4. Establishing a Relationship of Dialog Objects and Ontological Objects in the Relation Process

The definitions of DO (definition 8.1) and ontological object (definition 19.1) already indicated the close relation of both. The components in AIDE request the relation of DOs from the `domain model`. Creating a relation itself is however performed on ontological objects. Thus, we have to map DOs onto ontological objects and vice versa after the covering trees are obtained.
The `domain model` restricts itself on the core elements of a DO, which can be directly related to an ontological object. In the resulting relating trees, an identifier is attached to the affected ontological objects that enable components like the `dialog knowledge processing` to restore the original DOs from these ontological objects. Ontological objects from the trees without such a relation are the source for additional DOs in the o^2I-Trees.

21. Input Processing

The input processing – as well as the output creation presented in section 22 – is heavily affected by the application-blind design of the dialog engine AIDE enabling dynamic application setups.

[44]The constants in R_{domain} are presently manually adjusted on sample data from interactions with the system.

Thus, we will allow a glimpse at these two areas in DymaLog without going too much into detail, since our focus is on the analysis part inside Aide.

In principle, the framework allows the user to interact with the dialog system using different modalities. Figure 21-1 gives an overview on the input processing by DymaLog.

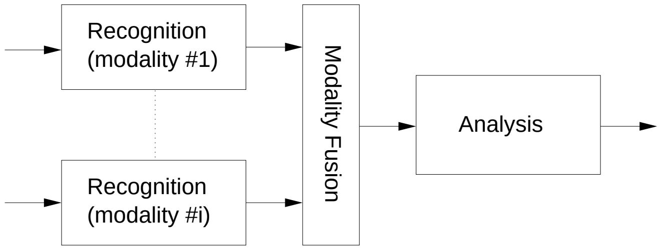

Figure 21-1: Outline of the input processing, excerpt from the outline of DymaLog shown in figure 6-1.

The multi-modal input is based on a set of recognizers, each being adapted to a certain modality. Common modalities are speech, pointing, or more advanced gestures. As the present input processing is oriented at the Spice system (Kellner and Portele [2002]), the single `recognition` results are combined into one structure for an integrated representation. Starting from the integrated representation, the `analysis` component identifies semantic entities. Other variants of performing multi-modal input `recognition` and `analysis` can of course be thought of.

21.1. Recognition of the User Input

As stated, before we currently rely on speech as input modality from the user. Changes in the knowledge sources related to the ASR might become necessary due to certain discourse states, changes in the application setup, or (de-)activating functionality inside applications.

For the ASR, the lexicon and the recognition network need to be updated. A recognition network is used instead of a language model, the major drawback being a more restricted `recognition`. At present, single lexica and networks for each application are integrated on top of a basis lexicon and network. The basis includes e.g. means to address list items.

Restricting on speech as only input modality yet makes the integration process of the `recognition` results obviously trivial.

21.2. Analysis of the Recognition Result

The aim of the analysis of the recognition result is the identification of semantic meaningful entities. For the speech input, a NLU based on a context free grammar is used. During the understanding process, DymaLog links text passages to semantic entities. The semantic entities contain up to three elements[45]:

- *Ontological Class and Domain.* Certain text passages reference ontological classes. Note that these classes can be both, specialized to certain applications or belonging to generic functionalities provided by the framework itself. Together with the ontological class, matching domains for this passage are derived. Both, class and domain derived from a text passage have not to be unique: the analysis result then contains the alternative derivations.

[45]For the present relaxation enabled through the framework, a special tag might be added to a semantic entity. See also section 10.11.

- *Value.* Semantic entities, which contain knowledge on an ontological class, optionally comprise a value bound to this class.
- *Task.* In addition or instead of an ontological class, a semantic entity may include a task according to table 8-1.

Examples for semantic entities are the ontological class representing a TV-channel `CTVChannel` in a `TV-set` domain with the value `CNN` and task SWITCH. The hour part `CHour` of a `time` contains the value `20`. Another example is the generic reference to the seventh list item `CListReference`, value equals `7`, which should be selected (task: SELECT). The domain – which is also obtained during the `analysis` – has not to be unique. Text sequences used to obtain a semantic entity can be spread over different domains, e.g. due to the reuse of classes through inheritance. The hypothesis selection process then identifies the best matching applicable domain indirectly through the selected hypothesis.
The semantic entities from the `analysis` form the basis of a transfer into DOs. Thus, the semantic entities relate to flat ontological structures. The existence of a task together with the ontological class in a semantic entity however is not broken down into two separate units, rather the task stays connected to the DO derived from the ontological class.

As for the ASR, the NLU is equipped with adapted knowledge sources to reflect changing needs from the dialog engine. The grammar applied in the NLU is presently basically the concatenation of a base grammar with a set of application specific grammars from the connected applications.

22. Output Creation

While the major parts of classical input processing techniques are adapted to dynamic application setups via their knowledge sources, the output creation receives its power from the combination of extensible markup language (XML) and web technologies. The reverse process of the input process also mirrors its structure, figure 22-1. The `analysis` is countered by the `generation` that

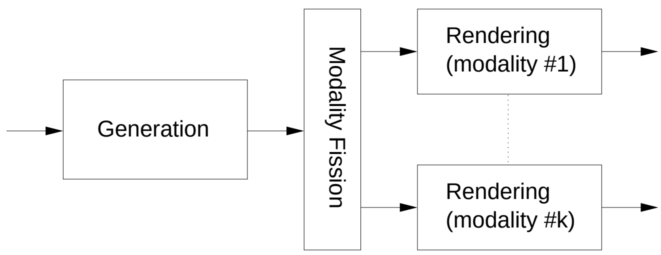

Figure 22-1: Outline of the output creation, excerpt from the outline of DYMALOG shown in figure 6-1.

transforms the hypothesis structure – which can be considered to represent a hierarchically structured arrangement of enhanced semantic entities – into representations suitable for `rendering`, e.g. sequences of words. The `generation` is capable of creating the input for `rendering` of different output modalities. A separate step to split up an integrated input for `rendering` into units for different modalities is not required. Thus, the counterpart of the `modality fusion`, the `modality fission`, performs the easy task to deliver the modality specific output to the proper renderer.

The current visual output is capable of presenting basically 'static' application related content, like some output values, tables, pictures, or movies. A separate display area presents more dynamic content related to the status of the dialog system in basic manner. An approach towards a more sophisticated output, especially for multimedia applications, is e.g. shown in Beckham et al. [2001].

22.1. Generating Input for Rendering from Hypothesis Structures

As an XML representation is defined and applicable for hypothesis structures (Burke et al. [2002]), XSLT can be used to transform these documents. Namespaces take over the task to partition the space of all hypothesis structures for the different application domains. Besides a generic system stylesheet, the applications provide separate stylesheets for their respective domains, one for each modality. For the different output modalities, the `generation` applies the generic stylesheet and application specific stylesheets applicable for the hypothesis structure obtained from the communication management, i.e. the stylesheets of domains contained in this hypothesis structure.
The partitioning according to the domains and modalities allows each application to deliver its own XML stylesheet language transformations (XSLTs) without dependencies on other applications, this holds analogously for the application of the stylesheets on the hypothesis structures.
The reuse of structures is mapped to imports of XSLT documents. If an ontological structure is reused in another ontological structure and the utilized structures deliver XSLTs for generation, the higher-level XSLT is allowed to base the transformation related to the imported parts on these stylesheets.

The only application specific elements in the `generation` are the knowledge sources, i.e. the stylesheets for the transformations. The component itself is generic in a sense that changes in the application setup updates the knowledge sources attached to the `generation` but leaves the `generation` unchanged.

22.2. Categories of Output

Note that the *output of the dialog system itself* is discussed here. Depending on the operations performed as consequence of the hypothesis structure delivered to an application, the direct output of an application may become noticeable for the user, e.g. music played by a MP3 player, changes in the light configuration of a room, or a ticket printed by a train ticket automat.
The output presented towards the user through the dialog system can be separated into two categories. On the one hand, the user is informed on the status of the dialog system itself, e.g. the activation status of the microphone or the state of the interaction with an application. On the other hand, the result of the application processing is presented to the user. Examples are a list of trains for a certain connection, the number of EPG entries matching given restrictions, or the success message of a device operation like switching the channel of a TV.

As indicated before, the `generation` at present takes over some tasks of the `communication management` that might become obsolete in a more sophisticated communication management. This comprises more explicit formulations of both, the conditions leading to certain responses and the realization of the response itself.

22.3. Rendering of Audible and Visual Output

DymaLog interacts towards the user with audible and visual output. The audio part mainly consists of spoken language (maybe SABLE annotated), earcons might be used to support the output. The visual output consists of the presentation of pictures and (structured) text. Feedback on the state of the dialog system and information related to the active applications is given. Examples for the rendering results are given in chapter III. Figure 22-2 shows the `rendering` in more detail.

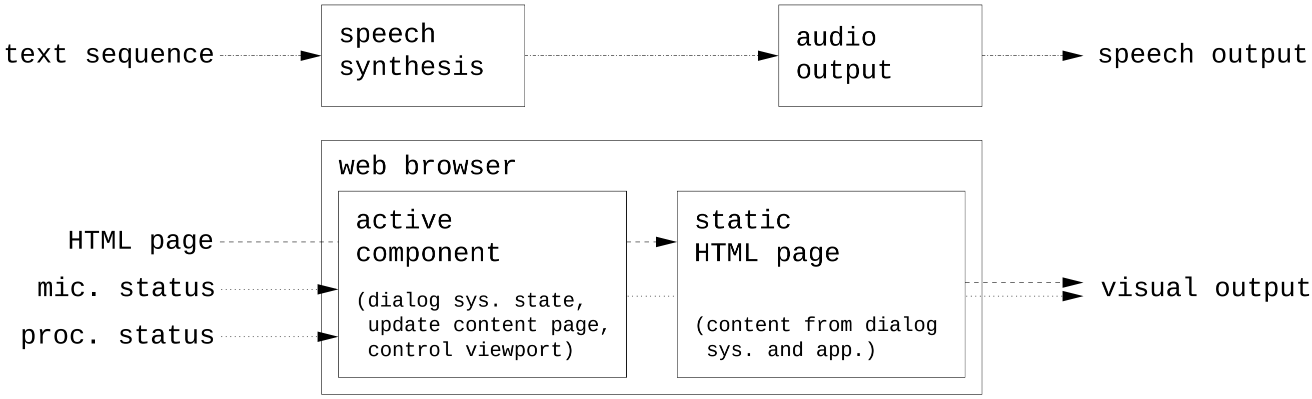

Figure 22-2: Rendering of spoken language and visual output in DymaLog.

22.3.1 Spoken Language Output. The stylesheets create text sequences that need to be communicated through spoken language to the user. A speech synthesizer is used to render the sequence for audio output. More advanced language generation targeting at application-independence is e.g. presented in Galley et al. [2001].

22.3.2 Visual Output. The stylesheets for the visual output are used to generate hypertext markup language (HTML) pages rendered by a web browser. Thus, the capabilities of HTML are used to show information to the user. For a common look, cascading stylesheets (CSS) determine parts of the design of the pages as rendered by the browser.

Since HTML is a structural language to describe content without means to observe changes in the surrounding outside the browser, it does not support automatic loading of a new page once it was generated. This leads us to an active component, embedded in the web pages.

22.3.3 The Active Dialog System State Monitoring and Visual Output Modality Control. The HTML pages created by the `generation` are not directly made visible by the web browser. Instead these pages are embedded into a frame-set that includes also an active component[46] for immediate feedback besides the HTML page. The active component serves three tasks:

Update of the Content Page. To cope with new visual output, the `generation` informs the active monitoring component as soon as a new HTML page is available. It then directs the browser to display this new page.

Feedback on the Dialog System State. The second element in the active component gives feedback to the user on the present state of the dialog system. A basic anthropomorphic character informs via facial expressions on the overall state of the dialog system. These states comprise

[46] For the active component, also a standard web technology is used. It is realized as a `Java` applet.

- *Sleeping.* The dialog system does not expect input to operate applications. However, as we see later, as an exception DymaLog still listens for certain phrases in order to change to an active state where the system is open for input from the user.
- *Listening.* The second idle state represented through the anthropomorphic character indicates a mode in which the dialog system waits for input from the user.
- *Analyzing/Processing.* After the `recognition` identified input that should be processed by the dialog system, the anthropomorphic character notifies the user that processing the input by the dialog system and applications takes place. Discrimination between the processing inside the dialog system and the applications respectively is conceivable. Note that in principle the framework includes the technical means to allow barge-in, at present however handling of barge-in is out of focus.
- *Responding.* After analyzing the user input and operating an application, the output is prepared. The character also accompanies the preparation and presentation.

The active component in the visual output is informed on changes of the dialog system state by two elements in the framework: the interface component for recording the audio signal from the microphone together with the ASR to detect if new user input is observed and the communication management.

Control of the Viewport. The pages displayed in the browser adhere to the restriction that the required width of the page must not drop under a certain level, whereas principally no restrictions are posed on the height. The framework supports this by providing the application-independent control of the viewport in the browser. Operations to control the viewport include relative movements and absolute addressing (Criterion 4.2.2(2)).

According to Lamel et al. [2000], the state of the discourse needs to be transparent to the user. Since the dialog system is a partner in the interaction with the user, DymaLog includes the representation of its state in the response to the user.
The usage of standard web technologies allows for a flexible choice of the device for the visual output. As long as a device supports the required technologies and can connect to the network, which also accommodates the dialog system, the visual output can use it. Examples are devices build around a touch screen (e.g. tablet PC, web tablet, iPronto, handheld/organizer), a web enabled TV-set, or a PC.

Chapter III.

The *Marvin* Spoken Language Dialog System

In order to prove the validity of the concepts introduced in DymaLog for automatic (spoken language) human–machine dialogs, we realized these concepts in the *Marvin* dialog system. By means of interactions between a user and the *Marvin* dialog system, we will demonstrate how the various aspects of our application-blind DymaLog introduced in chapter II contribute to allow the dialog system to communicate with the user. The DVD belonging to this thesis shows video recordings of a user interacting with the system.

At present, the focus to enable application-blind dialog processing in the sense of the previous chapters is on the dialog engine Aide according to figure 6-1, page 48. Consequently, not that much effort has been spent so far to optimize the automatic combination and integration of the knowledge sources, i.e. the core knowledge sources provided by the system and from the different applications.
This becomes especially relevant for the input processing which is borrowed from the *HomeLab* prototype, section 23.7. In ASR, generally recognition performance degrades with an increase of entries in the associated lexicon and the resulting extension of the language model and/or recognition network. For NLU, an increase in coverage of sequences, which can be mapped on semantic entities, may lead to additional ambiguities if the extension was not designed carefully. However, in the situation of dynamic application setups ambiguities are inevitable: consider a CE environment in which the same dialog system controls a TV-set and a DVD-recorder. The user input *"channel seven"* might be related to the TV-set (e.g. as a user reaction on *"which channel do you want to see?"*) or the DVD-recorder (e.g. *"which channel do you want to record at eight?"*).
To summarize, the performance of the input processing equipped with non-optimized knowledge sources requires 'experienced' users to interact with the dialog system, mainly for optimal ASR performance. Therefore, interactions shown here are dialogs of the system and users skilled with the *Marvin* dialog system.

At first, we look on the realization of DymaLog in a spoken language dialog system in section 23. The modules that realize the components of the framework are discussed, and the underlying platform is outlined.

Section 24 pursues section 23. An overview on the applications enabled for the use in the *Marvin* dialog system is given. The criteria catalog in section 4 is evaluated for the dialog system and the realized applications.

In section 25, the basic operation principles from DymaLog realized in the dialog system are explained by means of a sample interaction. The next sections also use sample interactions

to discuss the concepts introduced in the previous chapter.
The sections 26, 27, and 28 are based on interactions with the applications enabled for use with the *Marvin* dialog system. The realize features are investigated in order to show that the dialog system fulfills the requirements posed on the framework by the criteria given in chapter I.

23. Realization of DyMALog for Human–Machine Dialogs

The realization of DyMALog by the *Marvin* dialog system prototype reflects the division into a variety of components shown in figure 6-1.

Appendix A3 provides an overview on the code complexity of the *Marvin* dialog system. Appendix A4 allows to catch a glimpse on some of the annotation and editor tools. These tools support the creation of new applications and simplify the creation of corpora to train the parameters of the measure underlying the selection process.

23.1. Outline of the Marvin *Dialog System*

The screenshot in figure 23-1, and also figure 23-2, show the separation of the dialog system into modules. The major differences to the framework are

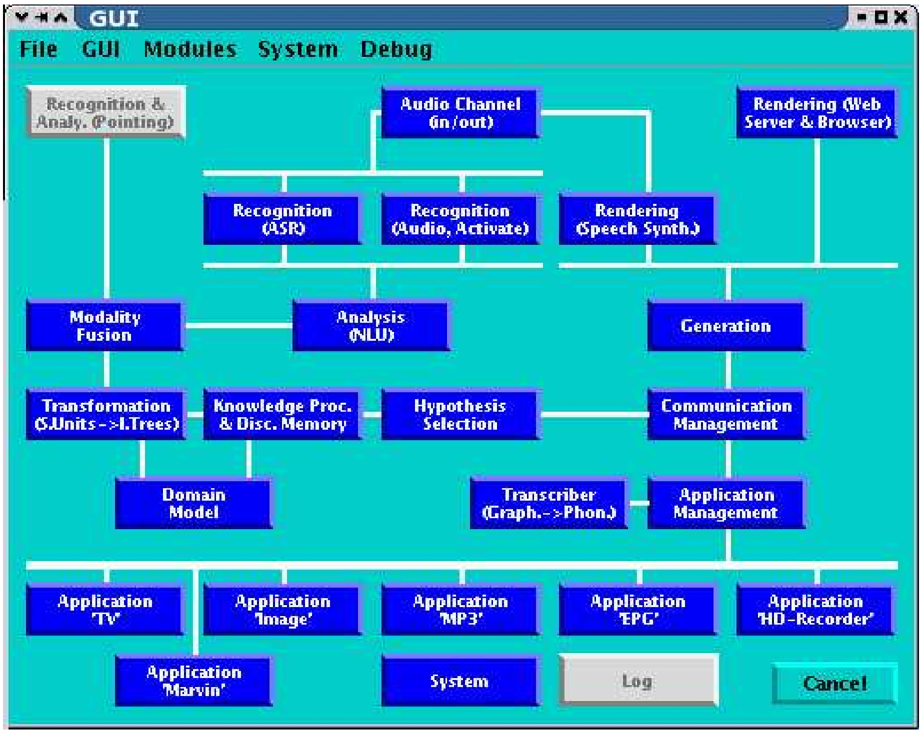

Figure 23-1: Structure of the *Marvin* dialog system realizing the framework presented in chapter II. Each box represents a single module, the lines indicate the flow of data, see also figure 23-2. The GUI presented in this screenshot is the technical GUI that informs on the status of the system. It is generally not visible to a user. The modules "`Recognition & Analy. (Pointing)`" and "`Log`" are not active in the shown setup.

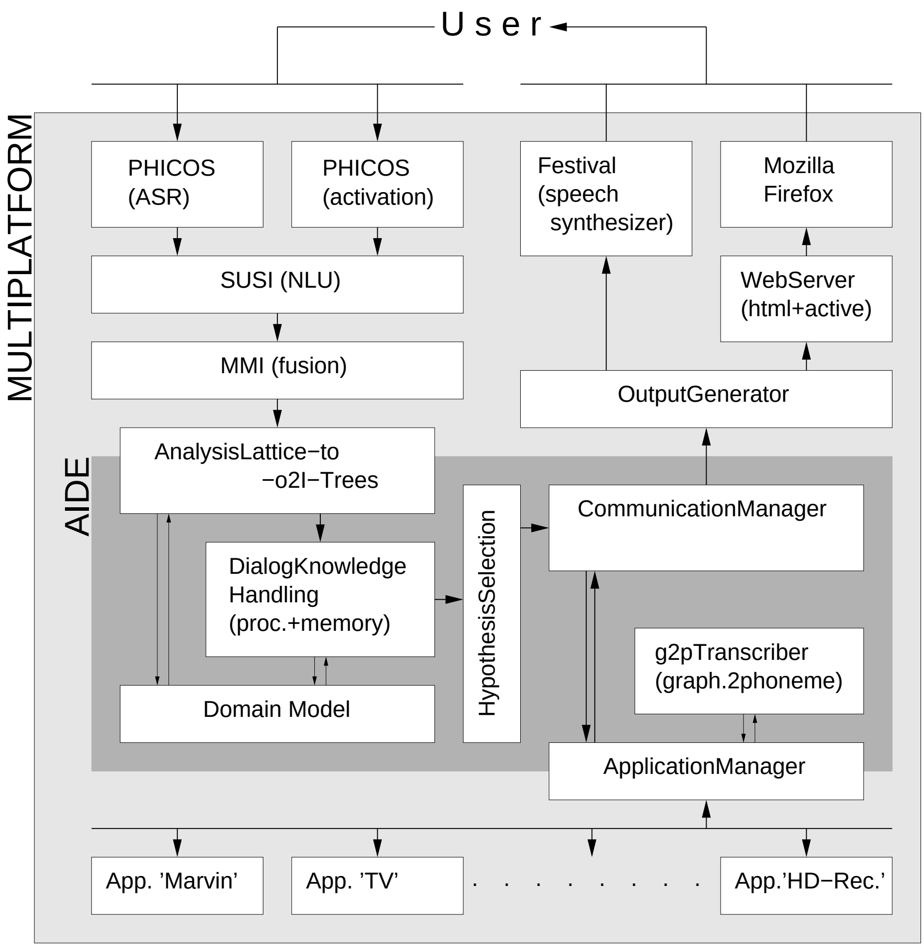

Figure 23-2: Overview of software modules and components in the *Marvin* dialog system, cf. figures 6-1 (page 48) and 23-1. The connections between the modules sketch the basic data-flow during the processing of the user input. Other flow of data is omitted, e.g. exchange of knowledge sources when a new application is connected to the dialog system. As indicated, the components are realized as modules within MultiPlatform.

- the close coupling of the `discourse memory` and `dialog knowledge processing` into the "`Knowledge Proc. & Disc. Memory`",

- the explicit breakdown of the components inside the input processing and output creation,

- the support of the "`Application Management`" by the "`Transcriber (Graph.`→`Phon.)`", and

- the listing of applications currently enabled for the operation through the *Marvin* dialog system.

After the thorough discussion of DymaLog in chapter II, we can concentrate here on the completion of the previous presentation with portions not being directly relevant for the framework but the system.

23.2. Input Processing

As noticed before, the present *Marvin* dialog system relies on spoken language input.

23.2.1 Automatic Speech Recognition. The ASR engine deployed inside the dialog system is the Phicos automatic speech recognizer. It is covered in more detail in Steinbiss et al. [1995]. To constrain the search during the recognition process, a network on word level describes the permitted combinations of words. It was favored compared to a language model based description of valid word sequences. The stronger restriction of permitted sequences had negative effects on the flexibility in recognizable speech input. However, the lack of training material in the different possible application setups did not allow us to train reliable language models, thus leading to the more robust network based approach.

For the analysis of the ASR result, the Susi NLU package is used (Kellner and Portele [2002]). Even though it is capable of weighting the rules applied during the understanding process, missing corpora for the estimation of these weights in the various application setups prevented us from determining appropriate results. As for the recognition lexicon, the fallbacks are equally weighted rules.

Note that the assignment of weights for the rules inside these networks and/or estimation of a language model in a dynamic application setup would require the models and mechanisms to update the overall knowledge sources and adapt the parameterization when changes in the setup occur (Rayner et al. [2001]). Especially, new applications must be integrable at later times. In addition, all applications would need justified parameterizations. In corpus-based approaches, corpora to estimate these parameters are required for each application.

23.2.2 Activation through Spoken Language and Identification. Figure 23-1 shows a second recognizer besides the "`Recognition (ASR)`", the "`Recognition (Audio,Activate)`". The *Marvin* dialog system supports two activation modes for spoken language input:

- *Activated.* The activated state represents the operational mode. The ASR listens to the user for input. The dialog system handles this input to operate the applications.

- *Deactivated.* In the deactivated mode, the dialog system does not process speech input by the user except for certain activation sequences.

A special recognizer, the "`Recognition (Audio,Activate)`" waits for a small set of fixed expressions to wake-up the system, i.e. change the state from deactivated to activated where the 'normal' ASR takes again over the task to handle spoken language input. The way back, from the activated state to the deactivated mode, is either explicitly initiated by the user (e.g. through the input *"go to sleep"*) or triggered after a certain period without input to the system.

The "`Recognition (Audio,Activate)`" is also responsible for the identification of the user. While listening for a restricted set of input sequences, especially the activation sequences, once input is obtained it performs a comparison of the features derived from the user input with reference features stored beforehand of a certain set of users. Unknown users are mapped to a generic profile.
The identification may e.g. be used for personalization and adaptations towards the user (Litman [2001]). A system to control applications in the CE environment might be used by different users inexperienced in the interaction with a spoken language based dialog system. Requirements for a publicly accessible dialog system comprise the speaker independence of the ASR and the use of natural and continuous speech (Giachin and McGlashan [1997]). To improve the performance of

the recognition, personalization can enable the use of user adapted ASR features. Lamel et al. [2000] demand the maximum possible performance of recognition and analysis for dialog systems. Furthermore, the identification of the user allows user-dependent defaults in the applications, rights management to access certain applications, user-related content in applications, dialog strategies, etc. (Litman et al. [2000]).

23.3. Dialog Engine Aide

The modules of the dialog engine Aide realize the concepts of DymaLog. The components find their counterparts in modules of the dialog system. For practical reasons, the integration of analyzed user input and previous discourse is closely coupled with the memory facility into the "`Knowledge Proc. & Disc. Memory`".

23.4. Output Creation

The *Marvin* dialog system utilizes the visual and audible output modalities for the presentation of the output from the dialog system towards the user. Operations inside the applications, however, may also lead to changes in the environment not being part of the *dialog system reaction*, e.g. switching a channel on TV or printing a train ticket.

23.4.1 Visual Output. A screenshot of the visual output creation result is shown in figure 23-3[1]. The output is basically divided into two major parts.

The left part hosts the active element being capable of providing feedback on the state of the dialog system. The comic character uses the facial expressions listed in section 23.4.2, also shown in figure 23-4, to inform the user on the state. Underneath, a bar segmented into three elements reflects the microphone level (similar to traffic lights: a yellow segment indicating a low microphone level is shown in the screenshot).
The active component drives the status line of the web browser. Besides browser messages that pop up, it displays the ASR result, segmentation information from the NLU result, and again the microphone level ('`recognized text - NLU segmentation - mic. level`').

The right area displays a HTML page. Its origin is the hypothesis structure and result structure, generally obtained during the handling of the user input, including the interaction with the applications. On top of an XML representation of the hypothesis structure and result structure, an XSLT stylesheet is applied to generate a HTML page. The reuse through building blocks was already discussed before.

The use of standard web technologies makes it easy to switch between different output devices[2]. Especially, decoupling the output device from the device hosting the dialog system is obtained as a side effect. The use of CSS supports a consistent look.

[1] The screenshots presented in this chapter are partly taken after the interaction of a user with the dialog system. Once the dialog is finished and the system went into the sleep state, the backward functionality of the web browser can be used to recover the browser-based visual output. However, for the screenshots taken on this basis the active component does not reflect the actual state during the interaction (i.e. 'presenting') but indicates that the system is in 'sleep' mode.

[2] The output presented by the web browser is partitioned by a frameset. A `Java` applet realizes the active component.

Figure 23-3: The user GUI of the *Marvin* dialog system rendered by the Mozilla Firefox web browser. The left stripe is reserved for the active component that informs on the system's state. HTML output based on the result structure uses the broad right area. The screenshot shows the system after the start-up phase. A set of applications is already connected to the system. The feedback character indicates that the system is not active now.

23.4.2 The Active Component in the Web Browser. The active component being part of the visual output serves two tasks: (i) it monitors whether updated HTML pages are available for displaying and (ii) informs the user on the state of the dialog system.

As soon as an updated page is available for rendering in the web browser, the active component is informed and triggers the web browser to reload the updated pages. The pages are provided by a basic web server running inside the *Marvin* dialog system, thus the pages can also easily be delivered via a network.

Besides the more technically appearing representation of the microphone level, the face of a basic character illustrates the internal state of the dialog system through its facial expressions. The most commonly used expressions are subsumed in figure 23-4. The character's face will also be referred to as 'Marvin'.
The expression "*asleep*" indicates that the dialog system is presently in the deactivated state, thus operating an application would require to 'wake up' the system again. During the "*waiting*" state, the system is activated and waits for speech input by the user. "*Listening*" indicates that audio signals are presently detected and the system assumes that these signals are speech input. While the speech input is processed, "*recognizing*" is shown, followed by "*analyzing*" which includes the NLU, the knowledge processing in the dialog engine, and the operation of the applications. Finally, "*presenting*" supports the audible and visual output that is presented to the user.

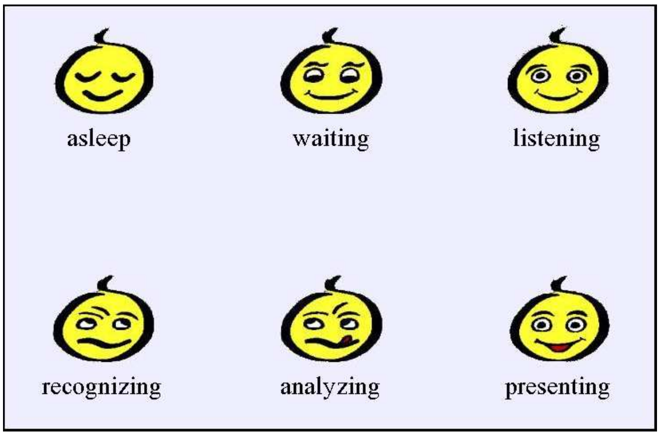

Figure 23-4: Facial expressions of the feedback character 'Marvin' to present the state of the *Marvin* dialog system towards the user.

23.4.3 Audio. The task of generating word sequences from the hypothesis structure and result structure is performed analogously to the generation of HTML pages described before.

The FESTIVAL speech synthesis system performs rendering the resulting 'sentences' (Taylor et al. [1998], Black and Taylor [1997], Festival Software [2005]).
The spoken language output contains paralinguistic expressions like "*a-ha*" and "*emh*". These sounds are used as acknowledgement to indicate towards the user that the system received 'something' and is busy handling this input. Earcons, i.e. roughly speaking a set of sounds, each sound being associated with a certain event, are presently not utilized in the *Marvin* dialog system. Earcons correspond to icons in GUI based operating systems and applications. They could e.g. be used to inform the user that the microphone is again ready to accept input after it was closed or be associated with certain applications such that the user knows which application she is currently mainly interacting with.

23.5. Applications

An overview on the set of *Marvin* enabled applications to verify the realization of DYMALOG in interactions of users with the dialog system is given in section 24.

23.6. The MULTIPLATFORM *Architecture*

The separation of the dialog system into a set of specialized modules is implemented on basis of the MULTIPLATFORM architecture, see figure 23-2. MULTIPLATFORM[3] was the reference platform of the Man-Machine Interfaces (MI) group at PHILIPS Research, Aachen (The SMARTKOM Consortium [2003]). MULTIPLATFORM does not require a central control instance but provides a set of blackboards that enable the communication between modules running inside the platform. The GUI shown in figure 23-1 gives a bird's-eye view on the setup with respect to the modules and basic information on their state, e.g. (not) active, idle, and processing. This technical GUI is not intended to be accessible by the user.
The development of MULTIPLATFORM was motivated by the necessity to provide a flexible envi-

[3]Actually, the PHILUS platform directly derived from MULTIPLATFORM provides the reference platform of the MI-group at PHILIPS Research.

ronment for the large scale Verbmobil research project (Wahlster [2000]). Verbmobil realizes a complex speech-to-speech translation system and was funded by the German government.

MultiPlatform provides a flexible tool to combine a set of functional units. The major application area for the platform is the arrangement of modules for spoken language dialog systems. However, also an information retrieval architecture has been realized at Philips Research on top of MultiPlatform. The partitioning of tasks into separate components allows an easy exchange of components, e.g. to evaluate alternative approaches to a certain problem, as long as the replacement adheres to the given interfaces.

23.7. Marvin*'s Predecessors*

The development of the Susi framework was a major milestone in the history of spoken language dialog systems at the Philips MI group (Aust [1998]). It includes capabilities for natural language analysis and dialog modeling, mainly for telephony applications. The work on this framework led to major contributions for the commercial success of Philips Speech Processing, Telephony[4].

Susi is the basis of several telephony applications. Two of the systems with a direct link to the commercial customers are Taba and Padis. The Taba system is a train timetable information system, initially developed to give information on connections between 1000 larger cities in Germany. It was the first major system and served as the first showcase of Susi (Aust [1998]). Another powerful application of Susi is the Padis directory assistance system (Kellner et al. [1997]). A demonstrator in our laboratories served the callers for quite some time.

Later developments enhanced the framework and applied it in more CE style applications (Kellner et al. [2000]). The multi-modal Spice system realizes spoken language access to an EPG. Among other things, Spice allowed the access to broadcasts by fractions of the (complete) title or content from the description. Examples for requests are *"give me info on Harry Potter [and the prisoner of Azkaban]"* and *"which movies with Clint Eastwood"*. During the participation in the SmartKom project, the Philus platform as the basis of modularized systems in spoken language processing has been developed on top of MultiPlatform at Philips research. The Phicos speech recognizer and parts of the Susi framework contributed to the available modules for the Philips adaptation of MultiPlatform. These modules enable speech recognition and analysis.

In the first dialog system on the basis of Philus, the predecessor of *Marvin* ('*HomeLab* prototype'), the mature Susi dialog manager has been used. The pre-*Marvin* dialog system is the first system of the MI group that allowed access to a set of CE applications (Kellner et al. [2001]). The dialog model and major parts of the applications are directly implemented in the Susi scripting language ('HDDL'). The system allowed us to gather experience in multi-application setups.

The findings derived from interactions with the pre-*Marvin* dialog system had major influence on the design of DyMaLog. Aide replaces the previous Susi dialog modules. As well, the input processing and output creation are adapted in the course of the transition.

[4]ScanSoft acquired Philips Speech Processing in 2003. ScanSoft also bought other major players in the field of speech processing.

24. *Marvin* Enabled Applications

The *Marvin* dialog system has been implemented to investigate the concepts that have been developed in DyMALog for the interaction between humans and machines. A meaningful evaluation implies that a set of applications is available to be controlled by the dialog system. As a starting point, several typical applications in the field of consumer electronics are realized. We will give an overview on these applications in this section.

Note that the applications are currently self-contained software packages. These packages may be used as a starting point to control real devices or interact with external services.

24.1. Application TV-set

The TV-set application emulates a TV device. The virtual device shows a TV-set. The channel content is suggested by screenshots of broadcasts, figure 24-1. The device implements vital functionality of a TV-set, like switching the channel relative ('channel up', 'channel down') or absolute ('channel ARD', 'channel CNN') and also relatively and absolutely changing the volume ('increase the volume', 'louder', 'mute'; figure 24-2), and powering the device on and off.

Figure 24-1: Virtual TV device application. The screenshot was token directly after the program was switched to channel 'ARD', which is programmed on channel number 1.

Figure 24-2: Virtual TV device application. The screenshot shows the reaction on the change of the volume to 'high'.

To transfer from a virtual device towards a real TV device mainly the driver level to map functionality on device operations is required, i.e. the calls to manipulate the virtual TV set have to be mapped to calls operating the real device.

24.2. Application Music Player

Large storage capabilities allow holding large media collections on a CE device. This includes devices equipped with harddisks (HDs) or flash memory to store music. A database stores a list of music titles. The music player application allows to browse through the music collection by addressing the desired artist, album title, and/or song title, figures 24-3 and 24-4. After selecting songs by specifying e.g. the artist and/or album, the player starts playing the selected songs. Basic capabilities for a music player like play, stop, and skip are also accessible via the dialog system.

Figure 24-3: Screenshot of the MUSIC PLAYER after the user requested the list of available songs by 'Maroon 5'.

Figure 24-4: Screenshot of the dialog system response after the user selected 'Harder to breathe' by 'Maroon 5'. The MUSIC PLAYER in parallel starts playing this song.

A simple playlist management is implemented in the MUSIC PLAYER, however the dialog system presently does not access it.

24.3. Application Image Browser

The IMAGE BROWSER application serves as an interface towards the increasing amount of digitally available pictures, especially due to the boom of digital cameras. The user can choose between different views on the picture album stored by the application, including an overview, a 'fullscreen' view, or a view on a picture with annotations, figures 24-5 and 24-6. In all these modes, navigating

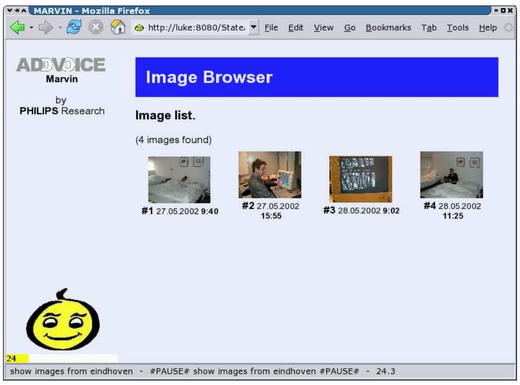

Figure 24-5: Overview of a selected set of pictures in the IMAGE BROWSER (screenshot).

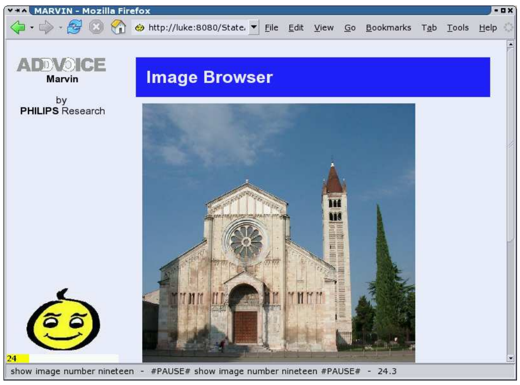

Figure 24-6: Screenshot of a fullscreen view of an image.

between the pictures and changing the application operation parameters defining the selection parameters for the collection is supported.

For the more advanced navigation in the image collection, the application assumes that the pictures are annotated. Depending on the motive shown by the picture, the annotation may comprise the location, date and time, names of persons shown on the picture, and a free description.

24.4. *Application* Electronic Programming Guide

The Spice demonstrator motivates the EPG. It allows browsing the list of broadcasts on TV. Criteria while scanning the program guide for interesting transmissions are the day of week, possibly relative to 'today', the channel, the genre, or the title. The EPG mainly provides two

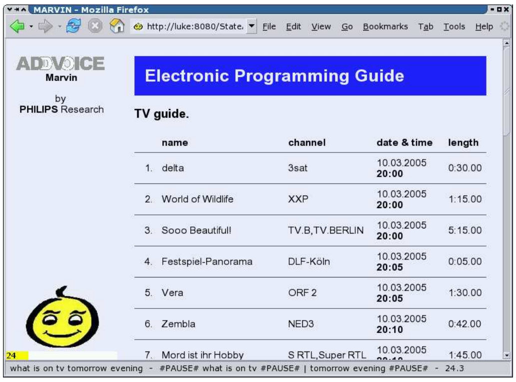

Figure 24-7: The TV program for 'tomorrow evening' provided by the EPG application (screenshot).

views: an overview of broadcasts in form of a table (figure 24-7) and a detailed view for a single transmission.

The program guide application especially cooperates with the HD recorder introduced next.

24.5. *Application* Harddisk Recorder

As substitute for the class of video recording devices – which especially includes the still commonly used VCR and nowadays state of the art DVD-recording devices – we selected a HD recording device. Harddisk recording devices presently become increasingly widespread in households, often combined with a DVD-recorder or integrated into a satellite tuner.
The recording application enabled for the control through the *Marvin* dialog system serves as an interface to access a set of recordings, figure 24-9, and allows the user to deal with timer entries to schedule the recording of broadcasts from TV, figure 24-8. I.e. the list of recordings and timer entries can be browsed and manipulated. Comparable to the songs in the music player, the recorded transmissions can be selected and played back. The timer entries defined by the user however do not actually trigger a recording – the system hosting the *Marvin* dialog system presently has no tuner device.

24.6. *Application* Activation and Identification *(aka* Marvin*)*

The *Marvin* dialog system utilizes two ASRs. The general ASR engine is responsible for processing the 'normal' spoken language input by the user. It recognizes the speech and transforms it to a transcription. The second recognizer is specialized on two tasks: robustly identifying a small fixed set of phrases from the speech of the users to activate the dialog system when it is deactivated and, in addition, identifying the user by means of the user's voice.

Note that the activation and identification is not completely decoupled from the dialog system – in contrast to the other applications introduced so far. To activate the system, it

Figure 24-8: Screenshot while defining a timer entry in the HD RECORDER application. The length still needs to be specified.

Figure 24-9: List of recordings (with two entries) stored by the HD RECORDER (screenshot).

interacts with the COMMUNICATION MANAGEMENT if an activation sequence is detected. Note further that the specialized ASR is a module running as part of the dialog system.
The identification of the user, which is an additional outcome by the specialized recognizer, may be used e.g. to personalize settings inside the dialog system, adapt applications towards the user's preferences, or access user specific content. Thus, it serves as a service module towards dialog system and applications.

Due to the interaction sequences used to activate the system (e.g. *"hello Marvin"*) and the capability of identifying the user, the ACTIVATION AND IDENTIFICATION application can be seen as the soul of the dialog system. Thus, this application is also informally referred to as MARVIN.

24.7. Application System

For completeness, we also list a meta application named SYSTEM. It is used to summarize the internal functionality inside the dialog system that can be addressed by the user during the interaction. Especially, it includes navigating in the discourse, referencing items in a list, and the control of the visual output.
The SYSTEM application does not subsume the processing methods used inside the dialog system, since this functionality builds up the dialog system. E.g., these methods include the integration of newly derived interpretations of the user input with the previous discourse, sharing of discourse information between applications, and the handling by the "`domain model`".

24.8. The Applications as Showcase of the Criteria for the Application-Independent Dialog Processing

The criteria for flexible dialog processing in dynamic application setups formulated in 4 are summarized in table 24-1 for the applications listed before (if relevant).
An '✓' indicates that an application presently makes use of a feature formulated in a certain criterion, thus being a part of DYMALOG. E.g. the description of the channel specified for the TV-SET application in its operation parameters is derived from a more general description given at another place. The operation parameters also use the same general description for the applications HD-RECORDER and EPG to define the channel. Thus, these applications automatically

FEATURE	*Marvin* ENABLED APPLICATION						
	System	Activation & Identification	TV	Music Player	Image Browser	HD-Recorder	EPG
Application-Independent Formulation of the Dialog Core (Section 4.1)							
Integration with **discourse history** *(Criterion 4.1(1))*	✓	✓[1]	✓[1]	✓	✓	✓	✓
Structured representation of application operation parameters *(Crit. 4.1(2))*	✓	✓[2]	✓[2]	✓[2]	✓[2]	✓[2]	✓[2]
Separation of logic: dialog and app's *(Crit. 4.1(3))*	–	✓[3]	✓	✓	✓	✓	✓
Automatic **sharing** of discourse info. between app's *(Crit. 4.1(4))*	✓	–	✓	–	–	✓	✓
App. **(dis-)connect** during **run time** *(Crit. 4.1(5))*	–	✓	✓	✓	✓	✓	✓
Interface supporting **decoupling** *(Crit. 4.1(6))*	–	✓	✓	✓	✓	✓	✓
Generic Dialog Functionality Supporting Human-Computer Interaction (Section 4.2)							
Resolution of **ellipsis** *(Crit. 4.2.1(1))*	✓	✓[1]	✓[1]	✓	✓	✓	✓
Resolution of **anaphoric references** *(Crit. 4.2.1(2))*	✓[4]	–	–	✓	✓	✓	✓
Analysis of **lists** and access to **list items** *(Crit. 4.2.1(3))*	✓	–	–	✓	✓	✓	✓
Serial **multi-tasking** with **task resumption** *(Crit. 4.2.1(4))*	–	✓	✓	✓	✓	✓	✓
Navigation in the discourse *(Crit. 4.2.2(1))*	✓	✓	✓	✓	✓	✓	✓
Control of viewport in the visual modality *(Crit. 4.2.2(2))*	✓	–	–	✓	✓	✓	✓
Dynamic (de-)activation of app's/dynamic update of knowledge sources *(Crit. 4.2.2(3))*	✓	–	–	✓	✓	✓	✓
Building blocks to support design of app's *(Crit. 4.2.3)*	✓[4]	✓	✓	✓	✓	✓	✓
Parallel Multi-Hypotheses Processing (Criterion 4.3)							
Multi-hypotheses safety	✓	–	–	–	–	–	–
Parallel hypotheses processing	✓[4]	–	–	–	–	–	–
Application Dependent Support of the Separation							
Access to **content by naming**	✓	–	–	✓	✓	✓	✓
Default values during processing by applications	–	–	–	–	–	✓	✓
(Semi-generic) **explicit relaxation** *(see also Crit. 4.1(2) and 4.1(4))*	✓[4]	–	–	–	✓	✓	✓
Miscellaneous							
Variable **number of semantic entities** in input	✓	✓	✓	✓	✓	✓	✓

Table 24-1: Features of DYMALOG presently utilized by the several applications, marked with '✓'; '✓^{n}' denotes features indirectly relevant for an application. A more detailed discussion can be found in section 24.8. *Remark:* The table presents features actually being deployed in the *Marvin* dialog system, and *not* the possibilities being available for the different applications (thus not requiring additional efforts). E.g., if an e-mail application would be available, the image browser could automatically share content with the e-mail application: when a user views a picture showing a friend of her, the input *"send an e-mail"* would canonically start the composition of a new e-mail and insert the e-mail address of the friend shown the picture.

share the channel: when switching the TV device to another channel, this channel may be used to restrict the set of broadcasts retrieved by the EPG during the next steps.
The marker '$\checkmark^n$' stands for an indirect relation between an application and the particular feature. The indirect relations are:

'$\checkmark^1$': The application makes use of a feature, but does not rely on it.
E.g. the TV-SET is a command-and-control application, each single input directed to this application usually triggers an 'atomic' operation. However, user input like *"switch to this channel"* may be integrated with a channel e.g. defined in the EPG.

'$\checkmark^2$': The application is obliged to implement this feature since it is part of the underlying framework. It is especially required for the representation of the application operation parameters, since the interface between dialog system and applications relies on this representation.

'$\checkmark^3$': The feature is only partly applicable for the application.
E.g. the logic of the ACTIVATION AND IDENTIFICATION application is formulated independent of the dialog system to a large extent, but it has a strong direct link into the dialog system since it implements the activation of the dialog system by speech.

'$\checkmark^4$': The application does not make use of this feature but provides means to the dialog system or other applications in order to realize certain functionalities.
E.g. the automatically performed integration of user input interpretation with the previous discourse provides means for relaxation by controlling the integration process. In order to make use of this feature, the applications have to create certain hooks in the knowledge sources to provide a hint that is evaluated during the automatic integration process.

A missing marker does not exclude the relevance of a certain feature for an application. It only indicates the present state of the *Marvin* dialog system underlying this text.

24.8.1 Applications and their 'Features'. The application-independent formulation of the dialog system has implications for all applications as can be concluded from the table. The sharing of discourse information depends on the formulation of operation parameters in the application related ontologies. Even though parts of these formulations are already based on more generic ontologies for most applications – like descriptions of time, place, and persons – the sharing of discourse information is useful for a subset of these applications. The IMAGE BROWSER may e.g. team with a phone application: if a picture shows a friend of you, a TELEPHONE application could use her name to call this person. However, there is presently no evident case for sharing with one of the other available applications.

The current set of applications utilizes most of the generic functionalities realized in the *Marvin* dialog system. The navigation is completely independent of the applications, but is of general use during the interaction. Since the ACTIVATION AND IDENTIFICATION and TV-SET applications do not provide lists with two or more elements as result, these applications do not make use of the list access nor require larger areas for the visual output.
All of the presently available applications make use of smaller building blocks. E.g. the TV-SET, EPG, and HD-RECORDER utilize a block that defines a ***channel***. Among other descriptions, a block to represent a ***person*** is used by ACTIVATION AND IDENTIFICATION to represent the user, by the EPG to describe an actor, and by the IMAGE BROWSER in annotations listing the persons shown on a picture.

Similar to the navigation, the dialog system inherently makes the variable number of semantic entities available that can be given in each turn, thus automatically used by all applications.

Table 24-1 also reflects that the selection of a hypothesis is performed before the applications are actually contacted: only the SYSTEM application must be multi-hypotheses safe. E.g. each hypothesis from the set of all hypotheses is separately investigated for a reference to a list item, and the references contained in a hypothesis are resolved.

Besides the generic functionalities, some applications show how the design of the applications can support the framework. Applications with a larger set of functionalities distinguish between global knowledge sources being available to the system as soon as an application is connected and specialized knowledge sources that become available as soon as an application is entered. The hierarchical structures allow the user to trigger semi-generic relaxation. The EPG and HD-RECORDER implement the relaxation. Also, addressing content inside an application by naming (e.g. *"record the movie tomorrow never dies"*) or applying default values for an operation parameter is dependent on the applications.

24.8.2 Connecting Applications to the System. After the start-up of the dialog system, applications may connect in order to be controlled by the system. As a default, the SYSTEM application is connected to enable the (semi-)generic functionalities. Also, the ACTIVATION & IDENTIFICATION is enabled by default since it delivers the functionality to wake up the dialog system from the sleep state. Remember that the *Marvin* dialog system switches into a sleep state when it was not accessed for a certain period of time. The system informs the user on the activation:

SYSTEM: *"I am activating 2 new applications: Marvin, System, please stand by."*
«the startscreen shown in figure 24-10 is replaced by the output shown in figure 24-11»

Figure 24-10: Screenshot of the welcome screen displayed to the user when the dialog system finished the start-up phase.

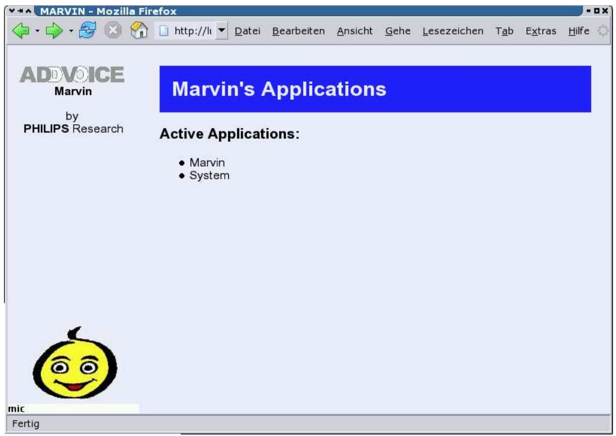

Figure 24-11: Screenshot indicating that one or more new applications were added to the system. The currently connected applications are listed.

When additional applications are connected to the system, a screen comparable to the figures 24-11 and 23-3 together with speech output (*"I am activating application* XYZ*, please stand by."*) is presented to the user. As soon as the additional application delivers its knowledge sources, the `application management` prepares the updated knowledge sources for the dialog system and distributes the knowledge sources.

Up to now, we did not look into synchronization of the interaction with the user with enabling the operation of new applications in the dialog system. I.e. the output that informs the user on new applications or the reconfiguration of components might interfere with the ongoing interaction. Actually, no interferences occur since no extensive (de-)connection of applications during run time is performed.

25. From Input to Output: A Complete Walk Through the Steps In-Between

During the next sections, fragments from interactions show the presented framework in action. We link these fragments to the concepts introduced in chapter II. The interactions with the applications are described and then analyzed to elaborate on the effects of the concepts on the interactions. The interactions, which illustrate a certain concept, are selected arbitrarily between all the applications that support the concept under consideration[5]. Even though complete interactions with the dialog system are reported, we concentrate on fragments. Note that sample interactions, being investigated due to a certain feature, mostly incorporate concepts discussed on the basis of other interactions.

The most basic application realized for operation by the *Marvin* dialog system is the TV-SET application. The following log shows the user interacting with the TV-SET:

USER#1 (U1): *"show channel WDR"*
SYSTEM#1 (S1): *"ok ... The TV is running with channel WDR."*
(the TV-SET is powered on, the channel switches to `WDR`, figure 25-1)
≪*the content frame inside the browser indicates that the user interacts with the* TV-SET *application*≫
U2: *"switch to RTL"*
S2: *"ok ... The TV is running with channel RTL."*
(the channel of the TV-SET is switched to `RTL`)
U3: *"decrease the volume"*
S3: *"emh ... Volume is now low."*
(the TV-SET indicates the change in the volume)
U4: *"set volume to mute"*
S4: *"er ... Volume is now mute."*
(the new volume level is shown by the TV-SET, see figure 25-2)
U5: *"switch off the TV"*

[5] To be even more precise, the selected interaction sequence to demonstrate the effect of a concept is chosen arbitrarily. I.e. other sequences inside the selected application or interaction sequences from other applications would also be suitable to investigate the effect of the concept.

Figure 25-1: TV-SET reaction on *"show channel WDR"*.

Figure 25-2: Screenshot of the response on *"set volume to mute"*.

S5: *"emh ... The TV is turned off."*
(the TV application triggers the TV-SET to power off)

The user decides to do something else. After a while, he may come back to watch TV again.

U6: *"turn on the TV"*
S6: *"ok ... The TV is turned on."*
(the TV application triggers the TV-SET to power on again)
U7: *"switch to channel arte"*
S7: *"aha ... The TV is running with channel arte."*
(the TV-SET switches to `arte`)
U8: *"set volume to medium"*
S8: *"ok ... Volume is now medium."*
(the change in volume is indicated on the TV-SET)

The main visual output modality for this interaction is the TV-SET screen. The device reflects changes of the application state. The interaction screen showing the visual output of the dialog system – not being directly controlled by the application – carries on presenting the status of the system through facial expressions. However, the content frame provides minimal output, showing that the user at present operates the TV.

25.1. Basic Steps During the Processing

For the first user input, U1, a more detailed view on the processing is given next. The command–and–control nature of a basic operation together with the absent of a discourse for the user input enables the pure view on the single steps.

The result of recognition and analysis is stored in a lattice[6]. For the input given in input U1, the different paths contained in the analysis-lattice are made of semantic entities for `CTuner` and `CChannelName`. For the `CChannelName`, the value `WDR` is extracted from the user input[7]. Two variants of the `CTuner` can be observed, either with or without a SWITCH task associated. If the EPG or HD RECORDER application were activated, the `CChannelName` could have been

[6]Examples of a word-lattice and an analysis-lattice are given in section A1.

[7]The output of both, `recognition` and `analysis`, are already pruned to limit the number of generated hypotheses. Therefore, only a restricted set of top candidates is contained in the lattices.

connected to these applications. Yet these applications are not considered during the dialog presented here, therefore only the connection to the domain TV of the TV-SET was drawn.

Inside the transformation, first a set of hypotheses is computed from the analysis-lattice. The hypotheses contain DOs derived from the semantic entities related to CTuner and CChannelName. Then, with support from the domain model, relations for CTuner and CChannelName are computed. At this stage, no trees need to be integrated into each other. The proposals for relations from the domain model can basically be adopted. For these two objects, a single hypothesis may consist either of a single object (CTuner *or* CChannelName) or include both objects (CTuner *and* CChannelName). The relation for a single object is of course trivial. In the latter case, the domain model delivers two variants for the relation:

1. the two objects lead to two trivial o^2I-Trees, each of these consists of only a single DO (TV:*{CTuner} or TV:*{CChannelName} respectively), and
2. a single o^2I-Tree made out of single path[8]

TV:*{CTuner}/TV:*channel*{CChannel}/TV:*channelName*{CChannelName}.

Keeping in mind that TV:*{CTuner} exists in two variants, either associated with a task object or not, different variants of the second case need to be considered. When a task object is associated,

TV:*{CTuner}/TASK="SWITCH"

is part of the o^2I-Tree. Otherwise, the resulting o^2I-Tree is restricted to the path given before.

Since no discourse is available and no generic functionality is addressed by one of the hypotheses derived from the user input, in principle the dialog knowledge processing just routes the hypothesis structures made out of the o^2I-Trees obtained in the transformation step forward to the hypothesis selection.

As already described in section 11, the hypothesis selection decides on the 'best' hypothesis according to a measure applied on each hypothesis separately. The selection is actually performed after the hypotheses are collected for a turn. The hypotheses are processed in parallel and in no predetermined order. Presently, only one hypothesis is selected, no matter if another hypothesis with the same or a close by rating exists. In case of more than one top-ranked hypotheses, the 'first' of these is selected.
Among other things, the parameters for the selection process are trained such that hypothesis structures with a single o^2I-Tree are preferred. In addition, the optimal number of tasks is considered to be one, but also influenced by their age, basically the younger the better. The hypothesis that best complies to the trained criteria is given in figure 25-3: a single o^2I-Tree with a single task object.

This 'standard' hypothesis does not require special actions from the communication management. First, an acknowledgement to indicate to the user that her input is processed is chosen randomly and immediately forwarded to the output generation, e.g. *"ok"* (see audio output S1 by the dialog system) or *"emh"*. Next, the application management transfers the selected hypothesis to the TV-SET application.

Internal rules of the TV-SET interpret the hypothesis. The TV device was in off state. The application logic triggers operations to switch on the device. If the TV is not running with channel WDR, also switching to this channel is initiated due to the hypothesis.

[8] A notation comparable to XPath is used to refer to elements inside the trees. From left to right the tree is addressed from the root towards the leaves: .../*levelN*/*levelN*+1/*levelN*+2/.... If an object, e.g. *actorList*, has two equal children, *actor*, these are accessed through *actorList*/*actor*[1] and *actorList*/*actor*[2].

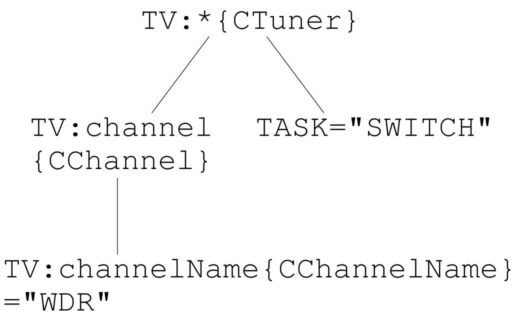

Figure 25-3: Selected hypothesis for user input U1 (section 25). Remember that only the core components and values of a DO are shown, further details are omitted (see also figure 8-1, page 61).

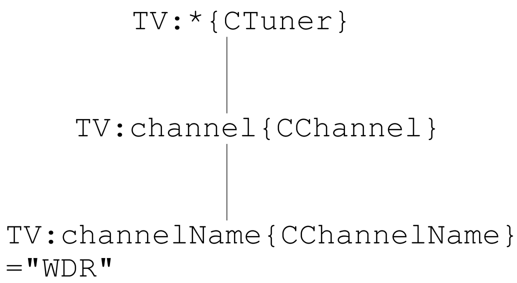

Figure 25-4: The result structure for user input U1 (section 25) generated by the TV-SET application as reaction on the hypothesis shown in figure 25-3.

Remember that the response of the application is divided into two parts.
The hypothesis structure, possibly modified, is the description of the operation designed around the application operation parameters, e.g. it may carry annotations to indicate which DOs need further specification to be able to perform an operation. It also includes the task(s), if available, being used to connect the operation parameters with potential actions to be taken by and application.
For input U1, the result structure basically mirrors the hypothesis structure, see figure 25-4. Since the result structure represents the outcome of the operation by an application, the result structure limits itself, in contrast to the hypothesis structure, on the core components of a DO, the relations of the DOs, and the value of a leaf DO if it exists. Also, the task is not part of the result structure. Therefore the only o^2I-Tree in the result structure is given by the path `TV:*{CTuner}/TV:`*`channel`*`{CChannel}/TV:`*`channelName`*`{CChannelName}="WDR"`. Figure 25-4 includes the information that the tuner is addressed, and the associated channel name is now `WDR`. But only based on the result structure one cannot decide if e.g. information on the presently selected channel was requested or – as in this case – the channel was switched.
The application returns the two structures to the `application management`. In the considered case, no actions by the `application management` are required (like the creation of new knowledge sources when a new application is connected, or when a 'larger' application is activated). The structures are delivered back to the `communication management`.

The `communication management` in its present realization just performs basic operations on the hypothesis structure and result structure. It includes a simple set of rules to extract a very restricted set of annotations for the structures. These can be considered as a pre-stage of 'speech acts' and may be honored by the output creation, but this is presently optional.
In addition, opening the microphone is triggered if suitable. For the application response on U1, i.e. the microphone will be opened right after the system's response. We will not elaborate on the activation in the remainder.
The output of the `communication management` with respect to U1 are the known hypothesis structure and result structure, enriched with the property 'communication act' is 'statement'. The 'communication act' infrastructure may serve as starting point for a speech act based management in future. Also, the `dialog knowledge processing` picks up the output to integrate the hypothesis structure into the `discourse memory`. The result structure is analyzed to detect

list structures in the application result, these are made available for reference.

The output `generation` is based on a set of rules formulated as XSLTs. In conjunction, the hypothesis structure and the result structure are transformed into a target representation suitable for rendering, taking into account also the 'communication act' property. The present target modalities of the dialog system are spoken language and (static) visual output.
The applicable XSLTs extract the task(s) from the hypothesis structure and determine the given communication act. Then the result structure is processed, i.e. the content in the result structure is transformed into a representation that can be rendered by the modality specific engines. Thus, the transformation depends on the content of the result structure, the task(s), and communication act.
For the audio output, a sequence of words is generated, possibly enriched by annotations to influence the speech synthesis. The text generated as response on U1 is given by S1, *"The TV is running with channel WDR."*. The visual output of the TV-SET application as response on U1 is an HTML page presented in the content frame only showing that the user presently interacts with the TV-SET, see figure 25-5. The user input towards the TV-SET generally leads to changes

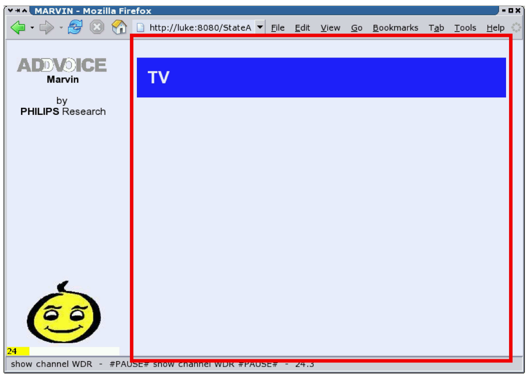

Figure 25-5: Visual output of the TV-SET application as reaction on switching the channel. The frame in red color on the right side contains the content area showing application related information.

of the TV device, e.g. the channel changes.
The generated output is finally presented to the user, rendered by either a speech synthesizer or a web browser.

We do not go into the details on how the active component monitors the state of the system. Basically, the component responsible for recording the audio channel regularly informs on the input level. The browser displays this. Furthermore, the active component listens on the communication between the single components. When certain conditions apply, a change of the facial expression of the comic style character represents this. E.g. it is detected that audio signals are sent to the ASR (show facial expression related to 'recognizing'), input is delivered to the NLU ('analyzing'), or a processing result is transferred to the output creation by the `communication management` ('interacting').
In addition, the status line of the browser is manipulated. The status line shows the best recognition result and the top scores outcome of the analysis as soon as they become available.

25.2. Integration Into the Previous Discourse – The Basic Case

After the system response S1, the `discourse memory` integrated the hypothesis structure given in figure 25-3. For input U2, the processing steps taken by the system are analog to the processing of U1. Of course, since the TV device is already turned on, the application does not need to turn it on before switching.

The hypotheses generated at the various stages also comply with the preceding turn. However, the `dialog knowledge processing` now starts from a non-empty discourse history in the `discourse memory`. In addition to the hypotheses directly derived from the user input, hypotheses including content from the discourse history may be generated.
The figures 25-6 and 25-7 show two hypothesis structures for the user input U2 received by the `dialog knowledge processing`. We will concentrate on the hypothesis represented by figure 25-6 and integrate it with the content of the `discourse memory` that originates from the hypothesis structure shown in figure 25-3.

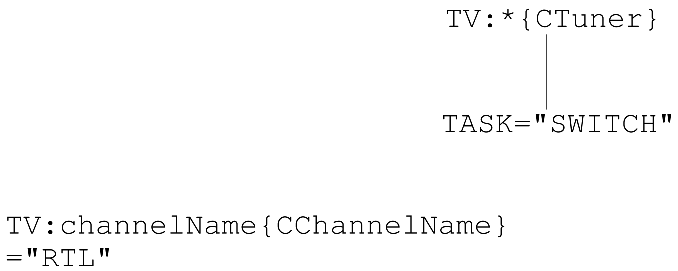

Figure 25-6: Hypothesis structure received by the knowledge processing for user input U2 (section 25).

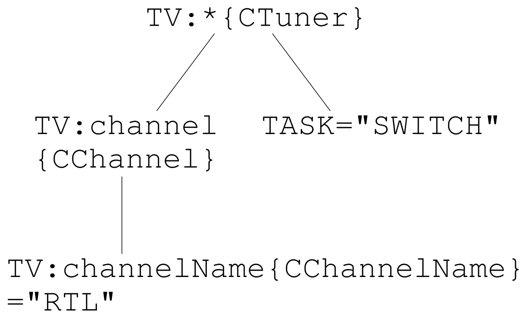

Figure 25-7: Alternative hypothesis structure with respect to figure 25-6 for user input U2.

The `domain model` receives the request to relate the (abstract) root-DOs of the involved trees: `TV:*{CTuner}`$_{U1}$, `TV:*{CTuner}`$_{U2}$, and `TV:*{CChannelName}`$_{U2}$[9].
The hypothesis structures from the `domain model` that relate the root-DOs carry a pointer to the original DO from the input. Figure 25-8 shows a single o^2I-Tree from the `domain model` to relate the given root-DOs. The resulting structure contains references to the source DOs, the figure also shows these. Inside the `dialog knowledge processing`, the references serve as

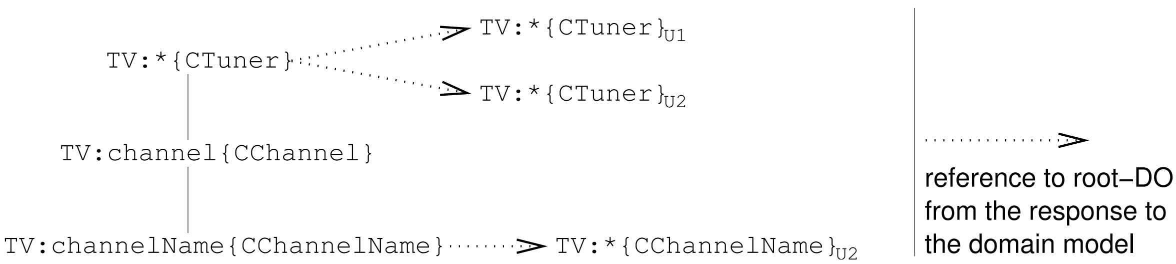

Figure 25-8: Relation structure by the `domain model` for the DOs `TV:*{CTuner}`$_{U1}$, `TV:*{CTuner}`$_{U2}$, and `TV:*{CChannelName}`$_{U2}$. The derived structure includes references to the input DOs as indicated in the figure.

[9]The marker $\cdot_{Un}$ is used to indicate to which turn the DOs belong.

anchor for the integration of the trees from the previous discourse (selected hypothesis for U1) and the user input U2. Since the entities from U2 are younger than the entities from U1, these are integrated first. An alternative 'relation' consisting of separate single DOs is presented in figure 25-9.

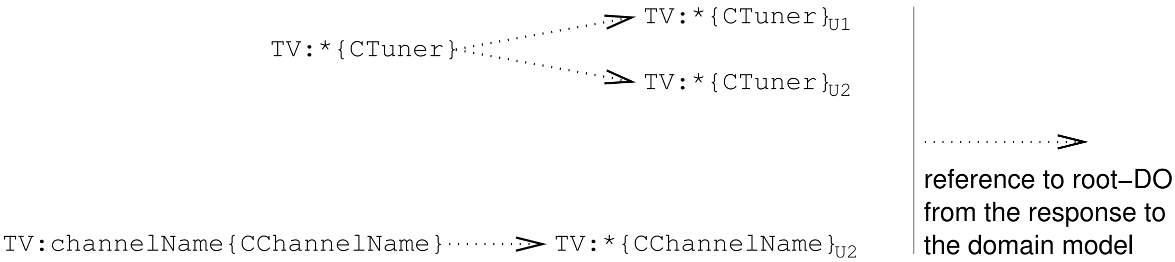

Figure 25-9: Alternative relation structure of TV:$*${CTuner}$_{\mathrm{U1}}$, TV:$*${CTuner}$_{\mathrm{U2}}$, and TV:$*${CChannelName}$_{\mathrm{U2}}$ compared to figure 25-8.

For the interaction reported in this section, the selection parameters were trained such that the hypothesis in figure 25-10 is selected. This hypothesis structure made out of a single

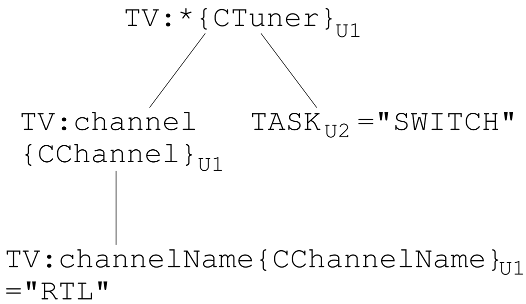

Figure 25-10: Actually selected hypothesis structure for input U2, adopting the hypothesis of U1 to input U2.

o^2I-Tree equals the hypothesis structure for U1. Modifications are an update of the value of the leaf TV:*channelName*{CChannelName}$_{\mathrm{U1}}$ from WDR to RTL and the (equal) TASK$_{\mathrm{U2}}$="SWITCH" overwrites the existing task object. For the discourse memory, this results in two updates of DOs and one addition of a task object. The task object relation to the already stored TV:$*${CTuner} is updated towards the newly added task. The second update concerns the TV:*channelName*{CChannelName} object. Its value is changed from WDR to RTL. The discourse memory remembers the changes such that the previous state can be restored on demand.

Two selected variants out of a larger set of alternative hypotheses are shown in the figures 25-11 and 25-12. Since the hypotheses belonging to a certain user input have a priori equal rights, a reparameterization of the hypothesis selection may lead to the selection of one of these interpretations.

Figure 25-11 starts from the relation of root-DOs given in figure 25-9. It behaves similar to the case discussed before. However, instead of updating the *channelName*$_{\mathrm{U1}}$ from the previous input, which is already stored in the discourse memory, an additional TV:*channelName*{CChannelName} is introduced for U2. Note that the hypothesis structure comprises two o^2I-Trees. One is a modification of the only o^2I-Tree of the selected hypothesis structure for input U1, the second is a trivial tree with a single DO.

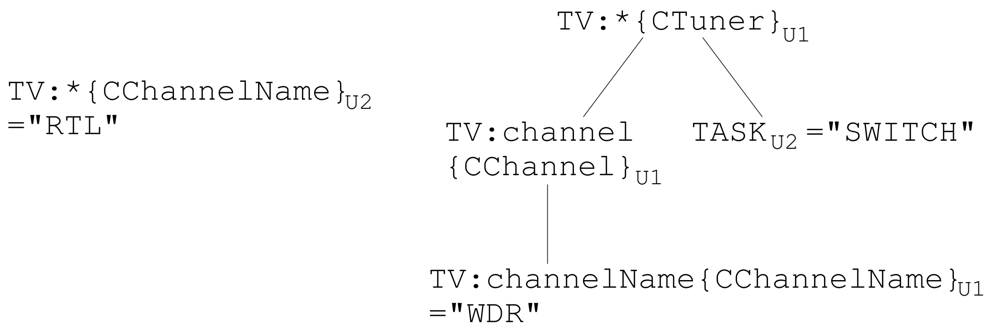

Figure 25-11: o^2I-Trees from a possible hypothesis structure for user input U2 (section 25) integrated with the hypothesis structure selected for user input U1.

The hypothesis structure in figure 25-12 is valid if the relation between TV:*{CTuner} and TV:*channel*{CChannel} is non-unique. U2 updates the Task$_{U2}$="switch" object, which origi-

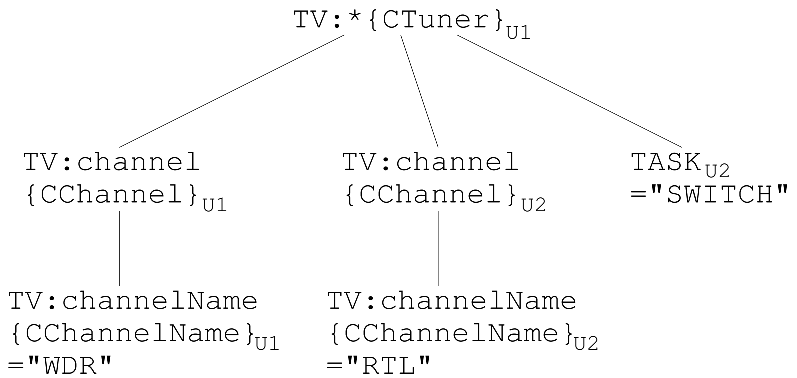

Figure 25-12: Integration of hypothesis structures for user input U1 and U2 (section 25). The parent-child relation between TV:*{CTuner} and TV:*channel*{CChannel} must be non-unique to be valid for this tree.

nates from U1. TV:*{CTuner}$_{U2}$ is merged with TV:*{CTuner}$_{U1}$, the effect is an updated TV:*{CTuner}$_{U1}$ which inherits the TV:*channel*{CChannel}$_{U2}$ child.

25.3. Influence of Tasks on the Operations

To show the effect of the task object consider the user input U7: *"switch to channel arte"* and U8: *"set volume to medium"*.

A hypothesis structure integrating input U7 with the previous discourse is shown in figure 25-13. The only task object in this figure, age 1, is associated with TV:*tuner*{CTuner}. The TV-set application honors the association. It deduces the wish to switch the channel to arte from the hypothesis structure and executes this operation. The additional objects TV:*power*{CPower} and TV:*volume*{CVolume} need not be taken into account to switch the channel of the TV.

A noteworthy property of the structure in figure 25-13 are the ages of the objects and their values respectively. In the given structure, a correlation between the location of the task object in the tree and the age of objects can be observed: the objects modified or introduced last are part of

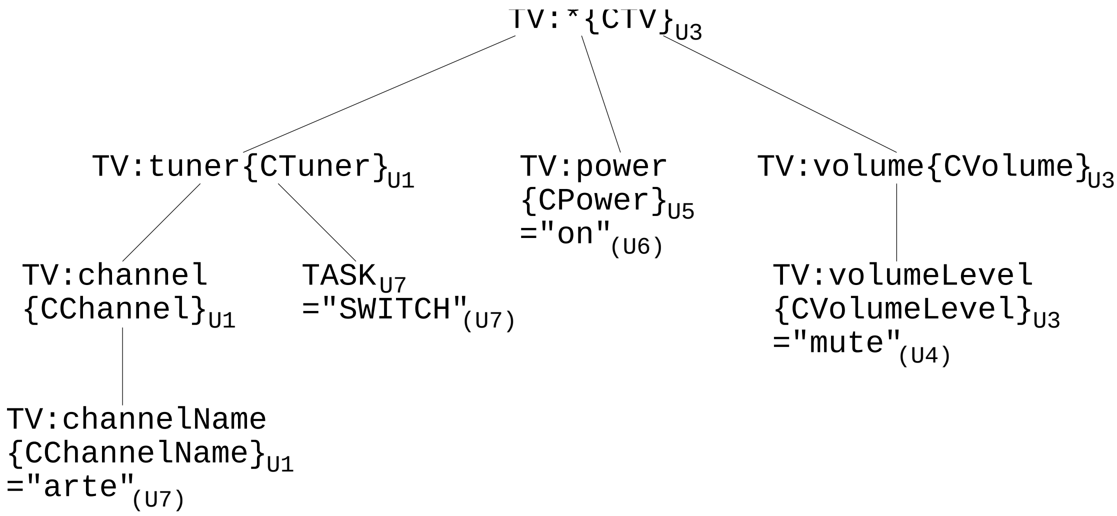

Figure 25-13: Possible hypothesis structure for user input U7: *"switch to channel arte"* (section 25). The indices at the DOs indicate the user input when a DO was introduced during the discourse. These objects may have been updated after their introduction. The bracketed indices at the values of the leaves show when the value given in the figure was introduced. The position of the task object determines the operation performed by the application as outlined in the text.

the subtree that also hosts the task object. This information can be used by the applications for the decision on the operation to perform, especially when multiple task objects are included in the hypothesis structure.

In figure 25-14, we show the hypothesis structure for user input U8 that basically resembles the hypothesis structure for input U7 given in figure 25-13. For the given hypothesis structure, the knowledge introduced by the current user input merges with the DOs that model the volume of the TV device. The task object is detached from the `TV:`*`tuner`*`{CTuner}`, the task object introduced with the present input U8 associates with `TV:`*`volume`*`{CVolume}`. For the TV-SET application, this association triggers the modification of the volume of the TV device from previously `mute` towards `medium`.
Again, the location of the task object conforms to the location of the most recently updated DOs.

A common strategy implemented in the present *Marvin* dialog system enabled applications is to identify the DOs with an attached task object. If more than one of these are found, the decision which objects are processed, and in which order, is based on the age of the task and the age of its associated DO. The present applications need a task object to perform an operation. Otherwise, the applications take a note of the DOs without actually changing the state of the application, like switching the channel or executing a query for titles in music database for a MUSIC PLAYER application.

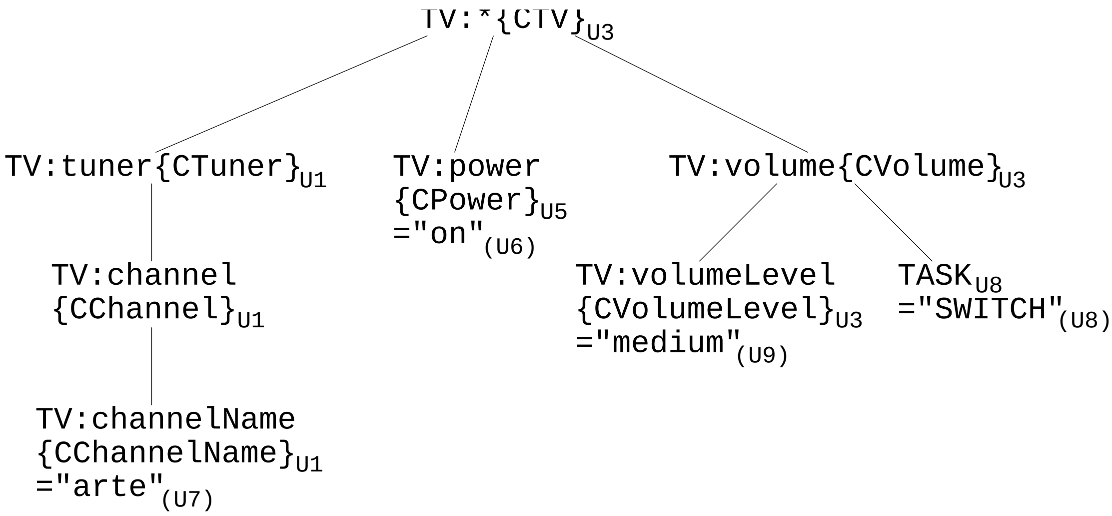

Figure 25-14: Possible hypothesis structure for user input U8: *"set volume to medium"* (section 25). Picking up the hypothesis structure from figure 25-13, DOs related to volume control are updated and the task object moved from `TV:`***`tuner`***`{CTuner}` to `TV:`***`volume`***`{CVolume}`. The structure as such remained constant.

26. Generic Functionalities and Ease of Application Design

The IMAGE BROWSER application serves as carrier to analyze concepts in DYMALOG around the notion of 'generic' functionality. As already discussed in chapter II, we can distinguish between application-*in*dependent and functionalities related to an application.
The application-independent generic functionalities are inherent in the dialog system. They support either the interaction between the system and a user as such, or support the user interacting with an application. Typical examples of the former are manipulations of an output modality, see section 26.3, and the navigation in the discourse that is discussed in more detail in the context of advanced manipulations of the `discourse memory`, section 27.2. An example of the latter given here is the analysis of list references, section 26.2. Applying generic means to resolve such references modifies the content for use by the applications.
Application-related functionalities intend to model areas which recur among different applications or even inside a single application, but which cannot abstract from the applications themselves. Examples which support the interaction are the (de-)activation of the 'specialized' content of an application, section 26.1, and the explicit relaxation 26.4. Execution of the functionality is split up between the dialog system and the applications. Another example are building blocks to ease the design of an application. E.g. a block may include the means for recognition, analysis, representation, and output creation related to date and time expressions.

Consider the following interaction of a user with the IMAGE BROWSER application, also the ACTIVATION & IDENTIFICATION application is involved.

(the system is started up, but since the user did not interact with the system it is in sleep mode)

USER#1 (U1):
"hello Marvin"

System#1 (S1):
"er ... Hello Jürgen."
(the system wakes up)
≪*the browser shows 'Jürgen, welcome.'*≫
U2: *"show images from Taiwan"*
S2: *"ok ... I did not find anything to do."*
(the system recognized '[show] channel [number 1]' since the specialized sections of the image browser knowledge sources are not (yet) active, for the elements in squared brackets no semantic meaning is derived)

The user addresses functionality of the IMAGE BROWSER, which is available only if this application was explicitly entered. The user enables these 'specialized' functionalities through the next utterance U3.

U3: *"go to the image browser"*
S3: *"ok ... I updated my knowledge. ... What images do you want to see?"*
≪*the* IMAGE BROWSER *welcome screen is shown in the browser window*≫
U4: *"show images from Taiwan"*
S4: *"ok ... I found 21 images."*
≪*21 pictures that match* `country="Taiwan"` *are shown in overview mode, figure 26-1*≫
U5: *"page down"*
S5: *"ok"*
≪*the content window scrolls downwards to show the next pictures, figure 26-2*≫
(the 21 pictures matching the restriction `country="Taiwan"` do not fit completely into the content frame)
U6: *"show details on picture number eighteen"*
S6: *"ok ... Image information: in memorial – taipeh – taiwan – during 'coling' taiwan – view from the memorial."*
(the 18th picture with `country="Taiwan"` is selected)
≪*the detailed view on the selected picture given by figure 26-3 is presented*≫

The pictures form a list that is identified by the dialog system. The pictures are numbered in the browser output. The user addressed a picture he is interested in by referring to the according position in the list.

U7: *"show fullscreen"*
S7: *"ok ... Switching to fullscreen mode."*
≪*the enlarged picture is shown in the content frame in conjunction with a selected subset of the pictures' annotation below, figure 26-4*≫
U8: *"next picture"*
S8: *"aha."*
≪*analog to picture 18, the* 19th *picture is shown in fullscreen mode*≫
U9: *"next picture"*

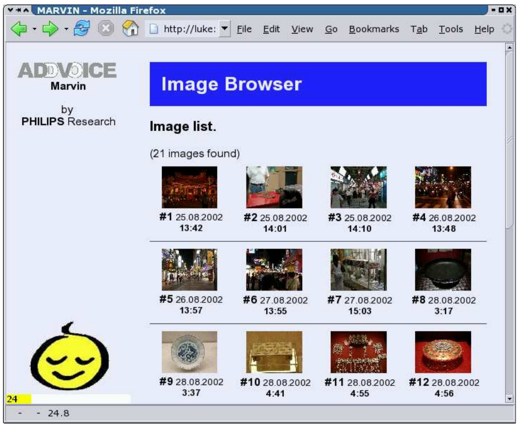

Figure 26-1: Pictures matching the restriction ***country***="Taiwan" shown by the IMAGE BROWSER in overview mode, see input U4 (section 26).

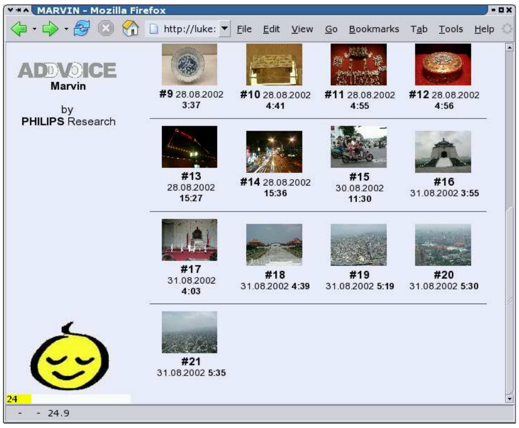

Figure 26-2: The picture overview given in figure 26-1 does not fit completely into the content frame. Scrolling down allows the user to view the remaining pictures (U5, section 26).

Figure 26-3: Detailed view on the 18th picture from the list of pictures with ***country***="Taiwan", selected by U6 (section 26). A thumbnail of the picture is shown on the left together with the pictures' annotation on the right side.

Figure 26-4: The 'fullscreen' view on the 18th picture. An enlarged picture is shown in the content frame, below a subset of the pictures' annotation, compare figure 26-3.

S9: *"ok."*

≪the browser contains picture 20 in fullscreen mode≫

U10: *"previous picture"*

S10: *"aha."*

≪again picture 19 is presented to the user in fullscreen mode≫

U11: *"show details on this picture"*

S11: *"ok ... Image information: in tower – taipeh – taiwan – during 'coling' taiwan – view to north."*

≪the detailed view on the 19th *picture is given≫*

U12: *"show overview"*

S12: *"Switching to overview mode."*

≪basically, figure 26-1 with pictures matching `country="Taiwan"` *is presented again to the user≫*

The user is finished with viewing pictures from `Taiwan` and decides to browse images taken at the 'CRE' exhibition in Eindhoven.

U13: *"show images from Eindhoven"*

S13: *"ok ... I found 4 images."*

(4 pictures matching *town*=`"Eindhoven"` are found in the picture database; since the application cannot find any pictures matching *country*=`"Taiwan"` *and* *town*=`"Eindhoven"`, it drops the older restriction in the application – this is part of the application logic)

≪the matches are shown in overview mode≫

U14: *"show images with Thomas"*

S14: *"ok ... I found 1 image."*

(only one picture fulfills *town*=`"Eindhoven"` and *firstName*=`"Thomas"`)

≪this picture is shown≫

U15: *"show details on this image"*

S15: *"er ... Image information: in CRE Natlab – Eindhoven – Netherlands – during CRE – Thomas Portele."*

(the viewing mode for the single matching picture is changed to 'detailed')

≪the selected picture is shown in 'detailed' mode≫

Finally, restrictions posed on the picture database are released and the user selects a single picture from the thumbnail view.

U16: *"show all images"*

S16: *"aha ... I found 62 images."*

≪the content frame shows the upper section of the overview page of the 62 pictures from the database≫

U17: *"page down"*

S17: *"ok."*

≪the viewport on the overview page scrolls down to show the next pictures not visible yet≫

U18: *"page down"*

S18: *"ok."*

≪*the page scrolls further down towards the bottom*≫

U19: *"show image number thirty"*

S19: *"emh."*

≪*the* 30th *picture is presented in fullscreen mode, figure 26-5; however, the present size of the browser cuts the picture in fullscreen mode*≫

U20: *"page down"*

S20: *"ok."*

≪*the lower part of the picture and parts of the associated annotation is now shown in the browser, figure 26-6*≫

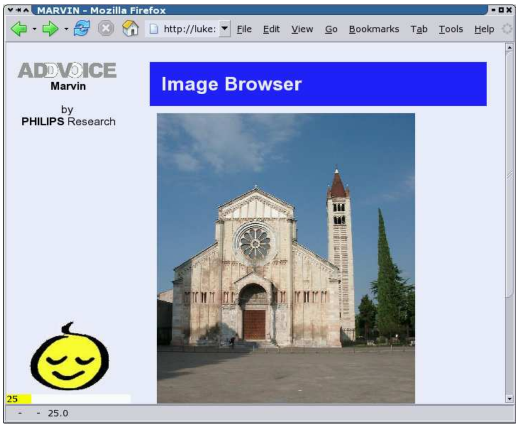

Figure 26-5: Example of a constellation of the browser dimensions and a picture where the fullscreen view does not fit the content frame.

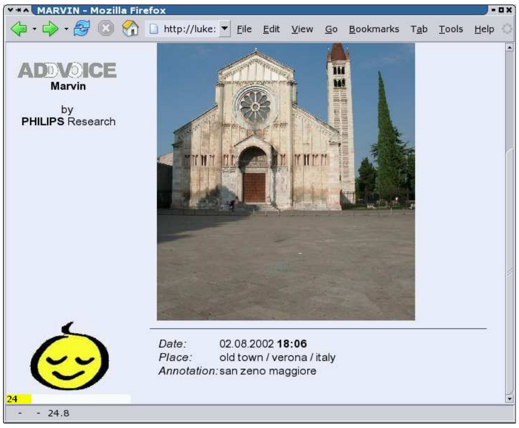

Figure 26-6: The lower part of the fullscreen view for the picture shown in figure 26-5.

U21: *"leave the image browser"*

S21: *"aha ... I updated my knowledge. ... You are leaving the image-browsing mode."*

≪*the browser shows the good bye screen of the* IMAGE BROWSER≫

After leaving the APPLICATION BROWSER, the 'specialized' functionalities are no longer accessible. The user input U22 places a request addressed towards specialized functionalities of the IMAGE BROWSER after these parts are unloaded. The recognition fails, especially no task object is extracted.

U22: *"show images from Italy"*

S22: *"er ... I did not find anything to do."*

(the selected hypothesis is build upon the recognized sequence 'shilin in guildford italy', no task is derived from the sequence)

Herewith, the excerpt from the interaction with the IMAGE BROWSER is finished. We will now go into the analysis of concepts from DymaLog that can be observed in the interaction fragment.

26.1. Separation of Applications Into Global and Specialized Parts

The application related knowledge sources may be partitioned by the application developer into *global* and *specialized* entities as introduced in section 2.2.

Global entities are available as soon as an application connected to the dialog system and distributed its knowledge sources.

The *specialized* elements become accessible when the user explicitly entered an application (*activation*) and vanish when the user explicitly exists the application (*deactivation*).

The dialog system strongly supports this segmentation, as we will see next. However, an application is free to implement a more granular partition of the knowledge sources. The applications are allowed to send updated knowledge sources whenever needed. Nevertheless, the responsibility of such a finer partition completely lies by the applications and is therefore a complete application property.
Furthermore, partitioning the knowledge sources mainly affects the NLU resources. The NLU grammar is used to derive the recognition network and lexicon, and together these knowledge sources determine the application operation parameters the user can address.

To support the partitioning on the highest level – i.e. global vs. specialized parts – a torso of knowledge sources is provided by the dialog system. The small ontology fragment to allow the (de-)activation of an application is given in figure 26-7. The objects ***applicationEnter*** and

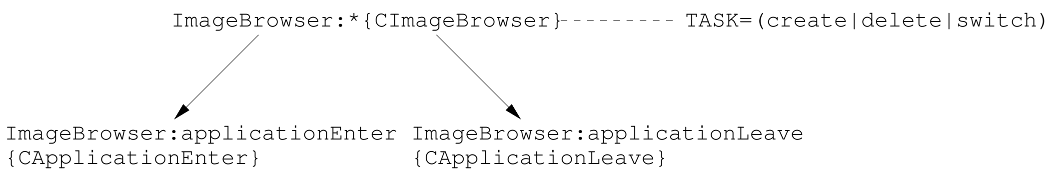

Figure 26-7: Ontology fragment for the activation and deactivation of an application.

applicationLeave are imported from the generic ontology provided by the dialog system. The application just specifies the terms that can be used by the user to name the application, in our example only the term 'image browser', and the generic grammar integrates these terms. The grammar contains phrases like:

'enter the <application name>',
'go to the <application name>',
'open the <application name>',
'leave the <application name>',
'delete the <application name>', or
'kill the <application name>'.

An application that makes use of the partition of knowledge sources as described here needs to implement the handling of structures according to the ontology fragment, i.e. publish the specialized sections of the knowledge sources upon activation and restrict to the global elements upon deactivation.

User input U3 performs the activation of the IMAGE BROWSER application. In utterance U21, the application is explicitly deactivated, thus the extend of functionalities is reduced again. In-between, the specialized functionalities of the IMAGE BROWSER are available as can be observed from the reported interaction. The reactions S2 and S22 on user input U2 and U22 respectively illustrate the failure of the attempts to access specialized functionalities that are not active, either not activated before or already deactivated again.

26.2. Accessing the n-th Picture

The access to list items is utilized in two places of the interaction: during user input U6 and U19. The procedure to resolve these references is of course analog in both cases, therefore it suffices to consider input U6.

The procedure outlined in section 16 naturally consists of three stages:

1. identification of lists in the result structure from the last user input,
2. identification of the reference in the current user input, and
3. integration of the referenced item into the hypothesis.

26.2.1 Analysis of the Result Structure from the Last User Input. Since the user input U5: *"page down"* targets at changing the viewport of the visual output modality and not on the IMAGE BROWSER itself, input U6 addresses the list that originates from user input U4: *"show images from Taiwan"*. During input processing, a DO of class `CImageList` with an associated SWITCH task is derived together with the DO ***`country`***`{CCountry}="Taiwan"`. These objects were combined into the hypothesis structure that was finally selected by the **`hypothesis selection`**, figure 26-8. The resulting result structure is sketched in figure 26-9. On the basis of

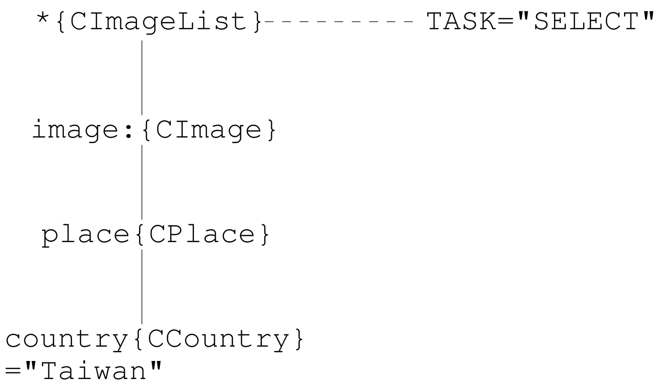

Figure 26-8: Illustration of the selected hypothesis structure for U4: *"show images from Taiwan"*.

the selected hypothesis structure, the IMAGE BROWSER queries the underlying picture database for pictures matching ***`country`***`="Taiwan"`. The output derived from the result structure is shown in figure 26-1. After user input U5, the viewport is changed and figure 26-2 is presented to the user.

The identification algorithm that analyzes result structures, section 16.1, first searches for lists on root level, i.e. if a set of single o^2I-Trees are given in the result structure. This does not apply in this case. The only o^2I-Tree in the result structure is searched for an inherent list. Since the root-DO ***`imageList`***`{CImageList}` has 21 subobjects ***`image`***`{CImage}`, the search for the inherent list succeeds already on the topmost level. The items of the list identified in this o^2I-Tree are tagged in figure 26-9 ('*item #...*'). The result structure is not further investigated for additional lists. Thus, a unique list that can be addressed was extracted.

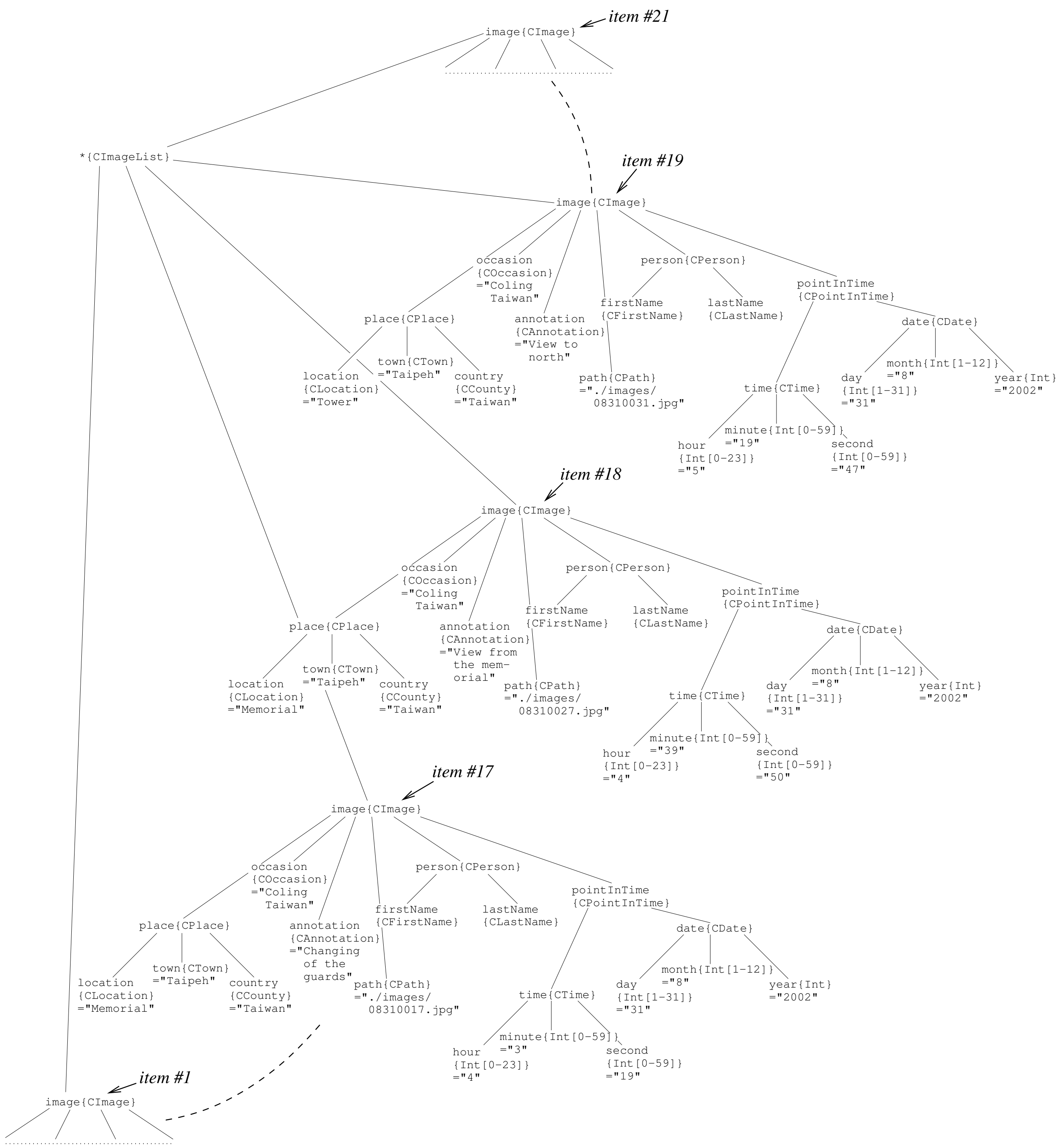

Figure 26-9: Sketch of the result structure generated by the IMAGE BROWSER for the user input U4: *"show images from Taiwan"*. The items #17 to #19 are shown in more detail. The 'item' markers already indicate the result of the list identification process. The result structure depends on the hypothesis structure that was delivered to the IMAGE BROWSER.

26.2.2 A Reference in the Present Utterance. Utterance U6 references the 18th picture. The interpretation of the user input actually underlying the selected hypothesis structure for input U6 is given in figure 26-10. From the input, already a hint on the targeted class is

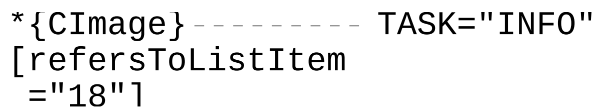

Figure 26-10: Interpretation of U6 (section 26) before resolving the list reference and integrating with the previous discourse. The hypothesis structure selected later by the `hypothesis selection` is based on this interpretation.

derived. However, also a completely generic reference is possible (*"... item number eighteen ..."*). A property attached to the extracted DO indicates the reference.

26.2.3 Porting the Referenced Item Into the Hypothesis. When the `dialog knowledge processing` receives this interpretation, it instantly tries to resolve the reference. In the given case, the list extracted previously contains 21 entries. Thus the 18th entry lies within the allowed range.

In this example, the actual integration of the referenced item into the hypothesis is straightforward. First, the referencing DO and the root-DO of the referenced item are merged. Next, the subtree below the root-DO of the referenced item is attached to the resulting object. Figure 26-11 contains the outcome of the integration for the interpretation in figure 26-10.

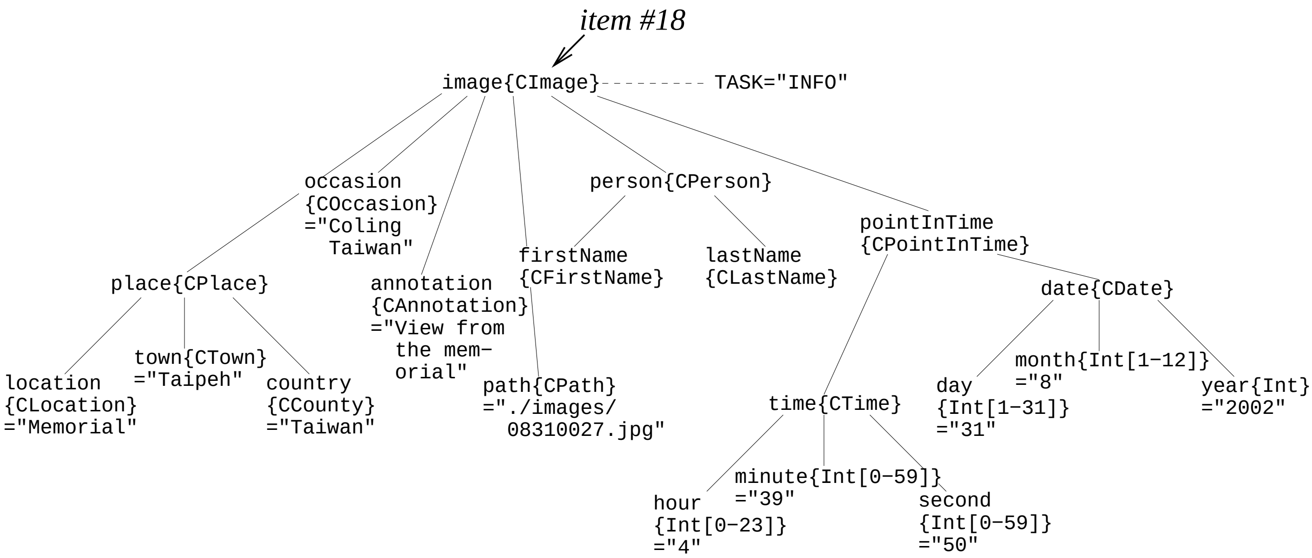

Figure 26-11: Result of the resolution of the list reference in the user input interpretation of figure 26-10 with the list contained in figure 26-9.

Note that in case of a generic list reference, the class of the referencing object is also a generic class. Therefore, the resulting DO is then redefined using the referenced object.

26.2.4 The Selected Hypothesis Structure. The user input interpretation integrating the list reference is also related to the previous discourse by the `domain model`, e.g. as shown in

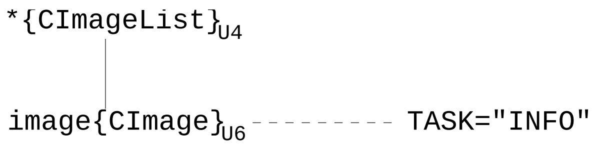

Figure 26-12: Relation of the user input interpretation for U6 and previous discourse (concentrating on the hypothesis structure selected for U4) by the `domain model` (section 26). The actually selected hypothesis structure is the integration of the input interpretation and the previous discourse into this skeleton.

figure 26-12. The integration of the tree derived directly from the user input and the previous discourse finally leads to the hypothesis structure in figure 26-13 containing a single o^2I-Tree that survived the selection process.

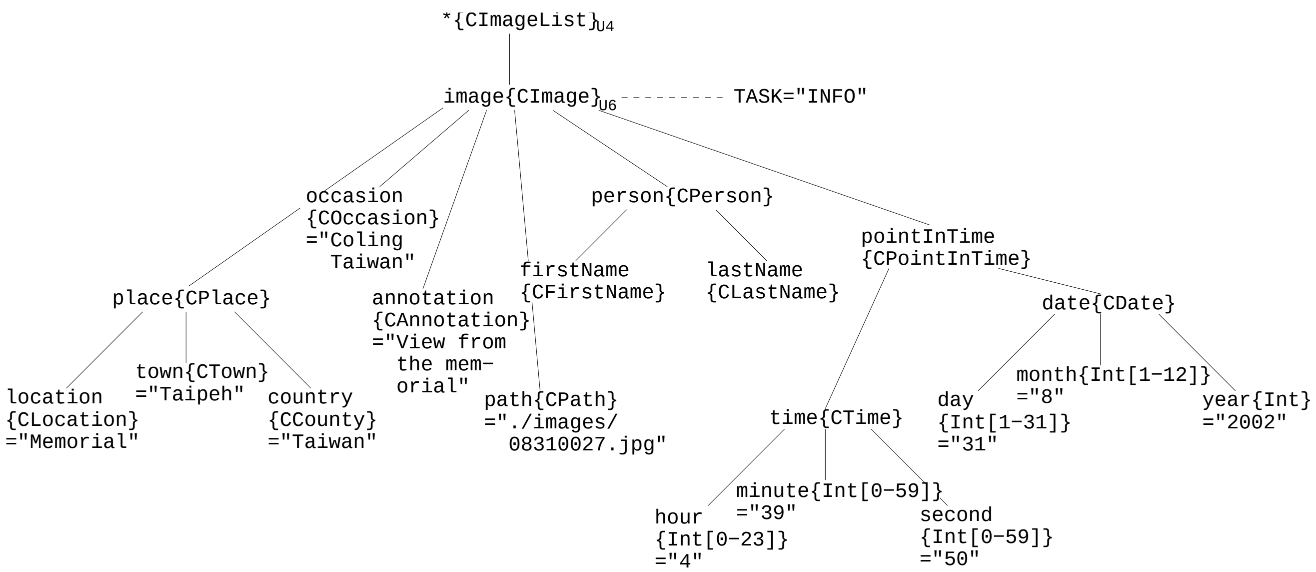

Figure 26-13: Selected hypothesis structure including the referenced list item number 18 and integrating the previous discourse for user input U6 (section 26).

Due to the INFO task object, the application interprets that the user wants the details on the associated ***image*** object. Thus the result structure basically contains the ***image*** object. That result structure– together with the task object – induces the output shown in figure 26-3.

The concepts to extract the lists from the result structure and to resolve the list references are inherent in the *Marvin* dialog system. In addition, generic means for ASR and NLU to address the list items are included, e.g. *"... the* n^{th} *item ..."*. However, the dialog system also provides anchor points to more closely couple the means to address list items and the applications. I.e. formulations like *"... number n"* or *"the* n^{th} *..."* are predefined and can be easily be incorporated by the application designer. If a class can be addressed by terms represented by '[class]', e.g. [class]$\in$ {image, picture}, then reasonable list references are *"[class] number n"* or *"the* n^{th} *[class]"*. This coupling is applied on the user input U6.

26.3. 'Next Page, Please' – Explicit Control of the Dialog System Components

User input U5: *"page down"* addresses system functionality. The processing up to the `communication management` does not differ from processing input related to 'normal' applications. Figure 26-14 shows the selected hypothesis structure for U5, a single DO that forms the only o^2I-Tree in the hypothesis.

```
System:*{CBrowserCmdName}="PageDown"
```

Figure 26-14: Selected representation of the user input U5: *"page down"*.

System functionality bypasses the normal processing inside the `communication management`. Once the `hypothesis selection` chose the best hypothesis, the affected components are contacted to execute the functionality if system functionality is addressed. In case of a 'page up/down' request, the active component running inside the browser interprets and executes the command. It results in scrolling down the viewport as shown in figure 26-2.

The `hypothesis selection` has no knowledge on the state of the single components building the dialog system. Therefore, system functionality is always handed over to the appropriate components. These components need to cope with such a request. E.g. if already the bottom most part of a page is shown, another *"page down"* delivered to the active component does not change the viewport anymore. Another option would be that the component decides to perform a wrap-around, and therefore jumps to the top.
In addition to the execution of the addressed functionality, a system reaction may be triggered, e.g. speech output to indicate that an operation was executed.

Note that the `discourse memory` does presently not integrate the selected hypothesis structure related to system functionalities.

26.4. Explicit Relaxation

The parameters for the selection of the best hypothesis of the system underlying the interactions presented here are trained such that the integration of the present user input and previous discourse is automatically performed. As a consequence, relaxation is *not* an automatic process (except an application designer explicitly formulated rules for relaxation inside a certain application, see S13).
On the other hand, parameters may be trained such that the present user input interpretation remains independent of the previous discourse. However, then the context from the previous discourse would be discarded.

The *Marvin* dialog system provides concepts to allow explicit relaxation, thus favors a setup in which the user input and previous discourse are integrated. The input U16: *"show all images"* makes use of these capabilities. The actually selected hypothesis structure related to U16 is contained in figure 26-15. The term 'all' is a central entity in some of the grammar rules. When the `NLU` processes the word-lattice from the `ASR`, one of these rules links to the `CImageList` and connects it to the special property ***`integrationCmd`*** with the value `stop`.
The hypothesis structure for the user input preceding U16 – input U15 – is analog to the structure shown in figure 26-13. Figure 26-16 shows the skeleton for the integration of U16 and U15. Since the tree for the present user input (U16) given in figure 26-15 is younger than the trees from the previous discourse, it is first integrated into the skeleton. When the tree from the

Figure 26-15: The selected hypothesis structure for U16: *"show all images"*. Explicit relaxation leads to the special property ***integrationCmd***. The integration process honors the property.

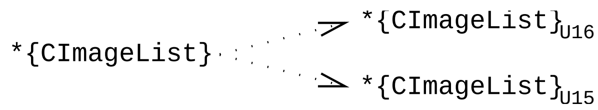

Figure 26-16: Integration skeleton derived from the `domain model` for user input U16 and U15 from the `discourse memory` (section 26). The single `*{CImageList}` that forms the skeleton is linked to the corresponding objects from U16 and U15.

previous discourse representing U15 is then integrated, the children of `*{CImageList}`$_{U15}$ are ignored due to the `*{CImageList}[`*`integrationCmd`*`]="stop"`. The final hypothesis structure that is actually selected by the `hypothesis selection` corresponds to figure 26-15.

When the IMAGE BROWSER application receives the hypothesis structure, the image database is queried for pictures without any restrictions. The result structure contains the database response, 62 images. The browser presents the images in overview mode.

26.5. Remarks on the Implementation of the Image Browser

The IMAGE BROWSER application stores the response of database queries as a list of pictures, including annotations. In addition, a certain position pointing to one of the list items can be stored. This enables

- switching between the different view modes (e.g. U6, U7, and U11) and
- stepwise moving in the list of pictures in single-picture mode (e.g. U8, U9, and U10).

Moving back from the fullscreen or detailed view to overview mode requires the application to remember the previously selected pictures, which is done by the mentioned list.

27. Operations on the Discourse Memory

An interaction with the MUSIC PLAYER application shows how the interaction can benefit from navigation capabilities. If further shows how the present input is enriched with content from the `discourse memory`, beyond overwriting existing content.

The dialog with the MUSIC PLAYER given next forms the basis of the illustration of the merger of user input with the previous discourse, section 27.1. Two applications of the navigation in the discourse are contained in the interaction. The backward navigation is used to *(i)* correct a misrecognition and *(ii)* restore a former state, section 27.2. In these cases, only one backward directed operation is performed in a row. More complex sequences of forward/backward directed operations are not shown here.

The interaction starts again from scratch. The first input awakes the ‘sleeping’ system.

(the system is started up, but since the user did not interact with the system it is in sleep mode)

USER#1 (U1):
”hello Marvin”

SYSTEM#1 (S1):
”ok . . . Hello Jürgen.”
(the system wakes up)
≪*the browser shows ‘Jürgen, welcome.’*≫

U2: *”go to the music player”*

S2: *”ok . . . I updated my knowledge. . . . What do you want to hear?”*
≪*the* MUSIC PLAYER *welcome screen is shown in the browser window*≫

The activation of the MUSIC PLAYER makes its specialized sections of the knowledge sources available for use in the dialog system. The user now starts to browse the music database.

U3: *”which artists do you have”*

S3: *”aha . . . I found these artists!”*
≪*the artists from the music database are listed in the browser, see figure 27-1*≫

U4: *”which albums by Eric Clapton”*

S4: *”emh . . . I found these albums!”*
(for the artist `Eric Clapton`, two albums are found)
≪*the matching albums are shown in figure 27-2*≫

Figure 27-1: Artists contained in the database of the MUSIC PLAYER application. The screen is shown as reaction on the user input U3: *”which artists do you have”* (section 27).

Figure 27-2: The albums by the artist `Eric Clapton` from the music database. The albums are requested in U4 (section 27).

U5: *”which songs”*
(recognized: *”which bands”*)

S5: *"aha … I found these artists!"*
≪the overview of artists as shown in figure 27-1 is again displayed in the browser≫

The misrecognition of U5 misleads the interaction in an unwanted direction. The user applies a navigation command to restore the previous state.

U6: *"go back"*
S6: *"Navigating back. …I found these albums!"*
(the overview on albums by `Eric Clapton`, figure 27-2, is redisplayed)
U7: *"which songs"*
S7: *"ok … I found these songs!"*
≪songs by `Eric Clapton` are shown in the browser frame≫

The previous discourse that is integrated with the present user input poses a restriction on the artist of a song. The music database returns songs by `Eric Clapton`.

U8: *"page down"*
S8: *"ok"*
≪the songs of `Eric Clapton` fit onto two pages in the browser, now the second page is visible as shown in figure 27-3≫
U9: *"go back"*
S9: *"Navigating back. … I found these albums!"*
≪again, the titles of albums from `Eric Clapton` are shown, see figure 27-2≫

Scrolling down the browser window is meta-communication. As such, the `discourse memory` does not store the hypothesis belonging to *"go back"*. Therefore, the navigation command annuls input U7 and 'ignores' U8.

U10: *"which songs from Time Pieces"*
S10: *"ok … I found these songs!"*
≪the screenshot in figure 27-4 lists the titles from the album `Time Pieces`≫

Figure 27-3: Second page with song titles of `Eric Clapton`. It is shown as reaction on utterance U8.

Figure 27-4: Song titles for the album `Time Pieces` requested by input U10.

U11: *"play I shot the sheriff"*

S11: *"er ... The Player is now playing."*

≪*the browser indicates that the* MUSIC PLAYER *starts playing* ***`I shot the sheriff`***≫

(`I shot the sheriff` is now playing)

After the last input, the music player starts playing the song `I shot the sheriff` on the speakers of the stereo equipment. The user could continue the interaction with the dialog system, but we stop the reproduction of the user-system dialog at this point.

27.1. Non-Trivial Integration

In section 25.2 (page 205), the integration of the present user input with the previous discourse mainly targets the update of a single node's content.
During the interaction with the MUSIC PLAYER, content from the previous discourse enriches the hypothesis structure representing the present user input. While processing user input U7, the discourse state after processing U6, shown in figure 27-5, is merged into the hypothesis structure. The representation of U7 is straightforward. Figure 27-6 contains a possible hypothesis structure

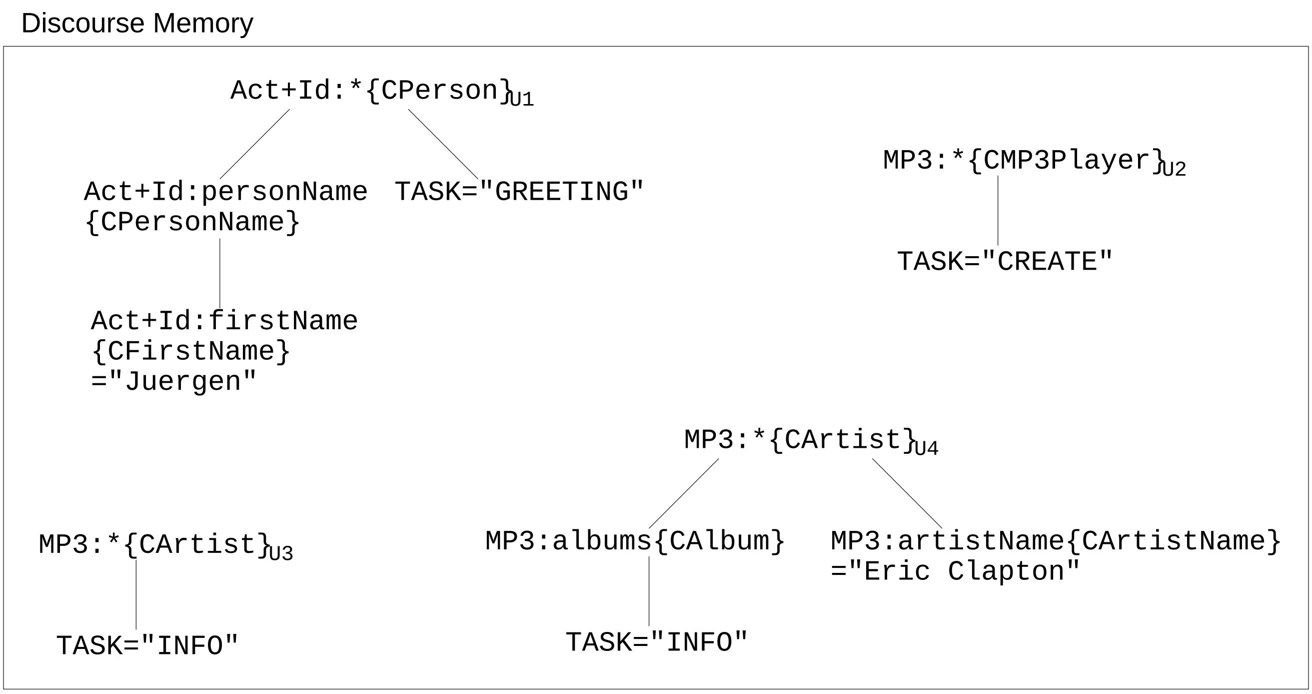

Figure 27-5: A set of o^2I-Trees that illustrate the content of the `discourse memory` after processing user input U6 (section 27). The selected hypothesis structure representing the user input U7 incorporates the structure with root-DO `MP3:*{CArtist}`$_{U4}$. The memory state corresponds to the state after processing U4.

to represent the input. This hypothesis structure is the basis of the hypothesis structure that is actually selected. The selected hypothesis combines figure 27-6 with the tree belonging to `MP3:*{CArtist}`$_{U4}$, figure 27-5. I.e. the `knowledge processing` identified this single tree as being relevant for the hypothesis in figure 27-6.

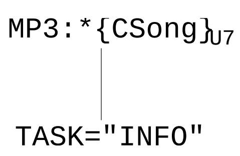

Figure 27-6: Representation of the user input U7 (section 27) before the integration with the content from the discourse memory.

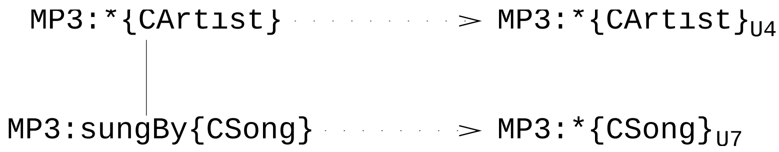

Figure 27-7: Relation of `MP3:*{CArtist}`$_{U4}$ and `MP3:*{CSong}`$_{U7}$ according to the underlying ontologies by the `domain model`.

In order to merge the two trees, the root-DOs are delivered to the `domain model`. The `domain model` computes valid relations of these DOs according to the available ontologies. One outcome of the relation process is given in figure 27-7.
The existing children of `MP3:*{CArtist}`$_{U4}$ are different from `MP3:*{CSong}`, which is also a child according to the `domain model`, see figure 27-7. Therefore, the integration of the trees into the skeleton from the `domain model` for the described setting is obvious. Figure 27-8 shows the actually selected hypothesis structure, which is built on the elements presented in this section. On the basis of this hypothesis structure, the MUSIC PLAYER decides to query the underlying

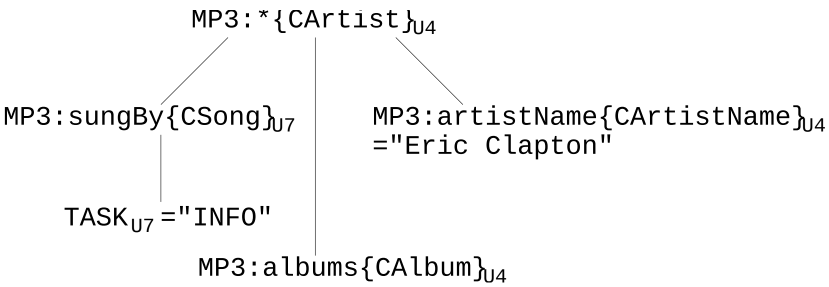

Figure 27-8: Selected hypothesis structure for the user input U7: *"which songs"* (section 27). It integrates the structure that can be directly derived from the user input (root-DO: `MP3:*{CSong}`$_{U7}$) with a structure from the `discourse memory` (root-DO: `MP3:*{CArtist}`$_{U4}$).

song database for songs from `Eric Clapton`. Finally, the web browser presents the list of songs. The user may e.g. select a song for playback.

27.2. Navigation

Navigation operations can be observed twice in the interaction with the MUSIC PLAYER. They serve two purposes. In user input U6, it is used to restore the state before misrecognition. In utterance U9, the user recalls former conditions and can then base the further interaction on that state.
As meta-communication that is directed towards the dialog system, the navigation in the discourse represents generic dialog functionality. Generic dialog functionalities are also covered in section 26.

Technically speaking, the processing required in both cases to execute the navigation is comparable. Therefore, it is sufficient to concentrate on U9. The state of the `discourse memory`

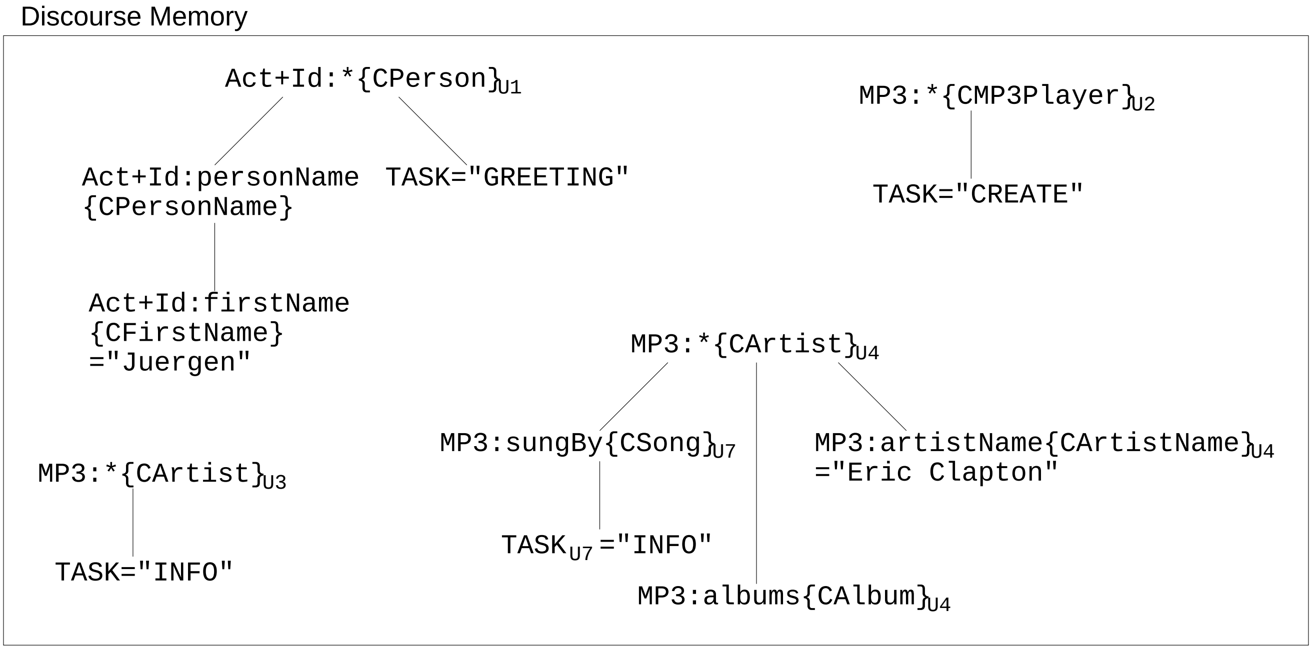

Figure 27-9: Illustration of the `discourse memory` on the basis of o^2I-Trees. The input of the dialog in section 27 up to utterance U8 is considered.

is illustrated in figure 27-9. It is similar to the state after processing user input U6 (figure 27-5), basically U7 adds to the memory.
Input U8: *"page down"* does not modify the `discourse memory`, but address the visual modality. The user inputs, which contributed to the `discourse memory`, are in order

$$\text{U1} \longrightarrow \text{U2} \longrightarrow \text{U3} \longrightarrow \text{U4} \longrightarrow [\text{U5}] \longrightarrow \text{U7}.$$

The bracketed entry relates to the misrecognized user input. User input U6: *"go back"* removes it from the `discourse memory`.

The effect of U9: *"go back"* on the memory is depicted in figure 27-10. The operation restores the `discourse memory` state after processing input U6 (or U4 respectively, figure 27-5). The additional content resulting from U7 is removed. The task object associated to `MP3:`***`albums`*** `{CAlbum}`$_{\text{U4}}$ becomes visible again.
The backward operation U9 reverted the impact of U7, and U8 did not influence the memory. Previously, U5 was reverted by U6. Thus, U4 becomes the latest 'valid' input that contributed to the `discourse memory`. The corresponding hypothesis structure is reconstructed and sent to the `communication management`. A flag indicates that a backward directed navigation operation restored the hypothesis structure. The output generated from this hypothesis structure is similar to the response on U4, figure 27-2. In addition, the user is informed that the given output results from a backward operation.

The processes being actually executed inside the `knowledge processing` and `discourse memory` are more complex than observable in the illustration of this section. The `discourse memory` needs to track the changes over time. This includes insertions into and deletions from the memory. Since the forward navigation operation can be regarded as an 'undo' of the backward directed operation, modifications of the memory due to such a backward operation must also be reversible. The *Marvin* dialog system does not restrict the number of consecutive backward or forward directed operations. Furthermore, not every user input contributes to the `discourse`

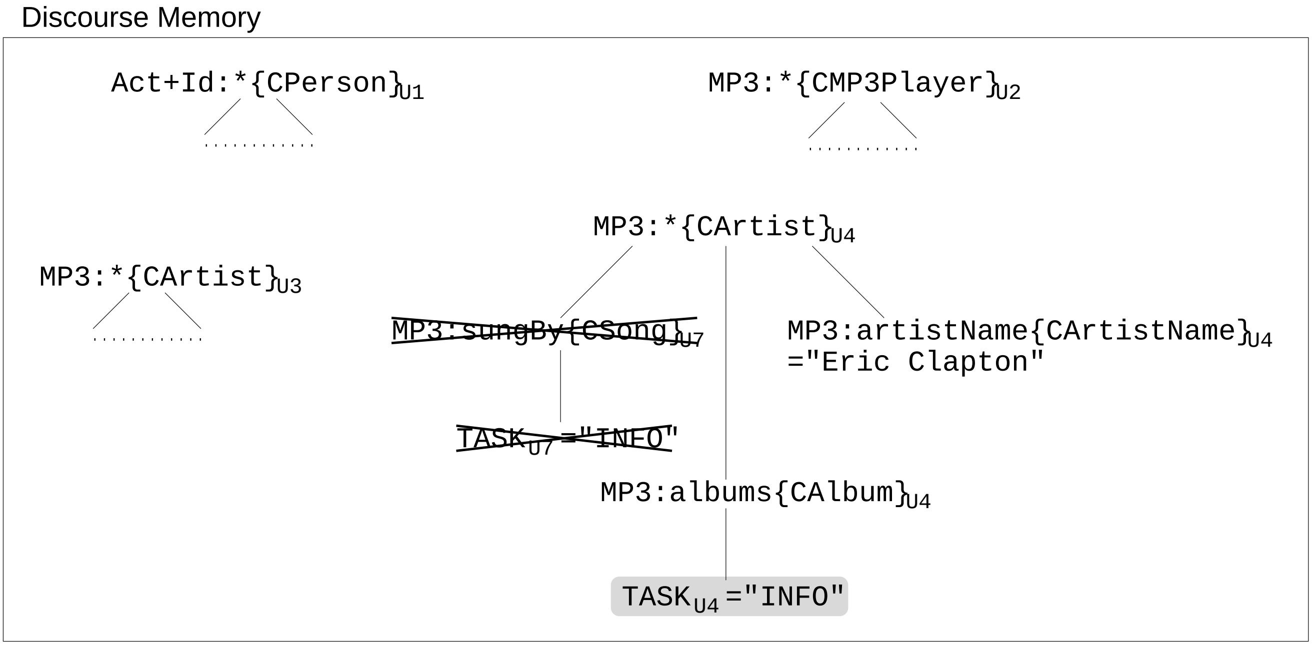

Figure 27-10: Illustration of changes in the `discourse memory` shown in figure 27-9 due to input U9. Additional content from U7 is removed from the memory, and previously vanished items become visible again. Unaffected structures are only indicated.

`memory`, e.g. hypothesis structures representing meta-communication are not integrated into the memory. More details on the internal procedures were given in section 15.

27.3. Interposed Switches between Applications

The automatic integration of the user input with the previous discourse as presented in section 27.1 even allows for interruptions of the interaction with one application by another application. This is demonstrated by means of another fragment of an interaction with the music player.

For the next depicted discourse, we assume that the user already interacted with the system. The TV-set is presently running and the user entered the music player application.

(the TV-set is running and the music player is activated)

(. . .)

U1': *"which albums by Ten Sharp"*

S1': *"emh . . . I found this album!"*

≪*one album for* `Ten Sharp` *is shown on the screen, see figure 27-11*≫

U2': *"turn off the TV"*

S2': *"emh . . . Switched off the TV!"*

(the TV-set is powered off)

U3': *"which songs"*

S3': *"aha . . . I found these songs!"*

≪*the screenshot in figure 27-12 lists the songs by* `Ten Sharp`≫

Figure 27-11: Albums for `Ten Sharp` retrieved from the music database as response on U1' (section 27.3).

Figure 27-12: Song titles of the group `Ten Sharp`. This screen is shown as reaction on U3' (section 27.3).

(. . .)

The procedure for integrating the user input with the previous discourse is similar to section 27.1. Starting from the `discourse memory` shown in figure 27-13, first the content relevant for the integration with the user input is identified. In the setup underlying the presented dialog, a valid selection from the `discourse memory` is the tree with root-DO `MP3:*{CArtist}`$_{U1'}$. The tree

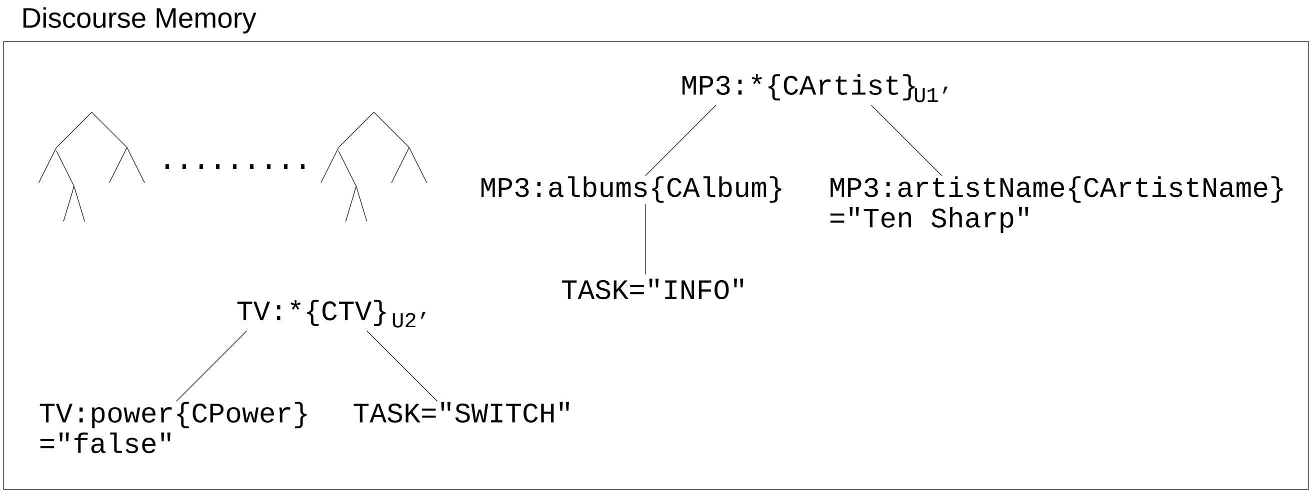

Figure 27-13: Illustration of the content from the `discourse memory` using o^2I-Trees. The state after processing U2' (section 27.3) – before U3' was received – is given in the figure.

with root `TV:*{CTV}`$_{U2'}$ is not of interest due to an incompatible mismatch of domains – the user input *"which songs"* is associated with the domain `MP3`.

Input U3' may again be represented as given in figure 27-6, analog to the representation of U7: *"which songs"* in the context of the previously shown interaction. One of the possible relations of root-DOs is analog to the relation shown in figure 27-7. Further processing is done completely analogously to the procedure in section 27.1 for integrating U7: *"which songs"* with the `discourse memory` depicted in figure 27-5.

28. Automatic Inter-Domain Sharing of Discourse Information

The combination of HD RECORDER and EPG highly benefits from the automatic sharing of discourse information. We will now look at an interaction sequence of a user with these two applications and focus on information sharing in the discussion. The session presented here starts where the user checks the TV program for the next evening.

(the system is up and running, the user may already have interacted with the system)

USER#1 (U1): *"go to the EPG"*

SYSTEM#1A (S1A): *"emh ... Please wait."*

Processing the user input resulted in an interpretation that activates the EPG application. In addition to global resources that became available when the EPG application connected to the dialog system, specialized resources are made available to enable access to the enlarged set of operations after activation.

I.e. the dialog system interacts with the EPG application. The `application management` computes updated knowledge sources, and distributes the new knowledge sources to the dialog system for use inside the affected components.

S1B: *"I updated my knowledge. ... What do you want to see?"*

≪EPG welcome screen is displayed≫

U2: *"what is on TV tomorrow"*

S2: *"ok ... I display 30 entries."*

≪the screen in figure 28-1 is shown in the web browser≫

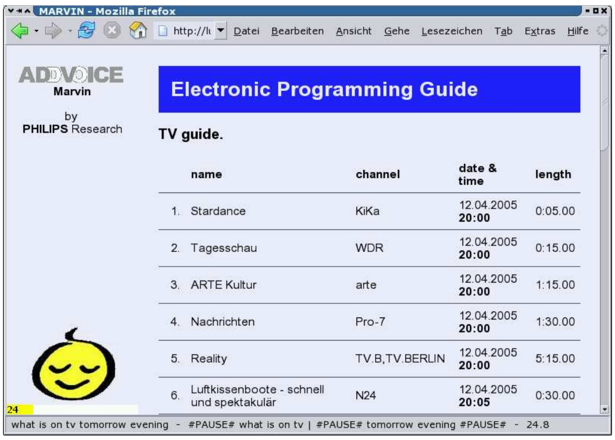

Figure 28-1: Screenshot of the response on U2: *"what is on TV tomorrow"*. The browser shows a section of a list with 30 entries.

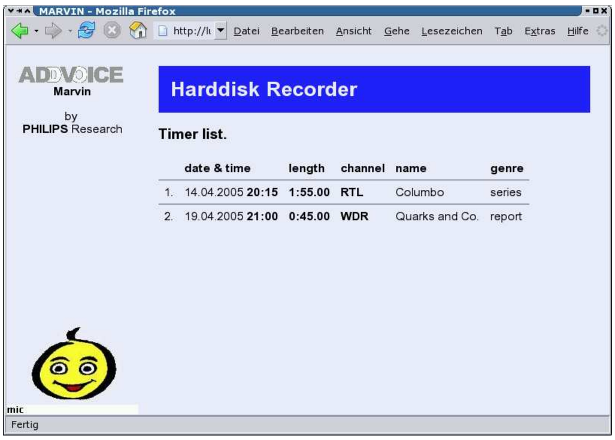

Figure 28-2: Screenshot of the response on U4: *"show my timer entries"*. Two entries are already programmed.

The user remembers that he wants to record a broadcast on channel `Pro-7`, but decides to check the already scheduled broadcasts first. The HD RECORDER, like the EPG, is an application that divides between global resources being always available and specialized resources. Therefore, the HD RECORDER needs to be activated to access the complete set of functionalities.

U3: *"go to the harddisk recorder"*

S3: *"er . . . I updated my knowledge. . . . Do you want to record a show or watch a recording?"*

U4: *"show my timer entries"*

S4: *"ok . . . I found 2 timer entries."*

≪the visual output is displayed in figure 28-2, the two timer entries present in the HD RECORDER *are shown≫*

The two applications of interest for this interaction are both accessible at this point. The user now starts to create a new timer entry and then requests support from the EPG to complete the creation.

U5: *"create a new timer entry"*

The interpretation of user input U5 represents the wish to add a new timer entry. However, due to a recognition error also `ZDF` is contained in the recognition result. The `NLU` maps it to a channel name.

S5: *"emh . . . When do you want a recording?"*

*≪the visual output of the system answer in figure 28-3 already contains an erroneous entry for the channel(-**name**): ZDF≫*

Figure 28-3: Screenshot of the response on U5: *"create a new timer entry"*. Due to misrecognition, the channel(-***name***) is specified as `ZDF`.

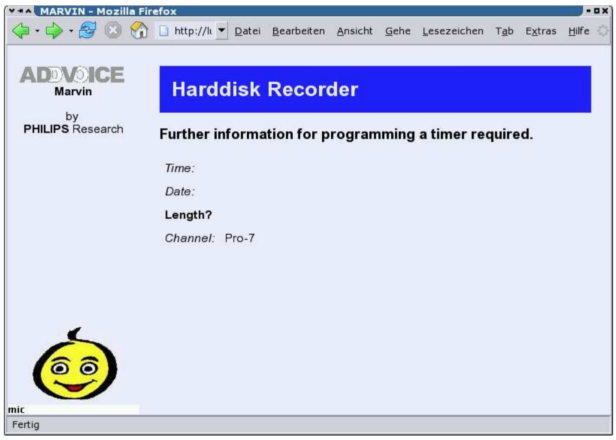

Figure 28-4: Screenshot of the response on U6: *"record on Pro-7"* to select the desired channel(-***name***).

Presently, the XSLTs explicitly formulate the acquisition strategy. The system requests the ***time*** and/or ***date*** by asking for a point in time. If the user would specify the ***time***, the application is programmed to insert the ***date*** of today if the resulting point in time is in the future, otherwise ***date*** is set to tomorrow.
The visual output explicitly indicates that the duration of the show is missing.

However, the user corrects the channel(-***name***) during the next input. The previous response strategy again applies and the result is analog to the previous response.

U6: *"record on Pro-7"*

S6: *"ok ... When do you want a recording?"*

*≪the specification of a new timer entry is shown in figure 28-4, only the channel(-**name**) is set≫*

The user reenters the application EPG. The specification of a broadcast in the `EPG` and a timer entry in the `HDRec` uses the ontological object ***channel***, consisting of channel-***name*** and channel-***number***.

The dialog system makes use of the resulting compatibility when the user accesses the program guide by automatically integrating the knowledge from one domain into the other. The constraint 'channel(-***name***) equals `Pro-7`' remains.

U7: *"what is on tv tomorrow"*

S7: *"emh ... I display 30 entries."*

≪figure 28-5 shows an excerpt from 30 entries delivered by the EPG *application, beginning with the top of the list.≫*

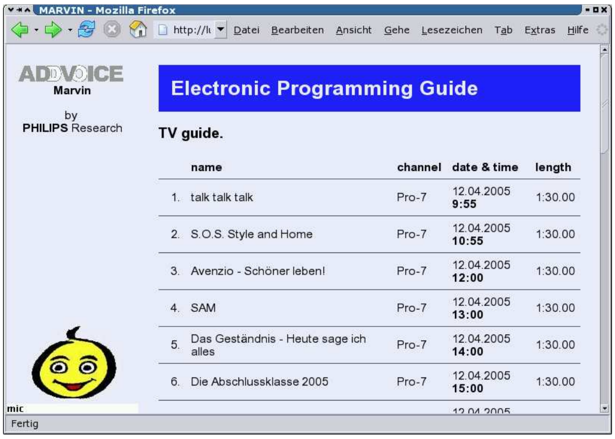

Figure 28-5: Screenshot of the response on U7: *"what is on tv tomorrow"*. Sharing of discourse information allows the EPG to take over the channel-***name*** from the domain `HDRec`.

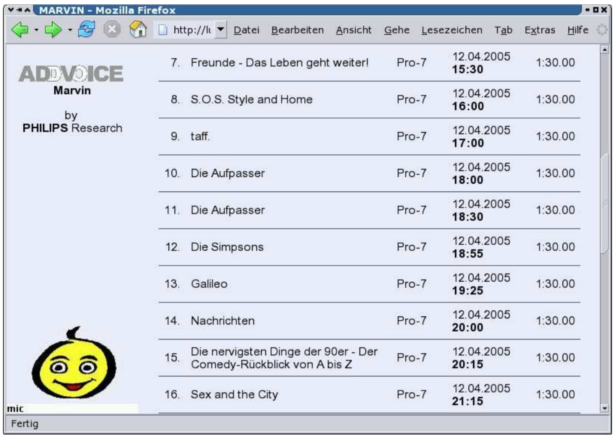

Figure 28-6: Screenshot of the reaction on U8: *"page down"*. The web browser scrolled approximately $\frac{3}{4}$ the height of the viewport down to show the next entries in the broadcast list.

U8: *"page down"*

S8: *≪after scrolling down, the next entries in the broadcast list are visible, figure 28-6≫*

One of the shows presented at the bottom of the page is the one the user searched for. He references its position in the list of broadcasts and issues the wish to record the 15th entry. I.e. the user intends to jump back from the EPG into the HD RECORDER.

U9: *"[record] entry number fifteen"*

S9: *"ok ... aha ... Details are shown on the screen."*

≪the hypothesis represented the selection of the 15th entry in the list of broadcasts, a more detailed view of this entry is shown in figure 28-7≫

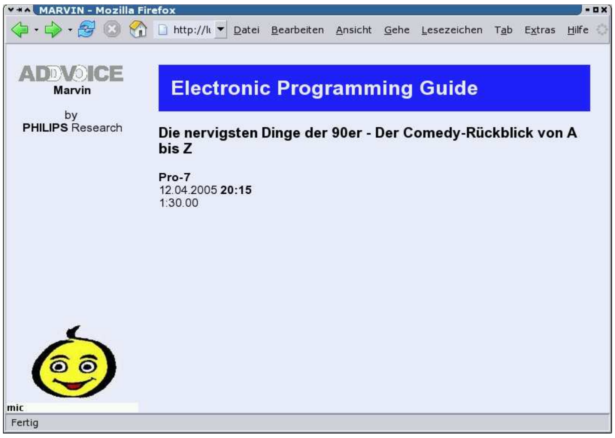

Figure 28-7: Screenshot of the reaction on U9: *"record entry number fifteen"*. The system interprets the input as *"entry number fifteen"*, therefore a detailed view on the entry is shown from the EPG perspective.

Figure 28-8: Screenshot of the reaction on U10: *"record this entry"*. The input operates on a result of the EPG application, the information is utilized to create the timer entry shown in the picture.

Even though a hypothesis is selected where the `recognition` correctly obtained *"record entry number fifteen"*, the `hypothesis selection` chose a variant where the `analysis` did not assign semantic content to the *"record"*. Thus, the interaction remained in the EPG application, and the 15th entry is selected for a more detailed view. The user intended to switch to the `HD recorder`. But actually, moving from the EPG application back into the HD RECORDER is postponed. The EPG entry number 15 is shown in details on the screen.

U10: *"record this entry"*

S10: *"ahm ... Created new timer entry."*

«the broadcast from the EPG *(figure 28-7) is taken over by the* HD RECORDER *to schedule a recording, figure 28-8»*

Creating the timer entry makes again use of sharing of discourse information. The required knowledge to program a new timer entry in the HD RECORDER is compatible with the corresponding sections of the representation of a broadcast in the EPG.

The next input queries the list of present timer entries. It also integrates knowledge from the previous discourse – yet the integrated knowledge is treated as criterion for the selection of the desired timer entries and yields to an empty selection of entries[10].

U11: *"show my timer entries"*

S11: *"ok ... I found 0 timer entries."*

«figure 28-9 shows the empty list of timer entries resulting from the integration of the previous discourse with the present user input»

Using explicit relaxation, the filter criteria are removed. The user input triggers a tag at the DO ***`timerSlots`***{`CTimerSlots`} that subsumes the timer entries, indicating that the integration with the previous discourse must not be performed on children of this DO.

[10]Note that the criteria copied from the previous discourse in principle match the newly generated timer entry. Actually, only a subset of the available information is stored for the timer entry. The current implementation compares the complete knowledge, thus this comparison fails, and no entry is selected.

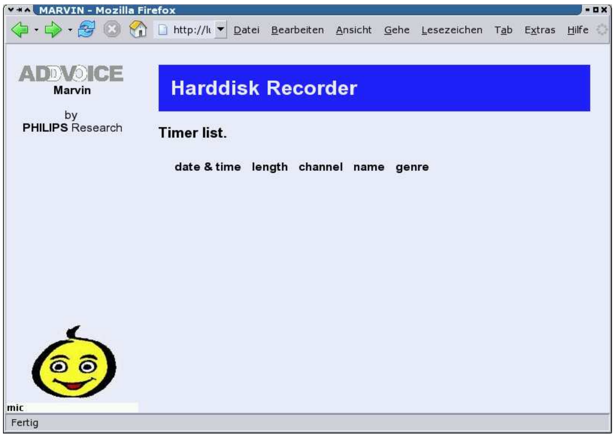

Figure 28-9: Screenshot of the reaction on U11: *"show my timer entries"*. The integration with the previous discourse defines filter criteria for the selection of timer entries, and therefore results in an empty result.

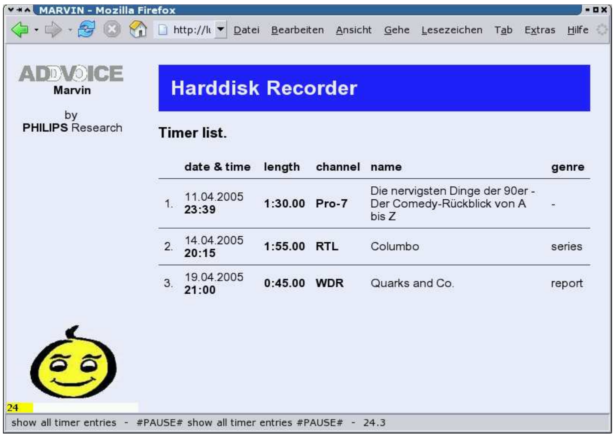

Figure 28-10: Screenshot of the reaction on U12: *"show* all *timer entries"*. The filter criteria that lead to the output in figure 28-9 are explicitly relaxed by the user.

U12: *"show all timer entries"*
S12: *"emh ... I found 3 timer entries."*
≪figure 28-10≫

Finally, the user notices an unwanted timer entry. The user removes this entry and the timer list is checked one more time.

U13: *"delete timer entry number two"*
S13: *"OK ... Deleted 1 timer entry."*
≪the system informs the user by the screen shown in figure 28-11 on the removal on an entry≫
U14: *"show all timer entries"*
S14: *"er ... I found 2 timer entries."*
≪the visual output of the timer list after removal of one entry is displayed in figure 28-12≫

Next, we elaborate on two of the phenomena that can be observed in the interaction presented before.

28.1. Sharing of Discourse Information Between the Applications EPG *and* HD Recorder

Incorporating discourse information from one domain in another domain can be observed in two places of the interaction. Between user input U6 and U7, a restriction on the channel is ported from the HD RECORDER to the EPG. From utterance U9 to U10, discourse information flows in the opposite direction of domains to schedule a new recording. Again, only effects being relevant for the finally selected hypothesis structure are considered.

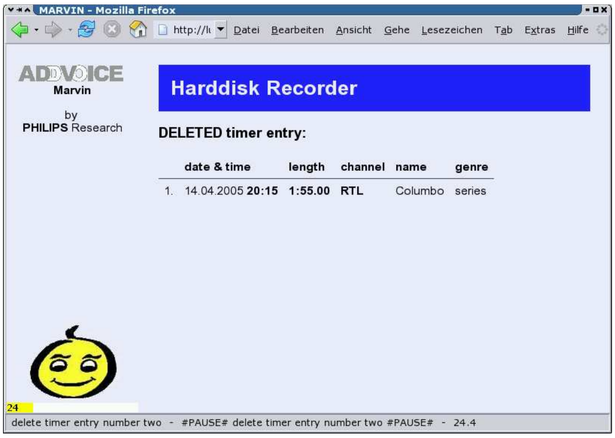

Figure 28-11: Screenshot of the reaction on U13: *"delete timer entry number two"*. The deleted entry is displayed.

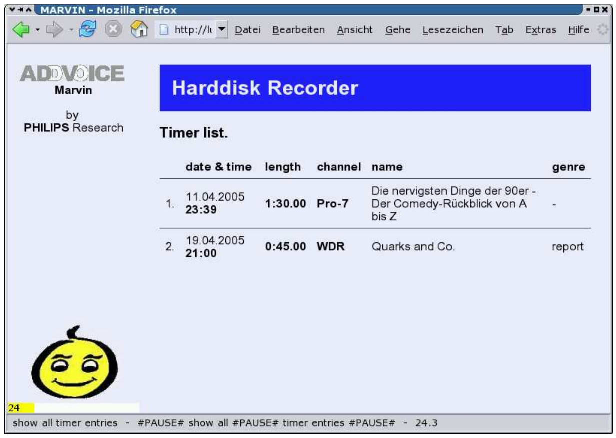

Figure 28-12: Screenshot of the reaction on U14: *"show all timer entries"* after the deletion of one entry.

'U6 → U7'. During the examination of the `discourse memory`, also the entities corresponding to the o^2I-Tree shown in figure 28-13 are considered. The root-DO `HDRec:`$*${`CTimer-`

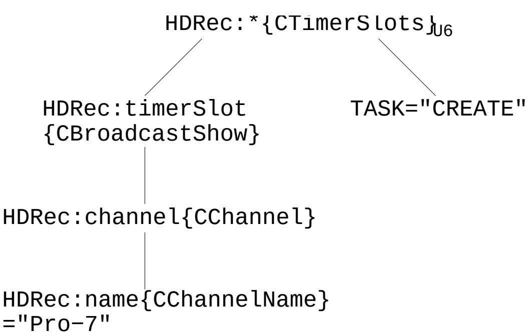

Figure 28-13: Illustration of content from the `discourse memory` that contributes to the actually selected hypothesis of input U7 (section 28). The topmost compatible DO for the EPG is `HDRec:`*`timerSlot`*{`CBroadcastShow`}.

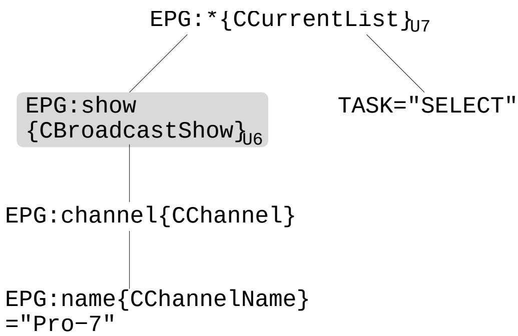

Figure 28-14: Hypothesis structure that integrates the hypothesis directly derived from the user input U7 with the compatible content from the `discourse memory` given in figure 28-13. The anchor of the integration is marked.

`Slots`}$_{U6}$ is not compatible with the `EPG` domain. However, the first proper child `HDRec:`*`timerSlot`*{`CBroadcastShow`}$_{U6}$ links to `EPG:`*`show`*{`CBroadcastShow`}.
For user input U7, one possible hypothesis structure consists of a root-DO `EPG:`$*${`CCurrentList`}$_{U7}$, and a task object TASK="SELECT" associated with this DO.
Analogously to the procedure applied in section 27.1, the hypothesis structure in figure 28-14 can be derived. It integrates the tree below `HDRec:`*`timerSlot`*{`CBroadcastShow`}$_{U6}$ into the hypothesis structure after transferring it into the domain `EPG` (marked with grey background in figure 28-14).

'U9 → U10'. The strategy for the incorporation of content from the `EPG` domain in to the `HDRec` domain is analog to the previously discussed case. The o^2I-Tree of figure 28-15 represents a section of the `discourse memory`. While investigating this figure, the topmost DO compatible to the `HDRec` domain is `EPG:`*show*{`CBroadcastShow`}$_{\mathrm{U9}}$.

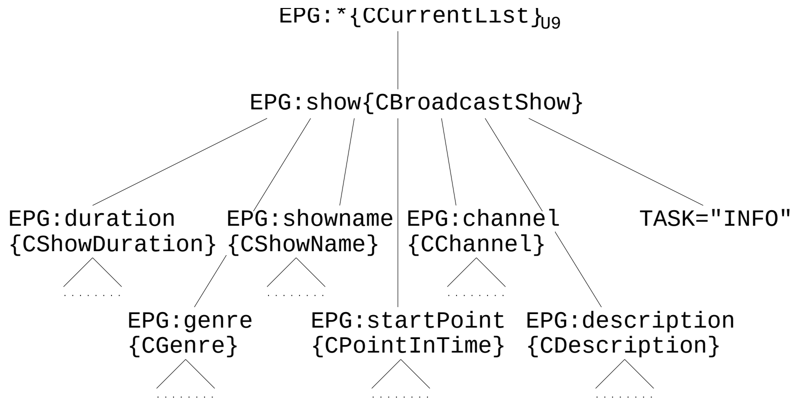

Figure 28-15: Illustration of the portion from the `discourse memory` that adds to the actually selected hypothesis structure for U10 (section 28). The topmost compatible DO needs to be identified.

The user input U10: *"record this entry"* before the integration with the previous discourse can be represented as a single DO `HDRec:*`{`CBroadcastShow`}$_{\mathrm{U10}}$ with an associated TASK=`"`CREATE`"`. The user input is again related to the relevant entities from the `discourse memory`. The finally selected hypothesis structure that is derived by the `hypothesis selection` is shown in figure 28-16. Before the integration, `EPG:`*show*{`CBroadcastShow`}$_{\mathrm{U9}}$ is transferred into the `HDRec` domain

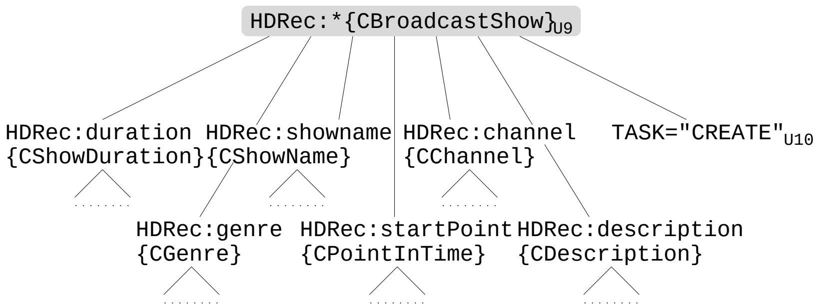

Figure 28-16: Integration of the user input U10 (section 28) with parts of the `discourse memory` leads to the hypothesis structure shown in the figure. The compatible DO from figure 28-15 merges with the single DO `HDRec:*` {`CBroadcastShow`}$_{\mathrm{U10}}$ that represents the user input.

and then merged with the interpretation of the user input obtained so far. The task CREATE lets the HD RECORDER schedule a new recording.

28.2. Resolution of Anaphoric References

Utterance U10: *"record this entry"*, which contains the demonstrative 'this', was already discussed directly beforehand in the context of sharing discourse information. Recognition and analysis need appropriate means to cope with a demonstrative. In the present implementation, a DO – `HDRec:*`{`CBroadcastShow`}$_{\mathrm{U10}}$ – is extracted for the demonstrative. The integration

algorithm already discussed before can also be applied here. After merging with the previous discourse, `HDRec:*`$\{$`CBroadcastShow`$\}_{\mathrm{U10}}$ may either be the root-DO of an originating o^2I-Tree, as in figure 28-16, or an inner object (similar to `EPG:`*`show`*$\{$`CBroadcastShow`$\}_{\mathrm{U9}}$ in figure 28-15). The handling of definite noun phrases is completely analog to the handling of a demonstrative, e.g. U10': *"record the entry"*.
The structure as such provides no hint for an application on the wanted action. Because of user input U10, a task object TASK="CREATE" is associated with `HDRec:*`$\{$`CBroadcastShow`$\}_{\mathrm{U10}}$. Thus, the resulting hypothesis structure comprises a task object, see figure 28-16. In the end, the operation performed by the HD RECORDER application is determined by the position and content of the task object in the hypothesis structure and, of course, the task type.

The resolution of a demonstrative or definite noun phrase thus relies on the concepts for the integration of the previous discourse and the current user input. Using this procedure, no explicit check is performed if the user input contained a demonstrative or definite noun phrase and, in this case, if a referenced item could successfully be identified and integrated.

Chapter IV.

Conclusions & Future Directions

Application-independent knowledge processing, the separation of dialog system and applications, and the flexibility to handle changing sets of applications affect a variety of areas in spoken language dialog processing. DYMALOG provides the comprehensive foundation to realize such a system. It addresses the crucial challenges, and also shows the benefits and the potential of such a universal approach. Furthermore, we showed that the realization of generic functionalities can benefit from this separation.
The complexity of this global approach has been broken down into separate areas for our investigations, yet these areas cannot be completely partitioned into disjoint parts. Naturally, we discovered – often strong – interdependencies between the areas. Partly, at some points the dependencies fan out as illustrated in the figures 2, 3, and 4 (pages 5, 6, and 7 in the introduction), relating the sections in this thesis.

Maintaining alternative interpretations affects major parts of the dialog system. Its data-driven optimization requires annotated corpora, and the presented parameter estimation algorithm GRAIL clearly outperformed parameter studies on such data.

Finally, the concepts for an application-blind dialog system developed in DYMALOG led to the *Marvin* dialog system. Focusing on the dialog engine AIDE, we were able to investigate its utilization in a connected environment.

29. Conclusions

We will recapitulate on DYMALOG, and the *Marvin* dialog system based on this framework, by means of the key questions formulated in section 5 (page 38).

***Q1**: Which methods allow the reduction, or even elimination, of the close interdependency between a dialog engine and its hosted applications?*

The major issues related to this question are the representation of information and processing of such information inside the dialog engine.

In this thesis, we introduced o^2I-Trees. These trees comprise the representation of application operation parameters and form the powerful basis of the application-independent operations. The o^2I-Trees are hierarchical structures with complex nodes, the dialog objects (DOs)[1]. The DOs include semantic content from the user input, the previous discourse, and the applications as well as content related to processing. The objects's domain defines its relation to the set of

[1]See definition 8.1, p. 59.

connected applications, and through instantiation a DO is connected to the 'real world'. The class property can be regarded as being the type of a DO.
In order to operate on the o^2I-Trees, major parts of such operations can be broken down to the DOs, depending mainly on their status and properties. For instance, we were able to introduce an innovative algorithm that merges the previous discourse with the present user input with minimal dependencies on the underlying applications. The integration of the trees itself is a recursive operation on the DOs, completely decoupled from the applications (see also *(iii)* below).

The o^2I-Trees are backed up by ontologies. We used ontologies for the definition of application operation parameters by each application independently. To derive the hierarchical structures, 'has-a' and 'is-a' relations connect the basic objects used inside the ontologies. These basic units link to the DOs. The ontological relations are translated into relations between DOs to build up the o^2I-Trees.
The proposed ontologies are a powerful instrument for the description of the application operation parameters. Import mechanisms allow an application designer to rely on existing parameter descriptions during the design of an application. To go beyond the representation of knowledge, also the generic handling of such imported parameters is made available for the use in the application implementation. Such building blocks are an offer to the application designer and lead to a more consistent behavior, but an application designer is free not to rely on the default building block.

The innovative o^2I-Trees show their potential at several places:

- representation of the interpretation of the user input as hypothesis structures,
- representation of the applications' processing result as result structures, a simplified variant of the hypothesis structures,
- interfacing between the dialog system and the applications through the `application management`,
- exchanging of interpretations between the modules of the dialog system, and
- foundation for several operations inside the framework.

The application-independent knowledge processing inside the dialog engine AIDE limits the dependency on the applications to a few, well-defined points of contact. More precisely, we should speak about 'minimized dependencies of the knowledge processing on the applications'. These steps are the transformation of the user input to semantic entities, i.e. `recognition` and `analysis`, the creation of relationships between the atomic DOs through the `domain model`, and the generation of a response, `generation` and `rendering`. The required knowledge sources are recognition and generation lexica, analysis and generation rules, and the relations of objects defined by the ontologies.
A major benefit of the application-independent knowledge processing based on o^2I-Trees is the automatic establishment of a relation between the current user input and the previous discourse. This corresponds to the automatic evaluation of the user input in the context of previous the discourse. DYMALOG applies a novel three-step procedure for the integration. These steps are *(i)* the identification of the content relevant for the integration process, *(ii)* the creation of relations between the 'topmost' elements, and *(iii)* the actual integration of the content from the previous discourse with the content extracted from the user input.
In the context of multi-application setups, our innovative approach allows us to expand the scope of content beyond a single application. During the integration process, the integration of the user

input and the previous discourse is not limited to matching domains. The formulation of the o^2I-Trees and the underlying ontologies together with the integration procedure on these o^2I-Trees allows to automatically share knowledge between applications. The dialog engine Aide does not rely on special rules that define processes to share the knowledge. Instead, anchors, which enable such sharing, are identified and used for the generation of additional alternative hypotheses.

Q2: *Can we generalize interaction concepts required for a variety of applications or related to meta-communication?*

The separation of the dialog system from the applications provides a good starting point to implement generic functionalities. As a showcase, the framework introduces concepts to *(a)* manipulate the state of the system itself, but also *(b)* elements that directly support user input targeted at the operation of applications, as explained next, on top of the separation of dialog engine and applications.

(a) The first area is e.g. covered by the navigation in the discourse history or the viewport control of the web-browser. In particular, the navigation feature showed its usefulness for the correction of misrecognitions, misunderstandings, and the like. I.e. it is a powerful means to correct failures. Nevertheless, it also invites the user to experiment with the system, since previous states can easily be recovered; the burden of restoring the wanted state by explicitly removing unwanted knowledge from the system and reentering the required content is drastically reduced. This manipulation of the dialog system is completely decoupled from the applications, but of course, the interaction of a user with the applications makes use of, and profits from, these tools during the discourse.

(b) To support the user in operating the applications, the realized examples of generic functionalities empower *Marvin* to automatically analyze lists in the results from the applications or to activate complex applications, which supply only a subset of their available functionality per default. DymaLog provides concepts that can be adopted and adjusted for use by the applications. The user perceives such functionality as part of an application, which consumes the functionality. These generic functionalities directed towards the applications are reusable entities.

Another type of generic functionalities are building blocks, e.g. to deal with persons or time and date. Building blocks ease the creation of applications. For the future, in general the more building blocks become available, the less effort must be spent to develop an application. Most applications in the *Marvin* dialog system already make use of building blocks. Such building blocks include the required knowledge sources, which can be included by the application specific knowledge sources, and classes with basic functionality (presently realized in `Java`). Some applications use the building blocks as they are, other applications extend the functionality. To enable the use of building blocks, besides the programmable parts of a building block also the knowledge sources, like ontologies, lexica, or output stylesheets, are modularized.

Altogether, DymaLog shows that we can generalize concepts focusing on two different aspects in an application-blind dialog framework. The first aspect is directly related to the interaction of a user with the dialog system. This aspect comprises the concepts directly addressable by the user during the interaction. Prominent examples are the navigation in the discourse and the access to list entries. The second aspect is directed towards the creation of applications and supports the application designer. Here, we have to mention building blocks in the first place. However, some of the concepts can be associated with both aspects: e.g. dealing with list items also supports an application designer by facilitating the development.

Q3: *On which principles can multi-application dialog systems base the handling of dynamic changes in the set of applications?*

The answer to this question is strongly dependent on the answer to Q1.

Dynamic changes in the set of applications require the dialog system to adapt to new conditions online. The dialog system

- needs to know how a user might address the available functionalities,
- needs to process the given user input in the light of the relevant discourse, and
- needs to react on events, either input by the user or generated by an application.

Given an appropriate division, the processing can be related either to the 'core' dialog system or to one or more applications. The novel methodology introduced by DyMaLog attaches the logic required to operate applications to an interface layer belonging to the application (or to the application itself). More general logic, not directly related to certain applications, becomes part of the application-blind dialog system. Providing the processing logic related to a certain application – which might or might not be connected to the dialog system at a certain point in time – does therefore *not* require this logic to become integral part of the processing logic inside the dialog system once the application is connected. The *Marvin* dialog system postulates that an application forms an atomic, self-contained entity, as far as possible.

The dependency of the *Marvin* dialog system on the applications is limited to knowledge sources. The dialog system in general needs background knowledge provided by an application to deal with utterances that address this application. The components, which consume these knowledge sources, need to quickly adapt to changes in the underlying knowledge sources. Since the knowledge sources from different applications need to be merged, the components, or a separate unit, must take over this integration task in a meaningful way.

Focusing on the dialog engine Aide, DyMaLog reduced the dependency on the applications to a single component, the `domain model`. Its main task is to establish relationships between the DOs.

The input processing and output creation show a stronger dependency on the knowledge sources provided by the applications than the dialog engine. The user input is prepared by the input processing for use inside the Aide. Therefore, knowledge to map the user input onto semantically meaningful entities is required, e.g. ASR lexica and NLU grammars. As the inverse operation, the output creation translates the concepts from the o^2I-Trees inside the hypothesis structures and result structures into a presentation given to the user. The dialog system needs to know how to map entities from these structures onto presentation objects.

The components inside the dialog system, which dependent on knowledge sources from the applications, are able to update their knowledge sources online. In the dialog engine and output creation, the affected components were newly developed for DyMaLog. Online updates of the knowledge sources were part of the requirement list, thus included right from the beginning. For the input processing and speech synthesis, already existing components were reused, such as the Phicos ASR, the Susi NLU, and the Festival speech synthesis. However, these components required significant enhancements to enable online changes of their knowledge sources.

At present, the `application management` prepares the knowledge sources for use within the components. This includes the integration of the various knowledge sources from the different applications, including the required bookkeeping. Usually, a large set of different applications comes along with a large complexity. By this, we mean that if the sizes of the knowledge sources increase, the chance of ambiguities for some user input increases, and also the risk for confusions

increases with the number of applications.
Among other things, a larger set of applications generally results in longer processing times, e.g. due to a higher number of cases that have to be considered or additional competing interpretations. Therefore, DymaLog parameterizes components to influence the number of newly generated hypotheses. For an online system, i.e. a system where the reaction times of this system are 'close' to a human conversation partner, faster machines are able to provide more alternative hypotheses, of course. Thus, additional interpretations are available in the hypotheses selection, the risk to overrule the 'best' interpretation too early is decreased.
To reduce the size of the active knowledge sources, larger applications allow to differentiate between global and specialized parts of the knowledge sources. The specialized parts are only activated on demand. The *Marvin* prototype showed the feasibility of this approach for a set of applications, yet the further exploration of scalability needs more research.

Q4: *How does a dialog system determine the best interpretation hypothesis for the user input, given the uncertainty and ambiguity in the underlying models and knowledge sources?*

During the integration process, DymaLog permits, and promotes, the creation of alternative interpretations of the user input in the context of the previous discourse. This strategy generally leads to a variety of competing hypotheses for the user input. The strategy for the selection of the optimal hypothesis is concentrated in a central place, the `hypothesis selection`. The modules do not need to restrict themselves on a single processing result for a given input. Presently, the system is trained to favor the integration of the user input with the most up-to-date information from the previous discourse belonging to the same application domain as the user input. I.e. due to the centralized decision on the result of the application-independent processing, we observe a consistent, predictable final outcome of this process. The outcome is predictable in a sense that the user perceives analog processing of her input, independently of the addressed application. If the system is trained to prefer the integration of the current user input with certain elements from the discourse history, the same strategy is applied to all user inputs, independently of the targeted application.

Naturally, the hypothesis generation, including especially the application-independent integration of the previous discourse and the present user input, takes a central role in DymaLog and involves several processing steps. Multi-hypotheses processing affects all modules up to the `hypothesis selection`. Permitting competing interpretations has major implications on the dialog system. For the online system, the hypotheses generated for a certain user input need to be assessed rapidly.
The hypotheses provide the basis of the actual selection of the 'best' hypothesis representing the user input. Ideally, the selection process delivers a unique outcome. DymaLog applies a parameterized measure on the hypotheses. The resulting rating describes the quality of each interpretation.
The rating is stochastically motivated. It combines a variety of single ratings. The actually applied measure is derived from an approximation of a probability measure. Therefore, we parameterized the global rating to compensate for these approximations. Due to its structure, the applied measure is easily extensible to integrate additional single ratings if required.
To determine the optimal parameter set for the selection measure, we collected corpora with the *Marvin* dialog system. The parameter set is estimated using these corpora. A prerequisite for the estimation is the annotation of the corpora. The annotation marks the hypothesis for each user input that a human annotator considers being the best representation of the input. During the annotation, we used a set of annotation rules to generate a consistent annotation. A change in these guidelines would generally lead to another parameter set for the selection measure, thus changing the selection strategy, and therefore the behavior of the dialog system. We introduced

a novel automatic rank-based parameter estimation method: GRAIL presented here clearly outperformed a major parameter study in the underlying criteria. The parameter study is heavily restricted due to the dimensions of the parameter space and the resulting computation times. The two-dimensional criterion consists of the mean rank of the marked hypothesis compared to the competing hypotheses and of the number of hypotheses with an equal rank compared to the hypothesis selected by the annotator.

Maintaining alternative interpretations influences the number of computations required to process some user input. Depending on the configuration of the modules in the dialog system, the mean number of hypotheses generated for each user input ranges from 1.9 to 155.8. To manage alternative hypotheses in a reasonable time frame, the modules, which have to deal with competing interpretations, together with the MULTIPLATFORM based PHILUS platform allow to process hypotheses in parallel. I.e. that generally a variety of hypotheses is processed simultaneously, each being at a different stage in the processing chain.
We showed that changes in the configuration of the modules, affecting the number of generated hypotheses, also affects the behavior of the system. The availability of additional hypotheses or absence of certain hypotheses can lead to different selected hypothesis compared to the 'standard' configuration of the modules. Especially, when the access to the discourse history is restricted, these differences become obvious.

Q5: *How can the proposed approaches be combined into a multi-application dialog platform?*

We presented the novel multi-application dialog framework DYMALOG, which targets at voice interfaces in connected environments with dynamic sets of applications. Clearly, this thesis covers different aspects for spoken language dialog systems, especially ranging from the conceptual and algorithmic work in the separation of the dialog engine and the applications to the mathematical formulation of the selection problem together with the corpus based methods for the parameter estimation. We were able to profitably combine these aspects in the dialog framework DYMALOG. DYMALOG introduces a dedicated application management component to mediate between the dialog engine and the applications; the `application management` routes the hypotheses and results. In addition, it makes the different knowledge sources from the applications available for the dialog system after combining them. The application-blind approach to the separation of the dialog engine AIDE and the applications proved to be a strong foundation for the realization of generic functionalities and the provision of building blocks. The first facilitates the interaction of a user with the dialog system, the later additionally supports an application developer in her development process.
Ambiguities and uncertainties in various models and knowledge sources lead to a variety of possible interpretations of the input into the system. In DYMALOG's strategy, a plurality of interpretations is derived from the input. The decision on the 'best' hypothesis is concentrated at the end of the interpretation process. I.e., DYMALOG maintains a variety of alternative interpretations for the same input, which may belong to different applications and imply different prerequisites in modules later in the processing chain. Therefore, the risk to overrule the globally 'best' hypothesis in some module due to a local sub-optimality of this hypothesis compared to other hypotheses in this module is removed[2].

DYMALOG is put into practice by the *Marvin* dialog system: the *Marvin* dialog system proves the feasibility of a unified speech interface to applications in a connected environment. Even with a basic model underlying the `communication management`, a variety of applications were successfully connected to the dialog system and operated by the user via speech. The

[2]Actually, for real-time operation the modules are capable of overruling hypotheses as mentioned earlier. Thus, the risk is strongly reduced and not completely removed in the real-time system.

DVD belonging to this thesis shows recordings from actual sessions of a user interacting with the *Marvin* dialog system.
The experiments with the *Marvin* dialog system, together with the applications prepared for use within this system, validated the potential of the underlying knowledge representation. After minor adoptions – after the system controlled the first applications – a stable state was reached soon. However, if future functionalities or applications demand for additional content in the hierarchical structures, the representation is open for corresponding extensions[3].

In order to evaluate the framework, a variety of applications is implemented, mostly related to the field of consumer electronics. The dialog system accesses the applications through a dedicated management layer. The notion of domains is used to relate objects and structures to applications. The same objects and structures can be related to different domains. In this case, competing hypotheses reflect this ambiguity. With the decision on the best hypothesis, the selection process also decides on the domains, and thus on the applications.
Preparing an application for the use with the dialog system poses only minor constraints on the development process. Generally, the application needs to implement the interface to the application management. The exchanged data is based on hypothesis structures together with result structures for the application responses, both relying on o^2I-Trees. Furthermore, a set of knowledge sources must be provided, e.g. lexica for recognition, or ontologies for the relation of DOs. As long as these constraints are fulfilled, the developer is free to choose the optimal means to realize a new application or '*Marvin* enable' an existing application. For the software part, the developer may select any programming language, yet a major advantage of object oriented languages like `Java`, `C++`, or `C#` is a canonical mapping of the DOs in the o^2I-Trees to classes, which can also reflect the relations of such DOs.
An application may comprise a dedicated hardware device, consist of a service (realized e.g. as web-service), or utilize and integrate a collection of devices or existing services. The *Marvin*-related part can be realized as a separate artifact, which controls a newly developed or existing application through a given programming interface and contains the logic needed by the application.

The selected interactions included in chapter III demonstrate a subset of the capabilities that can be presently addressed via the *Marvin* dialog system. Even though we concentrated on the dialog engine Aide and focused not so much on the input processing and output creation, we were able to realize a prototype with a variety of controlled applications related to the CE field. The excerpts show how the application-independent processing, including the generic functionalities, determines the interaction of user and applications. The applications also benefit from the 'semi-generic' functionalities, e.g. the combination of the selection of list elements, which, together with the (generic) control of the viewport inside the web-browser, allows dealing with long lists.

30. Future Directions

Several directions become available for further investigations now that DymaLog, which performs application-independent knowledge processing inside the dialog engine, provides the foundations for the separation of a dialog system from the applications.

[3]E.g. the topic of grounding was not discussed here. To enable grounding, we might enhance the DOs to contain grounding information, which the recognition and understanding parts could initially provide.

Advanced, Generalized Dialog Strategies. A major area is the realization of advanced dialog strategies. The literature contains many promising approaches, which need to be adapted and evaluated for use as part of DymaLog (see also section 17.6). For the utilization, the abstraction of the strategies (as far as possible) from the applications is required.
In this area, several topics are of particular interest. Examples are:

- realizing help functionality
  (including the scope of a help request: addressed to the dialog engine or an application; which application?),
- dealing with different users
  (either sequentially or, even more advanced, at the same time),
- addressing different applications in a single input
  (like *"turn off the radio and show channel CNN"*),
- grounding of information,
- user modeling,
- plan-recognition,
- over answering, and
- more advanced strategies for mixed-initiative communication management and improved turn-taking.

Generic Functionalities. Some of the topics could lead to an enhancement of the generic functionalities. In future scenarios, the set of applications may be extended beyond the system domain and the present CE applications. Even though the implemented applications cover different types of applications, a broader set of applications could help to identify candidates for additional generic functionalities and sharpen the existing generic concepts.

Handling of Multiple Hypotheses. Also in the field of multi-hypotheses processing, promising and exciting future directions can be identified.
A first direction is related to the *dialog strategies.* The present handling of equally rated best hypotheses could benefit from more advanced strategies, e.g. automatic identification of disambiguation features and the generation of disambiguation questions.
A second direction is a challenging extension of the use of *alternative hypotheses* also in the discourse history. Instead of storing only the best hypothesis, the first n best hypotheses for some user input might be stored. When the next user input is received, in addition to the first best hypothesis of the last input, the user input is also related with the $n-1$ rivals of this hypotheses. During the selection process, it could be decided that a rival suits better to the present user input. If the rival does not contradict the previously executed application operation, the dialog system could retrospectively alter its previous decision, and thus change its understanding of the previous interpretation in a non monotonic way.

Combination of Applications to Perform More Complex Tasks. Presently already in the focus of some research groups is the combination of several applications to perform complex tasks, either semi-automatically or automatically. The dialog system usually hides the complexity. For the *Marvin* dialog system, the canonical approach would be to assign a separate application with the task of combining existing applications, and not the dialog engine itself.

Barge-In. Another major research area DymaLog can benefit from is handling of barge-in. Especially in a dialog system that clearly separates between the dialog engine and the applications, the effects of barge-in on both units, and the required compensation operations, need to be carefully considered together.

Multi-Modal Input. A more short-term task is the addition of pointing gestures as input modalities, facilitated by the use of a web-browser as output device.

Recognition and Analysis. More detached from the dialog engine, the improvement of the recognition and analysis modules is required for user tests with 'real world users', together with a robust dialog strategy. This concerns both, the optimal creation of the knowledge sources utilized in the dialog system from the single knowledge sources delivered by the applications, and improvements in the underlying models and technology. As long as the number of applications and their functionalities is restricted, a proper dialog strategy may already allow naive users to interact with the system (as the literature and reality shows for fine tuned systems). Nevertheless, for advanced settings, further improvements in performance of ASR and NLU are essential.

Output Generation and Rendering. The use of available standardized technology in the output creation proved to be very powerful and flexible. However, more sophisticated solutions are presently being developed, e.g. to allow the user to interact with a virtual persona. Even though there is no urgent need to replace or enhance the present output creation, it would be interesting to evaluate the differences in interacting with the same framework, including aspects of 'efficiency', when different methodologies are used in the output creation. Ideally, the required knowledge sources would show a high conformance.

Chapter V.

Appendix

A1. Lattices Representing the Recognition and Analysis Result

The ASR and the NLU of the *Marvin* dialog system deploy lattices to represent their processing results. The word-lattice resulting from the recognition process serves as input for the analysis. For the utterance *"show images from taiwan"*, excerpts from these lattices are given in the figures A1-1 and A1-2.

A1.1. Word-Lattice

Figure A1-1 shows a subset of a complete word-lattice for the utterance *"show images from taiwan"*. The lattice represents alternative transcriptions of the same input due to uncertainty in the underlying models. Some edges are not shown in the figure, e.g. some redundant edges. The edges containing the text 'taiwan' (edge $5 \rightarrow 13$ and $5 \rightarrow 15$) originate from different pronunciation variants in the ASR lexicon and result in different recognition scores. The complete lattice contains 21 nodes and 35 edges.

The lattice only sketches the information content for each node and edge. In both cases, a more detailed example is also given in the figure. Another remarkable property of the graph is the recombination feature. Instead of fanning out the lattice into a tree, nodes following two different nodes may be identical, e.g. the different endpoints of the edges $5 \rightarrow 8$ 'ten' and $5 \rightarrow 11$ 'tenth' are later recombined into node 12 by the edges $8 \rightarrow 12$ 'one' and $11 \rightarrow 12$ 'one'.

A1.2. Analysis-Lattice

The lattice in figure A1-2 allows for a glimpse on a possible analysis-lattice derived from the word-lattice in figure A1-1. The additional arcs originate from ambiguities in the knowledge sources underlying the analysis process.
The illustration restricts itself to an excerpt from the complete analysis-lattice; it is limited to edges connected to the nodes contained in the best path inside the word-lattice. Even though, some edges are still omitted for clearness. Like the originator, the complete lattice has 21 nodes. However, the 35 edges from the word-lattice lead to 133 edges in the full analysis-lattice. As indicated by the excerpt from the lattice in figure A1-2, some of the resulting edges are redundant, i.e. already contained in the lattice, possibly only differing from each other in their scores.

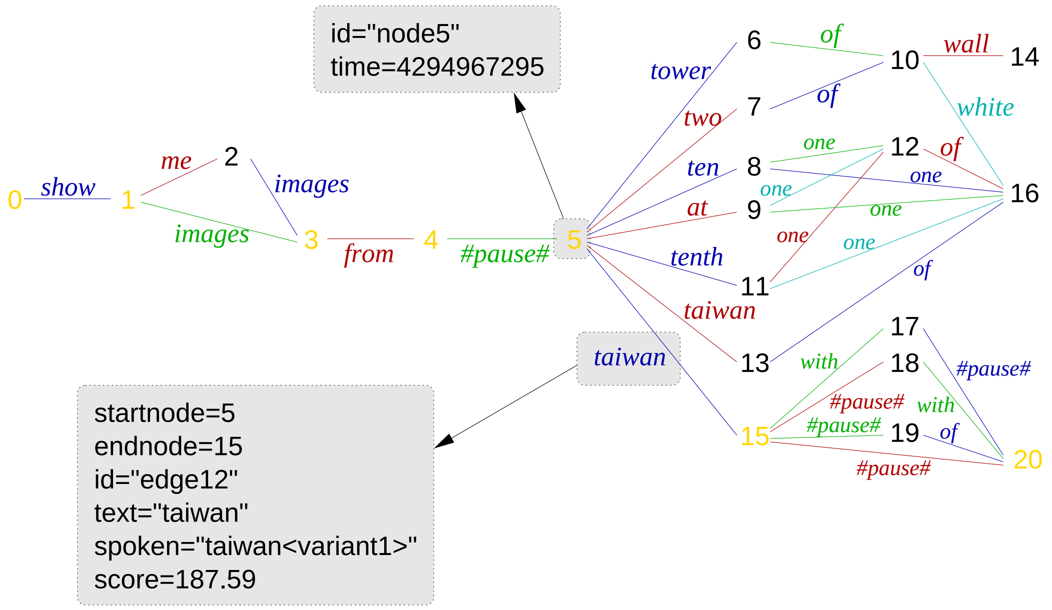

Figure A1-1: Excerpt from a word-lattice representing the speech recognition result for the utterance *"show images from taiwan"*. For clarity, only selected elements from each node and edge in the word-lattice are shown. Node 5 and edge 'taiwan' from node 5 to node 15 provide a more detailed look on the content related to a node and edge respectively in a word-lattice. The best path through the lattice according to the scores computed by the ASR is given by the node sequence $0 \rightarrow 1 \rightarrow 3 \rightarrow 4 \rightarrow 5 \rightarrow 15 \rightarrow 20$.

A2. Performance of Selected Weight Setups

The tables 13-3 and 13-5 in section 13 showed the performance of the hypothesis selection process for different weight setups on the corpora $\mathtt{C}_1$ and $\mathtt{C}_2$. For the sake of clarity, the tables provided a qualitative view on the weight setups.
Table A2-1 contains the exact weight setups, which were used during the hypothesis selection process.

A3. Code Complexity of the *Marvin* Dialog System

The PHILUS framework derived from MULTIPLATFORM together with the presently available components comprises nearly 250 000 lines of code, excluding the code from the MULTIPLATFORM framework.
The components newly developed for DYMALOG count about 57 700 lines of code (421 `C++` and `Java` classes). The components from the input processing are excluded, e.g. ASR and NLU. The dialog engine AIDE together with the output creation and the applications includes 45 200 code lines, the applications include 9 900 lines, mainly in the `Java` language. The remaining source code is contained in utilities and tools, e.g. GUIs like the hypotheses annotation tool to prepare the automatic learning of the weights, or the editor for ontologies and grammars. The rank-based parameter estimation will be listed separately. About 20 400 lines of code are written

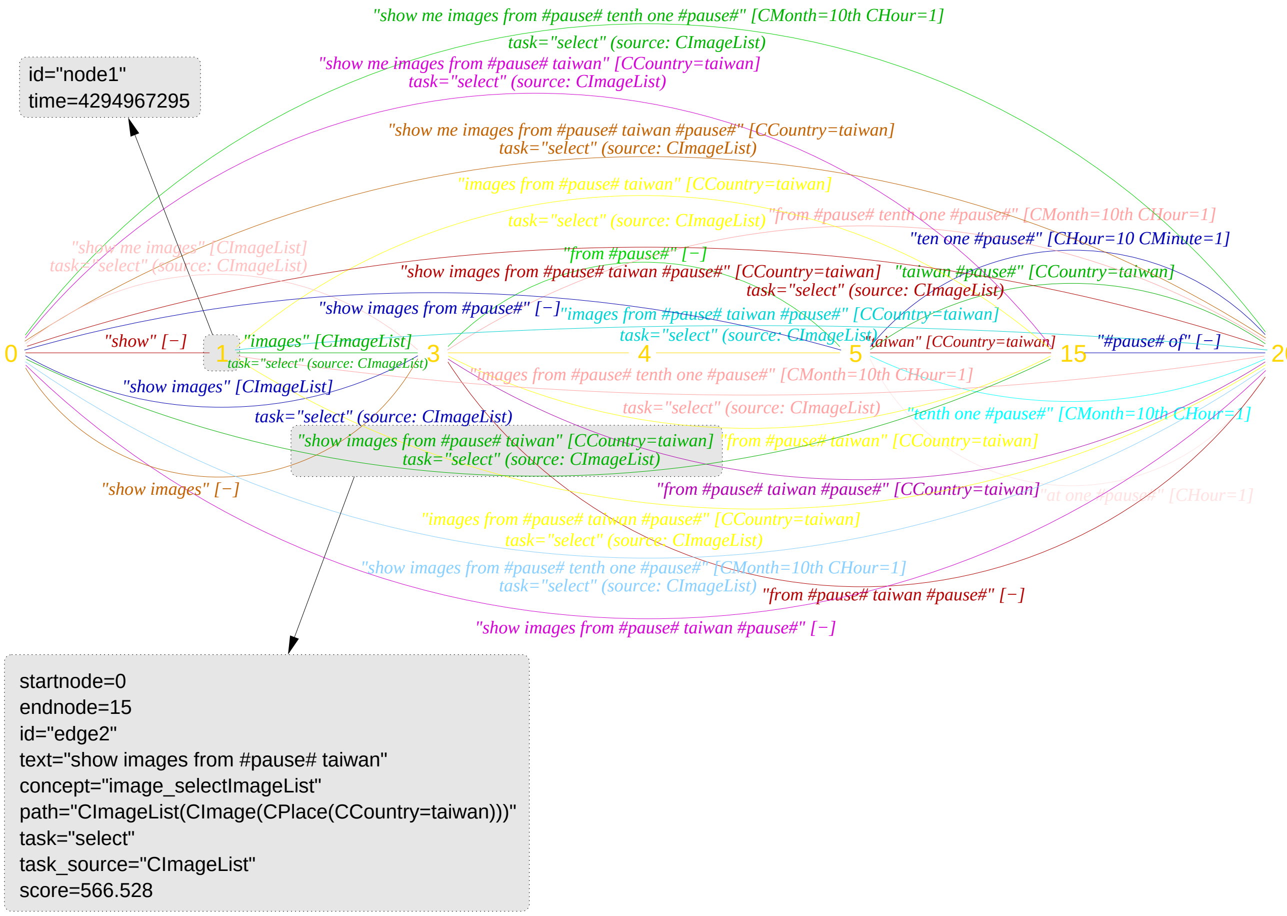

Figure A1-2: Excerpt from an analysis-lattice representing the analysis result for the utterance *"show images from taiwan"* based on the word-lattice given in figure A1-1. The excerpt is restricted to a major portion of the analysis for the best path 0 → 1 → 3 → 4 → 5 → 15 → 20 inside the word-lattice (according to the speech recognition scores). The nodes correspond to the nodes in the word-lattice. The edges contain the text sequence derived from the word-lattice (in quotes) and the computed class(es) of the entity, possibly enriched with some value (in square brackets, e.g. 'CCountry=taiwan'; a ' − ' indicates that no class was associated with a certain entity). A task is associated with some of the edges ('task="…"'), such a task is then also related to a class ('(source: …)'). A more detailed look for the information content of a node and edge respectively is given for node 1 and edge 0 → 15 'show images from #pause# taiwan'.

ω_1	ω_2	ω_3	ω_4	ω_5	ω_6	ω_7	ω_8	ω_9	ω_{10}	ω_{11}	ω_{12}	ω_{13}	ω_{14}	**mean rank**	**mean num. of rivals**
1														1.13333	5.38667
													1	1.0	10.3067
1									1					1.06667	2.50667
1							1							1.4	2.48
									1				1	1.0	5.10667
1							1		1					1.06667	1.17333
100		-1					10							1.49333	0.72
							1	1	1					1.0	4.28
100		-1						1	1					1.02667	0.72
100		-1					1		1					1.05333	0.146667
							1	1	1				1	1.0	2.68
91333.8	46.6514	-468.642	-193.493	82.9673	-6.34094	122.022	1449.98	2533.86	28315.1	7.82679	741.009	741.009	24795.5	1.01333	0.106667

Quantitative inspection of the weights sketched in table 13-3 (page 130).

ω_1	ω_2	ω_3	ω_4	ω_5	ω_6	ω_7	ω_8	ω_9	ω_{10}	ω_{11}	ω_{12}	ω_{13}	ω_{14}	**mean rank**	**mean num. of rivals**
1														2.80909	9.72727
													1	1.0	16.6182
1									1					1.76364	4.06364
1							1							1.81818	2.74545
									1				1	1.0	7.80909
1							1		1					1.53636	1.5
100		-1					10							1.8	1.05455
							1	1	1					1.0	4.3
100		-1						1	1					1.44545	1.35455
100		-1					1		1					1.52727	0.463636
							1	1	1				1	1.0	2.70909
91333.8	46.6514	-468.642	-193.493	82.9673	-6.34094	122.022	1449.98	2533.86	28315.1	7.82679	741.009	741.009	24795.5	1.1	0.381818

Quantitative inspection of the weights sketched in table 13-5 (page 133).

Table A2-1: Quantitative view on the weight setups $(\omega_M)_{\text{model } M}$ from tables 13-3 and 13-5, which provided a qualitative view.

in C/C++ and 24 800 lines are written in the Java programming language. The code of the dialog engine Aide, the output creation, and applications is distributed among 80 C++-classes and 298 Java-classes.
For the rank-based parameter estimation, the software contains about 6 200 lines of code, organized in 30 C++-classes. Of course, these numbers exclude lines from external software, like the numerical recipes package.

We counted lines in header and source code files in the C/C++ programming language, and the lines in Java source code. Comments, empty lines, and trivial lines (like bracket only lines) are excluded from counting, i.e. only 'real' code is counted. Further components, of which code is not counted, are the web-browser itself or the speech synthesis.

A4. Annotation Tools, Editors, and Viewer

A variety of tools has been developed around the *Marvin* dialog system during its evolution. In this section, we give an impression by screenshots on two editor tools that support the design of additional applications and the hypothesis annotation tool. The later is used to enable the annotation of corpora for the automatic training of parameters applied in the measure underlying the hypothesis selection process.

A4.1. Ontology Editor

Ontologies define the application operation parameters and their relations. The ontology editor show in figure A4-1 significantly simplifies the task of designing an appropriate ontology for a new application, or editing the ontology of an existing application.

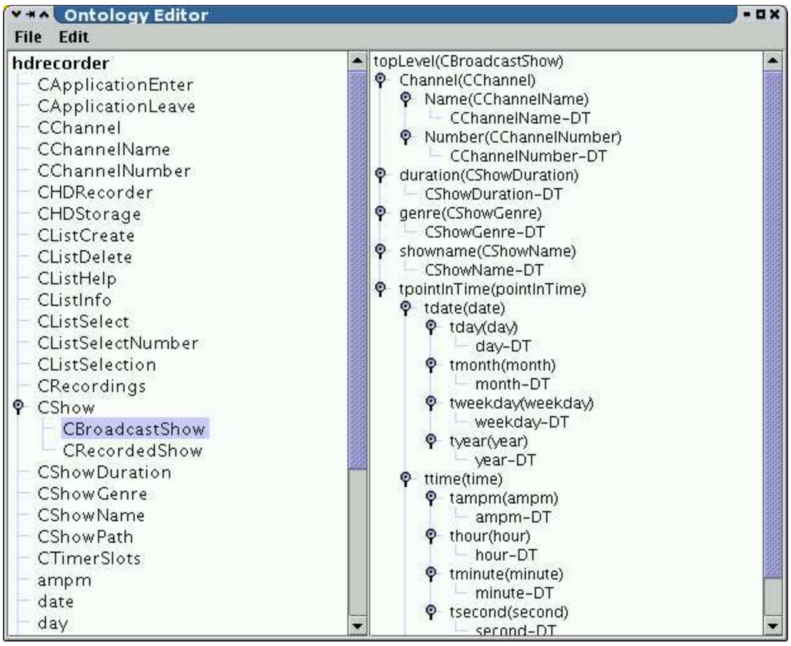

Figure A4-1: Screenshot of the ontology editor tool for showing a fragment of the domain HDRec of the HD-recorder.

The left panel shows an overview on the available ontological classes, including imported classes. Inheritance between classes is also indicated, `CBroadcastShow` is derived from `CShow`. The structure in the right panel shows the 'has-a' tree for `CBroadcastShow`, each object is represented in the form ***instance***`{class}`. The suffix '`-DT`' indicates an atomic type, e.g. string or integer.

The complete realization process for an ontology is integrated into the tool, e.g. setting up inheritance relations ('is-a') and member relations ('has-a'), and defining the value type of a leaf. It supports the association of an object with a set of allowed tasks. As already indicated, the tools allow the designer of an application to import already existing domains.

A4.2. Grammar Editor

Besides the ontologies, grammars play a central role in the creation and maintenance of applications. The grammars are also used to generate a recognition network for the ASR, and the lexicon is derived from this network.

Figure A4-2 shows a screenshot of the grammar editor. Each row stands for one rule. The

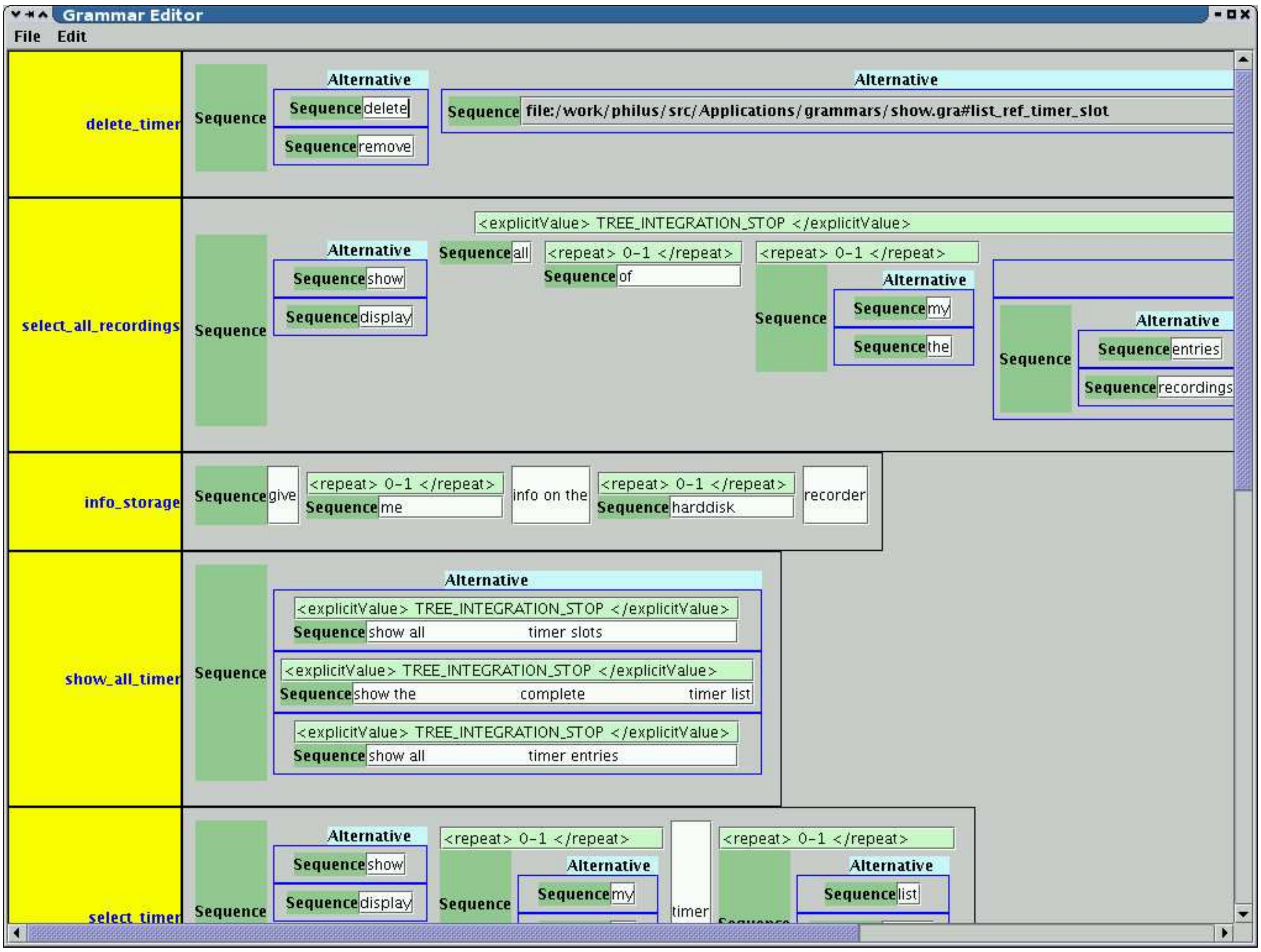

Figure A4-2: Screenshot of the grammar editor tool. The picture shows an excerpt of the HD-RECORDER application grammar.

left box in a row displays the rule's name. The right box defines the grammar rule. Rules are

associated to items from the ontology. For leaves, the procedure to extract values from the input can be defined in this editor. It is also possible to create an association to a certain task. Like for the ontology editor, the grammar editor offers the complete support for the creation and edit process, e.g. if a rule has a global or specialized scope, and means to incorporate dynamic entries from an underlying database are available.
The ontology and grammar editors are synchronized, i.e. for example that the grammar rules associated to a domain imported by the ontology editor are directly available for the grammar design after import. Therefore, rules from imported domains can easily be referenced.

A4.3. Annotation Tool for Training Corpora Creation

In order to train the parameters of the measure applied in the hypothesis selection process automatically, annotated corpora are required. For the parameter estimation introduced in section 12, in each dataset, which contains different hypotheses for some user input, one outstanding hypothesis (the 'best' hypothesis) must be identified.
The `hypothesis selection` of the *Marvin* dialog system allows to output the first n best hypotheses generated for each user input. The annotation tool shown in figure A4-3 enables the

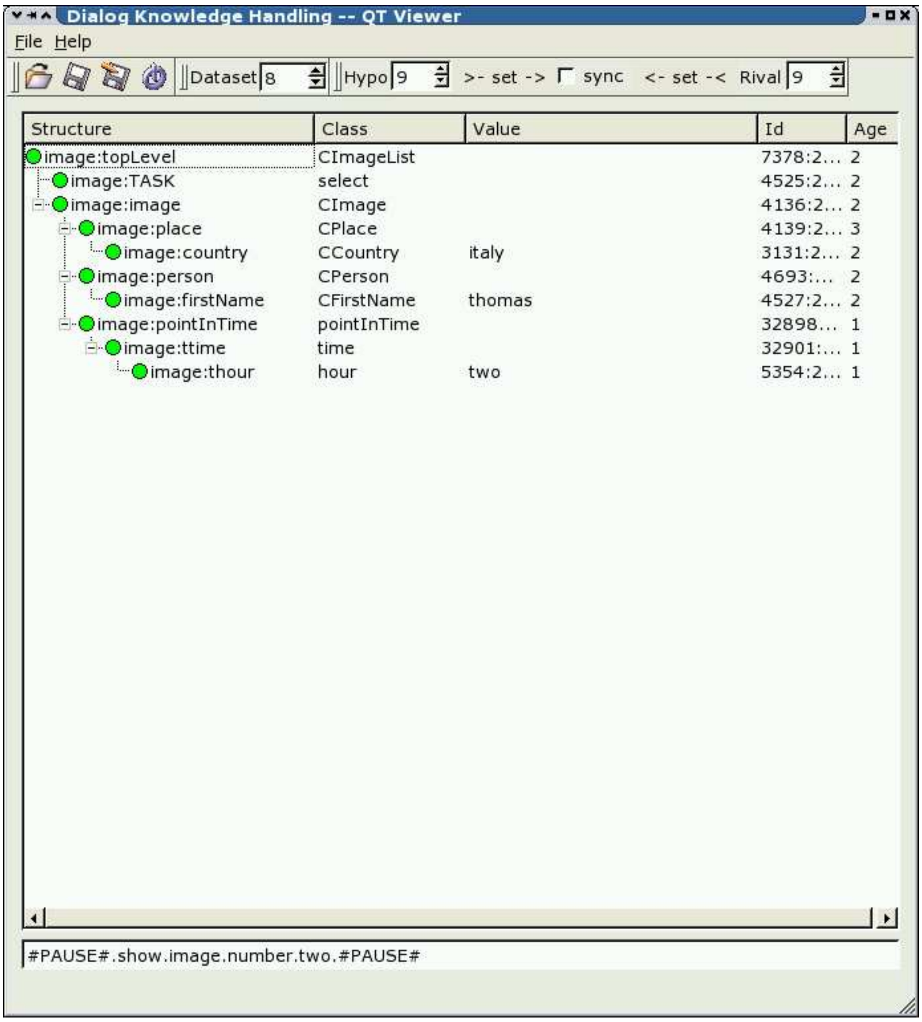

Figure A4-3: Screenshot of the hypothesis structure viewer and annotation tool. The 9th hypothesis in the 8th dataset is shown.

annotator to browse through the collected hypotheses. The structure of an o^2I-Tree is shown together with the value of leafs and age of the nodes. The 'id' is of special interest for debugging, it can be used to trace the evolution of an object over time at different processing stages.
The tool further supports the annotator in comparing competing hypotheses, figure A4-4. The

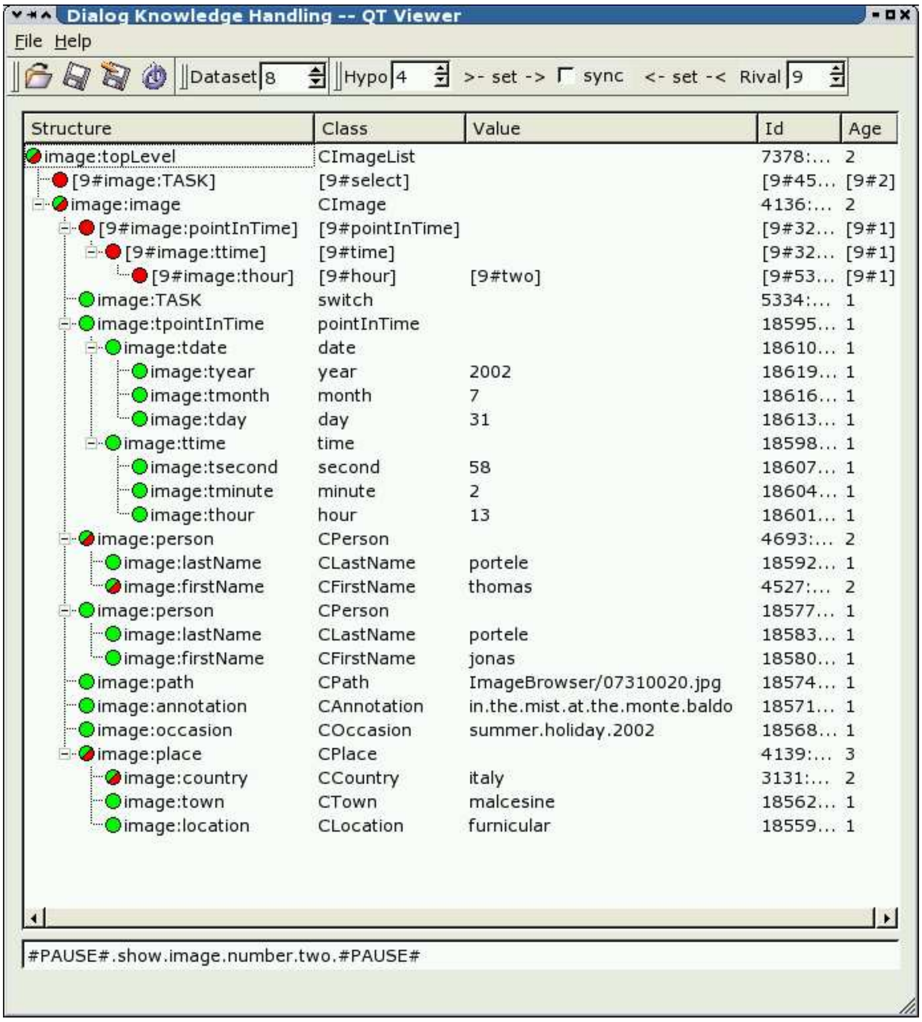

Figure A4-4: The hypothesis structures shown in the figures 13-2 and A4-3 are compared in this screenshot of the hypothesis structure viewer and annotation tool. Entities belonging to hypothesis #4 are marked by a circle (half-)filled with green color. Accordingly, the entities of the rival #9 are marked by a circle (half-)filled with red color, in addition entries of the rival #9 only entities are enclosed in squared brackets.

colored circles indicate overlap and differences between selected hypotheses.

However, the annotation remains a manual task. Introducing a new rating into the overall measure triggered a new data collection. The collected data was then annotated, and optimal parameters were estimated using Grail.

A4.4. Discourse Memory – Online Viewer

The module "`Knowledge Proc. & Disc. Memory`" allows catching a glimpse on the present state of the discourse. The presentation is to some extend similar to the view presented by the annotation tool described in section A4.3.

Figure A4-5 shows a screenshot of the memory after several turns. The viewer is updated every time the `discourse memory` integrates a new hypothesis. The relations and basic constituents of the DOs are presented.

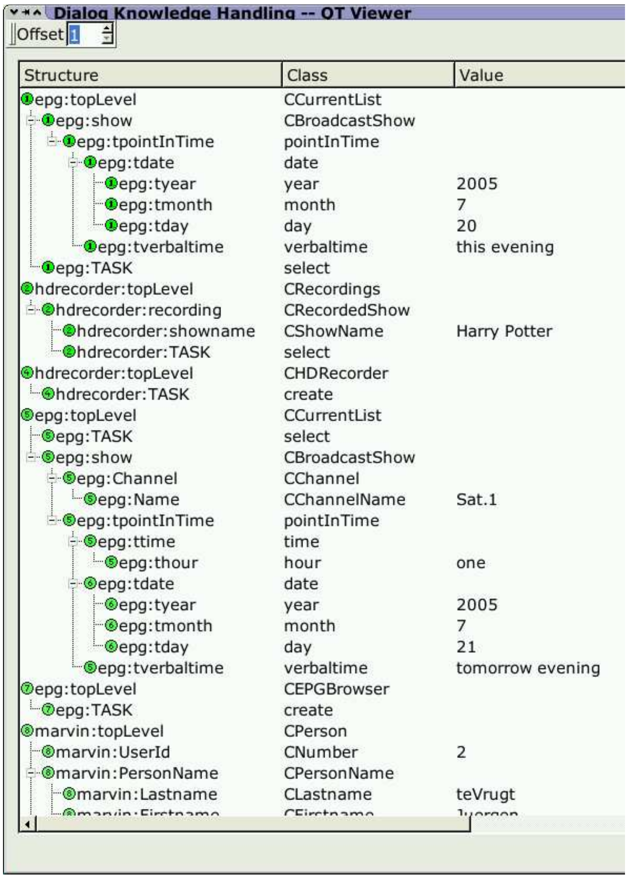

Figure A4-5: Online viewing tool to inspect the discourse state stored in the "`Knowledge Proc. & Disc. Memory`". The green circles contain numbers that indicate the age of a DO.

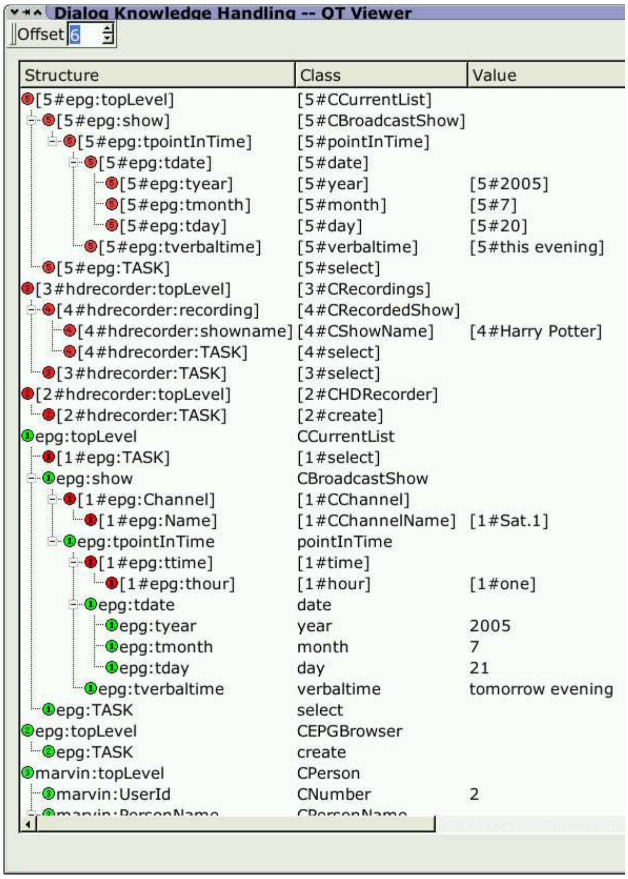

Figure A4-6: Previous states can be restored by specifying an 'offset'-age in the upper area of the window. In this example, an offset of 6 is shown (present turn: offset = 1). Red circles prefix the DOs, which will be introduced in later turns. The age given in the green circles is relative to the offset.

The viewer allows to recall previous states of the memory, figure A4-6. While navigating through the history, content that comes up in the future (relative to the selected state) remains visible in the screen, being tagged by a red circle showing a number to indicate the distance to the selected state.

The viewing tool allows a user to inspect the state of the "`Knowledge Proc. & Disc. Memory`" while interacting with the system, or directly after the interaction. It indicates the development of the `discourse memory`. As such, the integrated viewer was particularly useful during the iterative development cycles of the *Marvin* dialog system, e.g. to analyze the behavior of the system.

A5. 'Multi'- and 'Mixed'-Features in SLDS Demonstrators

Present dialog systems are used to perform research on a broad range of aspects. Among these aspects, 'multi' features listed below are frequently addressed: these aspects take the need to flexibly react on changing environments and to interact with different users into account.

For some research systems, we collected a number of such 'multi' features and present them in table A5-1. The meaning of the features listed in the table are:

- *Multi-domain.*
  Multi-domain capable systems are systems being deployed in different domains, e.g. a system that implements an interface to a cinema program information system and also an interface to a navigation application, like realized in SmartKom. We do not insist that a system covers different domains in a single instance.

- *Multi-application.*
  A system is a multi-application system if it is able to provide a user with access to different applications whereat she can switch between these applications during run time[1]. Such systems do not need to enable dynamic changes in the set of accessible applications during run time.

- *Multi-tasking.*
  Multi-tasking terms the capability of a system to switch between applications during runtime, suspending a first application in order to interact with second application, and later resuming the first application again. Thus, the term multi-tasking denotes a form of serial switching between applications during the discourse, in contrast to the multi-tasking feature of present operating systems like Microsoft Windows, where applications are executed parallel: an application may run in the background while the user interacts with another application in the foreground[2]. In our definition for the comparison in table A5-1, subdialogs are not considered to be separate applications.

- *Mixed-initiative.*
  During an interaction between a human and a machine, the human or the machine may take the initiative to drive the interaction. If the dialog system is capable of dealing with both – taking the initiative or acting in the passive role and letting the human forward the interaction – a dialog system can deal with mixed-initiative. Compared to several commercially available dialog systems where the initiative lies with the machine, such systems allow for a more flexible interaction.

- *Multi-modal.*
  A system that supports multi-modal input and output enables the usage of more than one channel for both, input and output. Frequent combinations are spoken language and pointing on a (touch-)screen as input modalities, and spoken language and graphics for the output. Of course, more advanced combinations with additional modalities can be thought of, or a system may implement multi-modality for either the input or output only.

[1]Some commercial telephony based dialog systems implement a gateway to different applications: in a first step, the user selects an application. Then, the system enables this application and the interaction of the user with the system is limited to this application. We will not consider such trivial multi-application systems here.

[2]Actually, on single processor (single core) machines, multi-tasking is also a serial process granting each application a very short time slot while cycling through all running applications.

	multi-domain[1]	multi-application[1]	multi-tasking	mixed-initiative	multi-modal	multi-party	multi-lingual[2]
Marvin	✓	✓	✓	✓	✓[3]	–	–
AdApt (Gustafson et al. [2000])	–[6]	–	–	○	✓	–	–
Galaxy (DARPA Communicator platform, Seneff et al. [1998])	✓	–	–	✓	–	–	○
GoDiS (Larsson et al. [2000])	–[6]	–	–	○	–	–	✓
Match (Johnston et al. [2002])	–	–[4]	○	✓	✓	–	–
Padis (Kellner et al. [1997])	–	–	–	✓	–	–	–
SmartKom (Wahlster [2006])	✓	✓	○	✓	✓	–	–
Spice (Kellner and Portele [2002])	–	–	–	✓	✓	–	–
Taba (Aust [1998])	–	–	–	✓	–	–	–
Verbmobil (Wahlster [2000])	✓	–	–	✓	✓[5]	–	✓

✓ feature supported
– feature not supported
○ feature support status unknown
[1] components of many systems can be adapted for use in different domains/ applications, yet often the demonstration systems are restricted to a single domain
[2] analog to [1]: the components of several systems allow translating the system to various languages, but each translation resides in a 'separate' dialog system
[3] multi-modal output is implemented, multi-modal input is prepared but presently not used
[4] access to subway *and* restaurant information
[5] Verbmobil features spoken input via different microphones, land-line phones, mobile phones, and typed input via the Internet
[6] based on TrindiKit, yet the system itself is implemented for a single domain

Table A5-1: Comparison of selected dialog systems regarding aspects that feature competing and/or complementing enhancements: 'multi' features. The Taba, Padis, and Spice systems share the same enabling technology, which has developed over time. The *Marvin* dialog system was developed in this tradition – it is partly based on the same fundamentals.

- *Multi-party.*
  Today, the robust and flexible interaction of a human with a computer is still a challenge for a dialog system. One step further, the simultaneous interaction of more than one user with a single system is even more complex (we restrict ourselves to a *single system*). E.g., such multi-party setups require adaptations in the recognition components (multi-speaker in ASR) and means to deal with the intentions and beliefs of different conversation partners in the dialog engine. The users may collaborate on a single task, each person may follow her own goal, or the users may even disturb the interaction of the other users with the system.
  First experiences in multi-party setups are obtained in meeting transcription systems, especially in the area of ASR.

- *Multi-lingual.*
  Multi-lingual dialog systems provide means to let users interact with a system in different languages. For example, consider a dialog system installed in a public place, which can be used by people speaking different languages. Ideally, such systems are capable of adapting their interaction language to the language of the present user.
  A system that can embed some words or small phrases of a foreign language into its major interaction language will not be denoted multi-lingual. E.g., a system implementing access to an EPG database for German speaking users should, to access broadcasts with non-German titles, be able to understand titles in foreign languages.

These 'multi'- and 'mixed'-features do not completely characterize a dialog system. However, they are relevant for the deployment of dialog systems in our daily life's.

List of Figures

List of Tables

Acronyms

AIDE application-independent dialog engine for dynamic environments

API application programming environment

AOP application operation parameter

ASR automatic speech recognition

CE consumer electronic

CSS cascading stylesheets

DO dialog object

DYMALOG dynamic multi-application dialog framework

GRAIL global-rank optimization algorithm

GUI graphical user interface

EPG electronic programming guide

HD harddisk

HS hypothesis structure

HTML hypertext markup language

IDE integrated development environment

o^{2}I-Tree object-oriented interpretation tree

KQML knowledge query and manipulation language

MDP Markov decision process

NLU natural language understanding

OAA open agent architecture

PC personal computer

PDA personal digital assistant

POMDP partially observable Markov decision process

RS result structure

UMTS Universal Mobile Telecommunications System

TRINDI task oriented instructional dialogue

W-LAN Wireless Local Area Network

XML extensible markup language

XSLT XML stylesheet language transformation

Index

Bibliography

Emile Aarts and Stefano Marzano, editors. *The New Everyday – Views on Ambient Intelligence.* 010 Publishers, Rotterdam, The Netherlands, February 2003.

Emile Aarts, Rick Harwig, and Martin Schuurmans. The Invisible Future. In Peter J. Denning, editor, *Ambient Intelligence*. McGraw-Hill Companies, October 2001.

Jan Alexandersson and Tilman Becker. Overlay as the Basic Operation for Discourse Processing in a Multimodal Dialogue System. In *Proceedings of the 2nd IJCAI Workshop on Knowledge and Reasoning in Practical Dialogue Systems*, pages 8–14, Seattle, Washington, August 2001.

Jan Alexandersson and Tilman Becker. The Formal Foundations Underlying Overlay. In *Proceedings of the Fifth International Workshop on Computational Semantics (IWCS-5)*, pages 22–36, Tilburg, The Netherlands, 2003.

Jan Alexandersson and Paul Heisterkamp. Some Notes on the Complexity of Dialogues. In Laila Dybkjær, Koiti Hasida, and David Traum, editors, *Proceedings of the 1st SIGdial Workshop on Discourse and Dialogue*, pages 160–169, Hong Kong, 2000. Association for Computational Linguistics.

J. Allen, D. Byron, M. Dzikovska, G. Ferguson, L. Galescu, and A. Stent. Towards a Generic Dialogue Shell. *Natural Language Engineering*, 6(3):1–16, 2000.

James Allen. *Natural Language Understanding.* The Benjamin/ Cummings Publishing Company, Inc., 2nd edition, 1995.

James F. Allen, Donna K. Byron, Myroslava Dzikovska, George Ferguson, Lucian Galescu, and Amanda Stent. Towards Conversational Human-Computer Interaction. *AI Magazine (American Association for Artificial Intelligence), Special Issue on Intelligent User Interfaces*, pages 27–37, Winter 2001.

Jens Allwood. Notes on Dialog and Cooperation. In K. Jokinen, D. Sadek, and D. Traum, editors, *Proceedings of the IJCAI Workshop on Collaboration, Cooperation, and Conflict in Dialogue Systems*, pages 9–21, Nagoya, August 1997.

Hiyan Alshawi and Shona Douglas. Variant Transduction: A Method for Rapid Development of Interactive Spoken Interfaces. In *Proceedings of the 2nd SIGDial Workshop on Discourse and Dialogue*, pages 11–19, Aalborg, Denmark, September 1–2 2001.

Egbert Ammicht, Alexandros Potamianos, and Eric Fosler-Lussier. Ambiguity Representation and Resolution in Spoken Dialogue Systems. In *Proceedings of the Eurospeech 2001 Scandinavia, 7th European Conference on Speech Communication and Technology, 2nd INTERSPEECH Event*, pages 2217–2220, Aalborg, Denmark, September 2001. 2nd INTERSPEECH Event.

Harald Aust. *Sprachverstehen und Dialogmodellierung in natürlichsprachlichen Informationssystemen*. PhD-Thesis (Aachener Informatik-Berichte), Rheinisch-Westfälische Technische Hochschule Aachen, Germany, 1998. Language: German.

Kent Bach. Communicative Intentions, Plan Recognition, and Pragmatics: Comments on Thomason and on Litman and Allen. In P. R. Cohen, J. Morgan, and M. E. Pollack, editors, *Intentions in Communication*, chapter 18, pages 389–400. MIT Press, Cambridge, MA, USA, 1990.

Amit Bagga, Tomek Strzalkowski, and G. Bowden Wise. PartsID: A Dialogue-Based System for Identifying Parts for Medical Systems. In *Proceedings of the 6th Applied Natural Language Processing Conference (ANLP)*, pages 29–36, May 2000.

Jennifer L. Beckham, Giuseppe Di Fabbrizio, and Nils Klarlund. Towards SMIL as a Foundation for Multimodal, Multimedia Applications. In *Proceedings of the Eurospeech 2001 Scandinavia, 7th European Conference on Speech Communication and Technology, 2nd INTERSPEECH Event*, pages 1363–1367, Aalborg, Denmark, September 3–7 2001.

Linda Bell and Joakim Gustafson. Positive and Negative User Feedback in a Spoken Dialogue Corpus. In *Proceedings of the 6th International Conference on Spoken Language Processing (ICSLP)*, volume 1, pages 589–592, 2000.

Nils Ole Bernsen, Hans Dybkjær, and Laila Dybkjær. Principles for the Design of Cooperative Spoken Human-Machine Dialogue. In *Proceedings of the 4th International Conference on Spoken Language Processing (ICSLP)*, pages 729–732, 1996.

Timothy Bickmore and Justine Cassell. Social Dialogue with Embodied Conversational Agents. In J. van Kuppevelt, L. Dybkjaer, and N. Bernsen, editors, *Natural, Intelligent and Effective Interaction with Multimodal Dialogue Systems*. Kluwer Academic, New York, 2006. To appear.

Alan W. Black and Paul A. Taylor. *The Festival Speech Synthesis System: System Documentation*. Human Communciation Research Centre, University of Edinburgh, Scotland, UK, 1997. HCRC/TR-83, `http://www.cstr.ed.ac.uk/projects/festival.html`.

Nate Blaylock, James Allen, and George Ferguson. Synchronization in an Asynchronous Agent-based Architecture for Dialogue Systems. In *Proceedings of the 3rd SIGdial Workshop on Discourse and Dialogue*, pages 1–10, Philadelphia, PA, 2002. Association for Computational Linguistics.

A. Blocher, J. Häußler, J. Kaiser, and N. Reithinger. SMARTKOM video demonstration. available at `http://www.smartkom.org`, 2003.

Hans Ulrich Block, Rudolf Caspari, and Stefanie Schachtl. Callable Manuals – Access to Product Documentation via Voice. *it – Information Technology*, 46(6):299–305, December 2004. Special Issue on Conversational User Interfaces.

Thorsten Bohnenberger. How Can an Intelligent Dialog System Plan Ahead? In Martin E. Müller, editor, *GI-Workshop on Adaptivity and User Modeling in Interactive Software Systems (ABIS–Adaptivität und Benutzermodellierung in interaktiven Softwaresystemen)*, volume 8th Workshop, pages 75–84, 2000. Osnabrück, Germany.

Thorsten Bohnenberger and Anthony Jameson. When Policies Are Better Than Plans: Decision-Theoretic Planning of Recommendation Sequences. In James Lester, editor, *Proceedings of the International Conference on Intelligent User Interfaces (IUI)*, volume 5th Conference, pages 21–24. Association for Computing Machinery (ACM), ACM Press, January 2001. Santa Fe, New Mexico.

Barry Brumitt, Brian Meyers, John Krumm, Amanda Kern, and Steven Shafer. EasyLiving: Technologies for Intelligent Environments. In P. Thomas and H.W. Gellersen, editors, *Proceedings of the 2nd International Symposium on Handheld and Ubiquitous Computing (HUC)*, Lecture Notes in Computer Science, pages 12–29, Bristol, UK, September 2000. Springer.

Dirk Bühler and Wolfgang Minker. An Architecture for Logic-Based Human-Machine Dialogue. In *Proceedings of the ISCA Tutorial and Research Workshop on Multi-Modal Dialogue in Mobile Environments (ITRW)*, Kloster Irsee, Germany, June 17–19 2002. Kluwer. Paper 50 (9 pp.), ISCA Archive, http://www.isca-speech.org/archive/ids_02.

Dirk Bühler and Wolfgang Minker. A Reasoning Component for Information-Seeking and Planning Dialogues. In Wolfgang Minker, Dirk Bühler, and Laila Dybkjær, editors, *Spoken Multimodal Human-Computer Dialogue in Mobile Environments*, volume 28 of *Text, Speech and Language Technology*, chapter 5, pages 55–76. Springer, The Netherlands, 2005a.

Dirk Bühler and Wolfgang Minker. The SmartKom Mobile Car Prototype System for Flexible Human-Machine Communication. In Wolfgang Minker, Dirk Bühler, and Laila Dybkjær, editors, *Spoken Multimodal Human-Computer Dialogue in Mobile Environments*, volume 28 of *Text, Speech and Language Technology*, chapter 11, pages 185–202. Springer, The Netherlands, 2005b.

H. Bunt, M. Kipp, M. Maybury, and W. Wahlster. Fusion and Coordination for Multimodal Interactive Information Presentation. In O. Stock and M. Zancanaro, editors, *Multimodal Intelligent Information Presentation*, volume 27 of *Text, Speech and Language Technology*, pages 325–340. Kluwer Academic, 2005.

Harry Bunt. Context Representation for Dialogue Management. In P. Bouquet, L. Serafini, P. Brézillon, M. Benerecetti, and F. Castellani, editors, *Proceedings of the 2nd International and Interdisciplinary Conference on Modeling and Using Context (CONTEXT)*, pages 77–90, Trento, Italy, 1999. Springer.

Carl Burke, Lisa Harper, and Dan Loehr. A Dialogue Architecture for Multimodal Control of Robots. In *Proceedings of the International CLASS Workshop on Natural, Intelligent, and

Effective Interaction in Multimodal Dialogue Systems, pages 23–26, Copenhagen, Danmark, June 2002.

Jaime G. Carbonell. Discourse pragmatics and ellipsis resolution in task-oriented natural language interfaces. In *Proceedings of the 21st Annual Meeting of the Association for Computational Linguistics (ACL)*, pages 164–168, Cambridge, Massachusetts, USA, 1983.

J. Cassell, T. Bickmore, M. Billinghurst, L. Campbell, K. Chang, H. Vilhjálmsson, and H. Yan. An Architecture for Embodied Conversational Characters. In *Proceedings of the 1st Workshop on Embodied Conversational Characters*, pages 21–30, Tahoe City, California, October 1998.

Ying Cheng, Anurag Gupta, and Raymond Lee. A Distributed Spoken User Interface Based on Open Agent Architecture (OAA). In *Proceedings of the 6th International Conference on Spoken Language Processing (ICSLP)*, volume 2, pages 743–746, Beijing, China, October 16–20 2000.

Jennifer Chu-Carroll. Form-Based Reasoning for Mixed-Initiative Dialogue Management in Information-Query Systems. In *Proceedings of the Eurospeech 1999, 6th European Conference on Speech Communication and Technology*, volume 4, pages 1519–1522, 1999.

Jennifer Chu-Carroll. MIMIC: An Adaptive Mixed Initiative Spoken Dialogue System for Information Queries. In *Proceedings of the 6th Applied Natural Language Processing Conference (ANLP)*, pages 97–104, Seattle, Washington, 2000.

Jennifer Chu-Carroll and Michael K. Brown. An Evidential Model for Tracking Initiative in Collaborative Dialogue Interactions. *User Modeling and User-Adapted Interaction*, 8(3–4): 215–254, 1998. Special Issue on Computational Models of Mixed Initiative Interaction.

Jennifer Chu-Carroll and Bob Carpenter. Dialogue Management in Vector-Based Call Routing. In *Proceedings of the 17th International Conference on Computational Linguistics (COLING)*, volume 1, pages 256–262, Montreal, Quebec, Canada, 1998.

Brady Clark, Elizabeth Owen Bratt, Oliver Lemon, Stanley Peters, Heather Pon-Barry, Zack Thomsen-Gray, and Pucktada Treeratpituk. A General Purpose Architecture for Intelligent Tutoring Systems. In *Proceedings of the International CLASS Workshop on Natural, Intelligent, and Effective Interaction in Multimodal Dialogue Systems*, Copenhagen, Denmark, June 28–29 2002.

Philip R. Cohen and Hector J. Levesque. Preliminaries to a collaborative model of dialogue. *Speech Communication*, 15(3–4):265–274, December 1994.

Dan Cristea, Nancy Ide, and Laurent Romary. Veins Theory: A Model of Global Discourse Cohesion and Coherence. In *Proceedings of the 17th International Conference on Computational Linguistics (COLING) and the 36th Annual Meeting of the Association for Computational Linguistics (ACL)*, pages 281–285, Montreal, Canada, August 1998.

Nils Dahlbäck, Annika Flycht-Eriksson, Arne Jönsson, and Pernilla Qvarfordt. An Architecture for Multi-Modal Natural Dialogue Systems. In *Proceedings of ESCA Tutorial and Research Workshop (ETRW) on Interactive Dialogue in Multi-Modal Systems*, pages 53–56, Germany, June 22–25 1999. Workshop on Interactive Dialogue in Multi-Modal Systems.

Lars Degerstedt and Arne Jönsson. Iterative Implementation of Dialogue System Modules. In *Proceedings of the Eurospeech 2001 Scandinavia, 7th European Conference on Speech Communication and Technology, 2nd INTERSPEECH Event*, pages 2193–2196, September 3–7 2001. Aalborg, Denmark.

Matthias Denecke. A Programmable Multi-Blackboard Architecture for Dialogue Processing Systems. In *Proceedings of the Workshop on Spoken Dialogue Processing, Association for Computational Linguistics (ACL)*, Madrid, Spain, 1997.

Matthias Denecke. Informational Characterization of Dialogue States. In *Proceedings of the 6th International Conference on Spoken Language Processing (ICSLP)*, volume 2, pages 114–117, Beijing, China, October 16–20 2000.

Matthias Denecke. Rapid Prototyping for Spoken Dialogue Systems. In *Proceedings of the 19th International Conference on Computational Linguistics (COLING)*, volume 1, pages 225–231, Taipei, Taiwan, August 26–30 2002.

Matthias Denecke and Alex Waibel. Dialogue Strategies Guiding Users to Their Communicative Goals. In *Proceedings of the Eurospeech 1997, 5th European Conference on Speech Communication and Technology*, volume 3, pages 1339–1342, Rhodes, Greece, September 22–25 1997.

Christine Doran, John Aberdeen, Laurie Damianos, and Lynette Hirschman. Comparing Several Aspects of Human-Computer and Human-Human Dialogues. In *Proceedings of the 2nd SIGDial Workshop on Discourse and Dialogue*, pages 48–57, Aalborg, Denmark, September 1–2 2001.

David Duff, Barbara Gates, and Susann LuperFoy. An architecture for spoken dialogue management. In *Proceedings of the 4th International Conference on Spoken Language Processing (ICSLP)*, volume 2, pages 1025–1028, Philadelphia, PA, USA, 1996.

Hans Dybkjær and Laila Dybkjær. DialogDesigner – A Tool for Rapid System Design and Evaluation. In Laila Dybkjær and Wolfgang Minker, editors, *Proceedings of the 6th SIGdial Workshop on Discourse and Dialogue*, pages 35–44, Lisbon, Portugal, September, 2–3 2005.

Laila Dybkjær, Niels Ole Bernsen, and Wolfgang Minker. Overview of Evaluation and Usability. In Wolfgang Minker, Dirk Bühler, and Laila Dybkjær, editors, *Spoken Multimodal Human-Computer Dialogue in Mobile Environments*, volume 28 of *Text, Speech and Language Technology*, pages 221–246. Springer, The Netherlands, 2005.

Jens Edlund, Gabriel Skantze, and Rolf Carlson. Higgins - A Spoken Dialogue System for Investigating Error Handling Techniques. In *Proceedings of the 8th International Conference on Spoken Language Processing (ICSLP)*, pages 229–232, Jeju Island, Korea, October 4-8 2004.

George Ferguson, James Allen, and Brad Miller. TRAINS-95: Towards a Mixed-Initiative Planning Assistant. In *Proceedings of the 3rd Conference on Artificial Intelligence Planning Systems (AIPS)*, pages 70–77, Edinburgh, Scotland, May 29–31 1996.

Festival Software. The Festival Speech Synthesis System Webpage. `http://www.cstr.ed.ac.uk/projects/festival/` and `http://www.festvox.org`, 2005.

Edward Filisko and Stephanie Seneff. A Context Resolution Server for the Galaxy Conversational Systems. In *Proceedings of the Eurospeech 2003, 8th European Conference on Speech Communication and Technology*, pages 197–200, Geneva, Switzerland, 2003.

Tim Finin, Richard Fritzson, Don McKay, and Robin McEntire. KQML as an Agent Communication Language. In *Proceedings of the 3rd International Conference on Information and Knowledge Management (CIKM)*, pages 456–463, November 1994.

Michael Fleming and Robin Cohen. Dialogue as Decision Making Under Uncertainty: The Case of Mixed-Initiative AI Systems. In *Proceedings of the NAACL-2001 Workshop on Adaptation in Dialogue Systems*, 2001.

Annika Flycht-Eriksson and Arne Jönsson. Dialogue and Domain Knowledge Management in Dialogue Systems. In *Proceedings of the 1st SIGdial Workshop on Discourse and Dialogue*, pages 121–130, 2000.

Yves Forkl and Michael Hellenschmidt. Mastering Agend Communication in EMBASSI on the Basis of a Formal Ontology. *Proceedings of the ISCA Tutorial and Research Workshop (ITRW) on Multi-Modal Dialogue in Mobile Environments (IDS)*, pages 52–54 (extended abstracts), June 17–19 2002. Paper 6 (13 pp.), ISCA Archive, http://www.isca-speech.org/archive/ids_02.

Michel Galley, Eric Fosler-Lussier, and Alexandros Potamianos. Hybrid Natural Language Generation for Spoken Dialogue Systems. In *Proceedings of the Eurospeech 2001 Scandinavia, 7th European Conference on Speech Communication and Technology, 2nd INTERSPEECH Event*, pages 1735–1738, Aalborg, Denmark, September 3–7 2001.

Andrew Garland, Lesh, Neal, and Candy Sidner. Learning Task Models for Collaborative Discourse. In *Proceedings of the NAACL-2001 Workshop on Adaptation in Dialogue Systems*, pages 25–32, June 2001. (Mitsubishi Electric Research Laboratories MERL, TR-2001-25, July 2001).

P.N. Garner, S.R. Browning, R.K. Moore, and M.J. Russell. A Theory of Word Frequencies and Its Application to Dialogue Move Recognition. In *Proceedings of the 4th International Conference on Spoken Language Processing (ICSLP)*, pages 1880–1883, Philadelphia, PA, USA, October 3–6 1996.

Egidio Giachin and Scott McGlashan. Spoken Language Dialogue Systems. In Steve Young and Gerrit Bloothooft, editors, *Corpus-Based Methods in Language and Speech Processing*, chapter 3, pages 69–117. Kluwer Academic Publishers, 1997.

James Glass and Eugene Weinstein. SPEECHBUILDER: Facilitating Spoken Dialogue System Development. In *Proceedings of the Eurospeech 2001 Scandinavia, 7th European Conference on Speech Communication and Technology, 2nd INTERSPEECH Event*, pages 1335–1339, Aalborg, Denmark, September 2001.

James Glass, Joseph Polifroni, Stephanie Seneff, and Victor Zue. Data Collection and Performance Evaluation of Spoken Dialogue Systems: The MIT Experience. In *Proceedings of the*

6th International Conference on Spoken Language Processing (ICSLP), volume 4, pages 1–4, Beijing, China, October 16–20 2000.

James R. Glass. Challenges for Spoken Dialogue Systems. In *Proceedings of the IEEE Workshop on Automatic Speech Recognition and Understanding (ASRU)*, pages 307–316, Keystone, Colorado, USA, September 1999.

Günther Görz. How Can Theory Contribute to the Construction of Scalable Speech Dialogue Systems? In *Proceedings of the First Workshop on Scalable Natural Language Understanding (ScaNaLU)*, pages 50–61, Heidelberg, Germany, May 23–24 2002.

H. Paul Grice. Speech Acts. In P. Cole and J. L. Morgan, editors, *Logic and Conversation*, volume 3 'Syntax and Semantics', pages 41–58. Academic Press, New York, NY, USA, 1975.

Olivier Grisvard and Bertrand Gaiffe. An Event-Based Dialogue Model and Its Implementation in MultiDial2. In *Proceedings of the Eurospeech 1999, 6th European Conference on Speech Communication and Technology*, pages 1155–1158, Budapest, Hungary, September 5–9 1999.

Barbara Grosz. Discourse and Dialogue. In Ronald A. Cole, Joseph Mariani, Hans Uszkoreit, Annie Zaenen, and Victor Zue, editors, *Survey of the State of the Art in Human Language Technology*, chapter 6, pages 187–208. Center for Spoken Language Understanding (CSLU), Carnegie Mellon University, Pittsburgh, PA, USA, 1996. A hardcopy of this document is available from Cambridge University Press ISBN 0-521-59277-1.

Iryna Gurevych, Robert Porzel, Elena Slinko, Norbert Pfleger, Jan Alexandersson, and Stefan Merten. Less is More: Using a Single Knowledge Representation in Dialogue Systems. In *Proceedings of the HLT-NAACL Workshop on Text Meaning*, pages 14–21, Edmonton, Canada, May 2003.

Joakim Gustafson, Linda Bell, Jonas Beskow, Johan Boye, Rolf Carlson, Jens Edlund, Björn Granström, David House, and Mats Wirén. AdApt – a Multimodal Conversational Dialogue System in an Apartment Domain. In *Proceedings of the 6th International Conference on Spoken Language Processing (ICSLP)*, volume 2, pages 134–137, Beijing, China, October 16–20 2000.

Eli Hagen and Fred Popowich. Flexible Speech Act Based Dialogue Management. In *Proceedings of the 1st SIGdial Workshop on Discourse and Dialogue*, pages 131–140, Hong Kong, October 7–8 2000.

Thorsten Herfet, Thomas Kirste, and Michael Schnaider. EMBASSI – Multimodal Assistance for Infotainment and Service Infrastructures. *Computers & Graphics*, 25(4):581–592, 2001.

G. Herzog, A. Ndiaye, S. Merten, H. Kirchmann, T. Becker, and P. Poller. Large-scale Software Integration for Spoken Language and Multimodal Dialog Systems. *Natural Language Engineering*, 10(3–4):283–305, 2004. Special Issue on Software Architecture for Language Engineering.

Julia Hirschberg, Diane Litman, and Marc Swerts. Identifying User Corrections Automatically in Spoken Dialogue Systems. In *Proceedings of the 2nd Meeting of the North American Chapter of the Association for Computational Linguistics (NAACL) on Language Technologies*, pages 208–215, Pittsburgh, Pennsylvania, USA, June 2–7 2001.

Judith Hochberg, Nanda Kambhatla, and Salim Roukos. A Flexible Framework for Developing Mixed-Initiative Dialog Systems. In *Proceedings of the 3rd SIGdial Workshop on Discourse and Dialogue*, pages 60–63, Philadelphia, USA, July 2002.

Michael Johnston, Srinivas Bangalore, Gunaranjan Vasireddy, Amanda Stent, Patrick Ehlen, Marilyn Walker, Steve Whittaker, and Preetam Maloor. MATCH: An Architecture for Multimodal Dialogue Systems. In *Proceedings of the 40th Annual Meeting of the Association for Computational Linguistics (ACL)*, pages 376–383, 2002.

Laurent Karsenty. Shifting the Design Philosophy of Spoken Natural Language Dialogue: From Invisible to Transparent Systems. *International Journal of Speech Technology*, 5(2):147–158, 2002.

A. Kellner, B. Rueber, F. Seide, and B.-H. Tran. PADIS – An Automatic Telephone Switchboard and Directory Information System. *Speech Communication*, 23(1–2):95–111, October 1997.

Andreas Kellner. Overview of System Architecture. In Wolfgang Minker, Dirk Bühler, and Laila Dybkjær, editors, *Spoken Multimodal Human-Computer Dialogue in Mobile Environments*, volume 28 of *Text, Speech and Language Technology*, chapter 7, pages 117–132. Springer, The Netherlands, 2005.

Andreas Kellner and Thomas Portele. SPICE – A Multimodal Conversational User Interface to an Electronic Program Guide. In *Proceedings of the ISCA Tutorial and Research Workshop (ITRW) on Multi-Modal Dialogue in Mobile Environments (IDS)*, pages 25–27 (extended abstracts), Kloster Irsee, Germany, June 17–19 2002. Paper 36, ISCA Archive, `http://www.isca-speech.org/archive/ids_02`.

Andreas Kellner, Sven C. Martin, Petra Philips, Thomas Portele, and Bernd Souvignier. Conversational User Interfaces for Consumer Electronics. In *Proceedings of the Philips User Interface Conference*, Eindhoven, The Netherlands, 2000. Proceedings available on CD-ROM.

Andreas Kellner, Dietrich Klakow, and Eric Thelen. The ADDVOICE Initiative: Current Issues in Conversational User Interfaces. In *Proceedings of the Philips User Interface Conference*, Eindhoven, The Netherlands, October 31–November 1 2001. Proceedings available on CD-ROM.

Katrin Kirchhoff. A Comparison of Classification Techniques for the Automatic Detection of Error Corrections in Human-Computer Dialogues. In *Proceedings of the NAACL-2001 Workshop on Adaptation in Dialogue Systems*, pages 33–40, Pittsburgh, PA, USA, June 2001.

Lori Lamel. Spoken Language Dialog System Development and Evaluation at LIMSI. In *Proceedings of the International Symposium on Spoken Dialogue*, pages 9–17, 1998.

Lori Lamel, Sophie Rosset, and Jean-Luc Gauvain. Considerations in the Design and Evaluation of Spoken Language Dialog Systems. In *Proceedings of the 6th International Conference on Spoken Language Processing (ICSLP)*, volume 4, pages 5–8, Beijing, China, October 16–20 2000.

Staffan Larsson, Peter Ljunglöf, Robin Cooper, Elisabet Engdahl, and Stina Ericsson. GoDiS - An Accommodating Dialogue System. In *Proceedings of the ANLP/NAACL Workshop on Conversational Systems*, pages 7–10, Somerset, New Jersey, 2000.

Renaud Lecœuche. Learning Optimal Dialogue Management Rules by Using Reinforcement Learning and Inductive Logic Programming. In *Proceedings of the 2nd Meeting of the North American Chapter of the Association for Computational Linguistics (NAACL) on Language Technologies*, pages 216–222, Pittsburgh, Pennsylvania, USA, June 2–7 2001.

Oliver Lemon, Anne Bracy, Alexander Gruenstein, and Stanley Peters. The WITAS Multi-Modal Dialogue System I. In *Proceedings of the Eurospeech 2001 Scandinavia, 7th European Conference on Speech Communication and Technology, 2nd INTERSPEECH Event*, pages 1559–1562, Aalborg, Denmark, 2001.

Oliver Lemon, Alexander Gruenstein, Alexis Battle, and Stanley Peters. Multi-Tasking and Collaborative Activities in Dialogue Systems. In *Proceedings of the 3rd SIGdial Workshop on Discourse and Dialogue*, pages 113–124, 2002a.

Oliver Lemon, Prashant Parikh, and Stanley Peters. Probabilistic Dialogue Modelling. In *Proceedings of the 3rd SIGdial Workshop on Discourse and Dialogue*, pages 125–128, 2002b.

Esther Levin and Roberto Pieraccini. A Stochastic Model of Computer-Human Interaction for Learning Dialogue Strategies. In *Proceedings of the Eurospeech 1997, 5th European Conference on Speech Communication and Technology*, pages 1883–1886, Rhodes, Greece, 1997.

Esther Levin, Roberto Pieraccini, and Wieland Eckert. A Stochastic Model of Human-Machine Interaction for Learning Dialog Strategies. *IEEE Transactions on Speech and Audio Processing*, 8(1):11–23, January 2000.

Bor-Shen Lin, Hsin-Min Wang, and Lin-Shan Lee. A Distributed Architecture for Cooperative Spoken Dialogue Agents with Coherent Dialogue State and History. In *Proceedings of the IEEE Workshop on Automatic Speech Recognition and Understanding (ASRU)*, pages 345–348, Keystone, Colorado, USA, December 1999.

Diane Litman, Julia Hirschberg, and Marc Swerts. Predicting User Reactions to System Error. In *Proceedings of the 39th Annual Meeting of the Association for Computational Linguistics (ACL)*, pages 370–377, Toulouse, France, July 2001.

Diane J. Litman. Improving Spoken Dialogue System Performance via Adaptation. In *Proceedings of the NAACL-2001 Workshop on Adaptation in Dialogue Systems*, Pittsburgh, PA, USA, June 2001.

Diane J. Litman and James F. Allen. Discourse Processing and Commonsense Plans. In P. R. Cohen, J. Morgan, and M. E. Pollack, editors, *Intentions in Communication*, chapter 17, pages 365–388. MIT Press, Cambridge, MA, USA, 1990.

Diane J. Litman, Michael S. Kearns, Satinder Singh, and Marilyn A. Walker. Automatic Optimization of Dialogue Management. In *Proceedings of the 18th International Conference on Computational Linguistics (COLING)*, volume 1, pages 502–508, Saarbrücken, Germany, 2000.

Karen E. Lochbaum. A Collaborative Planning Model of Intentional Structure. *Computational Linguistics*, 24(4):525–572, December 1998. Association for Computational Linguistics (ACL).

Susann LuperFoy, Dan Loehr, David Duff, Keith Miller, Florence Reeder, and Lisa Harper. An Architecture for Dialogue Management, Context Tracking, and Pragmatic Adaptation in Spoken Dialogue Systems. In *Proceedings of the 17th International Conference on Computational Linguistics (COLING)*, volume 2, pages 794–801, Montreal, Quebec, Canada, 1998.

Klaus Macherey and Hermann Ney. Scoring Criteria for Tree Based Dialogue Course Management. In *Proceedings of the ISCA Tutorial and Research Workshop (ITRW) on Multi-Modal Dialogue in Mobile Environments (IDS)*, pages 85–87 (extended abstracts), Kloster Irsee, Germany, June 17–19 2002. Paper 23 (15 pp.), ISCA Archive, `http://www.isca-speech.org/archive/ids_02`.

Klaus Macherey and Hermann Ney. Feature Functions for Tree-Based Dialogue Course Management. In Wolfgang Minker, Dirk Bühler, and Laila Dybkjær, editors, *Spoken Multimodal Human-Computer Dialogue in Mobile Environments*, volume 28 of *Text, Speech and Language Technology*, chapter 4, pages 55–76. Springer, The Netherlands, 2005.

Klaus Macherey and Hermann Ney. Features for Tree-Based Dialogue Course Management. In *Proceedings of the Eurospeech 2003, 8th European Conference on Speech Communication and Technology*, volume 1, pages 601–604, Geneva, Switzerland, September 2003.

Stéphane H. Maes. Elements of Conversational Computing - a Paradigm Shift. In *Proceedings of the 6th International Conference on Spoken Language Processing (ICSLP)*, volume 1, pages 130–133, Beijing, China, October 16–20 2000.

Colin Matheson, Massimo Poesio, and David Traum. Modelling Grounding and Discourse Obligations Using Update Rules. In *Proceedings of the 1st Meeting of the North American Chapter of the Association for Computational Linguistics (NAACL)*, pages 1–8, Seattle, Washington, USA, April 29–May 4 2000.

M.T. Maybury and W. Wahlster, editors. *Readings in Intelligent User Interfaces*. Morgan Kaufmann Press, 1998.

Michael F. McTear. Spoken Dialogue Technology: Enabling the Conversational User Interface. *ACM Computing Surveys (CSUR) Archive*, 34(1):90–169, March 2002.

Merriam-Webster OnLine. Dictionary. `http://www.m-w.org/dictionary/`, 2006. Version: March 23rd, 2006.

Renato De Mori. Recognizing and Using Knowledge Structures in Dialog Systems. *Proceedings of the IEEE Workshop on Automatic Speech Recognition and Understanding (ASRU)*, pages 297–300, 1999.

Masaaki Nagata and Tsuyoshi Morimoto. First Steps Towards Statistical Modeling of Dialogue to Predict the Speech Act Type of the Next Utterance. *Speech Communication*, 15:193–203, 1994.

David G. Novick and Stephen Sutton. What is Mixed-Initiative Interaction? In S. Haller and S. McRoy, editors, *Procedings of the AAAI Spring Symposium on Computational Models for Mixed Initiative Interaction*, pages 114–116. AAAI/MIT Press, 1997.

Yohei Okato, Keiji Kato, Mikio Yamamto, and Shuichi Itahashi. System-User Interaction and Response Strategy in Spoken Dialogue System. In *Proceedings of the 5th International Conference on Spoken Language Processing (ICSLP)*, volume 2, pages 495–498, Sydney, Australia, November 30–December 4 1998.

Tim Paek and David Maxwell Chickering. The Markov Assumption in Spoken Dialogue Management. In Laila Dybkjær and Wolfgang Minker, editors, *Proceedings of the 6th SIGdial Workshop on Discourse and Dialogue*, pages 35–44, Lisbon, Portugal, September, 2–3 2005.

Botond Pakucs. VoiceXML-Based Dynamic Plug and Play Dialogue Management for Mobile Environments. In *Proceedings of the ISCA Tutorial and Research Workshop (ITRW) on Multi-Modal Dialogue in Mobile Environments (IDS)*, pages 58–61 (extended abstracts), Kloster Irsee, Germany, June 17–19 2002. Paper 22 (11 pp.), ISCA Archive, `http://www.isca-speech.org/archive/ids_02`.

Botond Pakucs. Towards Dynamic Multi-Domain Dialogue Processing. In *Proceedings of the Eurospeech 2003, 8th European Conference on Speech Communication and Technology*, pages 741–744, Geneva, Switzerland, September 1-4 2003a.

Botond Pakucs. SesaME: A Framework for Personalized and Adaptive Speech Interfaces. In *Proceedings of the EACL Workshop on Dialogue Systems: Interaction, Adaptation and Styles of Management*, pages 95–102, Budapest, Hungary, April 2003b.

K.A. Papineni, S. Roukos, and R.T. Ward. Free-Flow Dialog Management Using Forms. In *Proceedings of the Eurospeech 1999, 6th European Conference on Speech Communication and Technology*, pages 1411–1414, Budapest, Hungary, September 5–9 1999.

Bryan Pellom, Wayne Ward, and Sameer Pradhan. The CU Communicator: An Architecture for Dialogue Systems. In *Proceedings of the 6th International Conference on Spoken Language Processing (ICSLP)*, volume 2, pages 723–726, Beijing, China, November 2000.

G.E. Pfaff, editor. *User Interface Management Systems: Proceedings of the Seeheim Workshop*, Seeheim, Germany, 1985. Springer Verlag, Berlin, Germany.

Roberto Pieraccini and Juan Huerta. Where do we go from here? Research and commercial spoken dialog systems. In Laila Dybkjær and Wolfgang Minker, editors, *Proceedings of the 6th SIGdial Workshop on Discourse and Dialogue*, pages 1–10, Lisbon, Portugal, September, 2–3 2005.

Roberto Pieraccini, Esther Levin, and Wieland Eckert. AMICA: the AT&T Mixed Initiative Conversational Architecture. In *Proceedings of the Eurospeech 1997, 5th European Conference on Speech Communication and Technology*, volume 4, pages 1875–1878, Rhodes, Greece, 1997.

Roberto Pieraccini, Sasha Caskey, Krishna Dayanidhi, Bob Carpenter, and Michael Phillips. ETUDE, a Recursive Dialog Manager with Embedded User Interface Patterns. In *Proceedings of the IEEE Workshop on Automatic Speech Recognition and Understanding (ASRU)*, pages 244–247, December 2001. Madonna di Campiglio Trento, Italy.

Roberto Pieraccini, Bob Carpenter, Eric Woudenberg, Sasha Caskey, Stephen Springer, Jonathan Bloom, and Michael Phillips. Multimodal Spoken Dialogue with Wireless Devices. In Wolfgang Minker, Dirk Bühler, and Laila Dybkjær, editors, *Spoken Multimodal Human-Computer Dialogue in Mobile Environments*, volume 28 of *Text, Speech and Language Technology*, chapter 10, pages 169–184. Springer, The Netherlands, 2005.

Massimo Poesio and David R. Traum. Conversational Actions and Discourse Situations. *Computational Intelligence*, 13(3):309–347, September 1997.

Thomas Portele, Silke Goronzy, Martin Emele, Andreas Kellner, Sunna Torge, and Jürgen te Vrugt. SMARTKOM-Home - An Advanced Multi-Modal Interface to Home Entertainment. In *Proceedings of the Eurospeech 2003, 8th European Conference on Speech Communication and Technology*, pages 1897–1900, Geneva, Switzerland, September 1–4 2003.

William H. Press, Saul A. Teukolsky, William T. Vetterling, and Brian P. Flannery. *Numerical Recipes in C++ – The Art of Scientific Computing.* Cambridge University Press, February 2002. 2nd Edition.

Dave Ragget. Task-Based Multimodal Dialogs. In *Proceedings of the International CLASS Workshop on Natural, Intelligent, and Effective Interaction in Multimodal Dialogue Systems*, pages 127–136, Copenhagen, Denmark, June 2002.

Ganesh Ramaswamy, Jan Kleindienst, Daniel Coffman, Ponani Gopalakrishnan, and Chlapathy Neti. A Pervasive Conversational Interface for Information Interaction. In *Proceedings of the Eurospeech 1999, 6th European Conference on Speech Communication and Technology*, pages 2663–2666, Budapest, Hungary, September 5–9 1999.

Manny Rayner, Beth Ann Hockey, and Frankie James. A Compact Architecture for Dialog Management Based on Scripts and Meta-Output. In *Proceedings of the 6th Applied Natural Language Processing Conference (ANLP)*, pages 112–118, Seattle, Washington, 2000.

Manny Rayner, Ian Lewin, Genevieve Gorrell, and Johan Boye. Plug and Play Speech Understanding. In J. van Kuppevelt and Ronnie Smith, editors, *Proceedings of the 2nd SIGDial Workshop on Discourse and Dialogue*, pages 126–135, Aalborg, Denmark, September 1–2 2001.

Charles Rich, Candace L. Sidner, and Neal Lesh. COLLAGEN: Applying Collaborative Discourse Theory to Human-Computer Interaction. *AI Magazine (American Association for Artificial Intelligence), Special Issue on Intelligent User Interfaces*, 22(4):15–25, November 2001. TR-2000-38.

Roni Rosenfeld, Xiaojin Zhu, Arthur Toth, Stefanie Shriver, Kevin Lenzo, and Alan W. Black. Towards a Universal Speech Interface. In *Proceedings of the 6th International Conference on Spoken Language Processing (ICSLP)*, volume 2, pages 102–105, Beijing, China, October 16–20 2000.

Nicholas Roy, Joelle Pineau, and Sebastian Thrun. Spoken Dialogue Management Using Probabilistic Reasoning. In *Proceedings of the 38th Annual Meeting of the Association for Computational Linguistics (ACL)*, pages 93–100, October 2000. Hong Kong.

A.I. Rudnicky, E. Thayer, P. Constantinides, C. Tchou, R. Shern, K. Lenzo, W. Xu, and A. Oh. Creating Natural Dialogs in the Carnegie Mellon Communicator System. In *Proceedings of the Eurospeech 1999, 6th European Conference on Speech Communication and Technology*, volume 4, pages 1531–1534, Budapest, Hungary, September 1999.

Alexander I. Rudnicky. Multimodal Dialogue Systems. In Wolfgang Minker, Dirk Bühler, and Laila Dybkjær, editors, *Spoken Multimodal Human-Computer Dialogue in Mobile Environments*, volume 28 of *Text, Speech and Language Technology*, chapter 1, pages 3–11. Springer, The Netherlands, 2005.

Dale Russell. A Domain Model centered Approach to Spoken Language Dialog Systems. In *Proceedings of the 6th International Conference on Spoken Language Processing (ICSLP)*, volume 2, pages 106–109, Beijing, China, October 16–20 2000.

Konrad Scheffler and Steve Young. Corpus-Based Dialogue Simulation for Automatic Strategy Learning and Evaluation. In *Proceedings of the NAACL-2001 Workshop on Adaptation in Dialogue Systems*, pages 64–70, Pittsburgh, PA, USA, June 4 2001.

J. Searle. *Speech Acts: An Essay in the Philosophy of Language.* Cambridge University Press, New York, USA, 1969.

S. Seneff, E. Hurley, R. Lau, C. Pao, P. Schmid, and V. Zue. Galaxy-II: A reference architecture for conversational system development. In *Proceedings of the 5th International Conference on Spoken Language Processing (ICSLP)*, volume 3, pages 931–934, Sydney, Australia, November 1998.

Stefanie Shriver, Roni Rosenfeld, Xiaojin Zhu, Arthur Toth, Alex Rudnicky, and Markus Flueckiger. Universalizing Speech: Notes from the USI Project. In *Proceedings of the Eurospeech 2001 Scandinavia, 7th European Conference on Speech Communication and Technology, 2nd INTERSPEECH Event*, pages 1563–1566, September 3–7 2001. Aalborg, Denmark.

Michael Skantze. Galatea: A Discourse Modeller Supporting Concept-level Error Handling in Spoken Dialog Systems. In Laila Dybkjær and Wolfgang Minker, editors, *Proceedings of the 6th SIGdial Workshop on Discourse and Dialogue*, pages 178–189, Lisbon, Portugal, September, 2–3 2005.

Bernd Souvignier, Andreas Kellner, Bernd Rüber, Hauke Schramm, and Frank Seide. The Thoughtful Elephant: Strategies for Spoken Dialog Systems. *IEEE Transactions on Speech and Audio Processing*, 8(1):51–62, 2000.

V. Steinbiss, H. Ney, U. Essen, B.-H. Tran, X. Aubert, C. Dugast, H.-G. Meier, M. Oerder, R. Haeb-Umbach, D. Geller, W. Höllerbauer, and H. Bartosik. Continuous Speech Dictation — From Theory to Practice. *Speech Communication*, 17(1-2):19–38, August 1995.

Paul A. Taylor, Alan Black, and Richard Caley. The Architecture of the Festival Speech Synthesis System. In *The 3rd ESCA Workshop in Speech Synthesis*, pages 147–151, Jenolan Caves, Australia, 1998.

Jürgen te Vrugt and Thomas Portele. Intention Recognition. In Wolfgang Wahlster, editor, SMARTKOM– *Foundations of Multi-Modal Dialogue Systems*. Springer, Germany, 2006. To appear.

Jürgen te Vrugt and Thomas Portele. Application-Independent Knowledge-Processing in a Task-Oriented Speech-Dialog-System. *it – Information Technology*, 46(6):306–314, December 2004. Special Issue on Conversational User Interfaces.

The EMBASSI Consortium. The EMBASSI Webpage. http://www.embassi.de, 1999–2003.

The SMARTKOM Consortium. MULTIPLATFORM (Source Code). http://multiplatform.sourceforge.net and http://sourceforge.net/projects/multiplatform, 2003.

The SMARTKOM Consortium. The SMARTKOM Webpage. http://www.smartkom.org, 1999–2003.

The TRINDI Consortium. TRINDI – *Task Oriented Instructional Dialogue*. The TRINDI Project Administrator, Department of Linguistics, Göteburg University, Box 200, S-405 30 Gothenburg, Sweden; mailto:trindi@ling.gu.se, http://www.ling.gu.se/research/project, May 2001a.

The TRINDI Consortium. TRINDIKIT. http://www.ling.gu.se/research/projects/trindi, 2001b.

Sunna Torge, Stefan Rapp, and Ralf Kompe. The Planning Component of an Intelligent Human Machine Interface in Changing Environments. In *Proceedings of the ISCA Tutorial and Research Workshop (ITRW) on Multi-Modal Dialogue in Mobile Environments (IDS)*, pages 28–30 (extended abstracts), Kloster Irsee, Germany, June 17–19 2002. Paper 40 (12 pp.), ISCA Archive, http://www.isca-speech.org/archive/ids_02.

Robert van Rooy. Conversational Implicatures. In *Proceedings of the 2nd SIGDial Workshop on Discourse and Dialogue*, pages 170–179, Aalborg, Denmark, September 1–2 2001.

Gert Veldhuijzen van Zanten. Adaptive Mixed-Initiative Dialogue Management. In *Proceedings of the 4th IEEE Workshop on Interactive Voice Technology for Telecommunications Applications (IVTTA)*, pages 65–70, Torino, Italy, September 29–30 1998.

Walther von Hahn, Wolfgang Hoeppner, Anthony Jameson, and Wolfgang Wahlster. The Anatomy of the Natural Language Dialogue System HAM-RPM. In L. Bolc, editor, *Natural Language Based Computer Systems*, pages 119–253. Hanser/ Macmillan, München, Germany, 1980.

Wolfgang Wahlster, editor. VERBMOBIL – *Foundations of Speech-to-Speech Translation*. Springer-Verlag, Berlin/Heidelberg, 2000.

Wolfgang Wahlster, editor. SmartKom– *Foundations of Multi-Modal Dialogue Systems.* Springer, Germany, 2006. To appear.

Wolfgang Wahlster and Rainer Wasinger. The Anthropomorphized Product Shelf: Symmetric Multimodal Interaction with Instrumented Environments. In E. Aarts and J. Encarnaçao, editors, *True Visions: The Emergence of Ambient Intelligence.* Springer, Heidelberg, Berlin, New York, Februar 2006.

Wolfgang Wahlster, Norbert Reithinger, and Anselm Blocher. SmartKom: Multimodal Communication with a Life-Like Character. In *Proceedings of the Eurospeech 2001 Scandinavia, 7th European Conference on Speech Communication and Technology, 2nd INTERSPEECH Event*, volume 3, pages 1547–1550, Aalborg, Denmark, 2001.

Wikipedia. The Free Encyclopedia. World Wide Web (WWW), August/September 2005. `http://www.wikipedia.org`.

Helen Wright Hastie, Rashmi Prasad, and Marilyn Walker. What's the Trouble: Automatically Identifying Problematic Dialogues in DARPA Communicator Dialogue Systems. In *Proceedings of the 40th Annual Meeting of the Association for Computational Linguistics (ACL)*, pages 384–391, Philadelphia, USA, July 2002.

Yunbiao Xu, Masahiro Araki, and Yasuhisa Niimi. A Multilingual-Supporting Dialog System Using a Common Dialog Controller. In *Proceedings of the Eurospeech 2001 Scandinavia, 7th European Conference on Speech Communication and Technology, 2nd INTERSPEECH Event*, pages 1283–1286, Aalborg, Denmark, September 3–7 2001.

Steve J. Young. Probabilistic Methods in Spoken Dialogue Systems. *Philosophical Transactions of the Royal Society (Series A)*, 358(1769):1389–1402, September 1999.

Bo Zhang, Qingsheng Cai, Jianfeng Mao, Eric Chang, and Baining Guo. Spoken Dialogue Management as Planning and Acting under Uncertainty. In *Proceedings of the Eurospeech 2001 Scandinavia, 7th European Conference on Speech Communication and Technology, 2nd INTERSPEECH Event*, pages 2169–2172, Aalborg, Denmark, September 3–7 2001.